Automatic rigging of animation skeletons with stacked hourglass networks and curve skeleton extraction

by

Robert F. Stewart

B.S., Kansas State University, 2017

B.S., Kansas State University, 2017

A THESIS

submitted in partial fulfillment of the
requirements for the degree

MASTER OF SCIENCE

Department of Computer Science
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2021

Approved by:

Major Professor
William H. Hsu

Copyright

Abstract

This thesis presents an approach to the inverse rigging problem that combines ways to extract skeleton poses using deep learning pose estimation networks. This process uses graphical modelling software like Blender and Autodesk Maya with Python library modules that implement mesh contraction algorithms. Mesh contraction processes can produce and extract curve skeletons. Curve skeletons contain properties consistent with general animated rig principles and provide valuable information about medial surfaces, including topological knowledge. They are locally centered, fully connected, and invariant to pose and scale. I extracted curve skeletons from a dataset of model resources and included the curve skeletons as a geometric input feature to a Stacked Hourglass Network. Stacked Hourglass Networks are state-of-the-art architectures that can identify joint and bone locations while learning local detailing within the global context. I adapted a variant Stacked Hourglass network with 3D volumetric input representation and experimented with five combinations of geometric features: topology with signed distance function, topology with signed distance function and (2) principle surface curvatures, topology with signed distance function and local vertex density, topology with signed distance function and local shape diameter, and topology with all geometric features (signed distance function, (2) principle surface curvatures, local vertex density, and local shape diameter). My results show topological information obtained from curve skeletons could be more useful than any combination of these features. I demonstrated and compared my results between ground-truth rigs and network predicted rigs using the average chamfer distance metric.

Table of Contents

List of Figures

List of Tables

Acknowledgments

I begin by thanking Xu et al.[1] for their work and access to source code, without which this thesis would not have been possible. I would also like to thank Srinivasan Ramachandran[2] and Philipp Schlegel[3] for allowing me to use their Python libraries for mesh contraction and skeleton extraction. I am much obliged to my major professor, Dr. William Hsu, for his advice and support these last two years and to my committee members, Dr. Mitchell Neilsen and Dr. Doina Caragea, for their guidance and willingness to take me under their wings.

Dedication

I dedicate this work to my parents: John and Marta Stewart. Without their love and sacrifice throughout the years, I would never have reached this milestone. I also dedicate this to my elder brothers: William and Patrick. Their encouragement have allowed me to pursue this dream. Now that it has finally come to fruition, I am no longer the baby brother. Finally, I also dedicate this to my late grandfather, Martin Burke.

Chapter 1

Introduction

Animation involves constructing a framework to govern pose, movement, and kinematics of a computer model, or mesh. The first process in building a framework is to develop a *rig*, a hierarchical virtual frame, embedded in the mesh, to serve as the skeletal structure of the object. A rig comprises a series joints and bones. Joints are vertices placed in an object, key locations that can translate or rotate parts of the body; bones are edges directly connecting joints and helping form substructures that move in conjunction with their parent joint (e.g., the hand and forearm move when a transformation is applied to the elbow). Skeletal animation in computer graphics has, however, long been painstaking and arduous in both academic research and industry.

1.1 Problem Statement and Scope

Reducing the complexity of the rigging task and helping to automate the rig construction process can be achieved by better understanding the medial surface of the object mesh, leading to better predictive accuracy for both joint and bone targets if applied as a feature in deep learning pose estimation models. In this thesis, I expanded on the approach proposed by Xu et al. (2019)[1] using volumetric CNNs with Stacked Hourglass networks and introducing topological knowledge as an additional geometric input. Mesh topology can be learned

through skeleton extraction methods that produce *curve skeletons*, stick-like 1D representations of 3D objects that capture the essential topology of the underlying object (Cornea et al., 2007)[4]. Skeleton extraction methods include approaches like mesh contraction (Au et al., 2008)[5], radial basis functions (Ma et al., 2008)[6], and incomplete point clouds (Tagliasacchi, 2009)[7]. The experiment designed in this thesis is used to extend the current state-of-the-art in automatic rigging by addressing the following research problems:

1. Obtain awareness of mesh topology and optimize mesh contraction run-time parameters to maximize fidelity of extracted curve skeletons in relation to their original mesh.

2. Develop deep pose estimation models with a high level of generality capable of producing versatile and robust rigs for various mesh types.

3. Investigate threshold limits of general rig criteria safeguards.

1.2 Motivation

In general, a rig has three basic criteria, which I have adapted from Bhatti et al. (2015)[8]:

1. The rig should be consistent and not cause behavioral transformation in unorthodox ways;

2. The rig should be predictable, meaning each joint and bone operates as intended;

3. The rig abides by single responsibility, i.e., one manipulator or controller per joint and one edge per bone.

Furthermore, the rig must include such properties as deformation of the mesh, applying rotational constraints, weight painting, skinning, and kinematics (Zeman, 2015)[9]. The current practice of constructing rigs for skeletonization is to tailor-make each rig to meet the model's specifications (Baran & Popovic, 2007)[10]. Often, this requires using a character rigger, someone who specializes in understanding geometry, physics, and anatomy to construct

the rig's framework and model its behavior believably and realistically. Not only is this complex and tedious, but, most importantly, it is time-consuming. This is compounded because each mesh requires its own rig for animation. Efforts to automate the rigging task thus far include skeleton embedding systems like the one used for Pinocchio (Baran & Popovic, 2007)[10]; widget creation structures (Bhatti & Shah, 2012)[11]; template-based inverse kinematics (Pantuwong & Sugimoto, 2012)[12]; machine learning systems like Mixamo (Adobe, 2020)[13]; and more recently, deep learning pose estimation models (Xu et al., 2019)[1].

After surveying the literature, automatic rigging most closely correlates to pose estimation. *Pose estimation* localizes joints in computer imaging (Toshev & Szegedy, 2014)[14]. Deep learning pose estimation networks approximate landmark locations in 2D pixels and 3D voxels for articulating limbs and identifying postures (Newell et al., 2016)[15], who also proposed Stacked Hourglass networks, a state-of-the-art approach for pose estimation in RGB imaging (Yang et al., 2017)[16]. The hourglass design uses autoencoders to contextualize the image within the global scope, capturing features at every scale of the resolution, while simultaneously using residual blocks to maintain knowledge of local details. These hourglass modules are then stacked, creating a series of modules where the output approximations of joint locations can be further evaluated and fine-tuned.

The approach proposed by Xu et al. (2019)[1] is a variant of the Stacked Hourglass network using 3D volumetric input representation and a second output branch for approximating bone landmarks. For every voxel, Xu et al. (2019)[1] included five geometric input features: signed distance function, two principal surface curvatures, local vertex density, and local shape diameter. Xu et al. (2019)[1] claimed each of these geometric input features increased the predictive accuracy of joint and bone locations, but none of these geometric features analyzed or obtained direct information about the medial surface, the surface portion of the body that comprises the middle of the shape (Dey & Sun, 2006)[17]. This omission is crucial because the ideal location for the animation rig is generally along the medial surface of the mesh.

1.3 Overview of Contributions

My research contributes to the field by showing how skeleton construction can be facilitated by utilizing mesh contraction algorithms and deep pose estimation models and by expanding on the work of Newell et al. (2016)[15] and Xu et al. (2019)[1]. The following list provides specific underlying contributions.

1. In this system, I extract the curve skeletons of each model in the dataset, provided by Xu et al. (2019)[1] using two mesh contraction implementations: Mesh Skeletonization (Ramachandran, 2018)[2] and Skeletor (Schlegel, 2020)[3]. *Mesh Skeletonization* is a Python package developed as a plugin for Blender 2.79. *Skeletor* is a modifed variant of MeshSkeletonization but developed as a more general use library for extracting skeletons. They are both implementations of the mesh contraction algorithm proposed by Au et al. (2008)[5].

2. I perform a parameter grid search for Mesh Skeletonization to optimize the extracted curve skeletons favoring a strong mesh contraction and increased smoothness for more accurate representation.

3. I adapt the hourglass module implemented by Xu et al. (2019)[1] to include the extracted curve skeletons as an additional geometric input feature for every voxel of a given model.

4. I perform an ablation study to determine the optimal threshold value for the Soft Non-Maximum Suppression method used by Xu et al. (2019)[1] to suppress duplicate joint vertices.

5. I perform five experiments using a combination of the topology data with each geometric input feature, including topology with principal surface curvature, topology with local vertex density, topology with local shape diameter, topology with all features, and topology only. Signed distance is also included in all experiments due to its innate requirement for converting a 3D mesh into volumetric representation.

6. I evaluate the results of my experiments by calculating the average chamfer distance between the ground-truth rig of the model with the predicted rig outputted by the altered Stacked Hourglass network.

Chapter 2

Background and Related Work

Skeletons provide intuitive, compact representations of an object; however, a rigorous definition of the word skeleton itself is still under debate (Golland et al., 2000)[18]. Dey and Sun (2006)[17] attempted to define curve skeletons as a subset of the medial surface defined by a medial geodesic function. This definition was disputed by Cornea et al. (2007)[4] because some existing, limited cases of line segments belonged to the curve skeleton but not the medial surface. The literature has since shied away from attempting to define such terms completely and has instead explored more underlying structures and properties that may apply to geometrically discrete or continuous cases.

2.1 Curve Skeleton Extraction Methods

Skeleton extraction algorithms have existed for several decades. Many of the more recent algorithms for 3D space often produce curve skeletons (Cornea et al., 2007)[4]. These simplified models are useful in many computer vision tasks, including object detection and recognition, shape matching, navigation, reduced-model formulation, and computer animation (Wang & Lee, 2008)[19]. Curve skeletons, however, have certain desirable properties that are particularly useful in animation. These properties include *homotopic* (topology preserving), invariant under isometric transformation, locally centered in the shape, fully connected, and

robust to posing and scaling (Cornea et al., 2007)[20]. These characteristics are pivotal to creating a rig for animation that encompasses general rig criteria.

Geometric extraction algorithms that use Voronoi diagrams have been popular (Au et al., 2008)[5]. The skeleton is extracted by approximating the medial surface and then using a Voronoi diagram to extract internal faces and edges connecting vertex sample points (Amenta et al., 2001)[21]. Although discrepancies occur between a curve skeleton and the medial surface of a 3D surface, approximations of the medial surface correlate well to a compact representation of the mesh. However, approximations straight from a Voronoi diagram are no guarantee of convergence to the medial surface as only a subset of Voronoi vertices converge to the medial surface (Dey & Zhao, 2004)[22].

In the literature, we also find proposals for 3D thinning algorithms. *Thinning* is a process by which pixels are removed from a figure when those pixels are unnecessary to topology preservation or are protrusions from the figure; these pixels are assigned to the figure's complement (Arcelli & Baja, 1985)[23]. In 3D thinning, the process is similar but uses voxels instead of pixels. The algorithm seeks to erode an object's layers, template-by-template, until theoretically only the skeleton remains (Ma & Sonka, 1996)[24]. One caveat to this approach is the produced skeleton may not have connectivity (Jin & Kim, 2017)[25]. To compensate for this deficiency, these algorithms generally act in parallel with a skeleton correcting algorithm. Jin and Kim (2017)[25] proposed analyzing the unit distance of voxels with 26-way connectivity and reconnecting removed voxels if the distance between voxels are within a threshhold value of $\sqrt{3}$.

Other approaches include Reeb graph construction and constriction on 3D polygonal meshes (Tierny et al., 2006)[26]; signed-distance functions examining the neighborhood of vertices on a mesh's surface (Shapira et al., 2008)[27]; object decomposition and partial recognition by 3D point clouds (Jayadevan et al., 2019)[28]; and radial basis functions (Ma et al., 2003)[6]. The method most pertinent to my approach is geometry mesh contraction.

Geometry *mesh contraction* is a skeleton extraction algorithm that applies a Laplacian smoothing operator, moving each vertex along its inner normal curvature, creating a zero-volume skeletal shape (Au et al., 2008)[5]. The Laplacian smoothing operator uses a cotangent

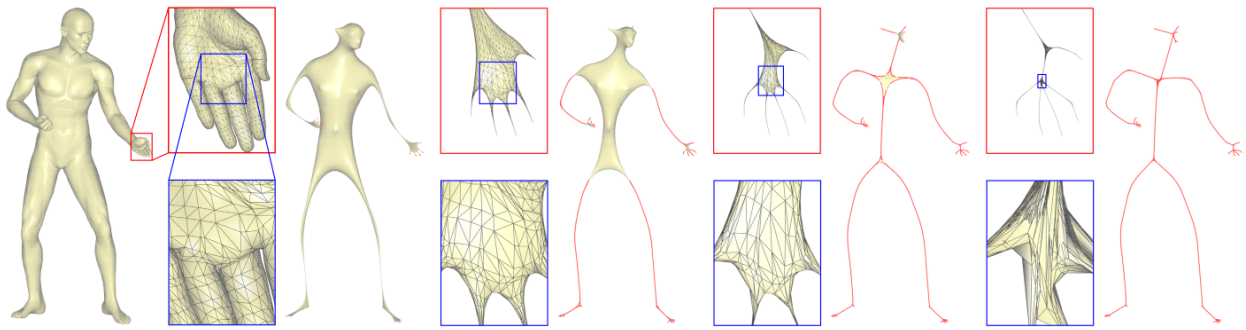

Figure 2.1: *This figure was produced by Au et al. (2008)*[5]*. It details the iterative process of the mesh contraction algorithm via the Laplacian smoothing operator. Each vertex in the mesh is moved along an inner normal direction towards the media surface of the mesh. The edge collapse process removes all unnecessary edges and faces, eventually forming the 1D curve skeleton.*

weighting measure such that

$$L_{i_j} = \begin{cases} \omega_{ij} = cot\alpha_{ij} + cot\beta_{ij} & \text{if } (i,j) \in E \\ \sum_{(i,k)\in E}^{k} -\omega_{ik} & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

where α_{ij} and β_{ij} are the opposite angles corresponding to edge *(i, j)* (Au et al., 2008; Desbrun et al., 1999)[5 29]. A series of edge-collapses maintain the connectivity of the skeleton, merging vertices and deleting respective faces, with a cost function containing two weight variables: sample-cost and shape-cost (Au et al., 2008)[5].

2.2 Skeleton Embedding Methods

The first well-known approach to automated creation of animation rigs was the Pinocchio system. *Pinocchio* is a skeleton embedding and skin attachment procedure that takes a static mesh and template skeleton as inputs, then learns a series of good and bad embeddings (Baran & Popovic, 2007)[10]. Its methodology of selecting potential joint locations included sampling points approximate to the medial surface, sphere packing, and drawing edges between intersecting spheres; it also used a discrete penalty function (e.g., short bones,

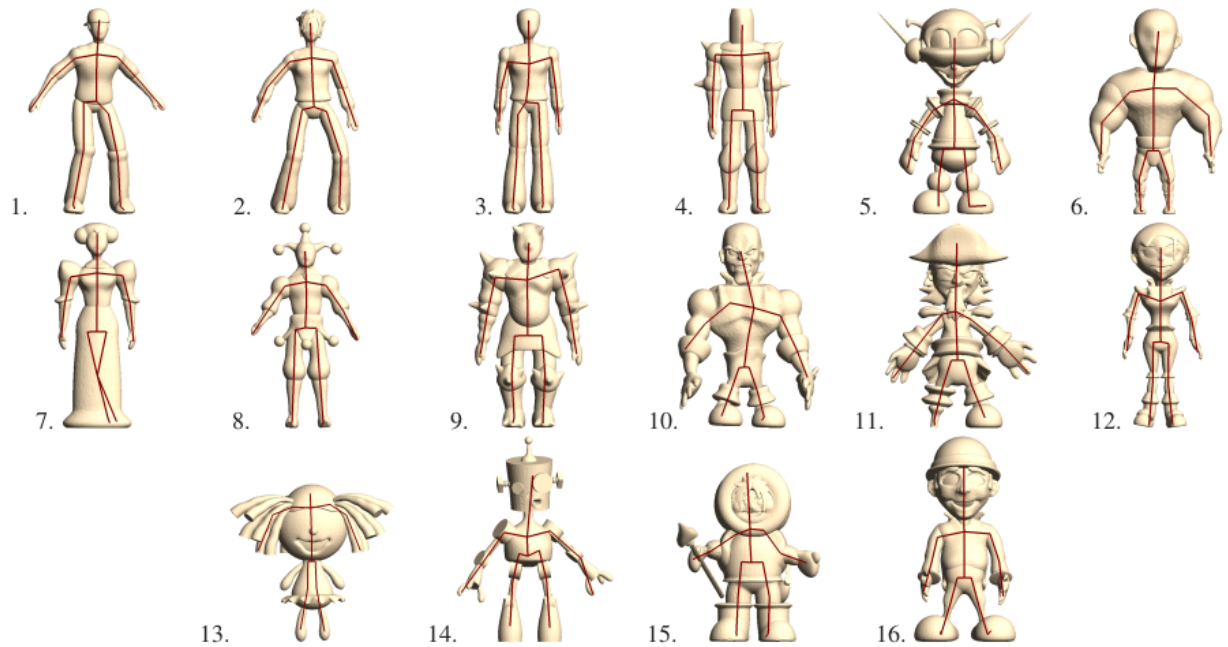

Figure 2.2: *This figure was produced in Baran and Popovic (2007)[10]. Based on the general rig criteria, 13 out of 16 models from their original test set produced qualitatively decent rigs. Models 7, 10, and 13 lack symmetry and consistency.*

zero-length bone chains, and improper orientation) (Baran & Popovic, 2007)[10]. Pinocchio tends to perform well on meshes resembling bipedal characters at rest with clearly defined symmetry. However, the system struggled with a mesh that had ambiguities like a ballerina dress obscuring the legs, hair-tresses resembling limbs, and asymmetrical poses. These deficiencies were attributed to automated weight generation, sensitivity to noise and outlier vertices, and the penalty function's inadequate filtering of similarly-shaped parts (Wang, 2012)[30].

To rectify Pinocchio's shortcomings, Wang (2012)[30] proposed using density-based clustering (DBSCAN) within the API at the Discrete Embedding stage, where the reduced skeleton input is matched to a good embedding. DBSCAN can find vertex clusters of arbitrary shape within a given epsilon threshold (Tran et al., 2013)[31]. This method of detecting clusters makes it not only robust but insensitive to noise because of DBSCAN's inherent ability to classify noisy vertices. By gathering these vertices as clusters and removing the cluster should a vertex be eliminated, incorrect vertex selection was solved (Wang, 2012)[30]. Using

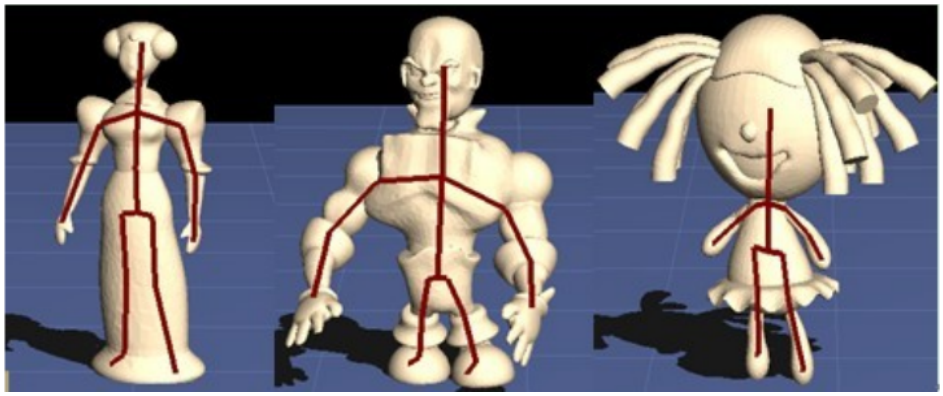

Figure 2.3: *This figure was produced by Wang (2012)[30]. After applying DBSCAN clustering and a corrective alignments process, models 7, 10, and 13 were fixed. The models are more consistent and symmetrical, and they do follow general rig criteria.*

DBSCAN and an adjustment strategy algorithm to tune symmetric joints allowed Wang (2012)[30] to correct the deficient rigs produced by the original Pinocchio (see Figure 2.3). However, this new system had a new set of limitations, most notably, arbitrary contortions and stretches caused models to suffer poor skeleton embeddings primarily because models exceeded the predetermined threshold limit.

2.3 Limitations of Existing Work

In spite of their usefulness, previous methods have limitations. Those limitations inspired the rationale for research into a new approach as described in this thesis. The Pinocchio system, for instance, is limited by the epsilon threshold used with DBSCAN to format a discrete skeleton embedding. When Wang (2012)[30] tested the limits of Pinocchio using DBSCAN, finding that models tended to fail because unwanted clusters were not be eliminated when the epsilon threshold was too low. Conversely, limbs were outright eliminated when the epsilon threshold was too high. Figure 2.4 shows these limitations by varying the epsilon threshold for Model 13 (Raggedy Ann) of the Pinocchio dataset. Wang (2012)[30] also used Pinocchio's bounded heuristic search to finish embeddings although the embadding would consequently not be labelled good or bad. This caused erratic behavior in a system where incorrect embeddings finished in a few seconds while a correct embedding took several minutes (Wang,

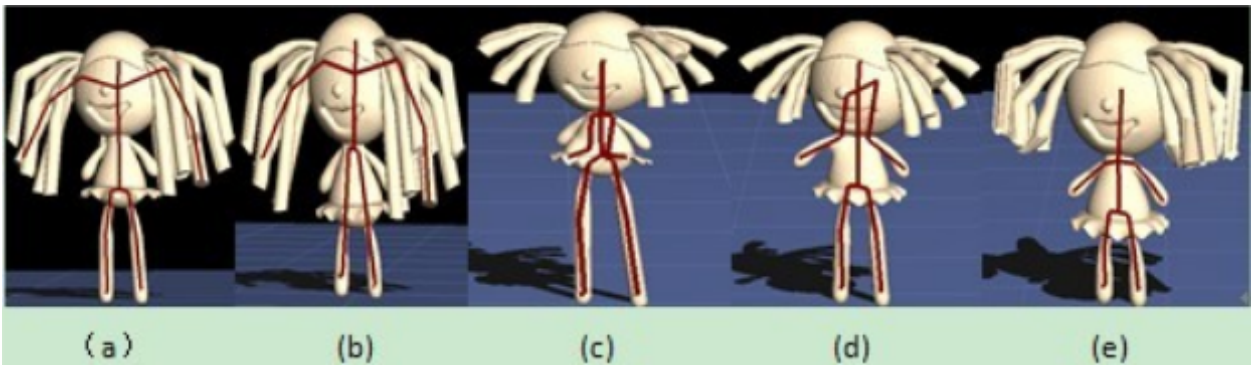

Figure 2.4: *Wang (2012)[30] produced this figure showing skeleton embeddings produced with varying epsilon thresholds, starting at 0.1 (a) to 0.6 (e).*

2012)[30].

In surveying the literature, I found Wang's (2012)[30] approach was the most recent development in methods for embedding and refining skeletons. This method has since been supplanted with the advent of deep learning pose estimation models. Newell et al.'s (2016)[15] seminal work introduced the concept of *Stacked Hourglass* networks capable of identifying and refining landmark locations for joint vertices directly from an image. Xu et al. (2019)[1] later adapted this approach using volumetric convolutional networks to read-in 3D voxel data instead of 2D pixel imaging. They also expanded Newell et al.'s work to simultaneously learn landmark bone location as well as joint vertices (Xu et al., 2019)[1]. Their approach is the first documented method applying deep learning pose estimation models to the automatic rigging problem.

For skeleton extraction, I applied Au et al.'s (2008)[5] mesh contraction approach. This technique is not only a seminal application for extracting curve skeletons, but also fast and invariant to pose and scale. It is the best mesh-based skeletonization algorithm (Sobiecki et al., 2013)[32]. Furthermore, this approach has two working implementations in Python: Mesh Skeletonization and Skeletor. Both libraries can interface with Blender's API. Figure 2.5 displays two sample curve skeletons extracted inside Blender, one using Mesh Skeletonization and the other using Skeletor.

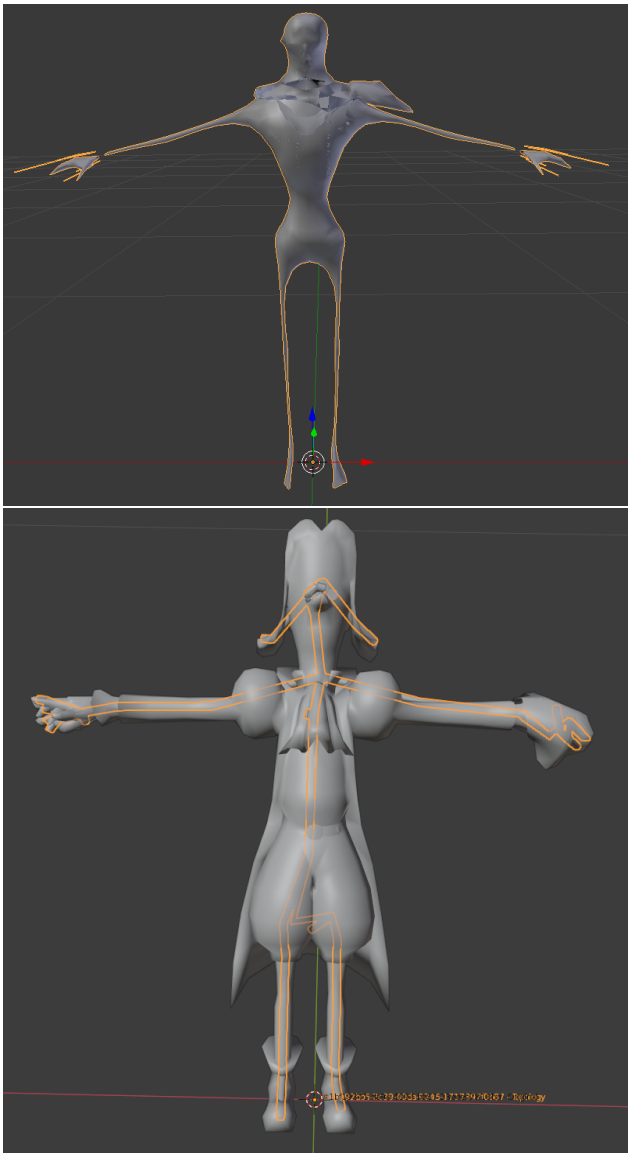

Figure 2.5: *The mesh on top is an extracted skeleton using Blender 2.79 with the Python library Mesh Skeletonization. The mesh on the bottom is an extracted skeleton using Blender 2.90 with Skeletor. Both mesh contraction parameters used 10 iterations, with an initial face weight of 1.0 and initial weighting factor of 1.0.*

Chapter 3

Methodology - Stacked Hourglass Network

Automating the rigging process is analogous to pose estimation. In both scenarios, our goal is to determine precise locations (joint vertices for rigs, pixels for imaging) of key points, limb articulation, and the posture (or pose) of the body (Newell et al., 2016)[15]. Ideally, automating the rigging process would simply involve inputting a mesh and have a procedure that can map a rig along the mesh's medial surface while observing the guidelines of general rig criteria. The Stacked Hourglass Network proposed in Newell et al. (2016)[15] is state-of-the-art in estimating landmark locations in RGB images (Yang et al., 2017)[16]. The hourglass design of the network was inspired by the need to capture detail at every resolution of the image while also retaining knowledge at the global scale (Newell et al., 2016)[15]. For this , the hourglass design is effective in predicting tasks involving joint and bone heatmaps (Huang et al., 2018)[33].

3.1 Network Architecture

The key motivation behind the design of the hourglass module is to capture information and context at every scale of the image; its function is to identify and associate features, including

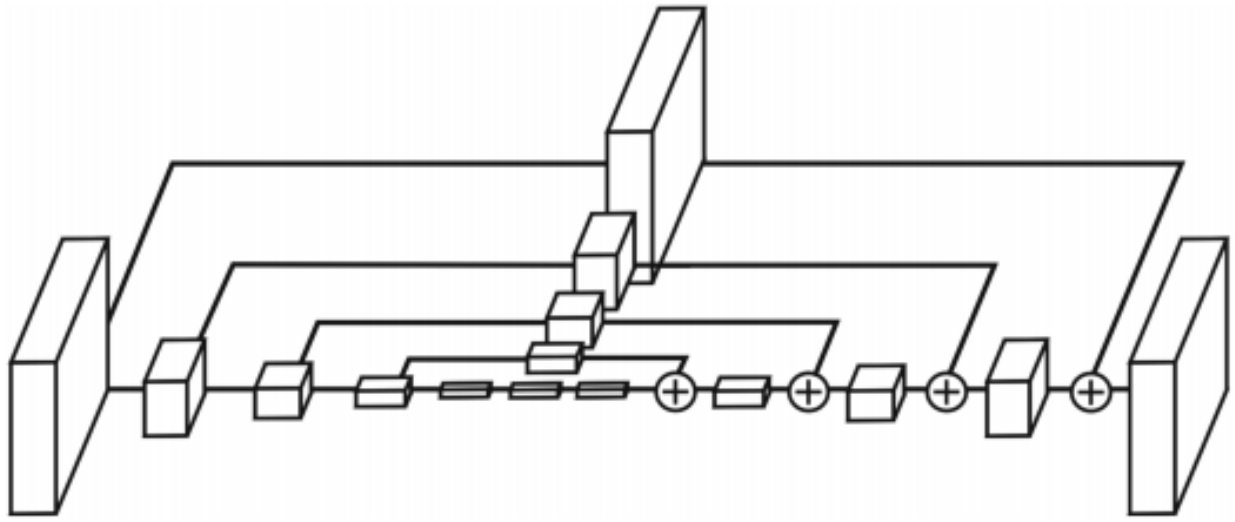

Figure 3.1: *An hourglass module presented in Newell et al. (2016)*[15]*. Moving left to right, the first-half of the residual blocks represent the encoder aspect; the second half the decoder. The "+" blocks denote the identity maps, connecting each residual block in the encoder region to their respective decoder residual blocks.*

object orientation, position of limbs, and spatial relationships of joints and bones (Newell et al., 2016). Figure 3.1 presents an abstracted hourglass module design. The initial layers of the hourglass modules are convolutional and max pooling layers, used to process features down to low resolutions (Newell et al., 2016). At this stage, the module attempts to identify and extract features within the global context. For every layer, the module processes and extracts features at each decreasing resolution until it reaches the lowest possible resolution. This portion of the module is also known as the encoder aspect.

After reaching the lowest possible resolution (represented by the inner-most block in Figure 3.1), the module enters the decoder aspect. Here, the module performs a top-down sequence of upsampling and combining features from the lower resolutions, using a sliding window model to overlap and incorporate adjacent resolutions (Newell et al., 2016; Tompson et al., 2014)[15,34]. To access details from earlier resolutions, every residual module of the decoder has an identity map back to its corresponding convolutional layer in the encoder (represented by the "+" sign). *Identity mapping* is an additional convolutional layer that simply outputs its inputs to the next connected layer, enabling residual learning (He et al., 2016)[35]. This method of using identity mapping allows the network to remember local

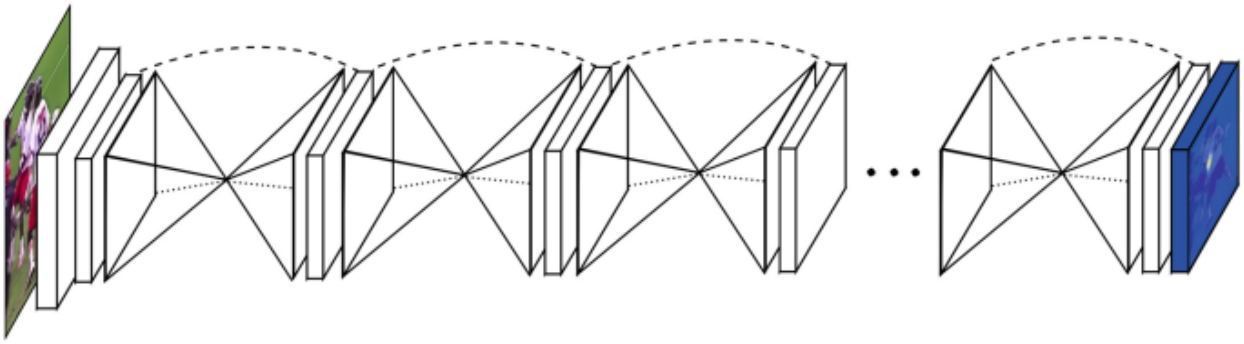

Figure 3.2: *A stacked hourglass network comprises a series of hourglass modules, with output of one module serving as input for the next. The final output is a refined heatmap. This figure is provided by Newell et al. (2016)*[15].

details lost in the resolution-scaling process. The final output of the network is a round of convolutions producing a set of *heatmaps*, a probability map detailing the per-pixel (or per-voxel) likelihood for key joint locations on the human skeleton (Newell et al., 2016; Tompson et al., 2014)[15,34].

Figure 3.2 displays an abstracted design of a stacked hourglass network where n number of hourglass modules can be cascaded together. Every output of one hourglass module is inputted into another where subsequent modules allow high level features to be processed, again and again, to further evaluate and revise higher order spatial relations (Newell et al., 2016)[15]. This process is called *intermediate supervision* where the prediction of intermediate heatmaps, generated from each hourglass module, are reevaluated within the global context of the shape, as opposed to its local features (Newell et al., 2016)[15]. With loss functions applied at the end of every module, intermediate supervision has an added effect of improving training speed and performance of the network (Park et al., 2018)[36]. However, choosing the number of stacked hourglass modules requires some care. An ablation study is recommended to determine the appropriate number of hourglass modules and identify where predictive performance saturates, as in Xu et al. (2019)[1].

3.2 Input Representation with Volumetric CNNs

The original model by Newell et al. (2016)[15] was designed for 2D pixel imaging. To adapt their network to 3D space, the input representation must be volumetric. This can be done by adapting the use of volumetric convolutional layers. *Volumetric CNNs* are convolutional neural networks with volumetric representations as their input features (e.g., depth maps, RGB-D images, and 3D voxel grids). The key attribute of these representations is depth information; 2D space cannot solve depth ambiguity (Wu, Finnegan et al., 2018)[37]. Among the first volumetric CNNs was 3DShapeNets, a convolutional deep belief network for object category recognition (Wu, Song et al., 2015)[38]; other examples are Voxnet, a real-time object recognition 3D convolutional neural network (Maturana & Scherer, 2015)[39]; OctNet, an octree network for sparse 3D data representation (Riegler et al., 2017)[40]; and V2V-PoseNet, a real-time 3D CNN for hand pose estimation (Moon et al., 2018)[41].

The input layer for the stacked hourglass network, adapted in Xu et al. (2019)[1], is an 88x88x88x5 volumetric convolutional layer. Each model from the dataset is a voxelized representation of the original mesh with dimensions 82x82x82 with 6-dim padding. Every voxel contains 5 potential geometric features: the signed distance function, local vertex density, (2) principal surface curvatures, and local shape diameter. The *signed distance function* represents surface intersections as zeros, free space beyond the surface of the mesh as positives, and occupied space as negatives (Newcombe et al., 2011)[42]. For each model in the dataset, the signed distance function is applied to the mesh to convert the voxel representation to a volumetric representation. *Local vertex density* is a Gaussian heatmap that identifies key landmarks on the mesh with a high density of vertices. Xu et al. (2019)[1] assumed that graphical modelers will often place more vertices than normal around key locations (e.g., neck, elbow, knee) for finer smoothing between different body parts. *Principal surface curvatures* indicate the rate of change in the direction of the normals at any point on the mesh's surface (Dharmasiri et al., 2017)[43]. *Local shape diameter* provides a link from the surface of the mesh to its volume through the medial axis transform and helps identify mesh partitions through segmentation (Shapira et al., 2008)[27].

3.3 Adapted Hourglass Module

The adapted hourglass module shown in Table 3.1, designed by Xu et al. (2019)[1], has an encoder that processes feature maps, capturing increasingly larger complex context of the input shape; a user controlled granularity parameter for pruning small or inconsequential bones; a decoder for accessing earlier feature maps containing lost local detail; two separate branches of residual blocks, each branch reducing dimensionality from 8 to 4 to 1; a sigmoid function output layer producing the intermediate predicted heatmaps for joints and bones. As in Newell et al. (2016)[15], the encoder has residual blocks that identity map their input to their respective layer in the decoder. The ReLU (rectified linear unit) layer further scales down the resolution by a stride of two.

The hourglass modules presented in Xu et al. (2019)[1] and Newell et al. (2016)[15] differ in two primary ways. The first is the user-controlled granularity parameter when the encoder has finished contextualizing. Xu et al. (2019)[1] included this parameter as a way for the user to control the granularity of the skeleton and to condition the network on variance in the model dataset, e.g., rigging fingers, and ears. The second way is Xu et al. (2019)[1] include a separate output prediction block responsible for producing bone heatmaps. Predicting joint and bone heatmaps are interdependent tasks because the placement of one affects the placement of the other. Therefore, the network must learn and contextualize features that will influence the placement of bones as well as joints.

The default value for the user-controlled parameter is 0.02, tuned through hold-out validation (Xu et al., 2019)[1]. In their ablation study, Xu et al. (2019)[1] determined that including this parameter yielded the best results in precision and recall. Intermediate supervision can be deployed to learn the inter-dependency between joints and bones. Even in 3D space, intermediate supervision is robust when using 3D voxels to resolve depth ambiguity (Wu, Finnegan et al., 2018)[37]. Xu et al. (2019)[1] found that four hourglass modules provided the best predictive accuracy performance. I retained these parameters because they represented the best configuration of the models tested by Xu et al. (2019)[1].

	Layers	Output
	Input volume	88×88×88×5
	ReLU(BN(Conv(5x5x5, 5→8)))	88×88×88×8
	ResBlock	88×88×88×8
Encoder	ReLU(BN(Conv(2x2x2, stride=2)))	44×44×44×8 for 1st module, 44×44×44×10 for the rest
	ResBlock	44×44×44×16
	ReLU(BN(Conv(2x2x2, stride=2)))	22×22×22×16
	ResBlock	22×22×22×24
	ReLU(BN(Conv(2x2x2, stride=2)))	11×11×11×24
	ResBlock	11×11×11×36
	Concat with control param.	11×11×11×40
	ResBlock	11×11×11×40
Decoder	ResBlock	11×11×11×36
	ReLU(BN(ConvTrans(2x2x2, stride=2)))	22×22×22×24
	ResBlock	22×22×22×24
	ReLU(BN(ConvTrans(2x2x2, stride=2)))	44×44×44×16
	ResBlock	44×44×44×16
	ReLU(BN(ConvTrans(2x2x2, stride=2)))	88×88×88×8
Prediction	ResBlock	88×88×88×4
	Dropout(ReLU(BN(Conv(1x1x1, 4→4))))	88×88×88×4
	Conv(1x1x1, 4→1)	88×88×88×1

Table 3.1: *An individual hourglass module adapted for predicting pose joints and bones in 3D space. ResBlock: The residual block is made of two volumetric convolutional layers with filters 3×3×3. Dropout layer with 0.2 probability. This table is provided by Xu et al. (2019)*[1].

3.4 Cross Entropy Loss Function

Newell et al. (2016)[15] used Mean Squared Error (MSE) as the loss function because the Gaussian heatmaps produced by that architecture are 2D and MSE is usually adopted for that domain (Sun et al., 2018)[44]. Xu et al. (2019)[1] chose the cross entropy loss function. *Cross entropy loss* combines the negative-log likelihood and log softmax loss functions into a single class and is often applied in multi-class classification problems (PyTorch, 2019)[45]. For pose estimation problems, this function is used to identify and select voxel locations with the highest probable weight. However, pose estimation differs from object classification in that no single voxel is assigned a true positive value because a huge penalty may be incurred

on neighboring voxels (Ryou et al., 2019)[46]. Therefore, Xu et al. used ground-truth and predicted value comparison with heatmaps instead of one-hot encodings.

The overall goal of the network is to predict joint heatmaps, $\hat{\mathbf{P}}_j$ and bone heatmaps, $\hat{\mathbf{P}}_b$ for every model in the dataset, specifically, a 3D isotropic Gaussian heatmap (Xu et al., 2019)[1]. These predicted heatmaps were compared with the ground-truth heatmaps produced from their respective animation rig using a cross entropy loss function for both joints and bones. Xu et al. (2019)[1] more formally defined the cross entropy loss function as

$$L = \sum_{s} \frac{1}{N_s} \sum_{v} M_s\,[v](L_j[v] + L_b[v]) \tag{3.1}$$

where $L_j[v]$ is the cross-entropy loss for joints and $L_b[v]$ is the cross-entropy loss for bones. $L_j[v]$ is defined as

$$L_j\,[v] = \hat{\mathbf{P}}_j(v)\,\log(\mathbf{P}_j(v) - (1 - \hat{\mathbf{P}}_j(v))\,\log(1 - \mathbf{P}_j(v))), \tag{3.2}$$

and $L_b[v]$ is similarly defined as

$$L_b\,[v] = \hat{\mathbf{P}}_b(v)\,\log(\mathbf{P}_b(v) - (1 - \hat{\mathbf{P}}_b(v))\,\log(1 - \mathbf{P}_b(v))). \tag{3.3}$$

M_{s} is the masked loss term introduced so all voxels outside the surface of the mesh have a loss of zero because these voxels dominate the loss function; N_{s} being the summation of all masked loss, formally defined as

$$N_s = \sum_{v} M_s[v]. \tag{3.4}$$

This cross entropy loss function was applied at at every intermediate supervisory step.

3.5 Soft Non-Maximum Suppression (Soft NMS)

Once the final predicted heatmaps for joint and bones are constructed, a soft non-maximum suppression step is applied to remove any duplicate joint vertex landmarks. *Soft non-maximum suppression* is an iterative greedy algorithm that selects relatively high scoring landmarks and decays the scoring probability of neighboring landmarks likely to cover the same object space (Hosang et al., 2017; Bodla et al., 2017)[47][48]. Its behavior is analogous to lateral inhibition in neuroscience, when excited neurons stimulate their surrounding environment, thereby inhibiting the activity of neighboring neurons (Cohen, 2011)[49]. We can regard joint vertices with the highest respective probability as the stimulated neuron. All other nearby joint vertices then have their probabilities inhibited. This step is vital for constructing a viable rig, consistent with the general rig criteria, because it keeps the rig from becoming too complicated and ensures only one joint for each landmark on the mesh.

The three hyperparameters for soft non-maximum suppression are threshold, kernel sigma, and kernel size. *Threshold* determines the range of overlap between the selected joint vertex and its neighbors; kernel *sigma* is the variance of the Gaussian kernel used to select joint vertices; kernel *size* is the diameter used for the Gaussian heatmap. Xu et al. (2019)[1] set the hyperparameter values of sigma to 15 and size to 11. These values could have been arbitrary or the result of Gaussian weighting functions; this was not specified in Xu et al. The threshold parameter was set at 0.02, and the decay for any joint vertices within the threshold neighborhood was set to half the original probability.

3.6 Rig Connectivity

The final step in the rig creation process is to merge all final joint vertices in the simplest manner while remaining consistent with the original mesh. Xu et al. (2019)[1] opted to use Prim's algorithm for minimum weight spanning trees (Kershenbaum & Van Slyke, 1972)[50], minimizing the cost function driven by the predicted bone heatmap where bones passing through low probability zones have a high cost; conversely, bones passing through high

probability zones have a low cost. Xu et al. (2019)[1] more formally defines this cost function as

$$\omega_{i,\,j} = -\sum_{v \in l_i, j} \log \mathbf{P}_b(v) \tag{3.5}$$

where i and j are joints with an edge between, $l_{i,\,j}$ is a line segment representing the edge, and $\mathbf{P}_{\mathrm{b}}(v)$ is the predicted bone probability of each voxel $v \in l_{i,\,j}$ that intersects the line segment.

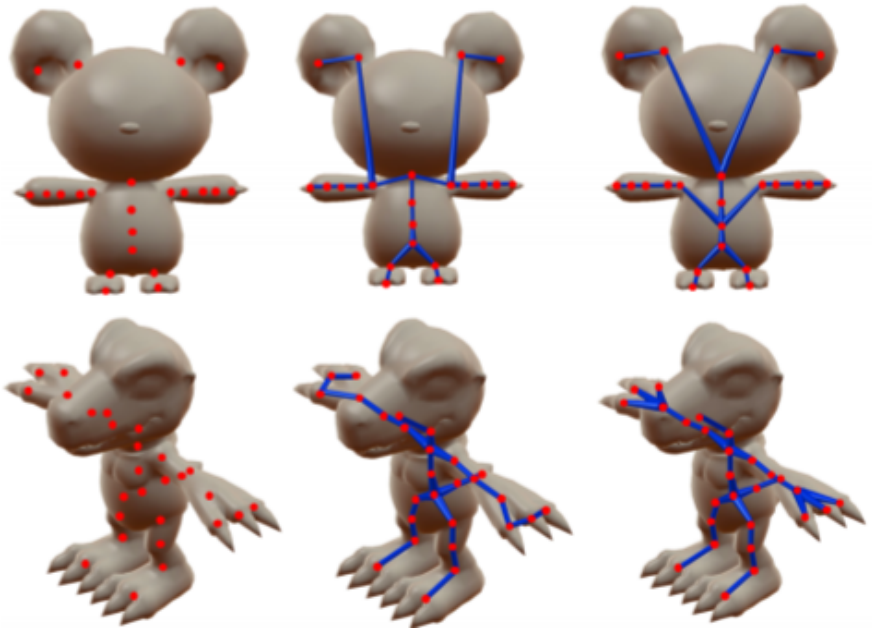

Figure 3.3: *Final landmark joint vertices produced by Stacked Hourglass Network (left). Rig connectivity using Euclidean distance (middle). Rig connectivity using Prim's algorithm (right). This figure comes from Xu et al. (2019)[1].*

Prim's algorithm was chosen over other edge cost functions, such as Euclidean distance, because Euclidean distance often produced qualitatively bad results like bone protrusions from the surface of the mesh (see Figure 3.3). Furthermore, as a minimum weight spanning tree algorithm, the final rig is structured as a strongly connected hierarchical graph. This format is often required for import into graphical modelling software such as Maya or Blender (Xu et al., 2019)[1]. Figure 3.4 shows the final rigs for various types of models: bipedal humanoids, quadrupeds, and flying mammals.

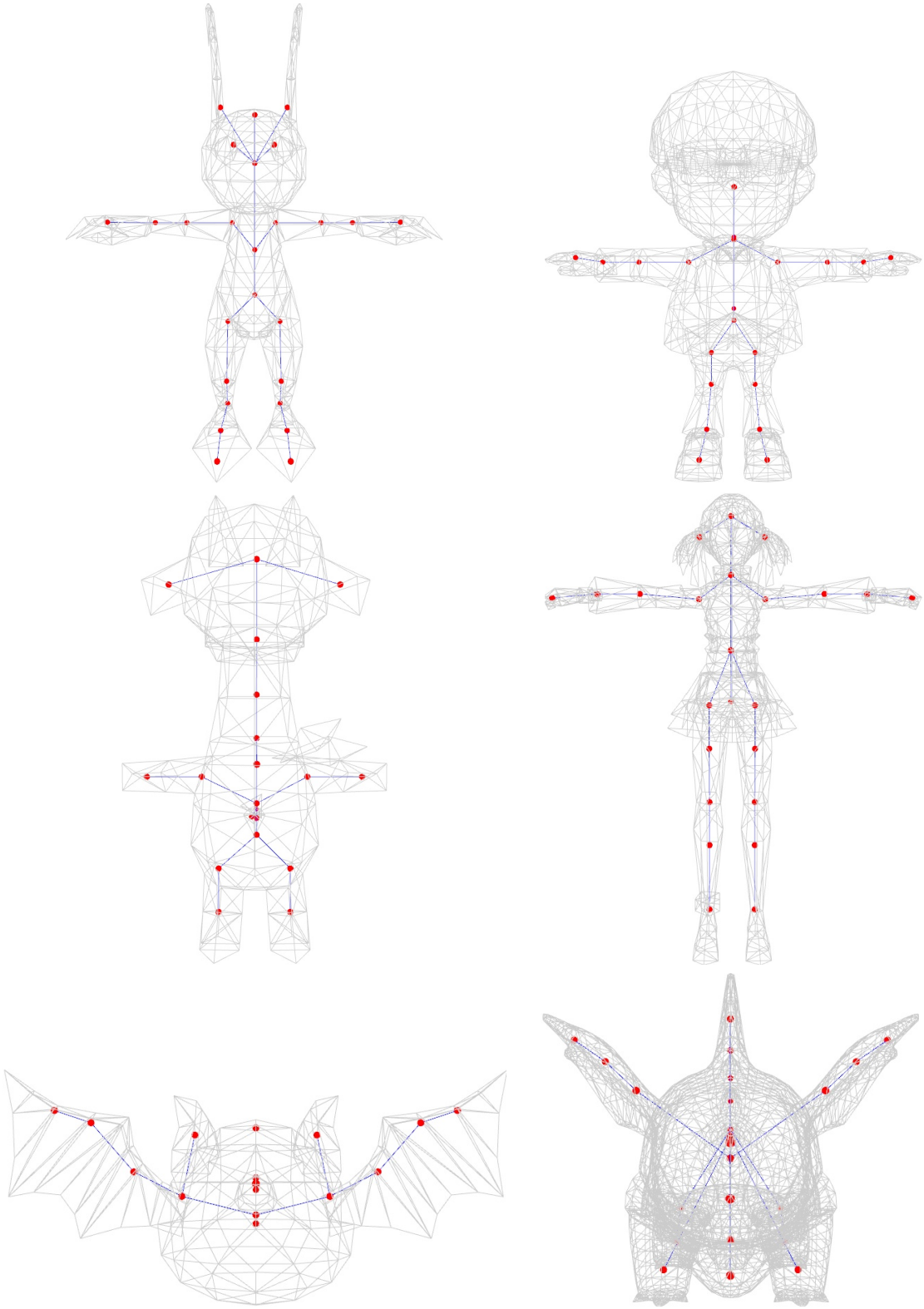

Figure 3.4: *Sample rigs after applying Prim's algorithm to connect all landmark joints together.*

Chapter 4

Experiment Design

4.1 Model Resource Dataset

I used the resource dataset compiled by Xu et al. (2019)[1] containing 3,277 meshes from Models Resource (2020)[51]; 3,193 meshes remained after removing duplicates. These meshes include bipedal humanoids, quadrupeds, animals, fish, and an assortment of fictional characters (e.g., robots, sentient dinosaurs, aliens). The diversity of meshes allowed the deep pose estimation models to be trained using many different shapes in different poses. These features increase the generality of the deep pose estimation model, allowing for better inference on unfamiliar object types. Each mesh is voxelized using *Binvox*, a program capable of reading 3D mesh files and rasterizing them into a binary 82^3 voxel grid (Min, 2020)[52]. This voxelization process converts the mesh into a volumetric representation capable of being read-in by the initial volumetric convolutional layer. The five input features: signed distance function, the two principal surface curvatures, local vertex density, and local shape diameter, are extracted during the process of creating the dataset.

4.2 Topology Extraction

Xu et al. (2019)[1] showed that each geometric input feature increased the predictive accuracy of the heatmaps for joints and bones. Skeleton extraction research shows high correlation between the skeleton of a mesh and its medial surface. Despite this, none of the five features analyze the medial surface of the voxel grid. Therefore, in the topology extraction step, I introduced a sixth geometric input feature: the curve skeleton. I believe curve skeletons will provide the Stacked Hourglass Network with features of the medial surface and increase predictive accuracy in joint and bone heatmaps. This is the intuitive placement for the rig because rigs also take the form of skeletons (Borosán et al., 2012)[53].

4.2.1 Mesh Skeletonization and Skeletor

To obtain awareness of mesh topology for every mesh in the model dataset, it is necessary to employ a skeleton extraction algorithm. I discovered two Python mesh contraction libraries that can interface with Blender: Mesh Skeletonization and Skeletor. Mesh Skeletonization is a Blender plugin designed for versions up to 2.79; its skeletonization process is based on the mesh contraction algorithm proposed by Au et al. (2008)[5]. Skeletor is a modified variant of Mesh Skeletonization designed as a general purpose library for skeleton extraction; however, this library was originally designed for neuron research (Schlegel, 2020)[3] with a core dependency of *NAVis* (NAVis - Neuron Analysis and Visualization), a library for analyzing and visualizing neuron morphology using a tree-like skeleton (Schlegel, 2018)[54]. Certain parameters and post-processing steps were influenced by this approach.

Figure 4.1 shows an example of using Mesh Skeletonization to extract the curve skeleton from a model and Binvox to then voxelize it. The results indicate that Mesh Skeletonization performs well to contract hands, feet, and limbs of the mesh. However, the torso, shoulders, head, and neck regions did not sufficiently contract to the medial surface. This particular example also shows the resulting curve skeleton may become fragmented, i.e., disconnected from multiple roots. This fragmentation propagates further during voxelization, as can be seen by the image on the right in Figure 4.1. The connectivity between the feet and hips has

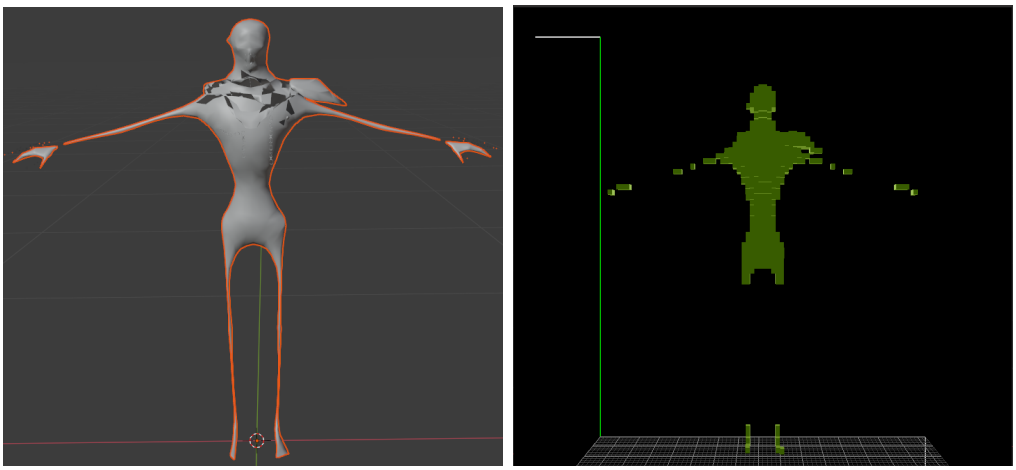

Figure 4.1: *Skeleton extracted in Blender 2.79 using Mesh Skeletonization (left). Voxelized skeleton after using Binvox to map the curve skeleton to dimensions 82^3 (right).*

disappeared completely. For very thin models like curve skeletons, Binvox must thicken the models by drawing all edges of the mesh. Figure 4.2 shows the final result after thickening.

Figure 4.3 displays another sample skeleton extraction using the Skeletor library in the image on the left. The mesh has fully contracted all parts of the body. However, as with Mesh Skeletonization, Skeletor produced fragmented curve skeletons. This required combining all fragments to form one cohesive whole. These fragmented sections can be combined using a minimum weight spanning tree (MST) algorithm. The NAVis library, which serves as the foundation of Skeletor, possesses an MST that can combine multiple roots based on Euclidean distance. However, the MST edge cost function cannot detect any edge connection that combines different parts and thus can produce an edge that protrudes outside the original mesh. The result is qualitatively bad connections like hands forming edges with faces. The right hand image of Figure 4.3 shows this.

For this reason, I chose to use the Mesh Skeletonization library over Skeletor for most extracted skeletons. I also discovered that the parameters for the mesh contraction process in Skeletor were highly sensitive and volatile. Parameters producing qualitatively good skeletons for one mesh would often produce drastically different results for another. Mesh Skeletonization tended to produce much better looking fits without much fine-tuning, despite its limitations.

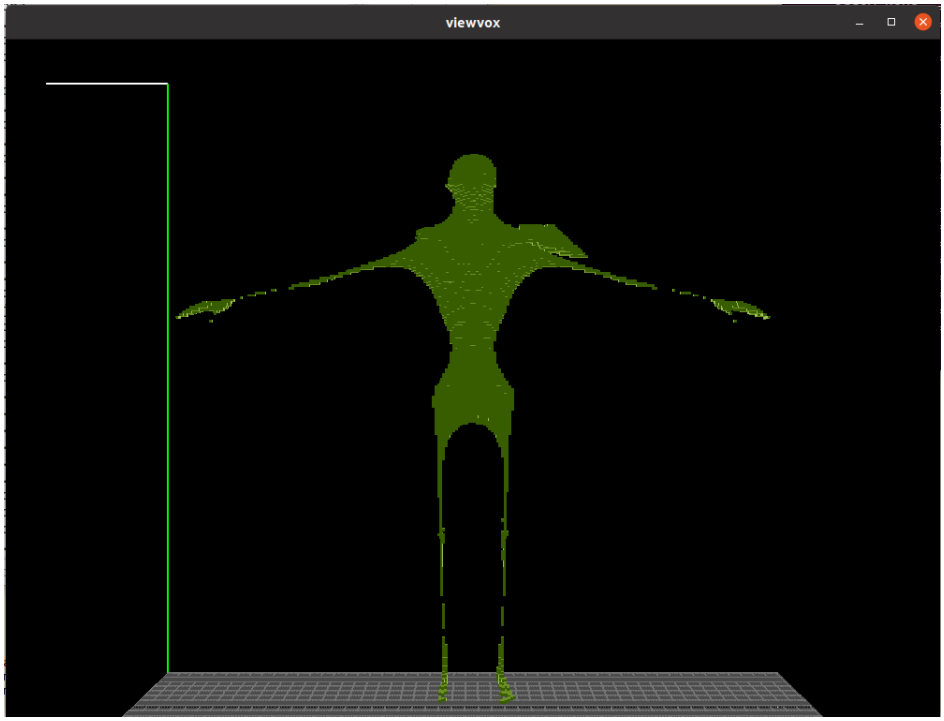

Figure 4.2: *Final voxelized skeleton after using Binvox to draw all edges of the mesh preventing further fragmentation.*

4.2.2 Curve Skeleton Extraction Across Model Dataset

I first imported each model from the dataset into Blender and applied a Triangulate modifer. Before running the mesh contraction process, triangulating the faces of the mesh is necessary because of the curvature flow of the Laplacian smoothing operator. This library has three input parameters: iterations (IT), initial face weight factor (SL), and initial weighting factor (WC). *Iterations* govern the total number of contraction steps. *SL* is the factor by which the contraction matrix is multiplied for each iteration; lower values cause slower contraction but ensure higher optimality. *WC* is the factor monitoring the contraction and weight constraints, determining the resulting smoothness of the contracted mesh (Schlegel, 2020; Ramachandran, 2018)[3][2].

I searched the parameter grid to ascertain the optimal extracted skeletons for the models in the dataset. This step is critical for optimizing the mesh contraction run-time parameters and to maximize fidelity of extracted curve skeletons in relation to their original mesh. For iterations, I used the values 1, 5, 10, and 20. For SL and WC, I used the values 1.0, 5.0, and

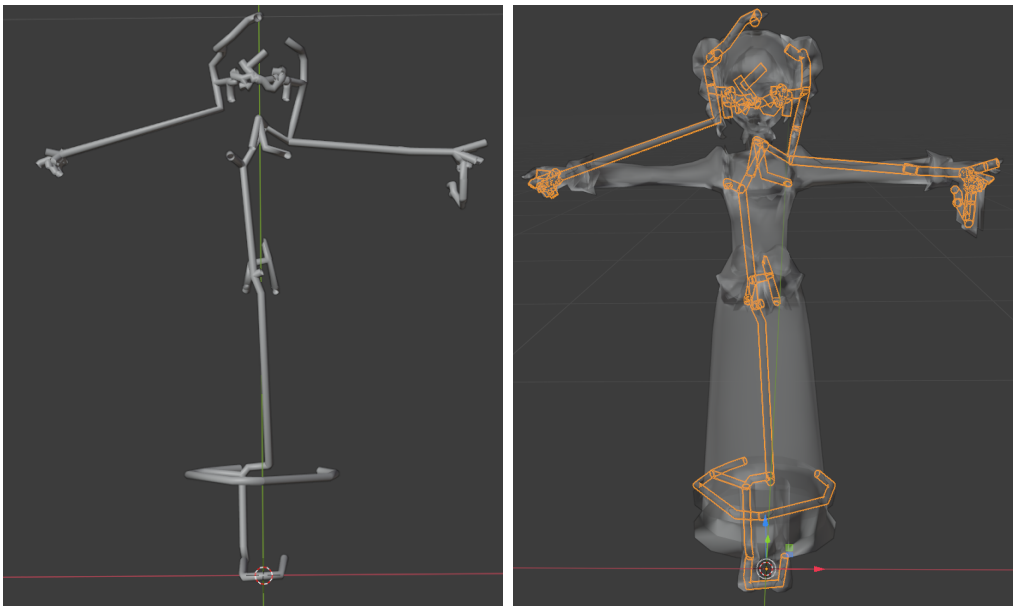

Figure 4.3: *Skeleton extracted in Blender 2.90 using Skeletor (left). Curve skeleton overlaid with the original mesh (right).*

10.0. My goal was to find the optimal parameters that create a strong contracted mesh as well as limiting distortion of the skeleton from the original mesh. Figure 4.4 showcases the results of my parameter sweep, indicating how far I could take the contraction step. During the edge-collapse step of mesh contraction, the algorithm may breakdown because of errors in the mesh (e.g., infinite values, duplicate vertices, degenerate faces (Schlegel, 2020)[3]). These failures are generally caused by a zero division error.

When I used 20 iterations on the dataset, 1,137 meshes failed during the contraction process, more than one-third of all meshes in the dataset. Therefore, I chose the extracted dataset with parameters: 10 (IT), 1.0 (SL), and 1.0 (WC). Not only does this dataset maximize the strength of the contraction, but the weight values ensured more accuracy in maintaining detail; with this dataset, only 34 meshes failed in the contraction step. I later used Skeletor to extract any skeletons Mesh Skeletonization could not, using the same parameters, weights, and Laplacian operators. I did not otherwise use Skeletor because of its great sensitivity to weight changes, volatile contractions, and need for a user to qualitatively match the extracted skeleton with the original mesh. The curve skeletons produced by Skeletor were also voxelized to dimensions 82^3 (See Figure 4.5).

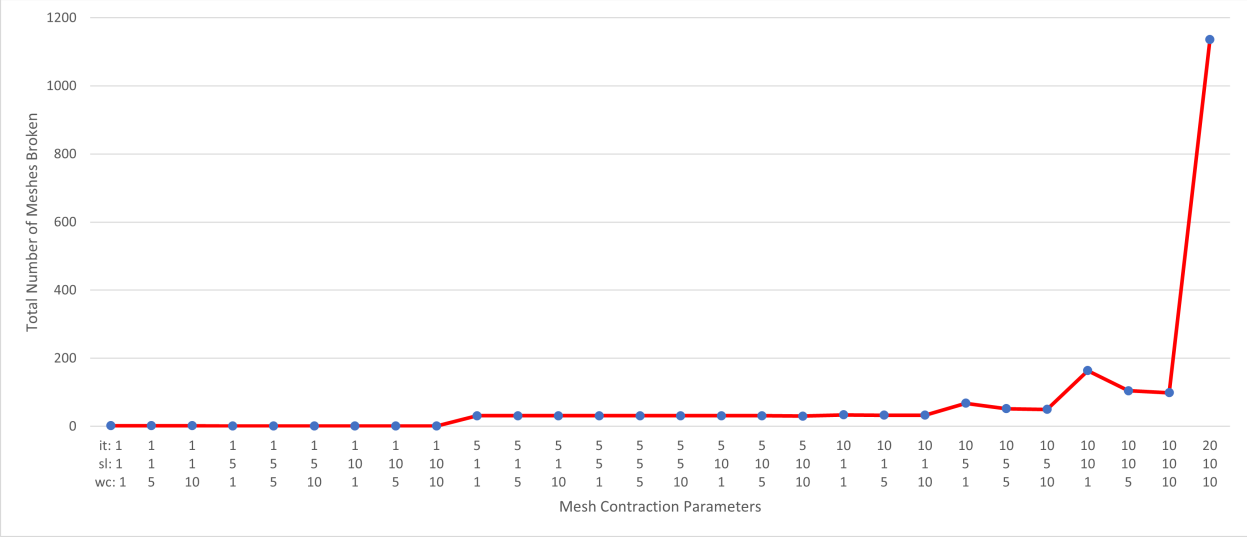

Figure 4.4: *Results after applying the mesh contraction step on the dataset with Mesh Skeletonization inside Blender; it: iterations, sl: initial face weight factor, wc: initial weighting factor.*

4.3 Adjusted Volumetric CNN

Voxelizing each curve skeleton to dimensions 82x82x82 meant the extracted curve skeleton dataset could be added as an input feature to the volumetric convolutional layer. Incorporating the extracted skeletons into the geometric input dataset with 6-dim padding brings the overall dimensionality to 88x88x88. The shape of the extracted skeleton is now congruent with the five input features, thus forming the final volumetric representation of 88x88x88x6. As mentioned in Section 3.3, I retained the five geometric input features, residual blocks, dropout layers with probability 0.2, user-controlled granularity value of 0.02, and four hourglass modules for intermediate supervision. This architecture layout and runtime parameters provided the best configuration of all the models tested and evaluated by Xu et al. (2019); they are control variables in my experiment. The extracted curve skeletons are my independent variable and their effect on producing versatile and robust rigs for various mesh types are my dependent variable.

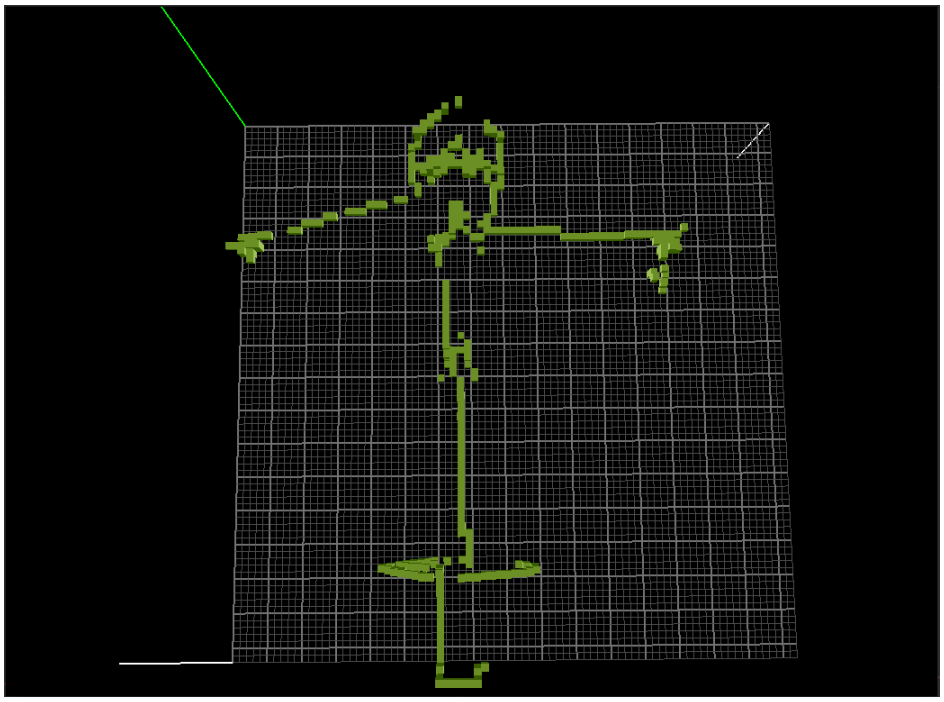

Figure 4.5: *Final voxelized sample skeleton produced by Skeletor. Voxelization was done with Binvox.*

4.4 Training

I followed the training/validation/testing procedures of Xu et al. (2019)[1]: 80% training, 10% validation, and 10% testing; 2,554 training meshes, 319 validation meshes, and 319 test meshes. The training dataset is used to fit the parameters of the deep pose estimation model; the validation set is used to evaluate the model's performance; the test set is a portion of the dataset locked away to prevent *peeking*, a consequence where a model evaluates itself on data it should not be able to view (Murphy, 2012; Russel & Norvig, 2013)[55][56]. This method of partitioning the dataset ensured evaluation of the model's performance was unbiased (Kuhn & Johnson, 2013)[57]. The primary boon of reduced bias is it leads to greater generality of the deep pose estimation model and produces viable rigs for unseen mesh types. Implementing and training the deep pose estimation models used PyTorch 1.2 with Cudnn 3.6. The total cross entropy loss function, equation 3.1 in Section 3.4, was the same. The Adam optimizer was used to minimize the loss function. The cross entropy loss function and Adam optimizer are kept as control variables in my experiment.

4.5 Soft NMS Threshold Ablation Study

To determine if the resulting number of vertices was sufficient to form a rig, Xu et al. (2019)[1] set a minimum requirement of 3 and a maximum requirement of 100 joints. These requirements helped safeguard the general criteria principles and ensured rigs remained simple, consistent, and predictive. However, these measures introduce limitations similar to Pinocchio with DBSCAN, as discussed in Section 2.3; varying the threshold parameter for soft non-maximum suppression resulted in disproportionate numbers of acceptable rigs. Therefore, I used an ablation study to observe how varying the threshold hyperparameter affected soft non-maximum suppression and thereby investigated the limitations of placing safeguards for general rig criteria.

4.6 Experiments Ran

I ran five experiments in all using a combination of topology, obtained via extracted curve skeletons, with the five geometric features: topology-only, topology and (2) principal surface curvatures, topology and local shape diameter, topology and local vertex density, and topology with all features. In every experiment, I also used the signed distance function because it is required to obtain volumetric representation of 3D meshes. I trained each of the five deep learning models over 50 epochs with a learning rate of 1e-4 on an Nvidia 2080 RTX. Depending on the available batch-size, training time took between 30 minutes and 50 minutes per epoch.

Experimenting with different combinations of geometric features is important in developing deep pose estimation models capable of producing rigs that are versatile and robust. Xu et al. (2019)[1] claimed that all geometric input features increased the predictive accuracy of their approach. I recreated their best model as my sixth experiment for comparison and evaluation of their approach against mine. I could thereby determine which combination of geometric features produced the best viable rigs.

Chapter 5

Results and Discussion

Figure 5.1 provides the results of my threshold ablation study for soft non-maximum suppression. I experimented with five different threshold values: 0.0002, 0.002, 0.02, 0.2, and 2; each value varied by a factor of 10. When the threshold was decreased, more vertices retained their original heatmap probabilities, resulting in more candidate joint locations. On the other hand, when the threshold was increased, the final rig has fewer candidate joint vertices. Inference was performed on the entire dataset of 3,193 meshes for each deep pose estimation model variant. Using the minimum-maximum requirement of 3-100 joints, threshold value 0.02 yielded the most rigs for five out of six experiments (including the original model by Xu et al. (2019)[1]. The only exception was the model variant featuring topology and shape diameter although the threshold value 0.02 came in second with 3,158 rigs produced versus 3,163 rigs for threshold value 0.002. Thus, I chose the set of rigs produced by the threshold value 0.02 for evaluation and comparison. Consequently, the 0.02 was also chosen as the best threshold value by Xu et al. (2019)[1].

Figure 5.2 displays the calculated loss values across the training, validation, and test sets. The loss functions for joint loss were the same as Equation 3.2, and for bone loss, were the same as Equation 3.3 in Section 3.4. These two equations, combined with the masked loss term in Equation 3.4, formulate the total loss in Equation 3.1 in Section 3.4. Based on the trends, local optimal convergence of the training loss comes between 10-15 epochs for

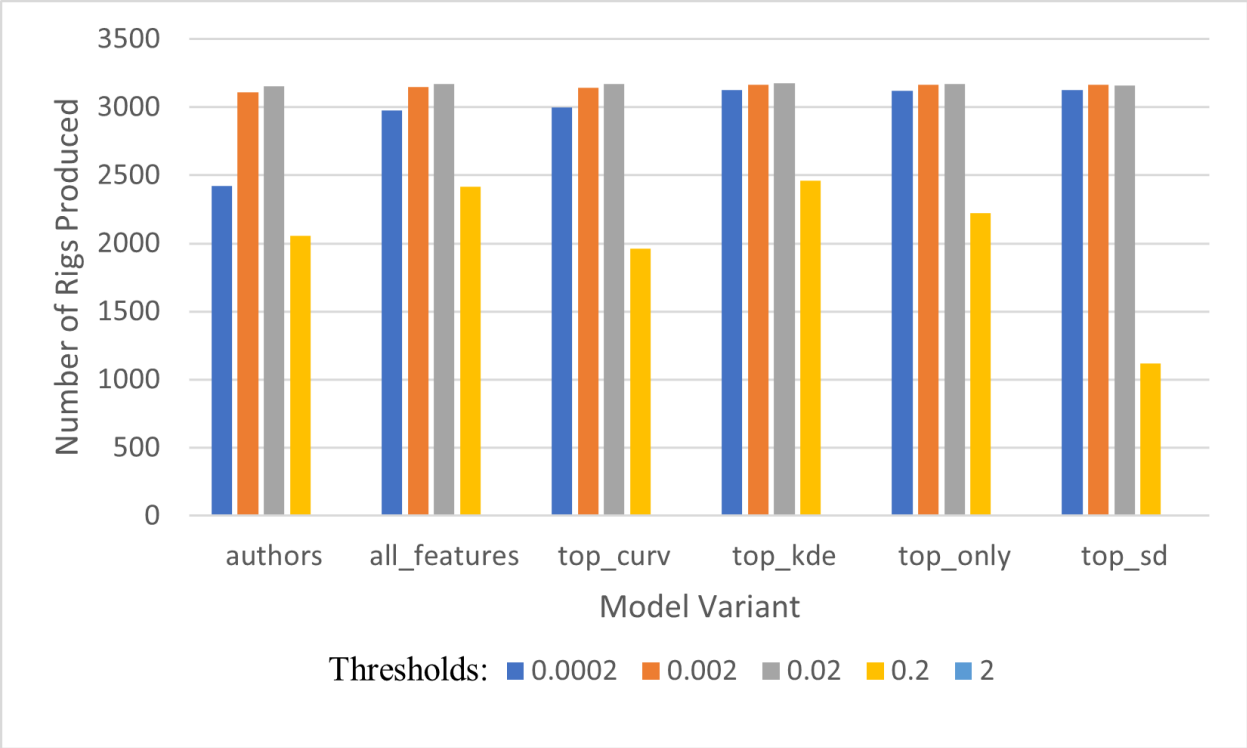

Figure 5.1: *Ablation study testing differing thresholds for soft non-maximum suppression. Threshold value 0.02 produced the most viable rigs in five out of six experiments using a dataset of 3,193 meshes: 3,154 author-trained model (Xu et al., 2019)[1]), 3,172 (all features), 3,170 (sdf+topology+curvature), 3,176 (sdf+topology+density), 3,170 (sdf+topology), 3,158 (sdf+topology+diameter). Threshold value 2 returned zero rigs across all experiments.*

both joint and bone loss. This remains true for both the validation and test sets. However, on the test set, topology with local shape diameter and topology with all features show spikes in the total loss. For both experiments, brief spikes occur in the joint loss between epochs 13-15 and epochs 25-28. Topology with local shape diameter also shows a notable spike between epochs 30-32 on the test bone loss. These spikes tended to smooth out with prolonged training, but the overall testing error beyond 35 epochs shows a slight upward trend, demonstrating characteristics of overfitting. Therefore, I recommend a cap limit of 20 epochs when using a learning rate of $1e^{-4}$.

For comparison and evaluation, I observed the chamfer distance loss function found in Pytorch3D. PyTorch3D calculates the *chamfer distance* between 2 meshes by first converting the mesh into a pointcloud, by uniformly sampling points on the surface of the mesh with probability proportional to the area of the face, then averaging the Euclidean distance across

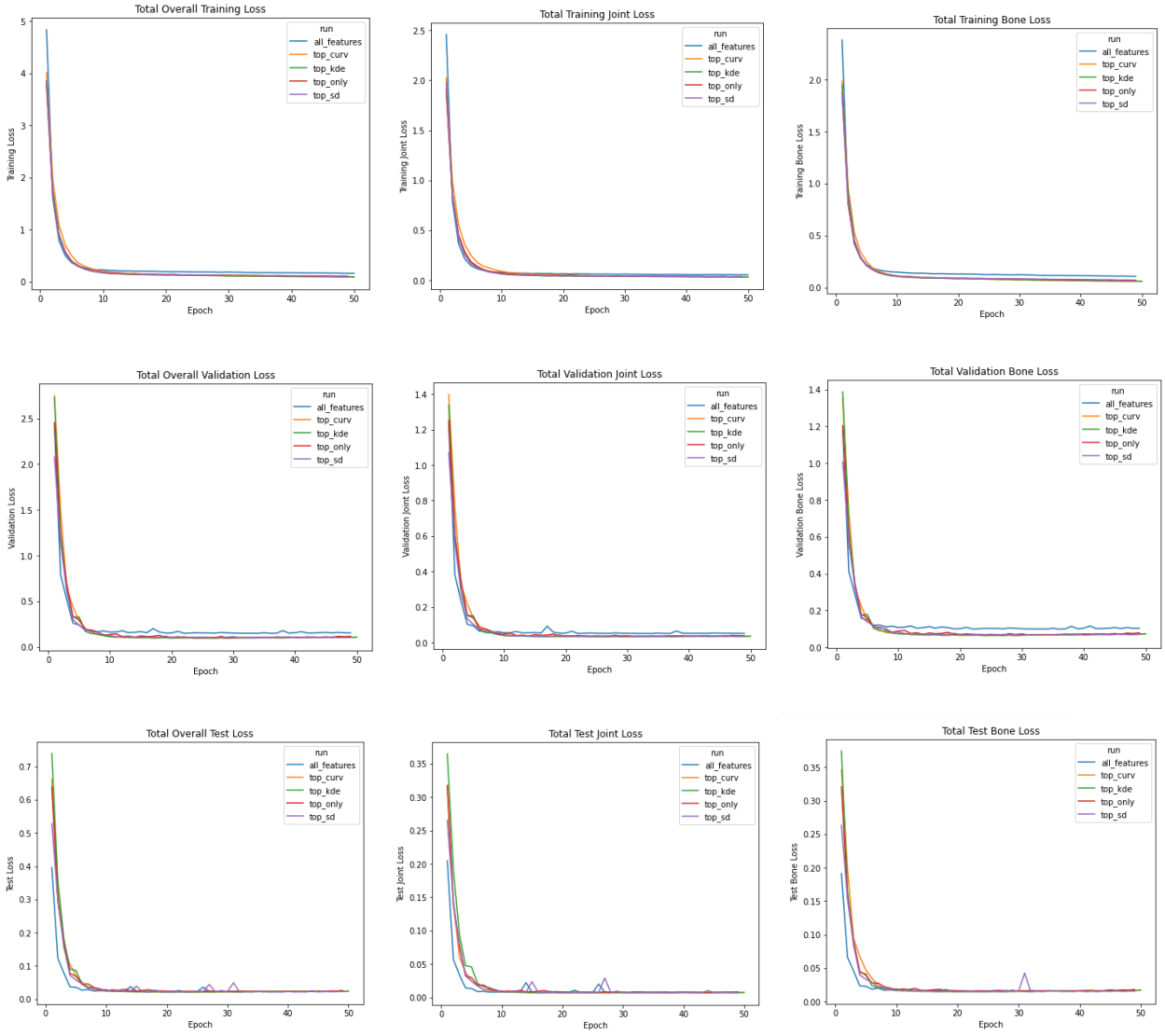

Figure 5.2: *Training, validation, and testing losses: total training loss (top-left), training joint loss (top-center), training bone loss (top-right), total validation loss (center-left), validation joint loss (center-center), validation bone loss (center-right), total test loss (bottom-left), total test joint loss (bottom-center), and total test bone loss (bottom-right).*

Deep pose estimation model variant	Average chamfer dist
sdf+topology	**1.7484e-06**
sdf+topology+curvature	1.7961e-06
sdf+topology+density	1.8629e-06
sdf+topology+diameter	2.2932e-06
all features	1.8571e-06
model by Xu et al. (2019)[1]	1.8370e-06

Table 5.1: *Evaluation of models, trained on different geometric inputs using topology, on the test set.*

all respective points (Facebook, 2020)[58]. I used Autodesk Maya 2020 to bind the resulting rigs to their original mesh. From there, I converted each of the final predicted rigs into meshes using Blender 2.79 and imported each mesh-rig dataset into PyTorch3D. The average chamfer distance was taken across all sample points in the cloud. I computed the average chamfer distance across all meshes in the dataset to produce my final metric.

Table 5.1 details the results for each variation of the deep pose estimation model on the test set. The best model is the topology-only model, which showed the lowest chamfer loss value. The model combining all geometric input features performed slightly worse than the model used by Xu et al. (2019)[1]. This is surprising given topology by itself performed better than any combination of geometric features. These results show a direct contradiction to the claim made by Xu et al. (2019) of predictive accuracy increasing with all geometric features. The models containing topology, shape diameter, and local vertex density showed depreciating results as expected; these models have fewer geometric inputs to provide knowledgeable features for the deep pose estimation model. However, their results are markedly worse than all other models.

Table 5.2 shows the results after inference is performed over the entire mesh dataset for each deep pose estimation model variant. These results seem to validate the behavior displayed in the test set other than a few notable exceptions. The model variant using topology and shape diameter performed much better. In fact, all model variants incorporating topology performed better overall than the trained model provided by Xu et al. (2019)[1]. Curiously, the only model variant that did not improve was the model including all geometric features; further proof of counter behavior to the behavior displayed in the model provided

Deep pose estimation model variant	Average chamfer dist
sdf+topology	**1.6327e-06**
sdf+topology+curvature	1.7989e-06
sdf+topology+density	1.6754e-06
sdf+topology+diameter	2.1050e-06
all features	1.8914e-06
model by Xu et al. (2019)[1]	1.9205e-06

Table 5.2: *Evaluation of models, trained on different geometric inputs using topology, using inference over the whole model dataset.*

by Xu et al. (2019)[1].

I speculate that bias does factor into the increased performance on inference over the whole dataset. Each model variant is trained using 80% of the dataset where models are fitted and fine-tuned. The inclusion of these meshes that impact the parameter tuning of the deep pose estimation models are bound to cause some undue influence. This is why we see different behavior in the independent evaluation of the test set. When analyzing the model variants: topology with principal surface curvature and topology with local shape diameter, we see the performance change minutely. This appears to be a symptom of underfitting where the model fails to capture the underlying variability of the dataset (Jabbar & Khan, 2015)[59]. Since the topology-only variant performs the best, I speculate the additional geometric features are causing the model variants to be biased towards those features. Conversely, the model variant featuring topology with local vertex density appear to display characteristics of overfitting given it performs poorer on the test set. To investigate this phenomenon further, I would begin by instituting k-fold cross validation to provide more reliable estimates and expose the model variants to new unseen types of data.

Chapter 6

Conclusion and Future Work

To conclude this thesis, the main contributions of this work is summarized, followed by remarks for current and future research of these contributions.

6.1 Summary

The chief objective of this thesis was to determine whether extracted curve skeletons provide better predictive accuracy than other geometric features, and from the results of the experiments, extracted curve skeletons can provide better geometric information for mapping animated rig joints and bones than any one of the five geometric features used by Xu et al. (2019)[1], including principle surface curvature, local vertex density, and local shape diameter. This provides further support for Zimmermann and Brox (2017)[60], Baek et al. (2019)[61], Chen et al. (2019)[62] who used 3D skeleton data with properties related to curve skeletons to train 3D pose estimation networks. These properties ensure the representation is simple, homotopic, locally centered, and fully connected. Topological data provided by curve skeletons, in conjunction with the signed distance function, produced the best predictive accuracy, when based on the average chamfer distance metric. This also reinforces the claim that the medial surface of an object is an optimal placement for automatically created rigs.

The work presented here and in Xu et al. (2019)[1] show the potential for practical uses of automatic rigging systems in 3D modeling software. By producing a viable rig, consistent with the general rig guidelines and criteria, character riggers need no longer start from scratch. Model datasets comprising various character templates will provide additional versatility and robustness for the architecture, providing character riggers with more templates to mold and adapt as needed. Even if a produced rig does not exactly meet the specifications of the rigger, this process saves time and reduces the complexity of rig construction.

6.2 Current and Future Work

The biggest limitation of using Python libraries for curve skeleton extraction is the fragmentation of skeletons. Whereas Mesh Skeletonization has no connectivity process, Skeletor can combine fragmented parts using a minimum weight spanning tree. Its edge cost function is based on the Euclidean distance between neighboring nodes. However, the extracted curve skeleton possesses no awareness of the original mesh's model space surrounding it. An edge-cost function penalizing bones protruding from the surface of the original mesh may give better qualitative results. Additional testing with other extraction methods would also be instructive. More accurate representation of the mesh topology should provide models with a better understanding of the medial surface, which in turn will make deep learning models such as the Stacked Hourglass Network more accurate.

During my initial round of testing, the model variant using topology and principle surface curvature, only outputted 5 candidate rigs after applying the soft non-maximum suppression step. Attempts to increase the parameter threshold, and thus remove the possible number of joint vertices, resulted in no observable change. However, after the threshold ablation study, this behavior disappeared. Additional testing and calibration on the soft non-maximum suppression parameters (kernel sigma and kernel size) may explain this behavior. Incorporating a post-processing convolutional network that learns non-maximum suppression may mitigate this (Hosang et al., 2017)[47].

The most interesting behavior in the experiment was that predictive accuracy depreci-

ated with additional geometric inputs features. This behavior was not seen in the original experiment by Xu et al. (2019)[1] where additional geometric input features increased the overall predictive accuracy. Future testing using k-fold cross validation may provide further insight into this phenomenon.

Bibliography

[1] Zhan Xu, Yang Zhou, Evangelos Kalogerakis, and Karan Singh. Predicting animation skeletons for 3d articulated models via volumetric nets. In *2019 International Conference on 3D Vision (3DV)*, pages 298–307. IEEE, 2019.

[2] Srinivasan Ramachandran. Meshskeletonization. https://github.com/aalavandhaann/Py_BL_MeshSkeletonization, Jan 2018. [accessed October 28, 2020].

[3] Philipp Schlegel. Skeletor. https://github.com/schlegelp/skeletor, Oct 2020. [accessed October 28, 2020].

[4] Nicu D Cornea, Deborah Silver, and Patrick Min. Curve-skeleton properties, applications, and algorithms. *IEEE Transactions on visualization and computer graphics*, 13 (3):530, 2007.

[5] Oscar Kin-Chung Au, Chiew-Lan Tai, Hung-Kuo Chu, Daniel Cohen-Or, and Tong-Yee Lee. Skeleton extraction by mesh contraction. *ACM transactions on graphics (TOG)*, 27(3):1–10, 2008.

[6] Wan-Chun Ma, Fu-Che Wu, and Ming Ouhyoung. Skeleton extraction of 3d objects with radial basis functions. In *2003 Shape Modeling International.*, pages 207–215. IEEE, 2003.

[7] Andrea Tagliasacchi, Hao Zhang, and Daniel Cohen-Or. Curve skeleton extraction from incomplete point cloud. In *ACM SIGGRAPH 2009 papers*, pages 1–9. 2009.

[8] Zeeshan Bhatti, Asadullah Shah, Ahmad Waqas, and Nadeem Mahmood. Analysis of design principles and requirements for procedural rigging of bipeds and quadrupeds

characters with custom manipulators for animation. *arXiv preprint arXiv:1502.06419*, 2015.

[9] Nicholas B Zeman. *Essential Skills in Character Rigging*. CRC Press, 2015.

[10] Ilya Baran and Jovan Popović. Automatic rigging and animation of 3d characters. *ACM Transactions on graphics (TOG)*, 26(3):72–es, 2007.

[11] Zeeshan Bhatti and Asadullah Shah. Widget based automated rigging of bipedal character with custom manipulators. In *Proceedings of the 11th ACM SIGGRAPH International Conference on Virtual-Reality Continuum and its Applications in Industry*, pages 337–340, 2012.

[12] Natapon Pantuwong and Masanori Sugimoto. A novel template-based automatic rigging algorithm for articulated-character animation. *Computer Animation and Virtual Worlds*, 23(2):125–141, 2012.

[13] Adobe. Mixamo: Animate 3d characters for games, film, and more. https://www.mixamo.com/#/, 2020. [accessed November 5, 2020].

[14] Alexander Toshev and Christian Szegedy. Deeppose: Human pose estimation via deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

[15] Alejandro Newell, Kaiyu Yang, and Jia Deng. Stacked hourglass networks for human pose estimation. In *European conference on computer vision*, pages 483–499. Springer, 2016.

[16] Jing Yang, Qingshan Liu, and Kaihua Zhang. Stacked hourglass network for robust facial landmark localisation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 79–87, 2017.

[17] Tamal K Dey and Jian Sun. Defining and computing curve-skeletons with medial

geodesic function. In *Symposium on geometry processing*, volume 6, pages 143–152, 2006.

[18] Polina Golland, W Eric, and L Grimson. Fixed topology skeletons. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition. CVPR 2000 (Cat. No. PR00662)*, volume 1, pages 10–17. IEEE, 2000.

[19] Yu-Shuen Wang and Tong-Yee Lee. Curve-skeleton extraction using iterative least squares optimization. *IEEE Transactions on Visualization and Computer Graphics*, 14(4):926–936, 2008.

[20] Nicu D Cornea, Deborah Silver, and Patrick Min. Curve-skeleton applications. In *VIS 05. IEEE Visualization, 2005.*, pages 95–102. IEEE, 2005.

[21] Nina Amenta, Sunghee Choi, and Ravi Krishna Kolluri. The power crust. In *Proceedings of the sixth ACM symposium on Solid modeling and applications*, pages 249–266, 2001.

[22] Tamal K Dey and Wulue Zhao. Approximating the medial axis from the voronoi diagram with a convergence guarantee. *Algorithmica*, 38(1):179–200, 2004.

[23] Carlo Arcelli and Gabriella Sanniti Di Baja. A width-independent fast thinning algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (4):463–474, 1985.

[24] C Min Ma and Milan Sonka. A fully parallel 3d thinning algorithm and its applications. *Computer vision and image understanding*, 64(3):420–433, 1996.

[25] Xun Jin and Jongweon Kim. A 3d skeletonization algorithm for 3d mesh models using a partial parallel 3d thinning algorithm and 3d skeleton correcting algorithm. *Applied Sciences*, 7(2):139, 2017.

[26] Julien Tierny, Jean-Philippe Vandeborre, and Mohamed Daoudi. 3d mesh skeleton extraction using topological and geometrical analyses. 2006.

[27] Lior Shapira, Ariel Shamir, and Daniel Cohen-Or. Consistent mesh partitioning and skeletonisation using the shape diameter function. *The Visual Computer*, 24(4):249, 2008.

[28] Vijai Jayadevan, Edward Delp, and Zygmunt Pizlo. Skeleton extraction from 3d point clouds by decomposing the object into parts. *arXiv preprint arXiv:1912.11932*, 2019.

[29] Mathieu Desbrun, Mark Meyer, Peter Schröder, and Alan H Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 317–324, 1999.

[30] Haolei Wang. Using density-based clustering to improve skeleton embedding in the pinocchio automatic rigging system. Master's thesis, Kansas State University, 2012.

[31] Thanh N Tran, Klaudia Drab, and Michal Daszykowski. Revised dbscan algorithm to cluster data with dense adjacent clusters. *Chemometrics and Intelligent Laboratory Systems*, 120:92–96, 2013.

[32] André Sobiecki, Haluk C Yasan, Andrei C Jalba, and Alexandru C Telea. Qualitative comparison of contraction-based curve skeletonization methods. In *International Symposium on Mathematical Morphology and Its Applications to Signal and Image Processing*, pages 425–439. Springer, 2013.

[33] Fuyang Huang, Ailing Zeng, Minhao Liu, Jing Qin, and Qiang Xu. Structure-aware 3d hourglass network for hand pose estimation from single depth image. *arXiv preprint arXiv:1812.10320*, 2018.

[34] Jonathan J Tompson, Arjun Jain, Yann LeCun, and Christoph Bregler. Joint training of a convolutional network and a graphical model for human pose estimation. In *Advances in neural information processing systems*, pages 1799–1807, 2014.

[35] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[36] Sungheon Park, Taehoon Kim, Kyogu Lee, and Nojun Kwak. Music source separation using stacked hourglass networks. *arXiv preprint arXiv:1805.08559*, 2018.

[37] Xiaokun Wu, Daniel Finnegan, Eamonn O'Neill, and Yong-Liang Yang. Handmap: Robust hand pose estimation via intermediate dense guidance map supervision. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 237–253, 2018.

[38] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1912–1920, 2015.

[39] Daniel Maturana and Sebastian Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 922–928. IEEE, 2015.

[40] Gernot Riegler, Ali Osman Ulusoy, and Andreas Geiger. Octnet: Learning deep 3d representations at high resolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3577–3586, 2017.

[41] Gyeongsik Moon, Ju Yong Chang, and Kyoung Mu Lee. V2v-posenet: Voxel-to-voxel prediction network for accurate 3d hand and human pose estimation from a single depth map. In *Proceedings of the IEEE conference on computer vision and pattern Recognition*, pages 5079–5088, 2018.

[42] Richard A Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J Davison, Pushmeet Kohi, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *2011 10th IEEE International Symposium on Mixed and Augmented Reality*, pages 127–136. IEEE, 2011.

[43] Thanuja Dharmasiri, Andrew Spek, and Tom Drummond. Joint prediction of depths, normals and surface curvature from rgb images using cnns. In *2017 IEEE/RSJ Interna-*

tional Conference on Intelligent Robots and Systems (IROS), pages 1505–1512. IEEE, 2017.

[44] Xiao Sun, Bin Xiao, Fangyin Wei, Shuang Liang, and Yichen Wei. Integral human pose regression. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 529–545, 2018.

[45] PyTorch. Crossentropyloss. https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html, 2019. [accessed November 24, 2020].

[46] Serim Ryou, Seong-Gyun Jeong, and Pietro Perona. Anchor loss: Modulating loss scale based on prediction difficulty. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5992–6001, 2019.

[47] Jan Hosang, Rodrigo Benenson, and Bernt Schiele. Learning non-maximum suppression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[48] Navaneeth Bodla, Bharat Singh, Rama Chellappa, and Larry S Davis. Soft-nms–improving object detection with one line of code. In *Proceedings of the IEEE international conference on computer vision*, pages 5561–5569, 2017.

[49] Ronald A. Cohen. *Lateral Inhibition*, pages 1436–1437. Springer New York, New York, NY, 2011. ISBN 978-0-387-79948-3. doi: 10.1007/978-0-387-79948-3_1379. URL https://doi.org/10.1007/978-0-387-79948-3_1379.

[50] Aaron Kershenbaum and Richard Van Slyke. Computing minimum spanning trees efficiently. In *Proceedings of the ACM annual conference-Volume 1*, pages 518–527, 1972.

[51] The VG Resource. The models resource. https://www.models-resource.com/, 2020. [accessed November 9, 2020].

[52] Patrick Min. binvox 3d mesh voxelizer. https://www.patrickmin.com/binvox/, February 2020. [accessed November 21, 2020].

[53] Péter Borosán, Ming Jin, Doug DeCarlo, Yotam Gingold, and Andrew Nealen. Rigmesh: automatic rigging for part-based shape modeling and deformation. *ACM Transactions on Graphics (TOG)*, 31(6):1–9, 2012.

[54] Philipp Schlegel. Navis - neuron analysis and visualization. https://navis.readthedocs.io/en/latest/, 2018. [accessed November 24, 2020].

[55] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[56] Stuart Russel, Peter Norvig, et al. *Artificial intelligence: a modern approach*. Pearson Education Limited, 2013.

[57] Max Kuhn, Kjell Johnson, et al. *Applied predictive modeling*, volume 26. Springer, 2013.

[58] Facebook. Pytorch3d. https://github.com/facebookresearch/pytorch3d/tree/18ce14cd31a2f1c99e16d727e1755156c360453a, 2020. [accessed November 7, 2020].

[59] H Jabbar and Rafiqul Zaman Khan. Methods to avoid over-fitting and under-fitting in supervised machine learning (comparative study). *Computer Science, Communication and Instrumentation Devices*, pages 163–172, 2015.

[60] Christian Zimmermann and Thomas Brox. Learning to estimate 3d hand pose from single rgb images. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[61] Seungryul Baek, Kwang In Kim, and Tae-Kyun Kim. Pushing the envelope for rgb-based dense 3d hand pose estimation via neural rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[62] Ching-Hang Chen, Ambrish Tyagi, Amit Agrawal, Dylan Drover, Rohith MV, Stefan Stojanov, and James M. Rehg. Unsupervised 3d pose estimation with geometric self-supervision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.