POLITECNICO DI TORINO

Master's Degree in Computer Engineering

Master's Degree Thesis

Machine learning for 3D human models remeshing for animation based on a semantic segmentation approach

Supervisors

Prof. FEDERICO MANURI

Prof. ANDREA SANNA

Candidate

EDOARDO NOVARA

12 2024

Abstract

Polygonal Meshes are the most widespread method of representing 3-dimensional shapes in Computer Graphics, with applications in a plethora of different fields. The need to generate meshes with a clean topology for simulation and animation drives an interest in algorithmic solutions that could automate the work of 3D artists. The goal of the thesis is designing a framework tailor-suited for remeshing realistic human 3D models for animation. To ensure a correct topology a machine learning algorithm is used to extract semantic features from an unstructured triangle human mesh. The semantic features are then converted to feature lines that guide a state-of-the-art remeshing algorithm. The 3D models obtained with the proposed pipeline are compared with the results of the standalone remeshing algorithm to evaluate how the performance improves.

Acknowledgements

First of all, I would like to thank my supervisors, Prof. Federico Manuri and Prof. Andrea Sanna for their invaluable feedback and help while researching, testing and finally writing the thesis.
Working on this thesis was the biggest challenge I faced in my academic career, and there were times I thought I wouldn't be on par with it. In those moments my family and Sofia were always by my side always offered their love and support. It is thanks to you that I was able to push through. I love you.
To all my friends in Marsala and in Turin, you mean so much to me and I hold dear to each of you. You bring so much joy in my life and I couldn't be any more grateful for that.
To all the wonderful people I met in PoliTo during those five years, I wish you only the best in life. Especially Agnese, whom I have always been able to count on as we shared the joys and sorrows of our university carreers.
Finally I want to acknowledge all the people openly sharing their knowledge in papers, software, media or answering in forums. Free access to information is the pillar of modern research and these people are the pillar of the community.

Edoardo

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Automatic remeshing is an open research field that has been a focal interest in computer graphics research for many years.
Meshes with an optimized topology are needed in many different scenarios, such as animation and numerical analysis. Despite the research efforts, automatic remeshing is still trailing behind the expertise of 3D artists especially for organic models, where artists can judge how to correctly place edge loops such that they flow smoothly around the semantic features of the shape. The goal of the thesis is combining automatic remeshing with machine learning techniques able to discern the semantic components as it is done natually by 3D artists.
The begginning of the thesis was spent studing state-of-the-art remeshing methods to evaluate which algorithm would benefit the most from machine learning methods and in which way. Taken into consideration the importance of 3D models in the context of cinema, XR and videogames, the research was narrowed to algorithms used to retopologise unstructured triangle meshes into quad or quad-dominant regular meshes. After compiling and testing multiple programs the Reliable Feature-Line Driven Quad-Remeshing [1] approach was selected for the pipeline. The main limitation of the algorithm is the usage of a dihedral angle technique to categorize feature lines, which works for models with sharp edges but is not able to recognize features in organic models. To overcome this restriction, a neural network for semantic segmentation was adopted, to substitute the lines extracted by the dihedral angle with edges that encompass the organic features of the model. The pipeline was first tested with DilatedToothSegNet [2] with the Teeth3DS [3] dataset with which it can be found pretrained on Github, as it is one of the most recent algorithms published (2024) that deals with semantic segmentation, and subsequently with MeshCNN [4] with the Human Body Segmentation Dataset [5]; in both cases the pipeline produced satisfactory results that yielded a visual improvement on the output. For the final pipeline, MeshCNN was chosen instead of DilatedToothSegNet as the respective dataset was deemed more relevant to the goal of the thesis. The

improvements on the output mesh with respect to the use of Reliable Feature-Line Driven Quad-Remeshing alone were measured with tools such as Blender, Meshlab and custom python scripts to compute other quality metrics such as the average deviation angle with respect to principal curvature directions, the average scalar jacobian and the average edge length deviation.

1.1 Context

1.1.1 Polygonal Meshes

Polygonal Meshes are one of the most common ways of digitally representing 3D shapes by efficiently approximating surfaces as a set of polygons. They are used in diverse scenarios, such as modeling, simulation, animation, architecture, mechanical engineering, medicine, virtual reality etc. Polygonal meshes can be described by their component elements, i.e. vertices, edges and faces.

Tris and Quad Meshes

The vast majority of polygonal meshes uses either triangles or quadrilaterals as their polygonal cells. While triangular faces have the obvious advantage of being the simplest two-dimensional elements and having a planar interior by construction, quadrilaterals can be better suited for graphical applications such as finite element analysis and deformation. Moreover, edges in quadrilaterals can be naturally aligned with the principal curvature directions of the mesh [6].
We call meshes that use triangles as polygonal cell triangle meshes, while meshes with quadrangular faces can either be quad meshes or quad-dominant meshes, if we allow for a small fraction of the faces to be 3-sided or 5-sided polygons.

Mesh Topology

A Mesh's Topology can be described as the arrangement and connectivity of its components. The topology of a mesh determines how the model reacts to numerical simulation, animation, how accurately represents a shape and how fast it renders. Some key concepts regarding topology are:

- Valence: The valence of a vertex is the number of incident edges. In quad and quad-dominant meshes, an internal vertex is said to be regular if its valence is 4, any vertex with a different valence number is called irregular or singularity [6].
- Edge Loops: Edge loops are a series of connected edges that form a continuous loop around a part of a model. They follow the natural contours and flow of

the object, making them critical to define semantic sections and to deform the mesh during animation.

- Manifold Meshes: A mesh is said to be manifold if every edge belongs to exactly two faces. A manifold mesh describes a shape without ambiguity.

- Tessellation Density: The tessellation density is the density of polygonal cells of a mesh. High tessellation leads to a more precise result that can capture finer details, while a lower tessellation can have faster render times. We can adaptively tessellate a mesh to get a finer resolution in critical areas that need to capture more detail or deform during animation.

- Isotropy: Isotropy describes the uniformity of the element shapes and their distribution across the surface. A mesh is said to be isotropic or to have high isotropy when the elements are uniformly shaped and sized, with consistent edge lengths and angles across the entire mesh. Isotropic meshes allow for even sampling of the surface, making them useful for applications where smoothness and uniform resolution are important, such as physical simulations, animation, or subdivision. A mesh is said to be anisotropic when the elements greatly vary in shape, size, or orientation. Anisotropic meshes are more often used to model static assets in real-time applications, where a higher number of polygons could lead to performance drops.

1.1.2 Meshes in Digital Animation

Meshes are ubiquitous in computer animation, whether for cinema, video games or commercials, and must go through a series of steps such as modeling, UV-mapping, texturing, rigging and animating before being composited in the scene.
Meshes used in animation must have certain topological requisites to ensure a correct rendering without visual artefacts, and this is particularly true for organic models that need to move in a life-like manner. Such meshes should have sufficient tessellation density on the joints, with well defined edge loops surrounding them, and a preferably symmetrical distribution of irregular vertices in areas that deform less. The edges should follow gracefully the flow of the shape.

Skinning

Meshes are animated by binding their vertices to a hierarchical structure of joints called a skeleton, in which each joint or bone controls the movement of a set of vertices. The influence of each bone on the vertices is determined by weight groups, which assign to each vertex a normalized scalar quantity called weight. The weight is used as a factor to determine the correlation between the bone's rotation and translation with that of the vertices. A topology that closely follow the direction of the bones deforms better and produces less visual artefacts.

1.1.3 Remeshing

Remeshing, also known in the industry as retopology or retopo, is the process of transforming a mesh's topology to reduce the number of vertices or to optimize it for numerical simulation and animation.
Remeshing can be manual, tool-assisted or fully automatic. Although manual and tool-assisted approaches are traditionally regarded as giving the best results, recent advancements in automatic retopology methods demonstrate significant improvements in quality each year.

Mesh Generation

Meshes can be created in many different ways. They can be modeled by 3D artists with sculpting, box modeling, procedural modeling, etc. They can be obtained from point clouds via photogrammetry or more recently, with the use of machine learning techniques [7].
Many of the aforementioned methods give as an output unstructured triangle meshes that are unfit for animation. Particularly, a pipeline comprised of sculpting and remeshing is considered as the industry standard as it gives high quality meshes with a clean topology.

1.2 Contribution

The main problem addressed by the thesis is the inability for fully automatic remeshing algorithm to recognize semantic sections of a model to guide the remeshing process, as it is done by professional 3D artists. A pipeline is proposed to address this problem and is then tested against the standalone remeshing algorithm used.

1.2.1 The Semantic Pipeline

The proposed pipeline, which will be referred to as 'semantic pipeline' in the following chapters, consists of the following steps:

- Neural Network: An unstructured triangle human mesh is given as input to a convolutional neural network that recognizes the semantic groups of the mesh and stores for each vertex its group ID in an .obj file

- Data Conversion: A custom python scripts reads the .obj file and extracts the information about the edge loops that border two different groups, which will be referred to as feature lines, then saves this information in a .sharp file
- Remeshing Algorithm: A remeshing algorithm receives as input the .obj file outputted from the neural network along with the .sharp file, then costructs a 4-RoSy tangent-vector field [8] aligned to the feature lines that guides the remeshing process.

1.2.2 Metrics

The metrics chosen to evaluate the quality of the output are the same used by the pipeline's remeshing algorithm [1]. To evaluate the animation capabilities of the mesh following the skinning process, the average angle of deviation from the principal curvatures has also been measured.

1.3 Expected Results

The reported data shows that the semantic pipeline increases the performance of the chosen remeshing algorithm under all chosen metrics, when tested with the human models from the chosen dataset. This translates to a final result that is hopefully easier to work with in the skinning process, and produces less visual artifacts when animated.

1.3.1 Structure of the Thesis

Chapter 2 will present an overview on automatic remeshing algorithms for quad meshes, with a particular focus on cross-field oriented methods. Additionally, Automatic Mesh Segmentation methods within the field of interest of the thesis will be discussed.
Chapter 3 will describe in detail the proposed pipeline and the script used to transfer data from the convolutional neural network to the remesher.
Chapter 4 discusses the motivation behind the metrics that have been chosen to evaluate the quality of the output, and shows through various tests how the semantic pipeline performs when fared against other remeshing algorithms.
Finally, chapter 5 declines the conclusions from the extracted data and analyzes possible future improvements both in the machine learning algorithm and the remeshing algorithm.
Appendix A show the scripts used to convert the results from the neural network and to measure some of the quality metrics chosen for the outputs.

Chapter 2

State of the Art

The analysis of the state of the art will give an overview on the most commonly used surface remeshing methods to optimize topology, with a particular focus on tris to quad remeshing through the use of cross fields. The algorithms will be organized chronologically by the respective paper's publication date, to highlight the advancements and the shifting trends in the remeshing field.
Lastly, as it is an important part of the semantic pipeline, an overview on relevant papers regarding mesh segmentation will follow.
The aim of this section is to give a comprehensive understanding on the research landscape and to justify the decision on the chosen algorithms, which will be done in section 2.3. The chapter directly refers to [6] and [9] as its main sources.

2.1 Overview of Remeshing Methods

Meshing algorithms can be mainly separated into local and global methods. Global methods are usually slower and may sometimes fail, since finding a global optimum is generally hard and sometimes not possible, however they produce far more accurate results with a reduced number of singularities. Local methods on the other hand are more resilient to imperfections in the input mesh and are able to better scale to larger meshes. Local methods can produce results instantaneously, although a greater number of irregular vertices will be present in the output mesh and the topology is locally optimized. Many advanced remeshing algorithms only partially use global methods to create the parameters that guide the local remeshing. This technique creates a good trade-off of execution speed and output quality.

2.1.1 Global Methods

Global algorithms are mostly based on parametrization, patches or fields.

- parametrization based methods act by mapping the 3D shape onto a 2-dimensional plane, where the problem of quadrangulation becomes trivial. The main challenge faced by this family of methods is defining a set of consistency conditions that ensure the quadrangulation to be consistent along the seams generated by the mapping.

- Patches based methods are also known as coarse layout based methods. instead of mapping the 3D model to a plane, this family of remeshing algorithms subdivide the surface of the mesh into a set of square patches that guide the final quadrangulation either by acting as a domain for a globally smooth parametrization or to be tessellated.

- A field is a value that varies smoothly over the entire surface. Fields are used to explicity determine the local properties of each quad element, such as the orientation of its edges and its size, and global properties such as the position of the irregular vertices.

Using Cross Fields for Remeshing

A field is defined as a physical quantity, represented by a scalar, vector, or tensor, that has a value for each surface point. A cross field assigns to each point a pair of vectors, which are used to construct a quad element by extracting the orientation and size of each vector. Cross fields are more often represented by a pair of vector fields: an orientation field that guides the direction and a sizing field to assign a size to each edge.
A relevant subclass of cross fields are n-rotationally-symmetric direction fields, in short n-RoSy fields [10]. n-RoSy fields represent rigidly coupled orthogonal crosses that are symmetric in n directions. the case where n=4, with $\pi/2$ rotational symmetry, is the most relevant for quad meshes as the properties of 4-RoSy fields are closely related to principal curvature directions.
In differential geometry, the two principal curvatures at a point on a surface correspond to the maximum and minimum curvatures, determined by the eigenvalues of the shape operator at that point. These curvatures quantify how the surface bends in different directions at that specific location. As previously discussed in section 1.1.2, aligning the mesh edges to the curvature of the shape is especially advantageous for the process of skinning as it leads to better deformation.

2.1.2 Local Methods

local algorithms are seldom used alone, as they are more suited for optimizing or coarsening a base mesh rather than comprehensively producing a new topology. They rely on topological operations such as swapping, splitting or collapsing edges. Some common local methods include:

- Central Voronoi Tessellation (CVT) [11]: in a Voronoi Tessellation, the shape is partitioned into regions known as voronoi cells based on a set of points determined by a random seed. The cells are delimited by assigning each point in the surface to the closest seed point. Voronoi Tessellation is said to be central when the generating point of each Voronoi cell is constructed iteratively to also be its mean. The final triangulation is then achieved by triangulating each cell. CVT obtains a high level of isotropy in the output meshes.
- Advancing Front [12]: the mesh is incrementally generated by starting from an initial boundary known as front and advancing inward. For each step a new polygonal cell, either triangular or quad, is generated by filling the space while maintaining the desired element size and shape.

2.1.3 Combining Global and Local Algorithms

Most commercial grade and experimental methods leverage the strengths of both of global and local approaches to deliver high-quality meshes while significantly reducing processing time. Some significant examples include:

- Instant Meshes [13]: Instant Field-Aligned Meshes integrates global and local meshing algorithms by introducing a new local smoothing operator, which is based on extrinsic energy. This operator is used to construct and optimize both the orientation and position of an N-rosy field, which then guides the quadrangulation process. Since the only global operations for this algorithm are the estimation of the alignment of the edges of the new mesh and element placement, it can execute in less than a second and can process meshes with several hundred million vertices. Since there's no global parametrization when computing the orientation and position fields, Instant meshes introduces a greater number of singularities in the final result compared to global methods.
- Quadriflow [14]: Quadriflow presents itself as an improved algorithm built upon Instant Meshes. Its goal is reducing the number of singularities in the final output. To do so, Quadriflow imposes on the construction of the position field using a Minimum Cost Network Flow problem which is solvable in polynomial time, generating a singularity-free field. This approach slows down the pipeline, as it claims executes the remesh a one-million triangle mesh

in 5 seconds. Furthermore, the position fields looses its natural alignment to the shape's features, resulting in a output that is not as precise in small detail areas of the mesh.

- Quadwild [1]: Reliable Feature-Line Driven Quad-Remeshing is a coarse-layout based method designed around preserving feature lines in hard surface meshes, with the goal of being an optimized method for automatic processing of CAD models. As its first step, Quadwild performs a dihedral angle technique to select the feature edges that will be preserved during remeshing. The algorithm then performs a first triangle-mesh optimization of the input using local operations to better process highly anisotropic meshes. An N-RoSy field is constructed so that it is aligned by construction to the feature edges and propagated over the surface. The N-RoSy field is used to guide the construction of paths that delimit the patches. Unlike other patch based methods, Quadwild allows for the creation of non-rectangular patches; however, thanks to the constaints imposed on the layout, the subsequent tessellation step is able to generate a regular quadrangulation for all patches.
  Quadwild was chosen for the remeshing step of the pipeline as it gives the best results amongst all considered methods.

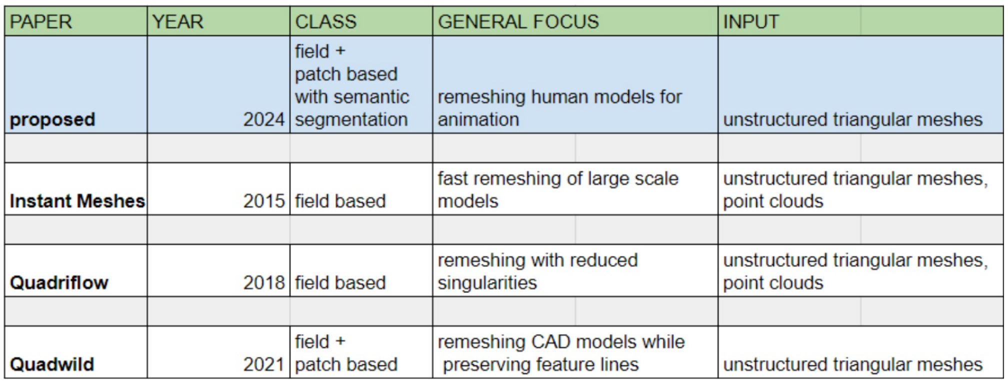
2.2 Relevant Studies on Mesh Segmentation

Mesh segmentation is an open field of research that is closely related to mesh manipulation. As it plays a major role in the proposed pipeline, basic knowledge of the state of the art for this topic will provide useful insights about its relation with the goal of the thesis and related arguments such as shape semantics. The presented data was extracted from [15].
Mesh segmentation is a process that involves dividing a complex mesh into smaller regions or components that can be considered as meaningful parts. This technique is widely used in various applications, such as shape analysis, animation, or object recognition, often as a preliminary step to improve the quality of such tasks. Breaking down a mesh into smaller components allows for more efficient processing for multi-threaded applications, gives better understanding of geometric features by highlighting their semantic role, and facilitates a variety of tasks in the 3D pipeline such as texture mapping, remeshing, and model simplification. Mesh segmentation techniques are related to image segmentation, machine learning and finite element partitioning.
Partitioning techniques can involve the faces, edges or even vertices of a mesh, although the most common approach is to define groups of faces.
Segmentation can have different objectives, and a meaningful distinction can be made between part-type segmentation, which focuses on dividing the object into

meaningful components, and surface-type segmentation, which uses geometric properties of a surface, such as its curvature, to created patches. part-type segmentation is based on the human understanding of images, as the segments can be considered as sub-meshes that have a meaningful semantic role in composing the totality of the shape. part-type segmentation has proved useful in defining skeletons for animation, as is the case in the paper Simultaneous shape decomposition and skeletonization [16].

Mesh Segmentation with Artificial Intelligence

More recently, Artificial Intelligence has been used in the context of semantic segmentation. This is the case with DilatedToothSegNet [2] and MeshCNN [4], the chosen segmentation algorithm for the proposed pipeline. Both CNNs are trained on a labeled dataset of meshes and take as input unlabeled meshes to perform segmentation.
Convolutional Neural Networks are conventionally used on images and work by sliding a small matrix known as filter or kernel on pixels to extract information such as edges, textures or shapes. In the context of meshes, convolutional filters are applied to a neighbourhood of faces or edges. Both MeshCNN and DilatedToothSegNet opt to use convolution on edges, but while MeshCNN uses a classical edge convolution combined with a pooling and unpooling strategy to simplify the mesh, DilatedToothSegNet introduces dynamic and dilated edge convolutions, which recompute the connectivity dynamically and expand the receptive field with Farthest Point Sampling (FPS).



PAPER	YEAR	CLASS	GENERAL FOCUS	INPUT
proposed	2024	field + patch based with semantic segmentation	remeshing human models for animation	unstructured triangular meshes
Instant Meshes	2015	field based	fast remeshing of large scale models	unstructured triangular meshes, point clouds
Quadriflow	2018	field based	remeshing with reduced singularities	unstructured triangular meshes, point clouds
Quadwild	2021	field + patch based	remeshing CAD models while preserving feature lines	unstructured triangular meshes

Figure 2.1: Recap of the algorithms reviewed in Chapter 2. In descending order: Proposed method, Instant Field-Aligned Meshes [13], QuadriFlow: A Scalable and Robust Method for Quadrangulation [14], Reliable Feature-Line Driven Quad-Remeshing [1]

2.3 Justification of the Pipeline

The motivation behind the proposed remeshing pipeline stems from the observation that, in a manual retopology process, a 3D artist typically starts by creating edge or face loops that define specific regions based on their understanding of the model's topology. This expert knowledge of the model's topology can, to some extent, be encoded in terms of semantic segmentation if we consider the definition of part-type segmentation discussed in chapter 2.2 as "a type of segmentation based on the human understanding of images", also discussed in A Survey on Mesh Segmentation Techniques [15]. By leveraging machine learning, it is possible to automate this segmentation process by identifying and defining semantically distinct areas of the mesh. This leads to the concept of automatically guiding the retopology process with a remeshing approach grounded in semantic segmentation of the mesh, ultimately streamlining the workflow and potentially enhancing the quality of the resulting topology.
While there exists multiple semi-automatic approaches to remeshing methods that aid segmentation, such as Surface remeshing with robust user-guided segmentation [17], where the user interacts with a Live-wire approach to define sharp features, a completely automatic approach is, to the extent of my knowledge, a novel idea in this field.
considering what has been discussed in the state of the art, Reliable Feature-Line Driven Quad-Remeshing [1] and MeshCNN [4] were chosen for the proposed pipeline for the following reasons:

2.3.1 Reliable Feature-Line Driven Quad-Remeshing

Reliable Feature-Line Driven Quad-Remeshing is the most recent remeshing algorithm discussed, for which the source code was also publicly available on Github. It presents itself as a versatile remesher that excels with hard-surface models such as mechanical components, but is able to work with organic shapes obtaining high quality results. The metrics presented in its paper show that it can produce pure quad meshes with a higher isotropy and less singularities than the other state of the art methods discussed below. Its efficacy was tested with the Thingi10K dataset, where it was able to succesfully remesh 9877 models.
Another reason that led to choosing this algorithm is its implementation and its similarities to the proposed approach: Reliable Feature-Line Driven Quad-Remeshing focuses on preserving feature lines, however it defines those lines only by the dihedral angle of edges. This works well for hard-surface models but it fails to recognize edges that divide semantic areas of a mesh. Such task is known to be well suited for Convolutional Neural Networks, which fall within the requirement for the thesis to use machine learning methods.

2.3.2 MeshCNN

MeshCNN is a versatile Convolutional Neural Network that takes as input meshes and is able to perform a variety of tasks, such as classification, segmentation, pooling and unpooling. It was choosen for the pipeline for its ease of use and availability, as it is one of the few networks publicly available that performs semantic segmentation on organic models. The authors of MeshCNN also provide a pre-trained version of the model. DilatedToothSegNet was also considered for the pipeline, as it is a more recent publication, but was ultimately discarded as converting the Human Body Segmentation Dataset in a suitable format for its use in combination with the network was non trivial.

Chapter 3

Pipeline Description

As previously stated, the goal for the thesis is to propose a remeshing pipeline that imitates the way a professional artist would retopologize an human mesh. This leads to defining two main problems:

- Understanding the mesh semantically and dividing it accordingly
- Using the segmentation results to guide the remeshing process

The proposed solution is based on using a segmentation network trained on human models to define the semantic regions of the target mesh, and defining the edges enclosing such regions as sharp features to preserve during remeshing.
In this implementation of the pipeline, a version of MeshCNN [4] trained on the Human Body Segmentation Dataset was chosen as the segmentation network, and Reliable Feature-Line Driven Quad-Remeshing [1] serves as the remeshing algorithm used.
The next section will explore the details of the proposed implementation.

3.1 Segmentation Network

While most approaches map the 3D mesh into a series of 2D projections to convert the irregular mesh data into a regular pixel grid, meshCNN operates directly on the irregular meshes by performing a version of pooling and convolution designed to work directly in the three dimensional domain.
In conventional Convolutional Neural Networks, convolution and pooling are two essential operations that work together to help extract features from images efficiently.

- Convolution: This is a mathematical operation where a filter (or kernel) slides over an input image (or feature map) to produce a new output feature map. The filter detects specific features, like edges, textures, or shapes, by computing the dot product between its weights and the input pixels within its receptive field. As a result, convolution highlights important spatial features in the input, which the network can use for tasks like classification or object detection.

- Pooling: Pooling is a downsampling operation that reduces the spatial size of feature maps, decreasing the number of parameters and computational load. The most common type is max pooling, which slides a window over the input feature map and takes the maximum value within each window. Pooling helps retain important features while making the model more robust to slight translations or distortions in the input image.

Together, convolution and pooling allow CNNs to focus on high-level patterns in data while reducing computation and making the model more efficient. In the context of mesh CNN, convolution and pooling operations are adapted to the structure of triangular meshes, focusing on edges rather than pixels.

- Convolution: In MeshCNN, convolutions are applied directly to the mesh edges. The convolutional neighborhood consists of the four edges of the two triangles sharing the target edge. This convolutional neighborhood structure allows to make the feature extraction process invariant to rotation, scale, and translation, by exploiting the natural geodesic connections within the mesh. While traditional CNNs trained on images need a regular grid structure, MeshCNN operates on edges using their relative geometric properties to interpret them as a network structure.
  It is imperative for convolutions to be invariant to tranformations as this allows the network to generalize better by recognizing objects or patterns regardless of their orientation or size in the data, rather than memorizing specific orientations.
  This issue is addressed by storing input edge features as a sorted 5-dimensional vector composed of the dihedral angle, two inner angles and two edge-length ratios for each face. The edge ratio is between the length of the edge and the perpendicular line for each adjacent face. Global ordering is not necessary for edges as convolution is performed on local neighbourhoods.

- Pooling: Pooling in MeshCNN is achieved through an edge collapse operation, which reduces the number of edges while maintaining the mesh's topology. The network learns which edges to collapse based on their feature importance, making the pooling task-driven. By collapsing edges that contribute less to the network's objectives, MeshCNN adapts the mesh resolution dynamically. This approach not only down-samples features but also exposes crucial shape details in areas of interest, allowing the network to retain and refine important

shape characteristics across varying mesh sizes.
Edge collapse is prioritized according to the strength of the features of the edge, which is taken as their $\ell 2$-norm. The $\ell 2$-norm (or Euclidean norm) of an edge's feature vector is the square root of the sum of the squares of its feature components. This value provides a single measure of the overall "strength" or magnitude of the edge's features. Edges with smaller $\ell 2$-norms are considered weaker or less informative and are thus collapsed first, while stronger edges with higher $\ell 2$-norm values are preserved.

MeshCNN performs mesh segmentation by classifying edges in 3D meshes as part of different semantic regions. To achieve this, individual edges gets a label assigned using supervised learning, indicating they belong in specific areas of the object according to how the model is trained. The released version of MeshCNN has been trained for testing on the COSEG and the Human Segmentation Datasets, both of which contain the meshes in .obj format and the ground truth labels in .seseg format. These datasets label each face, and these face-level labels are converted to edge-level labels to fit MeshCNN's edge-centered processing. To enable high-resolution segmentation, the network uses mesh pooling and unpooling layers: pooling layers progressively simplify the mesh by collapsing edges, while unpooling layers restore the mesh to its original resolution, enabling detailed segmentation. This combination allows MeshCNN to work on larger areas of the mesh and later refine the segmentation with unpooling. MeshCNN's segmentation process is well-suited to work on 3D meshes as it able to effectively adapt to diverse shapes and topologies; this makes the network resilient to variations in mesh triangulation, meaning it can be used to infer new results from meshes not present in the training set.

3.1.1 Implementation

MeshCNN is implemented in Python using the PyTorch library and has no visual interface. Shell scripts are provided to call the code and set the arguments to default values. In this section the relevant scripts for mesh segmentation will be briefly discussed.

- Training: the main scripts for the training phase are train.py and mesh_classifier.py. train.py calls the scripts to import the dataset and create_model.py, which in turn calls mesh_classifier.py in segmentation mode to create a ClassifierModel instance. The script loops through each epoch, feeding batches of data to the model and calling model.optimize_parameters() for each batch.
  the optimize_parameters method performs a forward pass to compute the model's predictions on the current batch of data and a loss calculation and

backward pass to calculate the loss function and gradients. An optimization step updates the model's weights based on the calculated gradients.

- Inferring: New predictions are inferred through the test.py script, which utilises the test(self) method from create_model.py to run a forward pass on the testing dataset, logging the overall accuracy for the epoch.
  For each mesh, the network outputs four version of the model in .obj format at 1500 tris (max quality), 1200 tris (80% of the original faces), 900 tris (60%) and 400 tris (27%). The information regarding the segmentation label is added in the edge section of the .obj file, as an integer in the range 0 to 7.

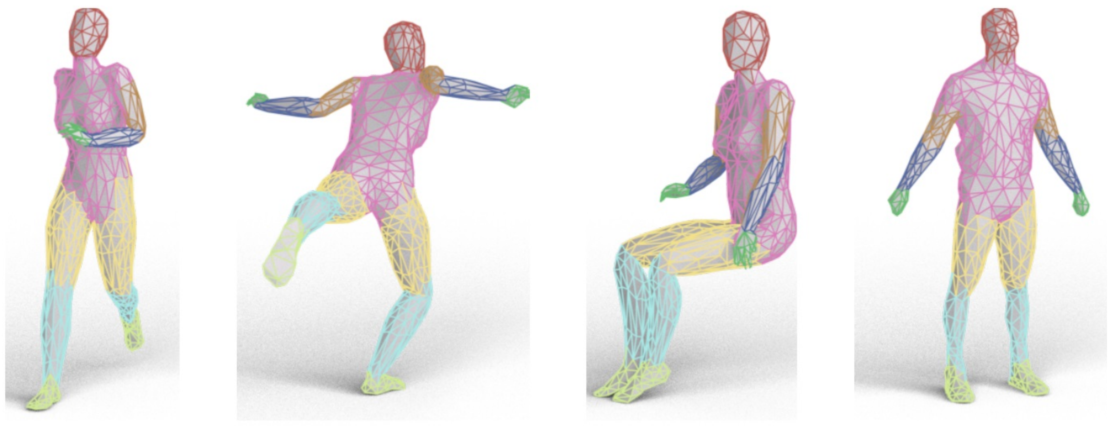


Figure 3.1: visualization of MeshCNN segmentation results

3.2 Dataset

The Human Body Segmentation Dataset was first introduced in Convolutional neural networks on surfaces via seamless toric covers [5]. The training set is composed of 370 models from the SCAPE [18], FAUST [19] and MIT animation datasets [20], while the test dataset is composed of the 18 sphere-like human models from the SHREC dataset [21].
All meshes in the training set were acquired via 3D scanning of real life subjects in various poses and manually segmented into eight labels (head, torso, upper arm, lower arm, hand, upper leg, lower leg, foot) according to the labeling used in Learning 3D Mesh Segmentation and Labeling [22]. Meshes in the test dataset are hand modeled, watertight and manifold human figures, both clothed and unclothed. All 18 meshes in the SHREC dataset are comprised of 1500 isotropic triangular faces

and 4500 vertices, and they have different scale and rotation to ensure invariance to geometric transformations.

3.2.1 Implementation

The implementation used is provided by meshCNN, along with a script to adapt other datasets in the correct format for the network.
Both the test and train datasets have their models stored in .obj format. The data on segmentation is provided by corresponding .eseg and .seseg text files with the same name as the mesh they're referring to.

- .eseg: edge segmentation id files are a list of length #edges, where each row gives the segmentation class of the edge.
- .seseg: soft edge segmentation ids files are a matrix #edges x #classes, with #classes=8 for the Human Body Segmentation Dataset. Seseg is computed from eseg, by looking at the segmentation ids of the 1-ring neighbors of each edge and is used to compute test accuracy. For a particular edge (i.e., a row in seseg), any column/s has a value between 0 and 0.5 to define the correct segmentation label. This allows for each edge to have more than one correct segmentation class, which is useful for boundary edges.

3.3 Extracting Results

As previously stated, the output of MeshCNN are .obj files containg additional information regarding the edge labels, represented as an integer from 0 to 7. This data needs to be converted in a format readable by Reliable Feature-Line Driven Quad-Remeshing, which is, according to the implementation, a text file in .sharp format structured as follows:

```
sn          //number of sharp features
t0 f0 e0     // for each sharp edge:  the first integer is 0 if the
edge is concave and 1 if convex, then the face and the index of
the sharp edge
...
tn fn en     // nth sharp edge
```

For the purpose of the pipeline, sharp edges are defined as the edges separating two different semantic areas of the model, i.e. all edges that belong to two faces in a different semantic group. This implies that the edges considered are not isolated but part of edge loops. Furthermore, said edge loops will be naturally placed in

proximity to the joints of the model, ensuring that the remeshing algorithm will preserve the principal curvature direction in areas more susceptible to deformation during animation.
To extract the label information from the output of meshCNN, a custom python script is used. The script, named featureExtractor, takes as input the path to the .obj file to use as input and the .sharp file to use as output. For a better understanding of how the algorithm works, basic knowledge of the .obj data format is needed.

Wavefront .obj File

OBJ is a geometry definition file format first developed by Wavefront Technologies for its Advanced Visualizer animation package. The file format is open and has been adopted by other 3D graphics application vendors.
in its most basic version, obj files contain:

- an array of vertices, indicated by the letter 'v' and their x, y, z coordinates.
- an array of faces, indicated by the letter 'f', and the id of the three constituent vertices. Vertices are noted in a counter-clockwise order by default, making explicit declaration of face normals unnecessary.

The output .obj files from meshCNN also have an array of edges indicated by the letter 'e', the id of the two constituent vertices and the id of the semantic group.

3.3.1 Code Implementation

The goal of the extracting script is to extract the semantic group boundary edges and convert their representation in one that is readable by the remeshing algorithm. For this purpose, we first create an Edge and a Face class.
The Edge class contains a dictionary where the key is the semantic label of the edge, while the value is a tuple (v1, v2), with v1 the Id of the first vertex and v2 the Id of the second vertex. The Face class is defined as an array of three elements from the edge class.
After opening the .obj file, the Face class is initialized from the lines starting with 'f', and initially setting the key values for each edge to 0. The Edge class is then initialized from the 'e' lines of the .obj file.
Following the initialization of both Face and Edge classes, the initial edge keys in the Face class are substituted with the correct keys by checking for each edge if either the tuple (v1, v2) or (v2, v1) belongs to a face.
To extract the boundary edges between two semantic groups, all edges that belong to a face where the other two edges have a different key, i.e. are in another semanti

group, are stored in a list of tuples as (index, face), where index is their index relative to the face (either 0, 1 or 2) and face is the face index. The list is printed in the output .sharp file along with its size. For this implementation, all edges are considered concave.
The full code can be viewed in Appendix A.

3.4 Remeshing Algorithm

Reliable Feature-Line Driven Quad-Remeshing falls into both field aligned methods and Coarse-Layout based Methods, as the patch layout is guided by the cross field calculated on the mesh. In the proposed pipeline, the dihedral angle measurement that defines feature lines is skipped, given that the edges to be marked as feature lines are listed in the .sharp file obtained from the previous step. The remainder of the algorithm follows the normal execution method as described in its paper, which is described below.

3.4.1 Steps Breakdown

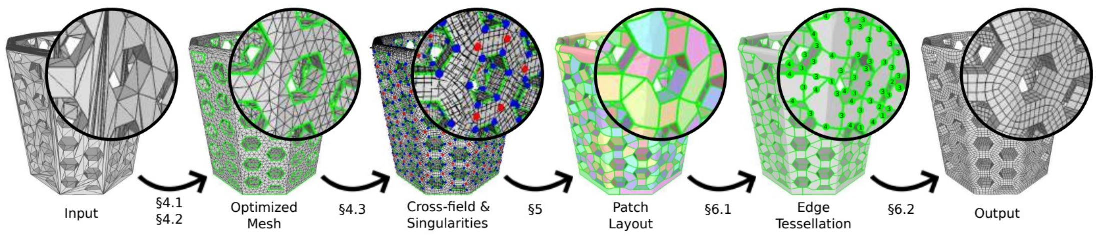


Figure 3.2: visualization of Quadwild's remeshing pipeline

Preliminary Input Mesh Optimization

This optimization step uses local operations such as edge flips, collapses, and splits following the approach of Hoppe et al. [23] to regularize edge lengths without disrupting any feature-lines by explicitly disallowing any operations that would interfere with them. This process is carried out in two passes:

- Uniform Target Edge-Length Pass: First, a uniform edge length, set to half the desired edge length in the final quad-mesh, is applied, resulting in an evenly shaped triangle mesh.
  In the proposed pipeline this steps acts similarly to adding a level of subdivision for the input mesh, as the desired edge lenght for the output is smaller than the average edge length in the input mesh.

- Adaptive Edge-Length Pass: A second pass adjusts edge lengths adaptively, based on triangle aspect ratios, to enhance local resolution around conglomerations of feature lines. This steps enforces the local resolution of the input mesh to be roughly adaptive to the complexity of its feature-lines.

While all 3D objects from the Human Body Segmentation test Dataset have moderately high isotropy, the remeshing pass allows for a better support of the subsequent cross-field construction and path tracing steps.

Cross-Field Definition

A 4-RoSy tangent-vector field [10] is constructed starting from faces adjacent to feature lines and propagated using the field propagation method described in Diamanti et al. [24]. Soft constraints are added during field propagation to reinforce alignment to principal curvature direction, similar to the method used in Panozzo et al. [25].
To ensure that each face is adjacent to at most one feature-edge, all faces adjacent to multiple feature edges are split. This step is crucial because it allows each face to receive a clear field alignment direction.

Patch Layout Decomposition

Reliable Feature-Line Driven Quad-Remeshing introduces the novel idea of allowing non-rectangular patches and T-junctions in the patch layout, while still constraining it such that it admits global consistency.
In this step of the pipeline, a series of field-aligned paths are traced along the surface, starting from converting feature edges into path edges. Paths are allowed to intersect only if they follow orthogonal directions of the cross field; they can either form closed loops, end at mesh boundaries or on other paths, generating T-junctions.
To prevent deformation along feature-lines, the mesh is cut open along these edges. This operation splits each feature-edge into two boundary-edges to which a direction from the cross field is assigned, and vertices at these edges are duplicated to create independent boundary-vertices. Any field singularities at these duplicated vertices are resolved by averaging the directions of adjacent faces, ensuring smoothness across boundary edges.
With the paths traced and edges split, the vertices along each path are classified based on the direction from the cross field they're aligned to. Vertices are labeled as straight, right-turn, left-turn, or U-turn, depending on the degree of rotation required for adjacent edges to align. This classification is crucial because patches must adhere to specific constraints on these angles to ensure valid quadrangulation.
To obtain a valid and high quality quadrangulation, patches must adhere to a set

of quality conditions. If it is impossible for a patch to do so, it must at least adhere to a set of validity conditions. These two sets of conditions guarantee that the final quadrangulation will not only be achievable but also have a higher quality.
Validity conditions for patches are:

- Topological constraint: The patch must be homeomorphic to a disk, i.e. it should have a single, unbroken boundary loop and no internal holes or disconnected components.
- Valence constraint: The patch must have between 3 to 6 edges
- Convexity constraint: The patch must be convex

Validity conditions are designed to be easily applicable as they are considered as a "fallback" strategy. Whenever possible, any patch should follow the following quality conditions:

- Geometric Condition: patches sides must approximately be the same length, with an added tolerance equal to the length of the shortest side.
- Valence Match: patches should contain at most 1 field singularity. If a patch is rectangular, it should not have any internal singularity. Non rectangular patches should have a valence equal to that of the internal singularity if present. This favors a more precise alignment of the final mesh to its cross field.

To satisfy these conditions, new paths are added iteratively in rounds by selecting a set of starting and ending nodes, with each round targeting specific objectives. All rounds introduce a large number of candidate paths, which are chosen if they contribute to satisfy the conditions or if they split a patch that does not meet said criteria, making problematic patches smaller. The rounds are performed as follows:

- Convexity enforcement round: For each non-convex vertex, candidate paths are traced to fix these configurations by either going straight or turning right, thereby adjusting the layout into a valid, convex shape. If non-convex patches remain after the first round, a second pass is performed. In this subsequent pass, candidate paths are allowed to end on T-junctions.
- Looped-paths round: To increase structural support for the final quadrangulation, a series of closed-loop paths are added. Such paths also simplify the layout and contribute towards meeting patch requirements.
- Border-to-border paths round: The final round resolves any remaining issues in the layout by connecting straight boundary nodes to each other. This round makes the patches fully subdivided and suitable for quadrangulation.

If after any round a patch fails to achieve the target, the round is repeated only within the irregular patch.
After all rounds have been performed, a cleanup phase removes any redundant paths that don't contribute to the layout's overall quality.
The tracing of a candidate path is interpreted as a shortest path search problem for a graph. First, an auxiliary directed weighted graph is derived from the mesh by representing each vertex as four nodes, to account for each direction in the cross-field. Cross field singularities are not represented in the graph, to ensure that each path segment follows consistently a single cross field direction. For a pair of nodes to be connected in the graph, they need to belong to vertices that share an edge, align with the direction of the cross-field, and have an angle of less than 45° between their direction and that defined by the field.
Since paths drawn along mesh edges can appear jagged, an additional virtual connection is added between nodes that are separated by two edges. This connection minimizes zigzagging by allowing the path to skip over intermediate nodes, reducing accumulated drift without adding length to the path.

Final Quadrangulation

The final patch quadrangulation requires to determine the number of edges on each side of all patches such that adjacent patches align. This is expressed as a global Integer Linear Program (ILP) that aims to minimize an objective function given by the sum of several weighted terms, subject to constraint such as:

- Parity Constraints: Each patch must have an even number of edges along its boundary to allow quadrangulation.
- Isometry Term: The term minimizes the deviation between actual and target edge lengths, promoting consistency in edge length across patches. This is computed as $\sum(s_i - \hat{s}_i)^2$, where s_i represents the number of edges assigned to a specific segment i along a patch boundary and $\hat{s}_i$ is the ideal edge count for segment i, calculated by dividing the actual geometric length of the segment by the target edge length. This target length is based on the desired quad size in the final mesh.
- Regularity Terms: These terms strongly encourage opposite sides of rectangular patches to have equal edge counts and non-rectangular patches to meet specific conditions. Specifically, 3-Sided patches should satisfy the inequality $\forall i,\ e_i \leq e_{i+1} + e_{i+2}$, with e_i the number of edges on side i, 5-sided patches should satisfy $\forall i,\ e_i + e_{i+1} + e_{i+4} \geq e_{i+2} + e_{i+3}$ and 6-sided patches should satisfy $\forall i,\ e_i \leq e_{i+2} + e_{i+4}$ and $e_i + e_{i+2} + e_{i+4} = e_{i+1} + e_{i+3} + e_{i+5}$. Conditions are strongly encouraged instead of enforced because imposing them might make the system unfeasible.

- Singularity Alignment Term: This term promotes the reciprocal alignment of singularities for non-rectangular patches by identifying pairs of either adjacent patches or patches separated by a sequence of rectangular patches and balancing edge counts on adjacent sides.

The last step is the individual quadrangulation of all patches. This either follows the simple strategy, which is equivalent to that proposed in Closed-form Quadrangulation of N-Sided Patches [26] or, if the patch only respects the validity conditions, the fallback strategy, which matches the method proposed by Pattern-Based Quadrangulation for N-Sided Patches [27]. To further enhance the final mesh a step of tangent space smoothing [28] is applied to improve quad shapes.

3.4.2 Implementation

The implementation of Reliable Feature-Line Driven Quad-Remeshing, known as Quadwild, is mostly written in C++ and C. Quadwild relies on C++ libraries such as Boost and BLAS for linear algebra operations, Gurobi for the global Integer Linear Program described in the final quadrangulation step and CoMISo (Constrained Mixed-Integer Solver) for geometry processing when dealing with organic meshes.
It does not have a visual interface but can be accessed through the terminal command ./quadwild <mesh> [.txt setup file] [.rosy file] [.sharp file]. The first parameter is the path to the .obj or .ply model to remesh, the optional parameters perform the following functions:

- [.txt setup file]: Contains the pipeline parameters. It enables or disables the initial remesh in the model setup step, specifies the dihedral angle for sharp features, balances the regularity and isometry of the final tessellation and determines the scale factor of the final quadrangulation
- [.rosy file]: Substitutes the 4-rosy field computed during the pipeline with a custom field described by the user
- [.sharp file]: Contains the information of the sharp features. It substitutes the dihedral angle technique normally adopted in the pipeline.

The program has multiple outputs that can be used to assess the various steps of the pipeline. The re-meshed triangulated mesh (suffix rem.obj), with the relative field and the sharp features, if automatically computed (.rosy and .sharp files); the mesh decomposed after the tracing (suffix rem_p0.obj); the patch decomposition (.patch file) containing the patch index for each triangle of the rem_p0 mesh; the files with suffix .corners, .c_feature, .feature that contain per patch information and the final quadrangulation before smoothing (quadrangulation.obj). The final quadrangulated mesh file is stored in the quadrangulation_smooth.obj file.

Chapter 4

Experimental Results

The goal of this chapter is to present a series of metrics that highlight the performance of the pipeline and topological quality of the output meshes.
The results are compared against Quadwild as a standalone program and the Blender 4.1 implementation of Quadriflow [14]. In addition, the pipeline is tested with reduced quality inputs and the resulting reduction in output quality is compared to Quadwild. Malformed meshes and meshes from the Tele-Aliens [29] dataset are also used as input to test the performance of meshCNN when tasked to apply segmentation to models that differ from the existing training dataset. Finally the pipeline is tested with different values in the .txt setup file.
In all tables, the pipeline is noted as SP, which stands for Semantic Pipeline.

4.1 Evaluation Criteria

4.1.1 Quality Metrics

To check the topological quality of the output and its suitability for animation, the chosen metrics are:

- number of vertices #V and number of faces #F: one of the main parameters in Quadwild is the target edge length. For a mesh with area A, the default target edge length is $A/10^4$. A smaller number of vertices directly relates to less singularities, as more regular structures require fewer additional vertices or complex connections to accommodate these irregularities.
  It is also observable that for all meshes the relationship #V-#F=2 is held. This demonstrates that the resulting meshes are topologically equivalent to spheres (i.e. they have no holes or boundaries) as they have the same Euler characteristic χ of a sphere.

- number of singularities #I: Skinning algorithms work best when weights can be smoothly distributed across a regular grid of quads.
  Irregular vertices directly impact the quality of animation, as breaking the regular grid leads to unpredictable stretching or compression around them, especially in areas with significant movements such as joints. A higher number of singularities directly translates to a worse performance when deforming the mesh in the animation process. The number of vertices, singularities and faces were measured by importing the meshes into Blender.

- average angle deviation from 90° AD(°): This metric measures how closely the angles within each quadrilateral cell approach the ideal 90° target for a perfectly square or rectangular shape. Maintaining angles close to 90° ensures that the mesh elements are as orthogonal as possible, which promotes uniformity and reduces distortion. Quad elements with angles significantly off from 90° can cause problems in deformation and interpolation, as they tend to stretch or compress unevenly, leading to inaccuracies in simulations and visual artifacts in animation. The average angle deviation was measured by importing the models into meshlab and using the "Compute topological measures for quad meshes" on the filters tab.

- average edge-length deviation from the average edge-size ED(%): This metric assesses the uniformity of edge lengths within the mesh by comparing each edge to the overall average edge length. Large deviations in edge length lead to irregular element shapes and sizes, which can result in localized stretching or compression, causing instability in simulations and less predictable deformations in animation.

- average Scaled Jacobian SJ: The Scaled Jacobian mesh quality criteria measures the deviation from the perfect element in the geometrical sense. This measure normalizes the range of reported values between [0,1] for a normal element, the value of 1 is considered a perfect element and 0 a element with a collapsed side. The scaled Jacobian measures the shape quality of each quadrilateral element by calculating the minimum of the Jacobian at each corner divided by the lengths of the 2 edge vectors.
  Reflecting both angle quality and shape distortion, this metric is a robust indicator of element shape quality, as it combines information on angles, scaling, and distortion within each element. Using it along the average angle deviation from 90° and the average edge-length deviation from the average edge-size allows for a well-rounded assessment of mesh quality. This multi-faceted approach ensures that no single quality issue is overlooked and that the mesh is robust and well-suited to organic deformations.

- average principal curvature deviation PCD(°): In differential geometry, the two principal curvatures at a given point of a surface are the maximum and minimum values of the curvature as expressed by the eigenvalues of the shape operator at that point. They measure how the surface bends by different amounts in different directions at that point. This metric assesses how well the edges of each quadrilateral element align with the natural curvature of the surface, which has a significant impact on deformation behavior, visual fidelity, and efficiency in animations. Aligning mesh edges with principal curvature directions allows the mesh to bend and flex more naturally along the surface's contours. In the context of 3D human meshes, quads aligned with muscle structures or facial features will deform in a way that respects the natural flow of the geometry, producing smoother and more lifelike motions. The average edge length deviation, scaled jacobian and principal curvature deviation were measured using python. The scripts are visible in Appendix A.
- sharp edge length SL: when confronting the pipeline with Quadwild, the sharp edge length is also noted. This measure does not translate directly to an improvement in quality but offers insight on how the number of edges considered 'sharp' is highly reduced in the pipeline.

All presented metrics, apart from the average principal curvature deviation and sharp edge lenght, were also chosen by the authors of Reliable Feature-Line Driven Quad-Remeshing to compare Quadwild with other remeshing algorithms.

4.1.2 Performance Metrics

to evaluate the performance of the proposed pipeline, along with that of Quadwild outside of the pipeline and Quadriflow, the execution time was measured. All performance tests were executed on a consumer-level laptop with Intel Core i7-8550U CPU, 12GB DDLR4 Ram memory, Nvidia GeForce MX130 with CUDA 12.4 and Ubuntu 22.04.4 as the OS.

4.2 Extracting Data From the Results

4.2.1 Quadwild Quality Comparison

For each mesh with maximum input quality (_0 subfix), the results are compared between the proposed pipeline, denoted by "with .sharp" in the method column, and Quadwild alone, denoted by "without .sharp". For both methods, the PCD(°), execution time and sharp edge length are given.
The average principal curvature deviation was only calculated for the high quality meshes, as the output meshes from reduced input quality are not suitable for

animation, nevertheless they prove useful to measure quality decline in the pipeline's output.
The execution time and total sharp edge length given by the .sharp file for the pipeline and the dihedral angle for the program are also compared.

	AD(°)		**ED(%)**		**SJ**		**PCD(°)**		**SL**		**T (m)**	
Mesh#	SP	QW	SP	QW	SP	QW	SP	QW	SP	QW	SP	QW
shrec_1_0	10.28	16.54	22.28	28.16	0.95	0.9	13.28	18.63	5.84	18.83	01:51	02:04
shrec_2_0	9.82	13.12	24.03	24.3	0.96	0.93	14	17.94	7.3	24.15	01:36	01:55
shrec_3_0	12.16	13.69	25.76	23.84	0.94	0.92	13.81	15.93	5.56	12.93	01:52	02:15
shrec_4_0	10.913	16.12	24.44	25.54	0.95	0.9	13.76	18.85	5.25	14.24	02:02	02:28
shrec_5_0	12.6	14.36	25.06	24.51	0.94	0.91	14.86	18.09	8.43	24.61	01:53	02:08
shrec_6_0	9.76	16.3	22.63	27.18	0.95	0.9	13.79	17.15	9.88	30.62	01:53	02:14
shrec_7_0	9.7	16.15	26.48	26.33	0.95	0.89	13.64	15.57	8.35	21.87	01:38	02:03
shrec_8_0	8.58	17.39	21.78	23.87	0.96	0.88	13.23	18.3	5.86	22.73	02:03	02:31
shrec_9_0	10.24	12.71	20.78	22.04	0.96	0.93	12.91	14.39	5.22	18.09	02:04	02:21
shrec_10_0	12.76	12.96	22.06	23.18	0.94	0.93	14.12	18.55	6.52	23.62	01:43	02:02
shrec_11_0	8.17	12.88	20.94	23.8	0.96	0.92	12.4	15.29	8.37	25.21	01:58	02:08
shrec_12_0	11.75	9.86	21.73	20.76	0.95	0.95	12.91	13.14	76.32	147.38	01:35	01:35
shrec_13_0	11.83	13.77	22.68	25.53	0.95	0.92	14.39	17.19	9.3	24.39	01:44	01:43
shrec_14_0	10.91	13.7	25.25	22.82	0.94	0.92	13.43	20.77	4.91	13.8	02:08	02:17
shrec_15_0	11.07	11.07	24.08	22.2	0.93	0.94	14.83	19.08	5.41	16.59	02:09	02:28
shrec_17_0	12.26	14.81	22.36	24.07	0.94	0.91	15.68	18.23	5.89	24.4	02:09	02:14
shrec_19_0	13.89	17.3	27.89	26.07	0.91	0.89	15.08	17.71	8.05	25.74	02:09	02:16
shrec_20_0	12.68	12.57	23.88	22.22	0.94	0.93	14.07	18.36	6.83	26.59	02:08	02:27
average	**11.07**	14.18	**23.56**	24.24	**0.945**	0.915	**13.9**	17.4	**10.35**	27.95	**01:55**	2:10

Table 4.1: Comparison between the proposed pipeline (SP) and Quadwild (QW) with default parameters and dihedral angle technique.

We can observe from Table 4.1 that the pipeline produces a higher quality result for every single mesh in the test dataset.
The average PCD for Quadwild is 17.40, compared to the average PCD for the pipeline of 13.90, a reduction of approximately 20%. Performance wise, execution speed is also roughly 11,5% faster in the proposed pipeline, with an average time of 01:55 minutes compared to 02:10 for Quadwild.
Sharp edge length is greatly decreased with feature lines extracted from meshCNN results. Excluding the outlier shrec_12_0.obj, which is simply scaled up to 9 times the size of the other meshes in the test dataset, the average edge length is just 6.88, compared to 21.67 derived from the use of dihedral angle, decreasing it by 68,25%. It is important to note that the sharp features extracted by both methods do not

refer to the same edges. Moreover in the case of the pipeline all sharp edges belong to closed loops by construction, while they might be isolated when the dihedral angle technique is used.
The Average Scalar Jacobian is increased from an average of 0.915 to 0.945. It is possible to observe that this improvement is mostly due to the average deviation from 90° angles, which has been reduced from 14.18° to 11.07°, while the average edge length deviation went from 24.24 to 23.56.

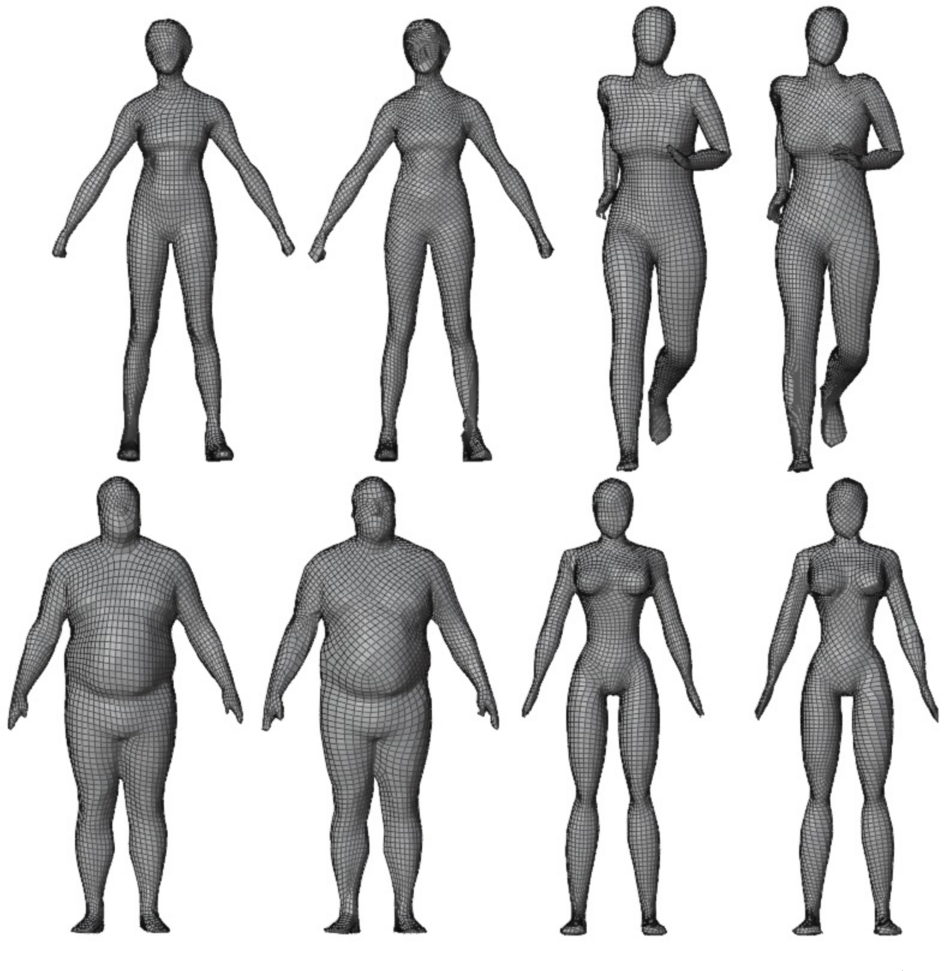


Figure 4.1: Visual comparison of the proposed pipeline's outputs (left) and Quadwild (right)

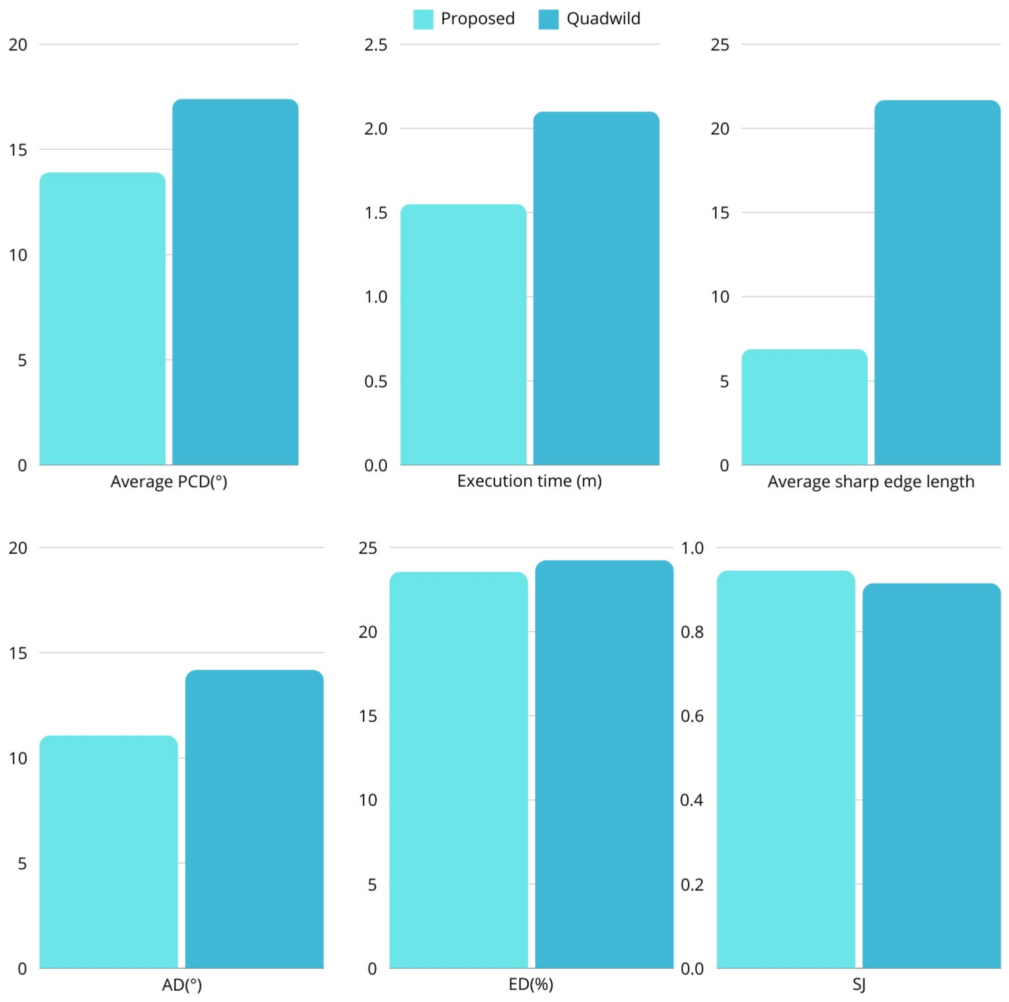


Figure 4.2: Quality and performance comparisons between the proposed pipeline and Quadwild with dihedral angle

4.2.2 Reducing Input Quality

MeshCNN outputs for each mesh four versions of the model with reduced face count. The first mesh with subfix _0 has 1500 trianguar faces, the mesh with subfix _1 has 900 triangular faces, subfix _2 has 600 faces and lastly subfix _3 has 400 faces. The pipeline was tested for all meshes with all versions to examine how reducing the quality of the input influences the output. The same test was repeated with Quadwild for comparison.

	#V		**#I**		**#F**		**AD(°)**		**ED(%)**		**SJ**		**T (m)**	
Mesh#	SP	QW	SP	QW	SP	QW	SP	QW	SP	QW	SP	QW	SP	QW
shrec_1_0	6054	6732	98	198	6052	6730	10.283	16.535	22.28	28.16	0.95	0.9	01:51	02:04
shrec_1_1	6392	10001	147	319	6390	9997	10.867	15.653	23.73	26.37	0.94	0.9	02:01	02:49
shrec_1_2	9315	13490	212	497	9300	13478	12.556	19.593	24.29	26.49	0.94	0.87	02:55	04:24
shrec_1_3	8372	12512	116	318	8370	12489	12.077	21.455	25.17	28.77	0.94	0.86	02:34	04:06
shrec_2_0	5073	5329	62	144	5071	5327	9.818	13.122	24.03	24.3	0.96	0.93	01:36	01:55
shrec_2_1	6019	6894	120	350	6015	6885	11.822	18.04	24.31	26.62	0.94	0.88	01:55	02:18
shrec_2_2	6136	7274	187	361	6119	7245	19.05	19.073	29.33	27.29	0.85	0.87	01:56	02:16
shrec_2_3	8006	11324	205	332	7982	11305	14.908	21.633	25.19	25.53	0.92	0.86	02:30	03:26
shrec_3_0	5971	6731	132	248	5965	6729	12.161	13.69	25.76	23.84	0.94	0.92	01:52	02:15
shrec_3_1	7240	7511	169	378	7237	7500	11.628	18.907	24.52	27.71	0.93	0.87	02:13	02:21
shrec_3_2	7731	10330	221	545	7714	10292	16.811	21.446	25.17	27.02	0.89	0.84	02:34	03:10
shrec_3_3	14933	18158	349	522	14905	18071	15.855	22.803	22.82	27.85	0.91	0.85	04:38	05:41
shrec_4_0	6361	7405	124	268	6359	7403	10.913	16.119	24.44	25.54	0.95	0.9	02:02	02:28
shrec_4_1	6747	7596	172	431	6740	7571	12.362	15.62	22.21	26.99	0.94	0.89	02:15	02:24
shrec_4_2	8206	11080	244	549	8192	11030	14.16	21.198	26.18	27.62	0.91	0.85	02:34	03:34
shrec_4_3	19360	20701	478	479	19306	20651	15.434	24.449	26.11	28.26	0.91	0.84	06:05	06:16
shrec_5_0	5935	6430	137	259	5929	6430	12.599	14.361	25.06	24.51	0.94	0.91	01:53	02:08
shrec_5_1	6191	6799	149	344	6184	6799	11.798	18.091	24.75	29.1	0.93	0.88	01:59	02:09
shrec_5_2	6512	8444	195	475	6504	8429	13.452	20.626	24.55	26.15	0.93	0.86	02:06	02:39
shrec_5_3	10242	14261	246	460	10227	14180	15.228	22.278	24.01	25.08	0.91	0.85	03:19	04:45
shrec_6_0	5955	6650	127	244	5953	6648	9.756	16.295	22.63	27.18	0.95	0.9	01:53	02:14
shrec_6_1	6313	8538	201	429	6508	8536	12.453	18.384	26.36	27.6	0.93	0.88	01:58	02:42
shrec_6_2	8026	11388	233	549	8018	11381	12.807	19.759	24.2	26.07	0.93	0.87	02:31	03:48
shrec_6_3	12370	14715	306	460	12359	14673	14.221	22.969	24.98	26.82	0.92	0.85	03:02	04:33
shrec_7_0	5432	6909	114	232	5430	6907	9.695	16.146	26.48	26.33	0.95	0.89	01:38	02:03
shrec_7_1	6002	7830	130	376	6000	7823	10.988	17.972	21.53	25.69	0.95	0.88	01:51	02:22
shrec_7_2	6706	9807	188	442	6701	9795	13.224	21.473	23.52	26.1	0.92	0.85	02:07	03:05
shrec_7_3	12859	15479	273	414	12856	15452	18.413	21.343	25.39	26.21	0.89	0.86	03:59	05:00
shrec_8_0	6285	7638	112	276	6283	7636	8.584	17.388	21.78	23.87	0.96	0.88	02:03	02:31
shrec_8_1	7056	7725	153	364	7054	7718	11.402	17.21	23.66	27.83	0.94	0.89	02:11	02:21
shrec_8_2	7748	10574	212	460	7742	10563	14.38	20.047	25.22	26.89	0.91	0.86	02:27	03:21
shrec_8_3	12346	17302	246	435	12337	17258	17.423	22.913	25.38	27.3	0.94	0.85	04:02	05:33
shrec_9_0	6033	6053	70	151	6031	6051	10.236	12.715	20.78	22.04	0.96	0.93	02:04	02:21
shrec_9_1	6429	8380	141	377	6419	8366	13.108	15.694	23.87	24.5	0.93	0.9	02:02	02:36
shrec_9_2	7781	10238	183	441	7779	10228	15.681	22.593	27.43	25.91	0.9	0.84	02:26	03:09
shrec_9_3	15119	18926	248	416	15113	18865	15.24	23.476	25.38	24.73	0.91	0.85	04:51	06:11
shrec_10_0	5725	5974	72	164	5723	5972	12.757	12.955	22.06	23.18	0.94	0.93	01:43	02:02
shrec_10_1	6048	7947	101	317	6046	7945	11.657	19.032	23.27	24.77	0.94	0.87	01:52	02:26

	#V		**#I**		**#F**		**AD(°)**		**ED(%)**		**SJ**		**T (m)**	
Mesh#	SP	QW	SP	QW	SP	QW	SP	QW	SP	QW	SP	QW	SP	QW
shrec_10_2	8435	9955	215	481	8412	9905	15.033	19.613	24.35	25.77	0.92	0.87	02:39	03:10
shrec_10_3	13715	17653	300	483	13705	17560	13.546	20.445	23.31	25.31	0.92	0.88	04:21	05:42
shrec_11_0	6213	7176	117	260	6211	7174	8.168	12.881	20.94	23.8	0.96	0.92	01:58	02:08
shrec_11_1	6582	8302	181	473	6579	8274	10.651	17.621	21.85	26.22	0.94	0.89	02:04	02:41
shrec_11_2	6424	8602	206	418	6417	8588	12.553	20.513	23.96	27.91	0.93	0.86	02:02	02:47
shrec_11_3	12260	15374	221	329	12256	15371	14.014	22.155	24.13	26.86	0.92	0.86	02:52	05:06
shrec_12_0	4758	5568	74	105	4756	5566	11.753	9.864	21.73	20.76	0.95	0.95	01:35	01:35
shrec_12_1	5694	7086	130	296	5689	7073	12.393	17.272	20.74	24.49	0.94	0.89	01:55	02:21
shrec_12_2	6212	8077	125	352	6210	8075	10.989	19.68	21.45	25.67	0.94	0.87	02:01	02:28
shrec_12_3	9729	12590	214	323	9696	12563	12.657	27.42	22.97	28.66	0.93	0.8	02:59	03:53
shrec_13_0	5669	5764	88	195	5667	5761	11.825	13.774	22.68	25.53	0.95	0.92	01:44	01:43
shrec_13_1	5596	7477	99	340	5594	7469	12.852	17.228	24.14	26.58	0.93	0.89	01:54	02:16
shrec_13_2	6002	8291	159	452	5998	8273	12.967	21.858	22.45	28.45	0.94	0.85	01:54	02:39
shrec_13_3	7777	10262	166	394	7772	10230	19.345	23.94	26.35	27.47	0.87	0.84	02:32	03:21
shrec_14_0	5572	6429	136	232	5570	6427	10.908	13.701	25.25	22.82	0.94	0.92	02:08	02:17
shrec_14_1	7560	8973	219	449	7556	8960	13.114	18.137	25.49	25.11	0.92	0.88	02:22	03:06
shrec_14_2	8468	11844	281	533	8463	11818	13.62	20.666	24.15	27.54	0.92	0.86	02:37	03:55
shrec_14_3	13082	16370	260	332	13076	16355	13.243	21.179	24.79	26.07	0.92	0.87	04:11	05:16
shrec_15_0	6402	7040	119	254	6400	7038	11.072	11.065	24.08	22.2	0.93	0.94	02:09	02:28
shrec_15_1	6937	8566	155	411	6935	8562	14.345	17.855	24.66	26.15	0.92	0.88	02:16	02:53
shrec_15_2	8982	11680	212	491	8974	11665	12.518	20.075	23.44	25.13	0.93	0.87	02:53	03:52
shrec_15_3	18993	22963	458	772	18961	22773	14.166	25.146	24.09	30.35	0.92	0.83	06:09	07:03
shrec_17_0	5354	6422	104	232	5352	6420	12.261	14.815	22.36	24.07	0.94	0.91	02:09	02:14
shrec_17_1	6512	8104	172	463	6506	8096	11.109	18.216	22.65	27.46	0.94	0.87	02:07	02:48
shrec_17_2	7619	10697	214	519	7601	10663	11.862	20.217	23.1	26.23	0.93	0.86	02:27	03:28
shrec_17_3	14357	17708	309	397	14353	17684	12.703	22.372	24.45	26.04	0.93	0.86	04:47	05:38
shrec_19_0	5758	6741	149	275	5751	6739	13.889	17.304	27.89	26.07	0.91	0.89	02:09	02:16
shrec_19_1	6451	8367	142	408	6449	8348	14.259	18.314	25.11	26.82	0.92	0.88	02:12	02:44
shrec_19_2	8220	10079	227	457	8213	10071	14.25	20.016	26.82	26.79	0.91	0.86	02:54	03:20
shrec_19_3	14625	16593	276	454	14603	16539	18.538	21.595	25.73	27.2	0.89	0.86	04:45	05:38
shrec_20_0	5554	6586	94	237	5552	6584	12.679	12.569	23.88	22.22	0.94	0.93	02:08	02:27
shrec_20_1	6869	8369	153	407	6867	8367	11.824	19.76	22.51	27.33	0.94	0.87	02:22	02:51
shrec_20_2	7354	9276	164	468	7351	9269	13.367	18.93	24.05	27.65	0.93	0.87	02:34	03:21
shrec_20_3	10913	12475	202	348	10901	12456	13.355	21.013	24.36	25.89	0.93	0.86	03:25	04:12
average_0	**5784**	6532	**107**	221	**5781**	6530	**11.08**	14.18	**23.56**	24.25	**0.95**	0.92	**01:55**	02:10
average_1	**6480**	8026	**152**	385	**6487**	8016	**12.15**	17.72	**23.63**	26.52	**0.93**	0.88	**02:06**	02:33
average_2	**7549**	10063	**204**	472	**7539**	10043	**13.85**	20.41	**24.65**	26.7	**0.92**	0.86	**02:25**	03:15
average_3	**12725**	15854	**271**	426	**12710**	15804	**15.02**	22.7	**24.7**	26.91	**0.92**	0.85	**03:50**	05:04

Table 4.2: Quality falloff comparison between the proposed pipeline (SP) and Quadwild (QW) - the subfix _0 indicates an input mesh with 1500 faces, _1 indicates 1200 faces, _2 900 faces and _3 400 faces

From the previous table, it is possible to observe how the proposed pipeline obtains higher quality in terms of the average Scalar Jacobian, the average angle deviation from 90° and the average edge-length deviation from the average edge-size for all versions of each mesh.
In addition, by plotting the quotient of the average of the statistics obtained from the proposed pipeline over the average of the Quadwild statistics, it is possible to appreciate how the proposed pipeline is less affected by a reduction in the input geometry

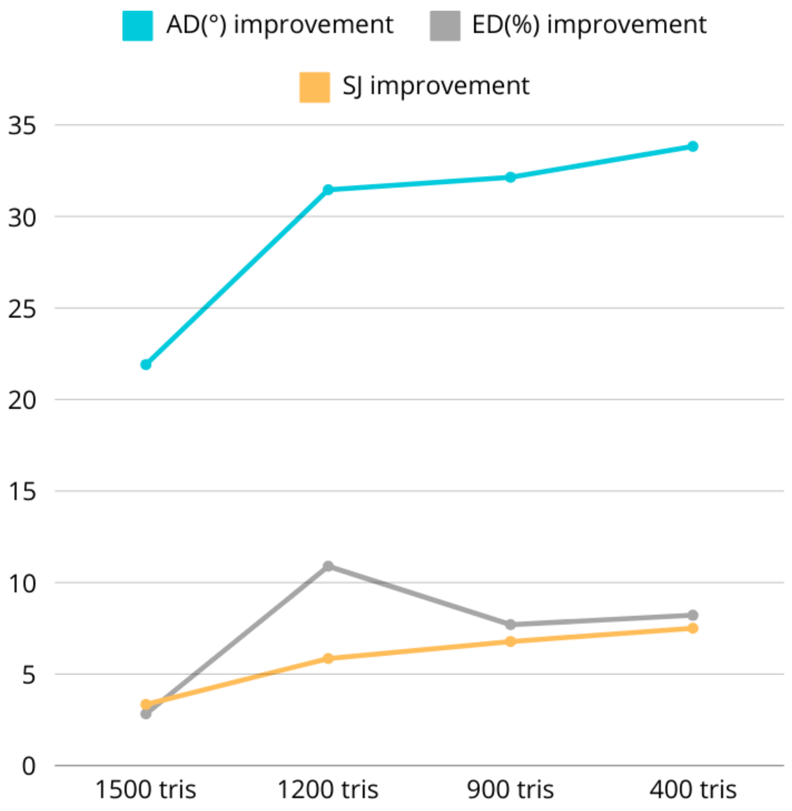


Figure 4.3: Percentage improvements in the proposed pipeline's statistics with respect to Quadwild while reducing input quality

The AD(°) metric shows the most significant difference in handling between the proposed pipeline and the Quadwild. As the input mesh resolution decreases, the angle deviation generally worsens for both methods, as visible in table 4.2. However, the pipeline shows a much slower rate of degradation, resulting in a substantial improvement of 33.83% over the baseline program when the mesh has only 400 triangles.

ED(%) handling shows an improvement that peaks around 1200 triangles at 10.89%, before slightly declining with further simplification. At 400 triangles, the ED improvement stabilizes at 8.21%.
SJ handling shows more modest improvements, with a gradual increase in the improvement rate as the resolution decreases. With the lowest quality input mesh, the improvement in SJ handling is 7.5%.

With lower lower input resolutions, the input meshes appear less organic, as smaller details are lost and different semantic areas are fused together. This is reflected by an improvement in the average percentage of singularities with the lowest resolution inputs, visible below.

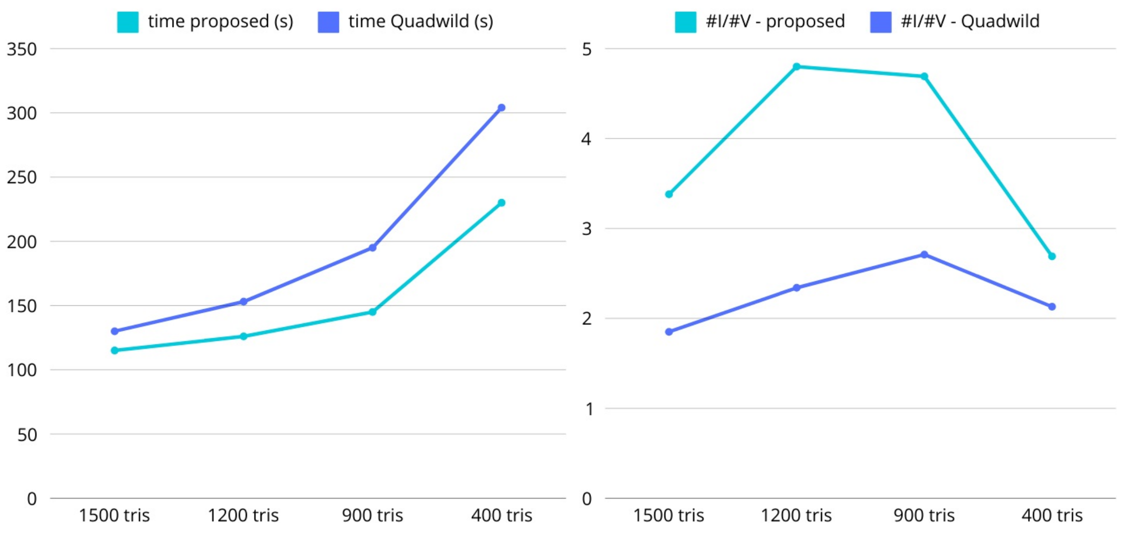


Figure 4.4: Left: average execution time - right: average percentage of irregular vertices in the meshes for the proposed pipeline and Quadwild

As it can be seen, while the proposed method performs best with the highest resolution, Quadwild improves the percentage of irregular vertices with the lowest quality input. This is possibly due to the sharper dihedral angles caused by the lower resolution. In both methods an improvement can be observed between the second lowest and the lowest resolution.
Execution time increases exponentially when lowering the quality of the input. This is possibly due to the larger surface area of the degraded meshes, where smaller areas such as arms and legs are joined together or with the torso region. Execution time is directly related to the number of vertices in the output, as the proportion between #V and time (s) is around 50 for all cases, both for the pipeline and for Quadwild.
Most meshes with the highest quality have Euler characteristic χ=2 in both methods,

which indicates the absence of errors in the final quadrangulation.
Three exceptions are present both for the proposed pipeline's results, and Quadwild's output. in the case of the proposed pipeline, meshes shrec_3 and shrec_19 have a vertex with valence 2 on the hand region, while mesh shrec_5 presents a hole in the right side of the head. For Quadwild the same problems occur for shrec_5 and shrec_19, while shrec_13 has a vertex of valence 2 on the hand region.

4.2.3 Malformed and Non-Human Meshes

A test was conducted on the pipeline to evaluate its performance with meshes from the Human Segmentation Dataset that were manually deformed in Blender and with non-human meshes from the tele-aliens set from the COSEG Shape Dataset [29].
The aim of this test is to evaluate how the chosen Convolutional Neural Network performs with inputs it was not trained on and how the metrics of the final quadrangulation are affected.

Deformed Human Meshes

The meshes used for this test are obtained by manually editing the models from the Human Segmentation Dataset using Blender. For each mesh, I applied translation, rotation and scaling operations to different semantic regions with the proportional editing tool, to achieve smoother deformations. This approach maintains the original topology of the meshes, as it only alters the position of existing vertices without adding or removing any element.

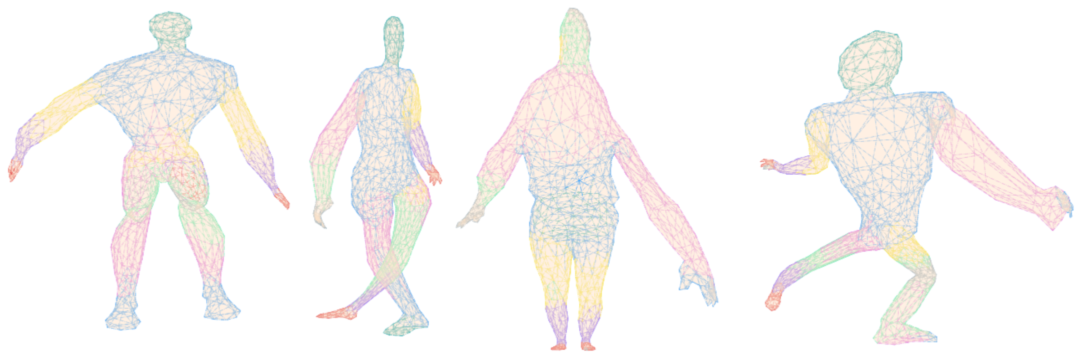


Figure 4.5: Visualization of the output of MeshCNN when segmenting the misshapen models

MeshCNN was applied to these deformed meshes to assess its ability to segment

them into coherent semantic regions. The results demonstrate that, despite the deformations, MeshCNN could largely partition the meshes into significant semantic areas. However, minor misclassifications are present, resulting in small patches of edges being assigned to incorrect categories.
Both the .sharp files from the original, unaltered meshes and those from the deformed meshes were utilized as guides in this retopology process. This is made possible by the fact that the .sharp file produced by the neural network encodes the topological structure of the vertices rather than their spatial positions.
The results presented in Table 4.6 indicate that using the .sharp file from the original mesh yields quality metrics comparable to those of the original models, underscoring the effectiveness of this approach in preserving the intended semantic structure despite vertex displacement.
The results obtained with the .sharp file from the deformed mesh however show a significant drop in quality. This demonstrates how the quality of meshCNN's segmentation is reduced when presented with an input not resembling the models in the train set.

mesh	**.sharp file**	**#V**	**#I**	**#F**	**AD(°)**	**ED(%)**	**SJ**	**PCD(°)**	**sharp edge length**
shrec__1_0	original	6419	109	6417	10.283	22.57	0.95	14.93	7.26
	deformed mesh	7259	280	7257	16.34	25.78	0.89	19.05	30.92
shrec__3_0	original	7205	142	7203	12.161	22.36	0.94	13.45	6.68
	deformed mesh	8337	258	8335	16.37	26.93	0.9	17.54	21.49
shrec__5_0	original	6360	153	6356	12.599	27.61	0.91	15.37	8.43
	deformed mesh	6891	287	6886	14.75	26.51	0.9	19.93	30.36
shrec__14_0	original	6568	121	6566	10.908	24.76	0.93	15.16	4.91
	deformed mesh	7538	329	7536	17.68	27.79	0.88	19.34	35.1
average	original	6638	131	6636	11.488	24.33	0.93	14.73	6.82
	deformed mesh	7506	288	7503	16.28	26.75	0.89	18.96	29.47

Table 4.3: Comparison of remeshing statistics for the deformed meshes using the .sharp file obtained from the original and the altered models

Non-Human Meshes

An additional test was conducted using the version of MeshCNN trained on human figures to segment the Tele-Aliens dataset, a collection of abstract models exhibiting a mix of organic and hard-surface topological features. In this test, the network successfully segmented only 11 out of the 28 meshes, corresponding to 40% of the total dataset.
Among the successfully segmented meshes, the resulting partitions generally did not correspond to visually meaningful semantic regions. Exceptions were observed

in specific tendril-like structures, which were occasionally segmented in a manner consistent with features such as arms or legs.

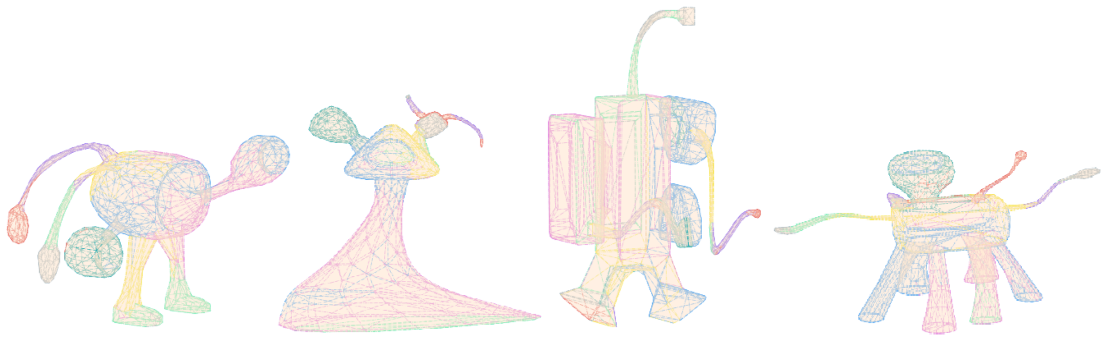


Figure 4.6: visualization of the output of MeshCNN when segmenting the meshes from the tele-aliens dataset

A notable advantage of this segmentation approach, when compared to methods based solely on dihedral angles, is the production of closed-loop sharp edges without T-junctions. As shown in Table 4.7, this property contributes to improvements in certain metrics for the final quadrangulation, particularly the Scalar Jacobian factor adn the percentage of singularities, two characteristics that are essential for generating topologically consistent meshes.
Despite that, the results for the principal curvature deviation metric (PCD°), which is the most critical to evaluate the deformability of the mesh in animation, were equal to or even worse than those achieved with Quadwild for most meshes, with an average improvement of only 2% that can be considered negligible. This highlights how an inaccurate segmentation by the neural network may negatively impact the suitability of the resulting meshes for animation.

	#V		**#I**		**#F**		**AD(°)**		**ED(%)**		**SJ**		**PCD(°)**		**T (m)**	
Mesh#	SP	QW	SP	QW	SP	QW	SP	QW	SP	QW	SP	QW	SP	QW	SP	QW
10_0	8357	10511	172	411	8350	10474	12.35	18.67	21.11	24.41	0.93	0.88	15.39	15.33	02:57	03:31
23_0	8278	11712	217	587	8268	11663	16.2	21.53	26.2	27.37	0.88	0.84	16.7	16.14	02:39	03:44
53_0	6298	6149	98	202	6290	6145	13.98	15.63	28.74	25.08	0.92	0.89	16.25	17.23	02:13	02:11
60_0	4754	7143	95	138	4759	7141	10.64	13.31	25.9	25.06	0.94	0.93	15.5	14.35	01:35	02:16
89_0	6986	9488	147	454	6984	9486	12.76	18.77	22.69	25.93	0.92	0.86	17.25	17.21	02:22	03:08
118_0	6358	7158	115	253	6352	7152	12.43	15.93	26.64	25.54	0.93	0.89	16.7	18.17	02:06	02:20
132_0	5986	7680	96	362	5984	7646	10.7	19.44	21.88	27.2	0.94	0.86	15.75	17.4	02:05	02:29
163_0	5556	6773	103	258	5554	6771	13.07	14.69	23.17	22.27	0.92	0.91	16.87	18.04	01:43	02:16
179_0	5559	6845	84	133	5557	6836	12.21	16.05	24.98	23.67	0.93	0.9	16.94	16.83	01:51	02:13
182_0	7728	9412	132	218	7723	9400	9.09	15.93	21.28	24.4	0.95	0.9	13.95	15.22	02:31	02:55
189_0	6488	10448	137	465	6486	10443	12.69	18.87	23.64	26.48	0.92	0.87	17.02	16.2	02:14	03:24
average	**6577**	8484	**127**	316	**6573**	8469	**12.37**	17.17	**24.2**	25.22	**0.93**	0.88	**16.21**	16.56	**02:09**	02:52

Table 4.4: Comparison of remeshing statistics of the tele-alien meshes, with the proposed pipeline (SP) and with Quadwild (QW)

While certain metrics show apparent improvement with the proposed pipeline, it is important to consider the hybrid nature of the Tele-Aliens dataset. These meshes are not purely organic and include sharp, hard-surface features. A visual comparison between the proposed pipeline and Quadwild reveals that the latter more effectively preserves these sharp features, a factor not captured in the evaluated metrics.

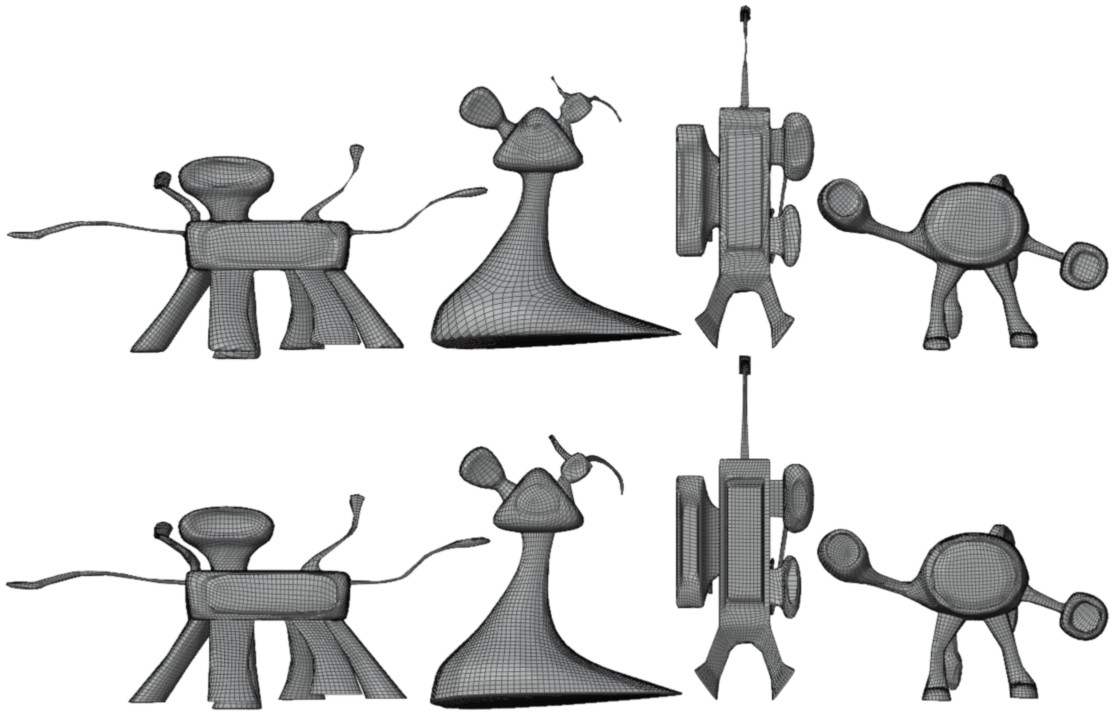


Figure 4.7: visual comparison of the remesh of tele-aliens model. Top: proposed method. Bottom: Quadwild

4.2.4 Changing Setup Values

The pipeline was tested with five different setup files. The setup file contains parameters that enable or disable the initial remesh, specify the dihedral angle for sharp features, the α value that balances regularity or isometry in the final tessellation (the lower the value, the higher the regularity and vice versa) and a scale factor for the final quadrangulation. In this experiment the quality of the output was measured with default settings, α=0.00 with and without initial remeshing, and α=0.04 with and without initial remeshing.

alpha	0.01	0.00	0.04	0.00	0.04
initial remesh	yes	yes	yes	no	no
#V	5572	1789	5538	1759	1760
#I	136	126	132	155	158
#F	5570	1787	5536	1757	1758
AD(°)	10.91	31.78	10.85	28.77	18.73
ED(%)	25.25	81.45	23.75	68.42	31.13
SJ	0.94	0.64	0.95	0.70	0.85
PCD(°)	13.43	24.22	13.24	23.93	21.46
time(m)	02:08	02:33	03:52	00:36	01:09

Table 4.5: Comparison of remeshing statistics of shrec_14_0.obj with different values of alpha in the setup.txt file, with and without the initial remesh.

In Table 4.8, we observe the influence of the α parameter and initial remeshing on the output quality and performance of the pipeline.
Regarding quality, increasing the α value, thereby enforcing greater isometry, enhances all quality metrics, with the most significant improvement being a 5.94% reduction in edge length deviation from the average. However, this improvement in quality comes at the cost of execution speed, with processing time nearly doubling compared to the default settings.
the highest contributing factor on execution speed is the initial remeshing, as tests conducted without it were substantially faster. This is likely due to the vertex count of the mesh not being increased when moving to the cross field computation and patching steps, which translates to less per-vertex operations.
Another notable finding is that the test with α=0.00 and initial remeshing produced an output with a vertex count comparable to tests without initial remeshing but resulted in even lower quality.
All tests produced final meshes with an Euler characteristic χ=2, indicating the absence of holes or boundary vertices.
From these results, we can infer that organic models particularly benefit from higher isometry settings.

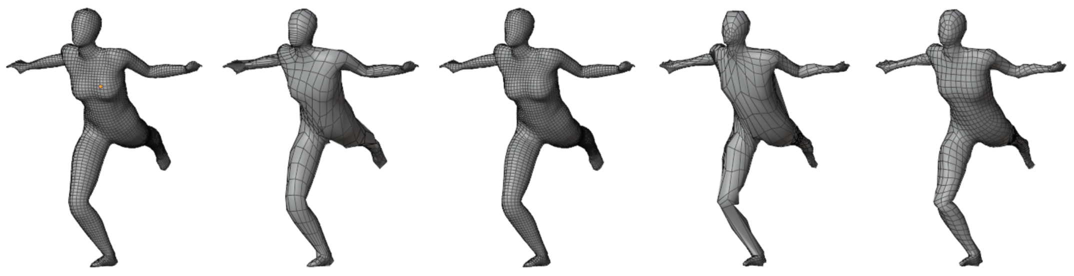


Figure 4.8: shrec_14 remeshed with the different setups. From the left: default, α=0.00 with remesh, α=0.04 with remesh, α=0.00 without remesh, α=0.04 without remesh

4.2.5 Quadriflow Comparison

The proposed pipeline is compared with the Quadriflow remeshing algorithm, with the highest quality meshes from the Dataset as input. Quadriflow is an evolution of instant meshes that aims to produce meshes with a low number of singularities in a short time, to be used interactively in modelling softwares such as Blender. For this comparison, Quadriflow was tested with the input mesh as-is and with an iteration of Catmull-Clark subdivision, to account for the preliminary remeshing executed by Quadwild. For this comparison, the average deviation from right angle AD(°) and average edge length deviation ED(%) are omitted, as the average scalar jacobian (SJ) alone was deemed sufficient to compare the quality of the quadrangular elements of the mesh between the two methods.

	#V		**#I**		**#F**		**AD(°)**		**ED(%)**		**SJ**		**PCD(°)**		**T(m)**	
Mesh#	SP	QF	SP	QF	SP	QF	SP	QF	SP	QF	SP	QF	SP	QF	SP	QF
shrec_1_0	6054	4755	98	76	6052	4753	10.28	12.36	22.28	25.17	0.95	0.94	13.28	15.59	111.38	5.8
shrec_2_0	5073	3745	62	56	5071	3743	9.82	9.1	24.03	23.37	0.96	0.96	14	14.71	96.01	4.73
shrec_3_0	5971	4824	132	82	5965	4822	12.16	14.57	25.76	24.39	0.94	0.93	13.81	15.44	112.68	6.04
shrec_4_0	6361	6028	124	114	6359	6026	10.91	7.11	24.44	19.22	0.95	0.97	13.76	15.04	122.13	6.04
shrec_5_0	5935	4825	137	74	5929	4823	12.6	8.26	25.06	23.18	0.94	0.96	14.86	14.39	113.67	5.27
shrec_6_0	5955	4885	127	88	5953	4883	9.76	8.65	22.63	22.48	0.95	0.95	13.79	14.56	113.25	5.06
shrec_7_0	5432	4444	114	80	5430	4442	9.7	9.13	26.48	20.48	0.95	0.96	13.64	14.02	98.13	4.8
shrec_8_0	6285	5768	112	96	6283	5766	8.58	6.24	21.78	19.7	0.96	0.97	13.23	12.74	123.69	5.51
shrec_9_0	6033	5808	70	84	6031	5806	10.24	7.45	20.78	19.59	0.96	0.97	12.91	13.23	124.42	5.47
shrec_10_0	5725	4961	72	70	5723	4959	12.76	7.11	22.06	20.54	0.94	0.97	14.12	13.61	103.1	5.26
shrec_11_0	6213	5957	117	106	6211	5955	8.17	9.98	20.94	19.42	0.96	0.96	12.4	12.82	95.35	5.78
shrec_12_0	4758	4303	74	64	4756	4301	11.75	12.59	21.73	18.78	0.95	0.96	8.97	12.57	95.62	4.01
shrec_13_0	5669	4761	88	82	5667	4759	11.83	12.59	23.88	22.72	0.95	0.94	09:21	12:57	103.59	5.2
shrec_14_0	5572	4151	136	104	5570	4149	10.91	11.58	25.25	24.05	0.94	0.93	10:19	14:24	128.97	5.18
shrec_15_0	6402	5703	119	120	6400	5701	11.07	7.48	24.08	19.73	0.93	0.96	19:55	19:12	129.32	5.72
shrec_17_0	5354	4815	104	100	5352	4813	12.26	7.96	22.36	18.75	0.94	0.96	16:19	08:52	129.19	4.74
shrec_19_0	5758	5230	149	92	5751	5228	13.89	9.94	27.89	22.62	0.91	0.94	01:55	02:38	129.64	5.28
shrec_20_0	5554	4389	94	82	5552	4387	12.68	11.66	23.88	23.31	0.94	0.93	01:40	05:45	128.02	4.94
average	5784	**4964**	**107**	87	5781	**4962**	11.08	**9.65**	23.63	**21.53**	**0.95**	**0.95**	**13.68**	14.58	114.34	**5.27**

Table 4.6: Comparison between the proposed method (indicated as SP) and the blender 4.1 implementation of Quadriflow[14] without Catmull-Clark subdivision (indicated as QF)

	#V		**#I**		**#F**		**AD(°)**		**ED(%)**		**SJ**		**PCD(°)**		**T(m)**	
Mesh#	SP	CC+QF	SP	CC+QF	SP	CC+QF	SP	CC+QF	SP	CC+QF	SP	CC+QF	SP	CC+QF	SP	CC+QF
shrec_1_0	6054	4413	98	76	6052	4411	10.28	11.37	22.28	27.38	0.95	0.95	13.28	15.42	111.38	6.52
shrec_2_0	5073	3711	62	80	5071	3709	9.82	11.35	24.03	26.37	0.96	0.94	14	17.68	96.01	5.38
shrec_3_0	5971	5779	132	84	5965	5777	12.16	12.53	25.76	20.3	0.94	0.94	13.81	14.46	112.68	7.1
shrec_4_0	6361	5210	124	88	6359	5208	10.91	7.53	24.44	21.25	0.95	0.97	13.76	13.01	122.13	6.95
shrec_5_0	5935	5058	137	84	5929	5056	12.6	11.18	25.06	23.47	0.94	0.95	14.86	14.92	113.67	6.26
shrec_6_0	5955	4832	127	70	5953	4830	9.76	7.14	22.63	21.33	0.95	0.97	13.79	13.62	113.25	6.17
shrec_7_0	5432	4981	114	76	5430	4979	9.7	10.04	26.48	21.32	0.95	0.95	13.64	15.04	98.13	5.66
shrec_8_0	6285	5868	112	80	6283	5866	8.58	11.42	21.78	20.08	0.96	0.95	13.23	14.59	123.69	6.51
shrec_9_0	6033	5220	70	66	6031	5218	10.24	8.05	20.78	20.75	0.96	0.97	12.91	13.33	124.42	5.93
shrec_10_0	5725	5870	72	60	5723	5868	12.76	6.53	22.06	19.14	0.94	0.98	14.12	12.58	103.1	5.92
shrec_11_0	6213	5560	117	88	6211	5558	8.17	9.77	20.94	22.34	0.96	0.96	12.4	12.66	95.35	6.59
shrec_12_0	4758	4216	74	68	4756	4214	11.75	7.13	21.73	20.42	0.95	0.96	8.97	11.64	95.62	4.85
shrec_13_0	5669	5005	88	80	5667	5003	11.83	10.11	23.88	19.47	0.95	0.95	09:21	15:50	103.59	5.58
shrec_14_0	5572	4822	136	80	5570	4820	10.91	12.01	25.25	23.79	0.94	0.94	10:19	21:36	128.97	6.25
shrec_15_0	6402	5823	119	88	6400	5821	11.07	9.17	24.08	20.56	0.93	0.96	19:55	09:21	129.32	7.03
shrec_17_0	5354	5158	104	100	5352	5156	12.26	6.55	22.36	17.95	0.94	0.97	16:19	02:38	129.19	5.34
shrec_19_0	5758	5253	149	86	5751	5251	13.89	7.15	27.89	21.82	0.91	0.97	01:55	14:52	129.64	5.91
shrec_20_0	5554	4555	94	72	5552	4553	12.68	7.92	23.88	22.39	0.94	0.97	01:40	09:36	128.02	5.53
average	5784	**5074**	107	**79**	5781	**5072**	11.08	**9.28**	23.63	**21.67**	0.95	**0.96**	**13.68**	14.34	114.34	**6.08**

Table 4.7: Comparison between the proposed method (indicated as SP) and the blender 4.1 implementation of Quadriflow[14] with Catmull-Clark subdivision (indicated as CC+QF)

In terms of performance, Quadriflow is approximately an order of magnitude faster than the semantic pipeline. This difference is expected, as Quadriflow generates its orientation field without relying on a global parameterization, solving a global minimum-cost network flow problem only for the position field, which can be computed in polynomial time. The Quadriflow paper also reports that the algorithm can remesh a one-million-triangle mesh in roughly 5 seconds, consistent with the results in Table 4.10.
For quality metrics, the proposed pipeline shows an advantage in the average scaled Jacobian (SJ) for 8 out of the 18 tested meshes, with 6 of these cases exceeding the SJ of Quadriflow when preliminary subdivision is applied. The proposed pipeline also demonstrates a lower average principal curvature deviation, which is a primary focus of the pipeline, in 12 out of the 18 meshes, outperforming Quadriflow with subdivision in 9 cases.
While the semantic pipeline generally yields a higher number of singularities than Quadriflow, it's essential to consider the total number of vertices as well. When examining the #I/#V ratio, the proposed pipeline achieves a lower singularity ratio in 10 meshes, with 4 instances where this ratio is also lower than that of Quadriflow with subdivision.
Both the proposed pipeline and Quadriflow consistently produce final outputs with an Euler characteristic of $\chi = 2$, except for the cases of shrec_5 and shrec_19 where the proposed pipeline includes boundary vertices, as discussed previously.

While Quadriflow is faster, it doesn't always guarantee a correct final quadrangulation. In 4 distinct meshes the algorithm failed to produce a valid quadrangulation. This problem is non-existent in our semantic pipeline, as the global optimization steps guarantee that a valid output is always reached, at the cost of a higher execution time.
In some other instances, Quadriflow presents better metrics than the semantic pipeline, but the output contains highly localized areas of irregularities. This is likely due to Quadriflow's position field not being aligned with the shape's features, therefore losing details in smaller areas.

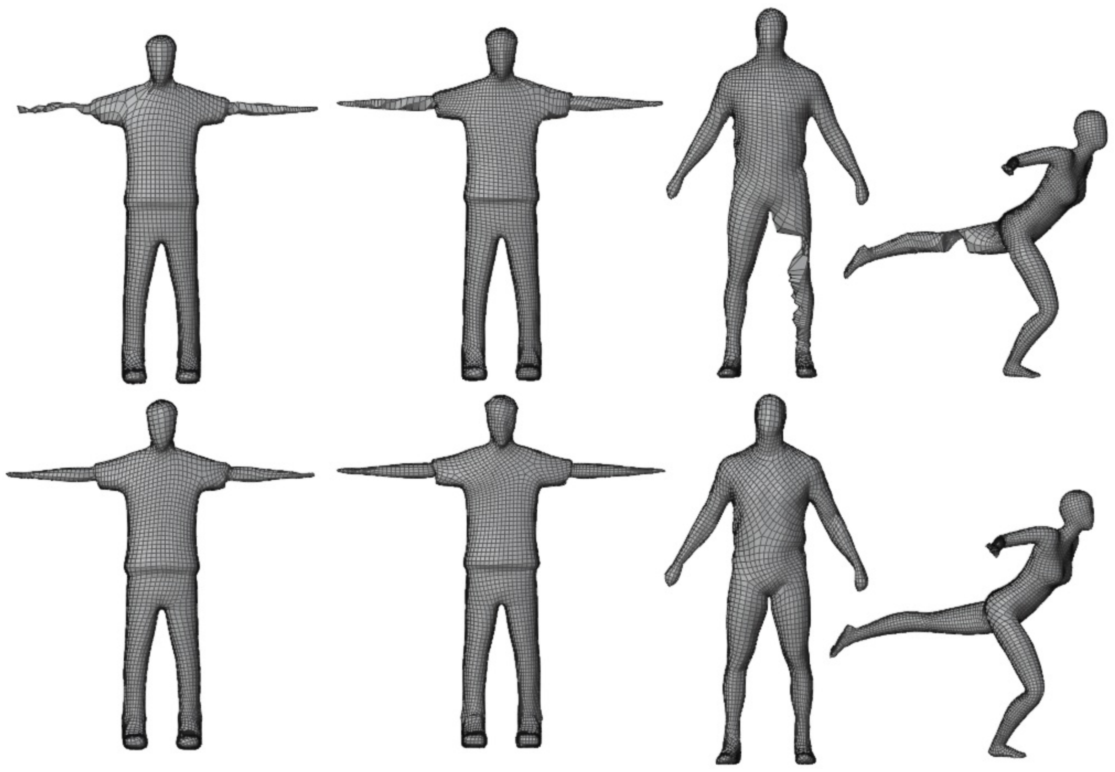


Figure 4.9: Top row: Errors in Quadriflow's quadrangulation. Bottom row: proposed pipeline

Chapter 5

Conclusion

This thesis presented a novel remeshing pipeline designed to enhance the topology of 3D human models for animation purposes, addressing limitations in existing automatic remeshing methods. Specifically, it focused on integrating semantic segmentation with Reliable Feature-Line Driven Quad-Remeshing to achieve results that match the semantic understanding of 3D artists, particularly for human models.

The proposed pipeline utilizes MeshCNN, a convolutional neural network trained on human models from the Human Body Segmentation Dataset, to identify and segment different semantic regions of a mesh. These boundaries of the semantic areas are then extracted with a python script and used to guide the remeshing process, which gives an output whose topology is optimized for animation and with higher quality. The remeshing algorithm aligns edges along the principal curvatures and critical feature lines, which ensures better deformation and fewer visual artifacts when the model is animated.

The results of the pipeline were evaluated against both the standalone implementation of the remeshing algorithm (Quadwild) and an alternative remeshing method (Quadriflow). The metrics show that the pipeline consistently produced more isotropic meshes, with improved edge alignment, fewer irregular vertices, and reduced angular deviation. The principal curvature alignment metric confirmed the pipeline's ability to create more natural and accurate edge flows around critical deformation areas, such as joints, which is essential for smooth animations.

In terms of performance, the pipeline not only achieved better quality metrics but also demonstrated enhanced efficiency, with a notable reduction in processing time and sharp feature edge length. The pipeline's robustness was further validated through tests on reduced-quality input meshes, confirming that it maintains higher topological integrity even under suboptimal input conditions. Furthermore, the pipeline was tested on malformed and non-human meshes that do not resemble elements from the training set, showcasing that eventual drops in performance related to erroneous segmentation make the output quality comparable in the worst case to that of Quadwild.

5.1 Future Work

The proposed pipeline for remeshing 3D human models shows better performance and resiliance to low quality inputs when compared to Quadwild as a standalone program when applied to human meshes. The semantic pipeline could be further enhanced in different directions, such as:

- Generalization to Different Organic Models: The pipeline could be applied to other organic models by adding to MeshCNN training datasets other models with similar features, such as mammals or different humanoid creatures.

- Optimization of Performance for Real Time Application: While the pipeline demonstrates reasonable execution times, additional optimization in computational efficiency could be achieved. The remeshing algorithm of the pipeline could be enhanced or substituted to achieve real time applicability, similarly to Quadriflow.

- Adaptive Edge Length: The current pipeline has a target edge length of $A/10^4$. While this guarantees a regular quadrangulation that achieves high quality metrics, a variable edge length could increase quadrangulation density around semantic edges, to obtain more accurate deformations when animating.

- Further unification of the pipeline: MeshCNN and Quadwild could be modified to work like an end-to-end system by directly using the edge weights derived from MeshCNN's feature confidence scores in the patch tracing step, or by introducing a feedback loop between the MeshCNN and Quadwild stages, where Quadwild provides edge quality metrics back to MeshCNN for a more refined segmentation.

- Integration with Real-world Animation Pipelines: Future works should focus on applying the insight provided by the results to explore the incorporation of the generated quad meshes into industry-standard animation workflows, including dynamic simulations or deformation tests. Measuring real-world improvements in animation quality could validate the pipeline's practical utility.

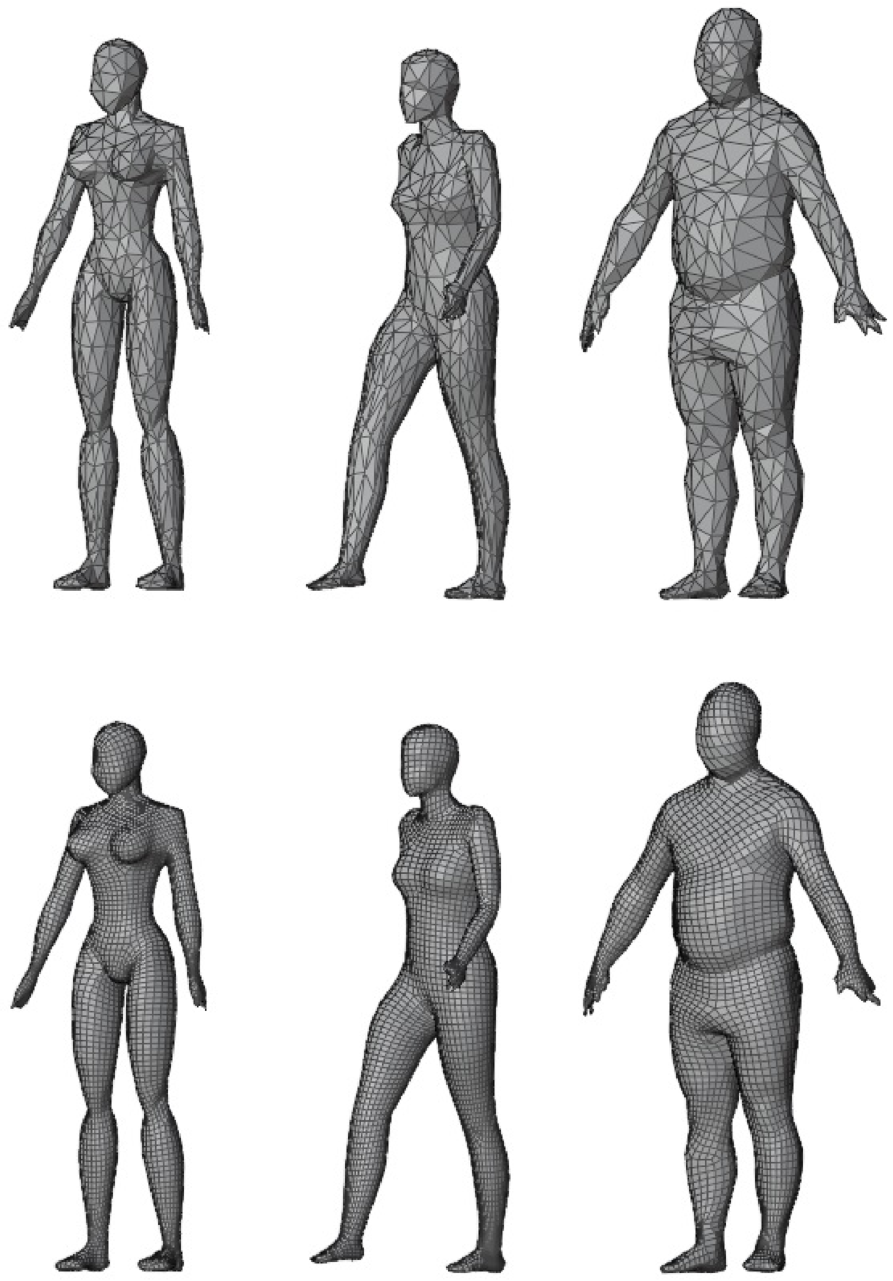


Figure 5.1: Initial meshes from the dataset and final quadrangulation using the proposed pipeline

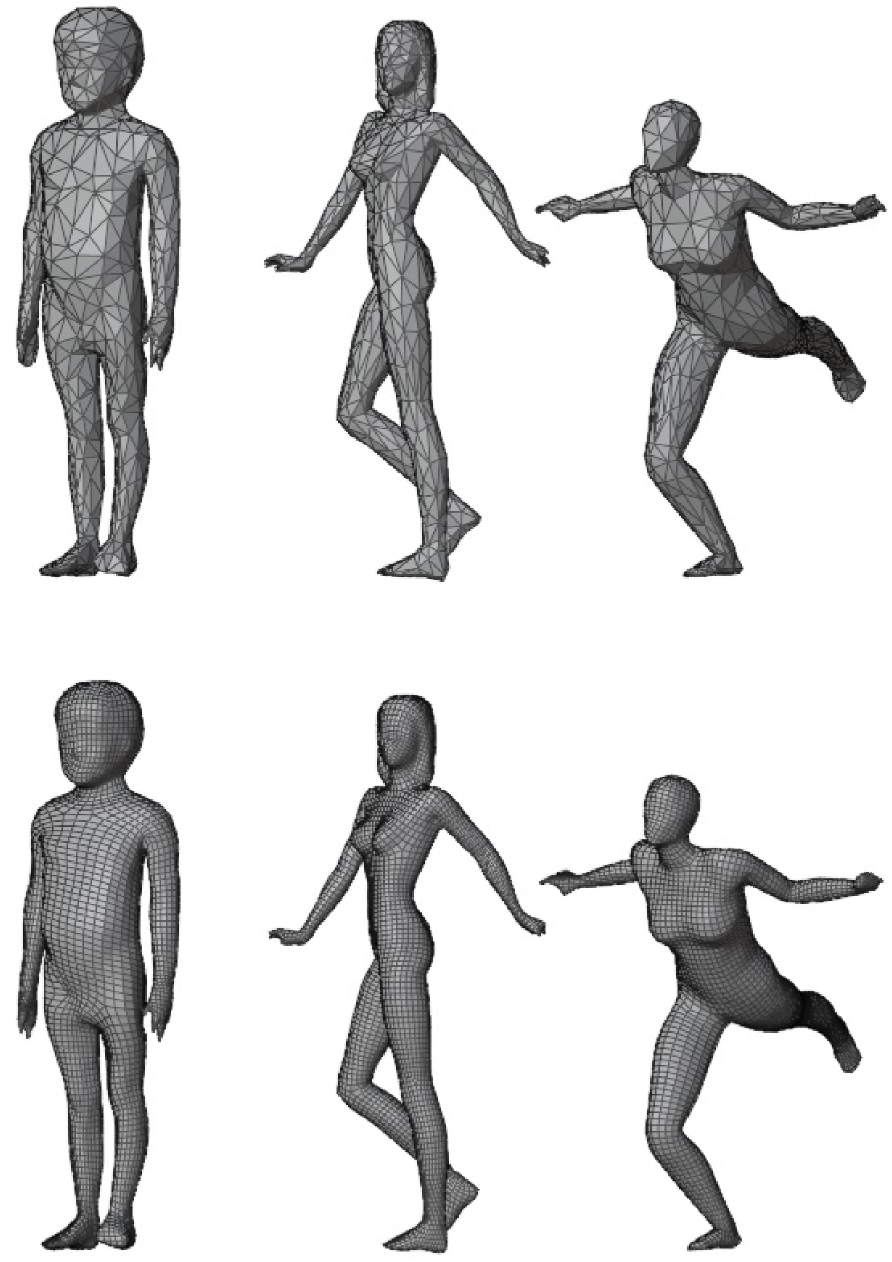


Figure 5.2: Initial meshes from the dataset and final quadrangulation using the proposed pipeline

Appendix A

Code

Algorithm 1 Extract segmentation boundaries from MeshCNN

```
class Face:
   def ___init___(self, v1, v2, v3):
      self.edges = {0: (v1, v2), 1: (v2, v3), 2: (v1, v3)}
   def substitute_edge_keys(self, edge_dict):
      for key, value in self.edges.items():
         if value in edge_dict:
            self.edges[key] = edge_dict[value]
         elif (value[1], value[0]) in edge_dict:
            self.edges[key] = edge_dict[(value[1], value[0])]
   def has_unique_group(self):
      groups = list(self.edges.values())
      return len(set(groups)) == 2
   def unique_group_index(self):
      groups = list(self.edges.values())
      for i in range(3):
         others = [groups[j] for j in range(3) if j != i]
         if groups[i] != others[0] and others[0] == others[1]:
            return i
      return None
   def ___str___(self):
      return f"Face({self.edges})"
```

```
    class Edge:
      def ___init___(self, v1, v2, edge_group):
        self.edge = {edge_group: (v1, v2)}
      def ___str___(self):
        return f"Edge({self.edge})"

    def process_obj_file(input_file):
    faces, edges, edge_dict = [], [], {}
    with open(input_file, 'r') as file:
      for line in file:
        parts = line.split()
        if not parts: continue
        if parts[0] == 'f':
          v1, v2, v3 = map(int, parts[1:4])
          face = Face(v1, v2, v3)
          faces.append(face)
        elif parts[0] == 'e':
          v1, v2, edge_group = map(int, parts[1:4])
          edge = Edge(v1, v2, edge_group)
          edges.append(edge)
          edge_dict[(v1, v2)] = edge_group
          edge_dict[(v2, v1)] = edge_group
    for face in faces:
      face.substitute_edge_keys(edge_dict)
    unique_faces = [(index, face) for index, face in enumerate(faces) if
  face.has_unique_group()]
    return unique_faces, edges

    def write_output(output_file, unique_faces):
    with open(output_file, 'w') as file:
      file.write(f"{len(unique_faces)}\n")
      for filtered_index, (original_index, face) in enumerate(unique_faces):
        unique_edge_index = face.unique_group_index()
        file.write(f"0,{original_index},{unique_edge_index}\n")

1: def main():
    parser = argparse.ArgumentParser(description="Process .obj file and
  output Face and Edge classes to a .txt file.")
    parser.add_argument('input_file', type=str, help="Path to the input
  .obj file")
    parser.add_argument('output_file', type=str, help="Path to the output
  .txt file")
    args = parser.parse_args()
    unique_faces, edges = process_obj_file(args.input_file)
    write_output(args.output_file, unique_faces)

2: if ___name___ == "___main___":
    main()
```

Algorithm 2 Compute Average Edge Length Deviation for a Mesh

```
Require: argparse, numpy, trimesh
    import argparse
    import numpy as np
    import trimesh

function COMPUTE_EDGE_LENGTH_DEVIATION(mesh_path)
        mesh = trimesh.load(mesh_path)
    if not mesh.is_volume then
            print("Warning: The mesh might not be watertight.")
    end if
        edges = mesh.edges_unique
        edge_lengths = mesh.edges_unique_length
        avg_edge_length = np.mean(edge_lengths)
        print(f"Average Edge Length: {avg_edge_length:.4f}")
        deviation = edge_lengths - avg_edge_length
        mean_abs_deviation_percentage = np.mean(np.abs(deviation)) / avg_edge_length * 100
        print(f"Mean Absolute Deviation: {mean_abs_deviation_percentage:.2f}%)
    return deviation
end function

function MAIN
        parser = argparse.ArgumentParser(description="Compute edge length deviation from average edge length of a mesh.")
        parser.add_argument("mesh_path", type=str, help="Path to the input mesh file")
        args = parser.parse_args()
        deviation = compute_edge_length_deviation(args.mesh_path)
        print("Edge length deviations computed successfully.")
end function

if __name__ == "__main__" then
        main()
end if
```

Algorithm 3 Calculate Average Scaled Jacobian

```
Require: argparse, vtk
    import argparse
    import vtk

 1: function CALCULATE_AVERAGE_SCALED_JACOBIAN(mesh_file)
 2:     reader = vtk.vtkOBJReader() if mesh_file.endswith('.obj') else vtk.vtkSTLReader()
 3:     reader.SetFileName(mesh_file)
 4:     reader.Update()

 5:     mesh = reader.GetOutput()

 6:     quality_filter = vtk.vtkMeshQuality()
 7:     quality_filter.SetInputData(mesh)
 8:     quality_filter.SetQuadQualityMeasureToScaledJacobian()
 9:     quality_filter.Update()

10:     quality_array = quality_filter.GetOutput().GetCellData().GetArray("Quality")

11:     total_scaled_jacobian = 0
12:     num_cells = quality_array.GetNumberOfTuples()
13:     for i in range(num_cells) do
14:         total_scaled_jacobian += quality_array.GetValue(i)
15:     end for
16:     average_scaled_jacobian = total_scaled_jacobian / num_cells if num_cells > 0 else None
17:     return average_scaled_jacobian
18: end function
```

```
function MAIN
    parser = argparse.ArgumentParser(description="Calculate the average scaled Jacobian of a mesh file.")
    parser.add_argument("mesh_file", type=str, help="Path to the input mesh file (.obj or .stl)")
    args = parser.parse_args()

    average_scaled_jacobian = calculate_average_scaled_jacobian(args.mesh_file)
    if average_scaled_jacobian is not None then
        print(f"Average Scaled Jacobian: {average_scaled_jacobian}")
    else
        print("No cells found in the mesh to compute the scaled Jacobian.")
    end if
end function

if ___name___ == "___main___" then
    main()
end if
```

Algorithm 4 Compute Edge Angle Deviation from Principal Curvatures

```
Require: argparse, numpy, igl, numpy.linalg.norm
    import argparse
    import numpy as np
    import igl
    from numpy.linalg import norm

function COMPUTE_PRINCIPAL_CURVATURES(obj_file)
    V, F = igl.read_triangle_mesh(obj_file)
    pd1, pd2, pv1, pv2 = igl.principal_curvature(V, F)
    return V, F, pd1, pd2
end function

function ANGLE_BETWEEN_VECTORS(v1, v2)
    v1norm = norm(v1)
    v2norm = norm(v2)
    if v1norm == 0 or v2norm == 0 then
        return 0.0
    end if
    v1_norm = v1 / norm(v1)
    v2_norm = v2 / norm(v2)
    dot_product = np.clip(np.dot(v1_norm, v2_norm), -1.0, 1.0)
    angle_rad = np.arccos(dot_product)
    return np.degrees(angle_rad)
end function

function COMPUTE_EDGE_ANGLE_DEVIATION(V, F, pd1, pd2)
    total_angle_deviation = 0.0
    total_edge_count = 0
    num_faces = len(F) // 2
    for face_idx in range(num_faces) do
        tri1 = F[2 * face_idx]
        tri2 = F[2 * face_idx + 1]
        quad_vertices = [tri1[0], tri1[1], tri1[2], tri2[2]]
        edges = [V[quad_vertices[1]] - V[quad_vertices[0]],
        V[quad_vertices[2]] - V[quad_vertices[1]],
        V[quad_vertices[3]] - V[quad_vertices[2]],
        V[quad_vertices[0]] - V[quad_vertices[3]]
        angle_deviations = []
```

```
            for i in range(4) do
                vertex_idx = quad_vertices[i]
                principal_directions = [pd1[vertex_idx], pd2[vertex_idx]]
                for edge in edges do
                    angle1 = angle_between_vectors(edge, principal_directions[0])
                    angle2 = angle_between_vectors(edge, principal_directions[1])
                    angle1 = min(angle1, 180 - angle1)
                    angle2 = min(angle2, 180 - angle2)
                    angle_deviation = min(angle1, angle2)
                    angle_deviations.append(angle_deviation)
                end for
            end for
            angle_deviations.remove(max(angle_deviations))
            total_angle_deviation += sum(angle_deviations)
            total_edge_count += len(angle_deviations)
        end for
        avg_angle_deviation = total_angle_deviation / total_edge_count
        return avg_angle_deviation
end function

function MAIN
    parser = argparse.ArgumentParser(description='Compute the average angle deviation of edges from the principal curvatures for a quadrilateral mesh.')
    parser.add_argument('input_obj', type=str, help='Path to the input .obj file')
    args = parser.parse_args()
    V, F, pd1, pd2 = compute_principal_curvatures(args.input_obj)
    avg_angle_deviation = compute_edge_angle_deviation(V, F, pd1, pd2)
    print(f"\nAverage Angle Deviation: {avg_angle_deviation:.2f} degrees")
end function

if __name__ == "__main__" then
    main()
end if
```

Bibliography

[1] Nico Pietroni, Stefano Nuvoli, Thomas Alderighi, Paolo Cignoni, and Marco Tarini. «Reliable feature-line driven quad-remeshing». In: *ACM Trans. Graph.* 40.4 (July 2021). ISSN: 0730-0301. DOI: 10.1145/3450626.3459941. URL: https://doi.org/10.1145/3450626.3459941 (cit. on pp. 1, 5, 9–11, 13).

[2] L. Krenmayr, R. von Schwerin, D. Schaudt, P. Riedel, and A. Hafner. «DilatedToothSegNet: Tooth Segmentation Network on 3D Dental Meshes Through Increasing Receptive Vision». In: *Journal of Imaging Informatics in Medicine* 37.4 (2024), pp. 1846–1862. DOI: 10.1007/s10278-024-01061-6. URL: https://doi.org/10.1007/s10278-024-01061-6 (cit. on pp. 1, 10).

[3] Achraf Ben-Hamadou et al. *Teeth3DS: a benchmark for teeth segmentation and labeling from intra-oral 3D scans.* 2022. arXiv: 2210.06094 [cs.CV]. URL: https://arxiv.org/abs/2210.06094 (cit. on p. 1).

[4] Rana Hanocka, Amir Hertz, Noa Fish, Raja Giryes, Shachar Fleishman, and Daniel Cohen-Or. «MeshCNN: a network with an edge». In: *ACM Transactions on Graphics* 38.4 (July 2019), pp. 1–12. ISSN: 1557-7368. DOI: 10.1145/3306346.3322959. URL: http://dx.doi.org/10.1145/3306346.3322959 (cit. on pp. 1, 10, 11, 13).

[5] Haggai Maron, Meirav Galun, Noam Aigerman, Miri Trope, Nadav Dym, Ersin Yumer, Vladimir G. Kim, and Yaron Lipman. «Convolutional neural networks on surfaces via seamless toric covers». In: *ACM Trans. Graph.* 36.4 (July 2017). ISSN: 0730-0301. DOI: 10.1145/3072959.3073616. URL: https://doi.org/10.1145/3072959.3073616 (cit. on pp. 1, 16).

[6] David Bommes, Bruno Levy, Nico Pietroni, Enrico Puppo, Cláudio Silva, and Denis Zorin. «Quad-Mesh Generation and Processing: A Survey». In: *Computer Graphics Forum* 32 (Sept. 2013). DOI: 10.1111/cgf.12014 (cit. on pp. 2, 6).

[7] Na Lei, Zezeng Li, Zebin Xu, Ying Li, and Xianfeng Gu. «What's the Situation With Intelligent Mesh Generation: A Survey and Perspectives». In: *IEEE Transactions on Visualization and Computer Graphics* 30.8 (2024), pp. 4997–5017. DOI: 10.1109/TVCG.2023.3281781 (cit. on p. 4).

[8] Amir Vaxman, Marcel Campen, Olga Diamanti, Daniele Panozzo, David Bommes, Klaus Hildebrandt, and Mirela Ben-Chen. «Directional Field Synthesis, Design, and Processing». In: *Computer Graphics Forum* (2016). URL: `http://graphics.tudelft.nl/Publications-new/2016/VCDPBHB16` (cit. on p. 5).

[9] Dawar Khan, Alexander Plopski, Yuichiro Fujimoto, Masayuki Kanbara, Gul Jabeen, Yongjie Jessica Zhang, Xiaopeng Zhang, and Hirokazu Kato. «Surface Remeshing: A Systematic Literature Review of Methods and Research Directions». In: *IEEE Transactions on Visualization and Computer Graphics* 28.3 (2022), pp. 1680–1713. DOI: `10.1109/TVCG.2020.3016645` (cit. on p. 6).

[10] Amir Vaxman, Marcel Campen, Olga Diamanti, David Bommes, Klaus Hildebrandt, Mirela Ben-Chen Technion, and Daniele Panozzo. «Directional field synthesis, design, and processing». In: *ACM SIGGRAPH 2017 Courses*. SIGGRAPH '17. Los Angeles, California: Association for Computing Machinery, 2017. ISBN: 9781450350143. DOI: `10.1145/3084873.3084921`. URL: `https://doi.org/10.1145/3084873.3084921` (cit. on pp. 7, 20).

[11] Qiang Du, Vance Faber, and Max Gunzburger. «Centroidal Voronoi Tessellations: Applications and Algorithms». In: *SIAM Review* 41.4 (1999), pp. 637–676. DOI: `10.1137/S0036144599352836`. eprint: `https://doi.org/10.1137/S0036144599352836`. URL: `https://doi.org/10.1137/S0036144599352836` (cit. on p. 8).

[12] J. Alan George. «Computer implementation of the finite element method». In: *Computer implementation of the finite element method*. 1971. URL: `https://api.semanticscholar.org/CorpusID:122680959` (cit. on p. 8).

[13] Wenzel Jakob, Marco Tarini, Daniele Panozzo, and Olga Sorkine-Hornung. «Instant field-aligned meshes». In: *ACM Trans. Graph.* 34.6 (Nov. 2015). ISSN: 0730-0301. DOI: `10.1145/2816795.2818078`. URL: `https://doi.org/10.1145/2816795.2818078` (cit. on pp. 8, 10).

[14] Jingwei Huang, Yichao Zhou, Matthias Niessner, Jonathan Richard Shewchuk, and Leonidas J. Guibas. «QuadriFlow: A Scalable and Robust Method for Quadrangulation». In: *Computer Graphics Forum* (2018). ISSN: 1467-8659. DOI: `10.1111/cgf.13498` (cit. on pp. 8, 10, 24, 41, 42).

[15] Ariel Shamir. «A survey on Mesh Segmentation Techniques». In: *Computer Graphics Forum* 27.6 (2008), pp. 1539–1556. DOI: `https://doi.org/10.1111/j.1467-8659.2007.01103.x`. eprint: `https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8659.2007.01103.x`. URL: `https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2007.01103.x` (cit. on pp. 9, 11).

[16] Jyh-Ming Lien, John Keyser, and Nancy M. Amato. «Simultaneous shape decomposition and skeletonization». In: *Proceedings of the 2006 ACM Symposium on Solid and Physical Modeling.* SPM '06. Cardiff, Wales, United Kingdom: Association for Computing Machinery, 2006, pp. 219–228. ISBN: 1595933581. DOI: 10.1145/1128888.1128919. URL: https://doi.org/10.1145/1128888.1128919 (cit. on p. 10).

[17] John Edwards, Wenping Wang, and Chandrajit L. Bajaj. «Surface Segmentation for Improved Remeshing». In: *International Meshing Roundtable Conference.* 2013. URL: https://api.semanticscholar.org/CorpusID:16644613 (cit. on p. 11).

[18] Dragomir Anguelov, Praveen Srinivasan, Daphne Koller, Sebastian Thrun, Jim Rodgers, and James Davis. «SCAPE: Shape Completion and Animation of People». In: *Seminal Graphics Papers: Pushing the Boundaries, Volume 2.* 1st ed. New York, NY, USA: Association for Computing Machinery, 2023. ISBN: 9798400708978. URL: https://doi.org/10.1145/3596711.3596797 (cit. on p. 16).

[19] Federica Bogo, Javier Romero, Matthew Loper, and Michael J. Black. «FAUST¦ Dataset and Evaluation for 3D Mesh Registration». In: *2014 IEEE Conference on Computer Vision and Pattern Recognition.* 2014, pp. 3794–3801. DOI: 10.1109/CVPR.2014.491 (cit. on p. 16).

[20] Daniel Vlasic, Ilya Baran, Wojciech Matusik, and Jovan Popović. «Articulated mesh animation from multi-view silhouettes». In: *ACM SIGGRAPH 2008 papers* (2008). URL: https://api.semanticscholar.org/CorpusID:230280 (cit. on p. 16).

[21] Daniela Giorgi, Silvia Biasotti, and Laura Paraboschi. «SHape REtrieval Contest 2007: Watertight Models Track». In: *SHREC competition* 8 (July 2008) (cit. on p. 16).

[22] Evangelos Kalogerakis, Aaron Hertzmann, and Karan Singh. «Learning 3D mesh segmentation and labeling». In: *ACM Trans. Graph.* 29.4 (July 2010). ISSN: 0730-0301. DOI: 10.1145/1778765.1778839. URL: https://doi.org/10.1145/1778765.1778839 (cit. on p. 16).

[23] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. «Mesh optimization». In: *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques.* SIGGRAPH '93. Anaheim, CA: Association for Computing Machinery, 1993, pp. 19–26. ISBN: 0897916018. DOI: 10.1145/166117.166119. URL: https://doi.org/10.1145/166117.166119 (cit. on p. 19).

[24] Olga Diamanti, Amir Vaxman, Daniele Panozzo, and Olga Sorkine-Hornung. «Designing N-PolyVector Fields with Complex Polynomials». In: *Comput. Graph. Forum* 33.5 (Aug. 2014), pp. 1–11. ISSN: 0167-7055 (cit. on p. 20).

[25] Daniele Panozzo, Yaron Lipman, Enrico Puppo, and Denis Zorin. «Fields on symmetric surfaces». In: *ACM Trans. Graph.* 31.4 (July 2012). ISSN: 0730-0301. DOI: `10.1145/2185520.2185607`. URL: `https://doi.org/10.1145/2185520.2185607` (cit. on p. 20).

[26] Marco Tarini. «Closed-form quadrangulation of N-sided patches». In: *Computers amp; Graphics* 107 (Oct. 2022), pp. 60–65. ISSN: 0097-8493. DOI: `10.1016/j.cag.2022.06.015`. URL: `http://dx.doi.org/10.1016/j.cag.2022.06.015` (cit. on p. 23).

[27] Kenshi Takayama, Daniele Panozzo, and Olga Sorkine-Hornung. «Pattern-Based Quadrangulation for N-Sided Patches». In: *Comput. Graph. Forum* 33.5 (Aug. 2014), pp. 177–184. ISSN: 0167-7055 (cit. on p. 23).

[28] Nico Pietroni, Davide Tonelli, Enrico Puppo, Maurizio Froli, Roberto Scopigno, and Paolo Cignoni. «Statics Aware Grid Shells». In: *Computer Graphics Forum* 34 (2015). URL: `https://api.semanticscholar.org/CorpusID:17664480` (cit. on p. 23).

[29] Yunhai Wang, Shmulik Asafi, Oliver van Kaick, Hao Zhang, Daniel Cohen-Or, and Baoquan Chen. «Active co-analysis of a set of shapes». In: 31.6 (Nov. 2012). ISSN: 0730-0301. DOI: `10.1145/2366145.2366184`. URL: `https://doi.org/10.1145/2366145.2366184` (cit. on pp. 24, 34).