Utrecht University

Master of Science

# Directional Field Design on Dynamic and Rigged Meshes

*Author:*
Constantinos Kyprianou

*Supervisor:*
Dr. Amir Vaxman

*A thesis submitted in fulfillment of the requirements for the degree of Master of Science*

*in*

Game and Media Technology

Department of Information and Computing Sciences

January 1, 2020

UTRECHT UNIVERSITY

# *Abstract*

Faculty of Science

Department of Information and Computing Sciences

Master of Science

**Directional Field Design on Dynamic and Rigged Meshes**

by Constantinos Kyprianou

We present a method for designing directional fields on discrete surfaces, that align to the movement of a dynamic mesh, which is animated by rigging and skinning. We exemplify how aligning the mesh to the direction of the movement serves to reduce discretization artifacts in the deformation. Our method designs a non-orthogonal and non-uniform field that respects creases and the movement patterns of the mesh, and uses it to create quad meshes.

# Contents

# Chapter 1

# Introduction

Meshing plays an important role in the quality of a dynamic geometric model, by influencing the visual and numerical quality of simulations working on this model. There is a plethora of methods for meshing, producing good results for *static* meshes. Most mesh generation methods focus on generating the mesh based solely on a specific model pose, and do not take into account any information about the animation and movement of the model to obtain compatible meshes. Such approaches leave many variables uncontrolled, like taking into account the position of the joints. Therefore, they are prone to discretization artifacts, like poor alignment to features. We propose a method to design a mesh that aligns with the direction of the movement of a geometric model, by considering its flexibility and movement profile.

We consider dynamic meshes that are animated by a *rig* skeleton, and move according to *skinning* weights. This is a very common paradigm for performing to perform character animation. In summary, the intuition behind this technique is that a set of "bones" is defined for the mesh, with each bone controlling a part of the mesh (e.g. thigh bone). The skinning weights are defined on the vertices of the surface mesh, and control how much each bone affects each vertex (some vertices are affected by multiple bones). The reader should not confuse this definition of skinning with *skinning methods*, as the latter refers to skin deformation methods.

There are several methods to compute skinning weights form a rig skeleton. We design a directional field on triangulated surfaces, which is aligned to the given skinning weights as much as possible. At the same time, it tries to preserve certain important properties, like alignment, smoothness and orthogonality. Since these properties almost always counter each other, the solution lies in finding the best balance between them. We formulate a quadratic energy and perform a global optimization to achieve this. More on directional fields in Chapters 2 and 3.

The basic intuition behind our method is that skinning weights are related to the movement of the surface. We consider bone projections on faces to obtain directions, and we utilize the skinning weights to define certain *per face dominance regions*. We consider three such configurations: 1-bone dominance, 2-bone dominance, and many-bone. In other words, for a single face we define how dominant the top weighted bone is, how dominant the 2 top weighted bones are, and so on. These regions control what we align to, and how, since we consider multiple directionals and blend them together. Nonetheless, aligning to several, and possibly conflicting, directionals at the same time requires using very general directional fields, such as PolyVectors. The methodology is described in detail in Chapter 4.

We design a directional field that is then used to obtain a quad parametrization which will be used to further obtain a quad mesh. Quad meshes are preferred when it comes to animation, as they naturally align to curvature directions and suit the modeling of certain parts, like limbs. A key feature that a quality mesh should possess is proper alignment near joints, as this is critical to obtain visually good deformations, which is also an area that quad meshes are suitable for. Despite all these, the automatic generation of quad meshes is a challenging task, and an important reason is the fact that many desirable quality criteria (e.g. alignment, orthogonality, smoothness) can conflict with each other.

Relevant works found in the literature do not base their methods on the skinning weights. Instead, they address compatibility to the animation by using other means, such as mesh sequences of key frames, or a database of examples. We aim specifically for dynamic alignment by using skinning weights, to obtain good meshes.

Quality meshes are critical to the performance of *skinning methods*. Skinning methods are responsible for skin deformation and are an important part of the animation pipeline. Common skinning methods have several limitations and cause deformation artifacts. Some such techniques are discussed in the literature, along with their drawbacks and main advantages. Bad quality meshes enhance those artifacts even more, and generally have heavy impact on the visual quality of the model's simulations. Our meshes perform very well regardless of the skin deformation technique applied.

More formally, we present a method that does the following:

- **Directional Field Design** Design a directional field on the surface of the mesh, aligned to the skinning weights in the best way possible.
- **Mesh Generation** Use the directional field that we designed to perform quadrilateral re-meshing in order to properly align the mesh to the properties defined by the field.

Finally, we produce and present results for various examples, both simple and more extreme. We explore and evaluate the different elements of our methods through the results section. Finally, we showcase our post re-meshing result, compared with the original input mesh.

# Chapter 2

# Related Work

## 2.1 Skinning Methods

Skinning methods are the part of the computer animation pipeline that controls *how* the mesh of the animated model deforms, according to a rig skeleton defined and controlled by the animator. We broadly divide skinning methods into three categories: **geometric**, **physical-based**, and **example-based**. Our contribution exclusively addresses the first one, but given a rig skeleton, it can be applied to methods of other categories as well.

**Geometric methods** deform the model mesh using the current pose of the character at any given time, by calculating where each vertex should be. Geometric skinning methods generally offer high performance and are suitable for real-time applications. These methods will be further split into *direct* and *variational*.

*Direct methods* explicitly compute the position of each vertex in the mesh by blending the deformation of the rig in a closed-form formula. The first model introduced in this category, and one of the most commonly used, is the linear blend skinning (LBS) [MTLT88], whose blending functions are affine transformations, each associated with a handle. A handle represents a guiding direction for the mesh deformation, and affects the vertices according to the skinning weight function for that handle. This method is widely used mainly by real-time applications, due to its computational efficiency, and straightforward GPU implementation. However, the simple linear blend skinning generates very noticeable volume loss artifacts. These are known as the *elbow collapsing artifact* (when joints are bent by ~180 degrees) and the *candy wrapper artifact* (when joints are twisted by a significant rotation). They are caused because of the linear combination of rotations that are relatively largely different. Furthermore, the weight assignment is a tedious process, which used to be done manually. An example is [JSW05], which requires the artist to manually specify the control handles. Now more state-of-the-art automate this process. Notable examples are [BP07; Jac+11; DDL14]. There are many existing approaches, and often the artists have to manually tune stuff to improve the results.

Non-linear techniques like [Ale02; For+07; YSZ06], prevent volume loss and generally create better deformations. However, they lead to other artifacts, as described in the course [Jac+14], and also make the process non-linear. One of the most common non-linear techniques is the dual quaternion skinning (DQS) [Kav+08]. This method avoids linear combinations of rotations by, instead, using unit dual quaternions to represent and blend rigid transformations for each vertex. Linear combination of unit dual quaternions further requires normalization, for a unit dual quaternion to be

obtained. The intermediate step introduced in DQS ends by converting each of the resulting unit dual quaternions to a transformation matrix for a single vertex. Despite overcoming the previous artifacts, DQS suffers from an artifact that creates buldges at joints while bending [KH14]. This issue is generally resolved manually, which is not preferable, and can be tedious. Finally, even though linearity is lost with dual quaternion skinning, every operation of the method is closed-form, and thus can be implemented very efficiently.

Our method requires the specification of an underlying skeleton along with the associated triangulated mesh, so that we can calculate skinning weights, that are our input. Therefore, it can be applied to all of the pre-mentioned methods and categories. In the category that follows, our technique will not be applicable to all of the methods, since a rig skeleton is not necessarily present. Therefore, skinning weights cannot be obtained from the standalone input used in such methods.

*Variational methods* pose the task as an optimization problem, minimizing an energy function, which typically measures deformation. Variational methods generally require iterative solvers. There are linear variational methods that have certain problems as discussed in [BS08]. This led to the development of non-linear techniques [Jac+12; SA07]. These are slower to optimize, their complexity grows non-linearly with the size of the mesh, and struggle to reach interactive framerates. However, [Jac+12] uses a reduced model, thus they combine the benefits of variational and direct methods. They minimize some shape energy, but instead of doing so on the shape itself, they restrict the available degrees of freedom and therefore have a lower-dimensional deformation subspace. Many variational methods, including [Jac+12; SA07], require both a skeleton and skinning weights, making them suitable to be used with our method.

## 2.2 Meshing

We define and cover two categories of meshing techniques: **dynamic** and **static**. We refer to the meshing techniques that mainly interest us as **dynamic**, meaning the ones that operate on mesh sequences, or a moving skeleton. Our goal is to ensure that the meshing is compatible throughout the animation. In an analogous manner, **static methods** operate on a given mesh pose, and don't take into account information regarding the movement. The number of works targeting dynamic meshes is very limited, when compared to the static ones.

Our work focuses on the generation of quadrilateral meshes, since they are best suited for joint alignment, and can always be converted to triangles for utilization of modern graphics cards [Bom+13]. At the same time, they offer natural alignment to curvature directions and are suitable for the modeling of most animated character parts, like limbs. There are certain quality criteria that characterize a quad mesh, like regularity, orthogonality, anisotropy, smoothness. However, despite all the positives of quad meshes, defining and satisfying these criteria is not trivial, as they compete with each other. Furthermore, there may be applicatoin specific criteria that a mesh is targeted at. For the aforementioned reasons and others, the task of automatically producing good quad meshes is challenging. [Bom+13] is a good source of information on quad meshes.

### 2.2.1 Static Meshing

[KNP07] presents a linear parametrization algorithm that takes as input a user-defined frame field and maps it to a single vector field on a branched covering. The singularities are defined by the input field. By optimizing an energy and using a Hodge decomposition, they produce an integrable vector field that is used to generate a quad mesh. [Jak+15] describes a method that uses a RoSy field to remesh a surface, using a local smoothing operator. The meshes produce exhibit high isotropy, and good alignment on sharp features.

[Dia+15] describes a framework for designing curl-free PolyVector fields that align to user-defined constraints, aiming to provide a better parametrization. While it can produce good integrable fields, they require non-linear optimization used to enforce integrability. [BZK09] is a field guided method based on an integer valued representation. Beginning with an input mesh, this method requires some constraints to be defined by the user, especially on sharp features. Using the input mesh, a cross field is generated, that interpolates those constraints. They automatically define singularities and use a mixed-integer optimization for global parametrization, which is used to extract a quadrilateral mesh. In practice, automatic methods like [BZK09] and [Dia+15] are highly desirable, because they avoid specifying singularities.

As standard practice, including aforementioned methods, meshes used in animation are the first part of the animated model that gets designed. The definition of a skeleton and any deformation information come *after* the mesh. Subsequently, the artist that creates the mesh must take into account the possible deformations that the model is going to undergo. This includes any amount of influence that the artist may have, depending on the technique being used. As a result, the process becomes tedious, significantly longer, and introduces room for error and inconsistency.

Even methods that are fully automatic, basically attempt to produce results that are on par to what an artist-influenced method would produce. Typically, they operate on a single pose of the model to be animated, again with no skeleton defined. As such, they lack any information concerning the animation and possible deformations of the mesh.

Currently, automatic static methods, that operate on a single character pose, cannot produce acceptable results for immediate use. There is always the need for manual work by an artist. The main issue lies in the fact that by operating on a single pose, we limit ourselves from capturing information about critical points that require focus, leading to discretization artifacts. For example, regions near the joints of a model's mesh are very critical, mainly in terms of proper alignment and density. However, by taking into account dynamics, we enable ourselves to be able to enforce the properties that we want at those critical areas.

### 2.2.2 Dynamic Meshing

In [Yao+09], the user specifies a skeleton in the model, represented as a collection of line segments. Then, they construct what they call a poly-pipe, which is an approximated representation of the original model created by inflating the skeleton, and consists of only quad faces. Finally, the model is parameterized by the poly-pipe, and compatible quadrangulation is performed. This method requires significant amount of input. An example-based method that also works with sketching is presented in [Mar+15].

They create a database with quadrangulation rules/patterns, that are extracted from models manually designed by artists. The user is able to interactively sketch the desired flow of the edges, by drawing lines on the surface. Afterwards, the system queries the database to find appropriate patterns to fill the interior of each sketch. The consistency and compatibility, in this case, are aimed to be attained by utilizing these manually designed patterns.

[Mar+13] presents a quadrangulation method, compatible with the animation, that takes as input a *sequence* of shapes (deformations of a single triangle mesh) with point to point correspondence. As such, each individual triangle has a known sequence of deformation sampled by key frames. Analyzing these deformations for each specific triangle, a cross field is initialized, taking into account several factors, which is used to extract a quad mesh. This method depends on the poin to point correspondence. In a similar setting, [MH16] takes as input a collection of 3D shapes, explore their correspondence (not necessarily point to point), and extract feature lines independently for each shape, that are used to align separate cross fields. Next, they generate quad meshes from the fields and they proceed to design a cut graph that aligns all of the shapes in a common parametric domain. The independent nature of the algorithm has an impact on non-aligned regions. The most recent method, presented in [Aze+17], aims for an *approximate* consistency between a pair of shapes. Given a pair of triangle meshes, with a given correspondence, a cross field is calculated on each shape mesh, and a magnitude of consistency is enforced, which can be altered. Finally, the two meshes are quadrangulated separately, maintaining the consistency. This method allows for focus *only* on specific shapes of the object.

Generally, using the skeleton instead of a sequence of key frames can be advantageous. Firstly, it allows for automatic detection of where the joints are and where the deformation happens, whereas when using a key frame sequence one has to estimate that. And secondly, there isn't any issue about correspondence, which is required when dealing with mesh sequences. Correspondence can either be there when the meshes are designed, which is good; or created later, which is inconvenient and can cause problems. On the other hand, dealing with a mesh sequence rather than a skeleton can be advantageous when it comes to features and animations that aren't associated with the skeleton, like facial features, since we wouldn't be able to distinguish the important parts with a skeleton.

## 2.3 Directional Fields

The method presented in [IBB15] creates a non-orthogonal frame field on a sketched object to enrich the shading and texturing of the sketch, when it's given textures. The authors discuss how their fields can be of direct use to quad meshing algorithms, where smoothness, orthogonality or non-orthogonality can be desired. [CDS10] presents a method to create a smooth directional field on discrete surfaces, based on a set of singularities. It revolves around how we can map one tangent space to another and ultimately create a connection to move a vector along a surface consistently (independent of the path taken). These connections have several uses in mesh processing.

For detailed general information on directional fields, including synthesis, design and applications we refer the reader to [Vax+16]. This survey introduced a consistent taxonomy of fields, as there are term inconsistencies in the literature. When some form of direction field synthesis is needed in an application, the question arises of

which synthesis method (using which representation and which discretization) is best suited.

In order to optimize for the "best" directional field for a given application, the notion of "best" has to be defined and formulated. For this, there are multiple factors to be considered, like the topology of the field, different constraints, and the fairness of the field. The most common way to measure the fairness of a field is by using the Dirichlet energy, measuring how variable, or rather, non-similar, the field is between adjacent tangent spaces. In the case that other objectives are desired, such as orthogonality and feature alignment, there may be competition between all of the properties, and a balance is required.

Our goal is to design the directional field that is the "best" for the purpose of dynamic meshing. Initially, the target is to obtain a smooth cross field that aligns to the weight functions predefined on the model. We are interested in the alignment and smoothness currently, but other properties may appear necessary along the way. We make use of [Kn13] to obtain the globally optimal result from a combination of energies. Furthermore, it allows the control of the influence of each energy, to possibly adapt results even better.

As an extension, we aim to obtain a PolyVector field [Dia+14], which is a more general and less restricted form of directional fields than the cross field. No vector is necessarily related to another in terms of symmetry and magnitude. For this, we need to align two vectors instead of one, which is more challenging, but at the same time more rewarding. PolyVectors offer better alignment because they are more flexible, given that they are handled properly. However, orthogonality is not default and needs to be controlled by extra energy terms. Furthermore, when working with PolyVectors, we can also vary the density of the discretization in spot of significance, primarily near the joints.

# Chapter 3

# Background

In this Chapter we detail the key components required for our approach.

## 3.1 Rigging and Skinning

In computer animation, the standard way of animating characters or objects with joints is by using a skeleton. This is a popular approach, since reducing the animation work to the manipulation of a skeleton is simple and intuitive. This concept was introduced in [MTLT88]. With such techniques, a character is represented by two parts: a hierarchy of connected bones (skeleton or rig), and a surface mesh (or skin). Typically, a polygonal mesh is used, which is a visual representation of an object that is to be animated, as a list of *vertices* $V = \{v_1, \ldots, v_n\} \in \mathbb{R}^3$ with 3D positions and a list of *faces* $F$, with consequently a set of *edges* $E$. The faces are formally the *constant* connectivity of of the vertices, and are the element to be rendered.

We work with such a mesh $\mathcal{M} = \{V, E, F\}$, to which a *rigging* was made. Rigging is the process of creating the skeleton for a character (or *articulated* object), given the character's visual representation. It is typically done manually by the artist. A rig is a hierarchy of a set of transformation matrices, each representing a bone. These transformations are the one thing that changes throughout the animation, and are what guides it. We represent the rig as a graph $G = \{J, B\}$, composed of a list of joints $J = \{j_1, \ldots, j_n\} \in \mathbb{R}^3$, and connectivity list of bones $B$. Figure 3.1 provides an illustration of this process.

The mesh deforms accordingly by considering the transformations prescribed for the rig, in a process called *skinning*. In this work, we target skinning using precomputed weights: Every pair of bone $b \in B$ and mesh vertex $v \in V$ are assigned a (time-independent) weight $w_{b,v}$ which are a partition of unity over the bones: $\sum_{b \in B} w_{b,v} = 1$. Consider the transformation $T_b$ prescribed to a bone, then the transformation of a vertex position is:

$$v' = \left( \sum_{b \in B} w_{b,v} T_b \right) v \tag{3.1}$$

.

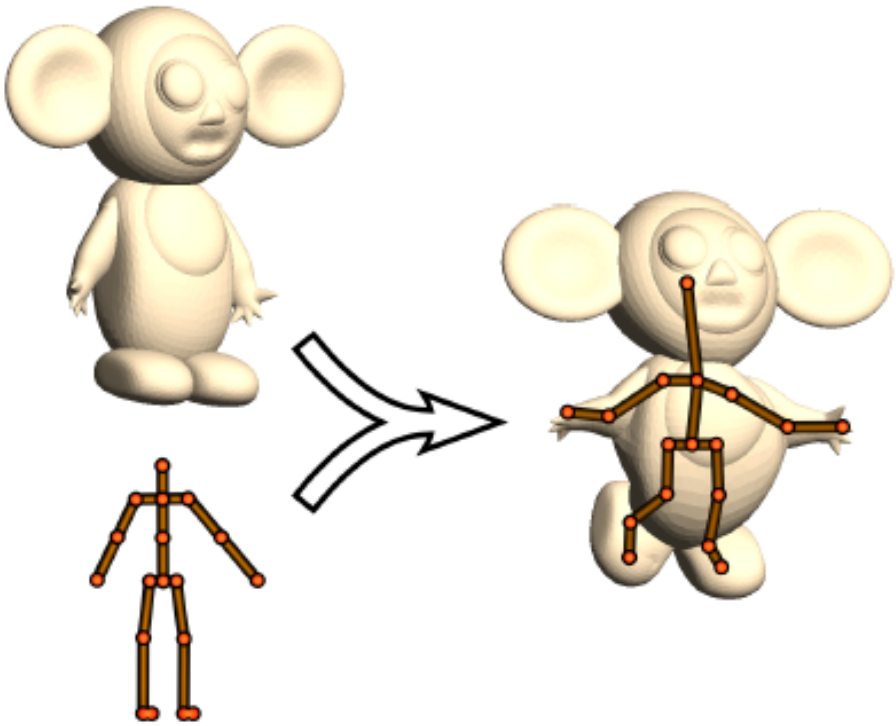

FIGURE 3.1: Example of applying a rig to a mesh. And after skinning showcasing deformation by animating the rig. Figure obtained from [BP07].

## 3.2 Directional Fields

Directional fields are the assignment of a directional object to every *tangent space* in some *tangent bundle* on an ambient space. A directional object is a collection of vectors and directions (vectors without magnitude), usually without predefined ordering. Directional fields discussed in this work, like many similar ones, are defined on the surfaces of triangle meshes.

More specifically, we use a face-based representation in which the planes that support the triangles are the tangent planes. Given a mesh $M = \{V, F\}$, we consider the plane defined on every face $f \in F$ as a single tangent space, and the entirety of $F$ as the tangent bundle. According to this definition, since one directional object is defined on each face, the directional field is piece-wise constant.

### 3.2.1 Connections

We need to perform comparisons between the vectors (or directional objects) in the directional field in some way, so that we obtain desired properties, such as smoothness. But since the vectors on the faces are defined in *different* tangent spaces, a way is needed to map one space to the other. A solution to this is to use a *connection*, which describes how a vector will change when moving from one tangent space to another, as it moves along a surface.

We use the discrete Levi-Civita connection, defined on the *complex* space $\mathbb{C}$. Complex numbers exhibit a useful property regarding symmetry, by allowing the mapping of multiple vectors to a single one. For details, we refer [CDS10]. This connection is used to compare two vectors $u_f$, $u_g \in \mathbb{R}^3$ that are in the tangent spaces of the adjacent faces $f$, $g \in F$. This requires the vectors to be in *local* (2D) complex number representation $u_{f_c}$, $u_{g_c} \in \mathbb{C}$, which is trivial, by setting one coordinate as the real part and the other as the imaginary part. Note that from this point everything we refer to in this subsection will be in complex form, without labeling it. The comparison is done through the common edge $e \in E$ (set of all edges) of the two faces, which is respectively converted to the two complex numbers (vectors) $e_f$, $e_g \in \mathbb{C}$. For *normalized* vectors, LC-parallelity between $u_f$, $u_g$ is defined as:

$$u_f \bar{e}_f = u_g \bar{e}_g, \tag{3.2}$$

where $\bar{e}$ is denoted as the complex conjugate of $e$, used to obtain the angle between each pair of vectors.

### 3.2.2 Frame Fields

We use an $N$-PolyVector field, which is a set of $N$ tangent vectors, and each $N$-PolyVector is piece-wise constant. In particular, we make use of *frame fields*, that are 4-PolyVector fields, composed by two 2-RoSy fields. Therefore, the above formulation has to be extended to general directional objects, instead of a single vector. In this section we will cover the formulation regarding frame fields.

A frame field is characterized by the following polynomial:

$$P(z) = (z^2 - u^2)(z^2 - v^2), \tag{3.3}$$

where $u$, $v$ are the two 2-RoSy fields. Frame fields are utilized through their complex coefficients $x_0 = u^2v^2$ and $x_1 = -(u^2 + v^2)$, that are vectors themselves. These coefficients exhibit certain properties. $x_0$ represents the cross of bisectors between $u$ and $v$, while $x_1$ measures their deviation from being a perfect cross (if $x_1 = 0$ then $u$ and $v$ form a perfect cross) [Dia+14]. Consequently, a *cross* field (4-RoSy field) is a special case of frame field, characterized by $P(z) = (z^4 - w^4)$, featuring four equal vectors that form a perfect cross.

Based on the equation 3.2, the Levi-Civita Parallelity regarding frame fields becomes:

$$x_{f,0}(\bar{e}_f)^4 = x_{g,0}(\bar{e}_g)^4, \tag{3.4}$$

$$x_{f,1}(\bar{e}_f)^2 = x_{g,1}(\bar{e}_g)^2. \tag{3.5}$$

As this is defined using the coefficients, the final real vector factors can be trivially obtained by solving the field polynomial for these coefficients. In the same manner as before, cross fields are a more restricted case, with a therefore simpler formulation, represented by $(w_f)^4(\bar{e}_f)^4 = (w_g)^4(\bar{e}_g)^4$.

## 3.3 Energy Minimization

Since we are interested in several field properties (e.g. alignment, smoothness, orthogonality), we will be using quadratic energy minimization to achieve the final result balancing all of the properties. Each of the properties is represented by a different energy. We will only be dealing with quadratic energies, from which we derive a linear system to solve, containing all of the energies.

Using the LC connection (eq. 3.4, 3.5), the **smoothness** energy to be minimized for *cross fields* is defined as:

$$E_{smooth} = \sum_{e(f,g)\in E} \left|(w_f)^4(\bar{e}_f)^4 - (w_g)^4(\bar{e}_g)^4\right|^2, \tag{3.6}$$

and for *frame fields* as:

$$E_{smooth} = \sum_{e(f,g)\in E} \left|x_{f,0}(\bar{e}_f)^4 - x_{g,0}(\bar{e}_g)^4\right|^2 + \left|x_{f,1}(\bar{e}_f)^2 - x_{g,1}(\bar{e}_g)^2\right|^2. \tag{3.7}$$

The optimal smoothness is for every pair of vectors (or directional objects) to be LC-parallel across a common edge, which equals zero smoothness energy.

In the case of frame fields, **orthogonality** comes into play, which measures how orthogonal the pair of 2-RoSy fields is. This is an important property, as it helps in avoiding degenerate polygons when we perform the meshing. Orthogonality is controlled by the energy:

$$E_{orth} = \sum_{f\in F} \left|x_{f,1}\right|^2. \tag{3.8}$$

However, an uncontrolled minimization of this will simply reduce the field to a cross field, defeating the purpose of using general directional fields in the first place. We control it by minimizing it together with other energies, namely smoothness and alignment, while defining each one's influence. Given an **alignment** energy $E_a$, that is to be discussed in the next section, the **total** energy of the system will be of the

form:

$$E_{total} = \lambda_1 E_{smooth} + \lambda_2 E_{align} + \lambda_3 E_{orth}, \tag{3.9}$$

with $\lambda_x$ representing the influence of the associated energy.

Each of the energies is of the form $|Ax - b|^2$, where $A$ is the coefficient matrix and $b$ is the constants vector. Therefore, the final $E_{total}$ will be of that form as well. Minimizing such energy is commonly proven to be equivalent to solving the following derived linear system:

$$\overline{A^T} Ax = \overline{A^T} b, \tag{3.10}$$

where $\overline{A^T}$ is the *conjugate transpose* of the *complex matrix* $A$.

## 3.4 Quad meshing

The most common way to remesh a triangle mesh into a quadrilateral mesh is to compute a frame field on the faces, and then integrate them to a parameterization of surface to the plane. This is obtained by solving, for example, a Poisson equation [BZK09]. Depending on the method used, the resulting parametrization can vary, possibly affecting the number of singularities and quad length distortion. A grid tessellation on the plane is then textured on the mesh by the inverse parameterization, to become the consequent quad mesh. This process requires the cutting of the mesh so that it has disc topology, where all the singularities are on its boundary, and then taking care that the grid is *seamless* across the cut, both in alignment and in integer transitions of the parameterization.

# Chapter 4

# Methodology

## 4.1 Skinning-Guided Directional Fields

Our purpose is to design a directional field on a surface so that the resulting quad mesh would deform as naturally as possible with the deforming mesh. The principles of "natural deformation" that we ascribe to are:

- The mesh aligns to the shape of the bone in a region dominated by the movement of one bone.
- The mesh aligns to creasing features, which are defined by regions where two or more bones meet.
- The mesh is otherwise as smooth and isotropic as possible.

In *relation to curvature-guided alignment*, which was explored in the 2 section, a rigging indicates how the mesh is going to deform. For static meshes, there are similar rules, but incentivize alignment to principal curvature directions for quality quad meshing. While an intuitive rigging is usually correlated with the principal curvature directions, it is not always the case. Moreover, curvature directions are often noisy and hard to align to directly, whereas a rigging provides a simple guiding skeleton. Later on in the results section we will provide some comparison of different approaches.

## 4.2 Dominance Regions

To decide how much to align to bones, and to which, we require a method to determine how much a face is dominated by such bones. We consider a single face $f$, with a set of skinning weights $\{w_{f,b}\}$. Let us assume, without loss of generality, that the bones are locally re-indexed so that the skinning weights are in descending order: $\left\{w_{f,1}, \cdots, w_{f,|B|}\right\}$. We define the weight $\omega_k$ as determining how much the first $k$ bones in the sorted order are dominant, as follows:

$$\omega_{f,k} = k \cdot (w_{f,k} - w_{f,k+1}) \tag{4.1}$$

In that way, a face is perfectly controlled by a single bones only if it has weight 1, perfectly by two bones only if they have a mutual weight of 0.5 (since $w_{f,2} \leq 0.5$ by definition as $w_f$ are a partition of unity), and so on. In Figure 4.1 we show how the dominance scores look for a typical rig. These regions are exclusive for each face, and

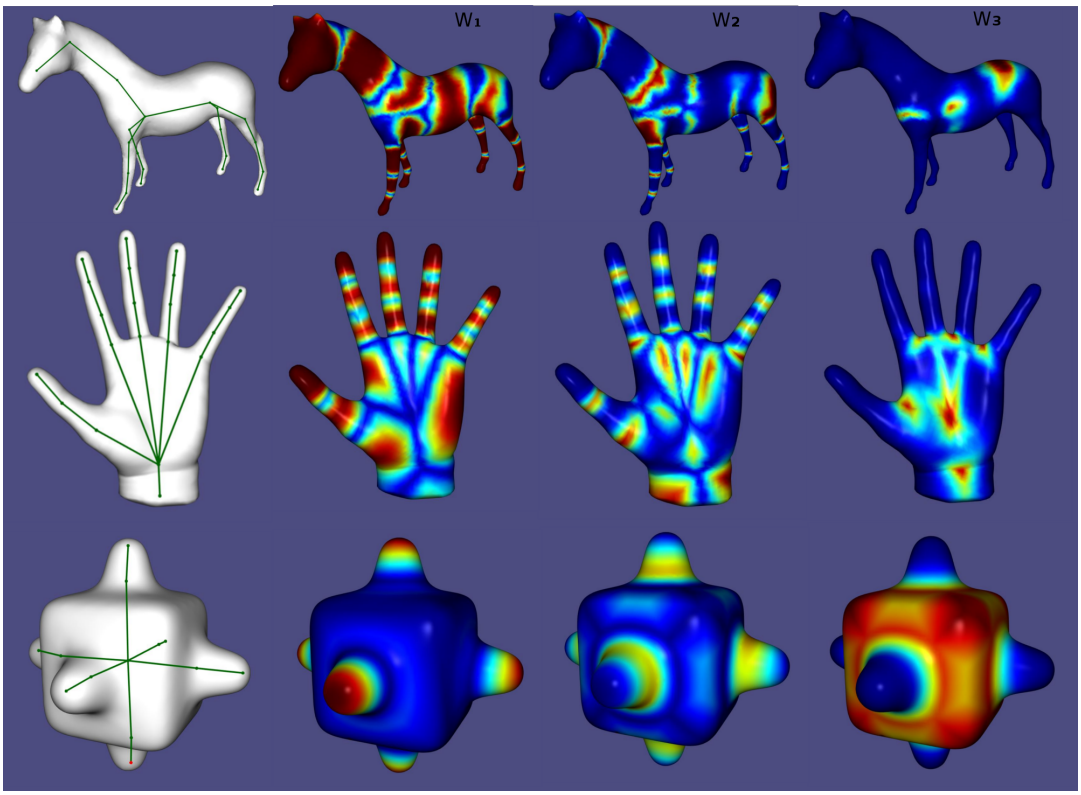


FIGURE 4.1: Dominance regions for the entirety of the rigged mesh, showing $\omega_1$ to $\omega_3$ from left to right respectively. The top two are common rigs for animations, whereas the third has a non-trivial rig.

therefore we can display them on the entire mesh, unlike the weight functions that have to be displayed per bone.

## 4.3 Soft Alignment Constraints

The purpose of giving dominance scores is to align the directional fields to the bones correlating on how much these bones are dominant. For this, we first need to define alignment to bones, and then form the alignment constraints.

**Bone alignment** Consider the normalized vector of the bone's direction $\vec{b}$ (the sign does not matter). Then, we use its projection $\vec{b}_f$ on the face:

$$\vec{b}_f = \vec{b} - \langle \vec{b}, \vec{N}_f \rangle \vec{N}_f, \tag{4.2}$$

with $\vec{N}_f$ being the normalized normal of face $f$. A visualization of this process can be seen in the left part of Figure 4.2, illustrated by the red line. The obtained projection is a tangent vector on $f$, which can therefore be represented as a complex vector, as described in 3. Such projection can be close to zero; nevertheless, we note that this happens for non-dominant bones, or cases where alignment is not of particular importance (e.g. tips of fingers). Otherwise, projections that are close to zero mean that the rigging is not fine enough. The algorithm does not need to assume anything about the projection.

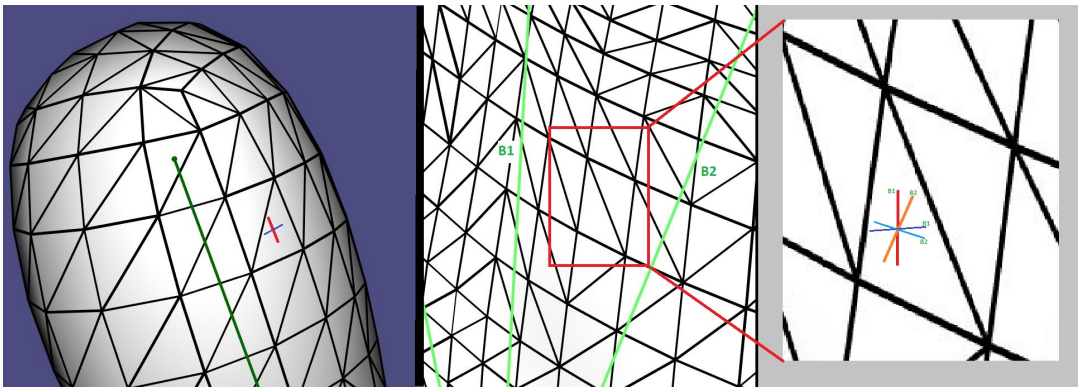


FIGURE 4.2: Illustration of the projections of the bone(s) on a triangle face. The bones are illustrated as green lines. **Left:** Example projection of the bone in a 1-zone, highlighted in red. The blue line represents the perpendicular to the projection. **Right:** Example of the projections in a 2-zone. The red and orange lines represent the main projections, while the blue and light blue lines represent the perpendiculars respectively.

**Alignment to dominant bones** We create a soft alignment constraint for the case of a single dominant bone as the unit cross-field, or formally:

$$c_0^1 = \left(\hat{b}_{f,1}\right)^4, \tag{4.3}$$

$$c_1^1 = 0, \tag{4.4}$$

with $\hat{b}_{f,1}$ being the normalized $\vec{b}_{f,1}$, where the purpose is to try and have a perfect quad that is exactly aligned to the bone.

For the case of two dominant bones, we assume that this is the case close to a joint, and therefore we have to vary the density so as to align to the crease. The density is controlled using these scaling policies:

$$D = \left|\langle \vec{G}_{f,\omega_1}, \vec{b}_{f,1} \rangle\right| \tag{4.5}$$

$$s_1 = \begin{cases} 4 \cdot D, & \text{if } angle(D) > 45 \\ 4, & \text{otherwise} \end{cases}$$

$$s_2 = \begin{cases} 4 \cdot (1 - D), & \text{if } angle(D) > 45 \\ 1, & \text{otherwise} \end{cases}$$

where $\vec{G}_{f,\omega_1}$ is the *normalized* gradient of $\omega_1$ at face $f$ and $0 < angle(D) < 90$ degrees. The constant values in these metrics have been derived as fitting, after repeated experimentation. The distinction of cases based on the angle is performed to differentiate between two possible configurations of the two dominant bones: a) forming an obtuse angle, and b) forming an acute angle. This doesn't necessarily require the bones to be connected, but these angles can be imagined by considering

the closest edge of the bones as the connection point. The first case translates to $D$ giving a "*small*" angle, whereas giving a "*large*" angle for the latter, and therefore 45 was chosen. The objective is to obtain density *along* the bones for the case (a), and *perpendicular* to the the bone for case (b). We base this assumption for movement to having a representative rest pose as input.

Using these, the actual coefficients are created for face $f$, starting with the projections of the relevant bones:

$$u = \vec{b}_{f,1}, \tag{4.6}$$

$$v = \vec{b}_{f,2}, \tag{4.7}$$

followed by two initial sets of coefficients $t_{0,1}$ & $t_{1,1}$ and $t_{0,2}$ & $t_{1,2}$, one for each projection. The initial projections of this are visualized in Figure 4.2. Initially representing perfect crosses, they are completed with the addition of the calculated scaling policies respectively:

$$t_{0,1} = (s_1 \cdot u)^2 \cdot \left(s_2 \cdot u^{\perp}\right)^2, \tag{4.8}$$

$$t_{0,2} = (s_1 \cdot v)^2 \cdot \left(s_2 \cdot v^{\perp}\right)^2, \tag{4.9}$$

$$t_{1,1} = -\left((s_1 \cdot u)^2 + \left(s_2 \cdot u^{\perp}\right)^2\right), \tag{4.10}$$

$$t_{1,2} = -\left((s_1 \cdot v)^2 + \left(s_2 \cdot v^{\perp}\right)^2\right). \tag{4.11}$$

These are later blended to obtain the final coefficients for the two dominant bones:

$$c_0^2 = \frac{w_{f,1} \cdot t_{0,1} + w_{f,2} \cdot t_{0,2}}{w_{f,1} + w_{f,2}}, \tag{4.12}$$

$$c_1^2 = \frac{w_{f,1} \cdot t_{1,1} + w_{f,2} \cdot t_{1,2}}{w_{f,1} + w_{f,2}}. \tag{4.13}$$

In the case of more than two bones, we assume that the region does not have specific alignment (for instance, a joint region with many bones). As such, we opt to obtain the smoothest and most isotropic cross field.

The field we finally align to is then blended with the dominance weights, only considering the first two dominance regions:

$$c_0 = \frac{\omega_1 c_0^1 + \omega_2 c_0^2}{\omega_1 + \omega_2}, \tag{4.14}$$

and similarly for $c_1^1$ and $c_1^2$. Note that we don't have a field for the cases $k \geq 3$, and that is since there is no proper alignment at such regions; our alignment energy in the next section would be nulled in that case.

## 4.4 Putting the energies together

Our variables are the $x_{f,0}, x_{f,1}$ PolyVector coefficients per face $f \in \mathcal{F}$, where we minimize the following quadratic energy:

$$\{x_0, x_1\} = argmin\left(E_{align} + E_{smooth} + E_{orth}\right), \tag{4.15}$$

$$\begin{aligned} E_{align} &= \sum_{f \in \mathcal{F}} \alpha(f) Area_f \left(|x_{f,0} - c_{f,0}|^2 + |x_{f,1} - c_{f,1}|^2\right) \\ E_{smooth} &= \sum_{(f,g) \in \mathcal{E}} \beta(f,g) Diam_{f,g} \left(|x_{f,0} - x_{g,0}|^2 + |x_{f,1} - x_{g,1}|^2\right) \\ E_{orth} &= \sum_{f \in \mathcal{F}} \beta(f) Area_f \, |x_{f,1}|^2 \end{aligned} \tag{4.16}$$

The energies follow the forms that were discussed in previous sections, with the addition of certain factors. Starting with the factors that control the global weight of each energy per face:

$$\alpha(f) = 0.6\omega_{f,1} + 0.2\omega_{f,2}, \tag{4.17}$$

which has the purpose of favoring $\omega_{f,1}$ over $\omega_{f,2}$, because we want faces with higher $\omega_{f,1}$ to enforce their alignment throughout the mesh. Subsequently, and derived from $\alpha(f)$:

$$\beta(f) = 0.2\omega_{f,1} + 0.4\omega_{f,2} + 0.5\omega_{f,3}, \tag{4.18}$$

$$\beta(f,g) = \frac{\beta(f) + \beta(g)}{2} \tag{4.19}$$

where the constants used are the remainder from $\alpha(f)$ when subtracting from 1, divided to be split between smoothness and orthogonality. There is no incentive to favor either smoothness or orthogonality over the other, hence the split. $\beta(f,g)$ represents the smoothness case, since it is performed on pairs of faces. The higher $\alpha(f)$ the more the face will try to align, and otherwise it will resort to being as smooth and orthogonal as it possible.

Additionally, $Area_f$ is the area of the face divided by the area of the largest face, and is applied to alignment and orthogonality, to enforce relative impact based on size of face. $Diam_{f,g}$ is the diamond area of the respective edge, composed of 1/3rd of each of the two involved faces.

Finally, this system is solved by liner least squares, using a sparse Cholesky solver, to obtain the resulting PolyVector field.

# Chapter 5

# Results

The full process of our quad re-meshing algorithm, including input and intermediate computation steps, is shown in figure 5.1. Some deformations in the form of poses of this same mesh can be seen in figure 5.2. Our algorithm has been implemented using libigl [JP+18] and Directional [Vax+], and almost all of our illustrations have been visualized through those as well. We use libigl-native file formats and data structures, and therefore any *other* type of input is converted to be able to be used seamlessly. Namely, the mesh file format, for which we convert from several popular input formats (.obj, .off, etc), and the skinning weights format, which will be expanded on next.

In our experiments, the rigs are represented as a graph of vertices and edges, using a libigl-native format. Unlike skinning weights, all of our rigs have been created natively, so no conversion was necessary from other formats. For most of our illustrations, we will be using the Bounded Biharmoninc Weights (BBW) [Jac+11] as skinning weights, because they are more convenient to compute, due to compatibility with our implementation. We use libigl to automatically convert the input surface mesh to a tetmesh and then compute BBW based on that result, since BBW requires the input to be a tetmesh. However, we only use the skinning for the surface mesh for our method. Later on in this section we will also use Geodesic Voxel Binding (GVB) [DDL14], which is and expand on their comparison. GVB is created and exported using Maya 2017, and then converting it to be compatible with libigl. For the purpose of comparing, creating GVB weights is very tedious, due to having to match the already tedious BBW, and that requires manual labor. More on the comparison of the two will come later.

Due to the composite nature of our method's input, there are several factors influencing the results: skinning weights, skeleton quality, rest pose, to name a few. Therefore, for the results we will attempt to evaluate how our method performs when each of those factors changes. In some cases, an intentionally bad input will be used, to showcase how the method adapts to that, and/or how dependent it is on a decent input. Furthermore, the method will be evaluated against static quad meshing methods, to show the advantages of our dynamic meshing.

Initially, we will be testing our method on trivial case, for which we know how the result should be like; typically, these cases are simple and intuitive. The purpose at this point is to verify the correctness and robustness of our method. The two examples that we are going to use for this phase are shown in figure 5.3, along with the respective rigs. Given these meshes, what the result should be is trivial; in terms of where the singularities should be, how the directional field should be aligning, and how the resulting parametrization should look like. At this point, we assume good quality

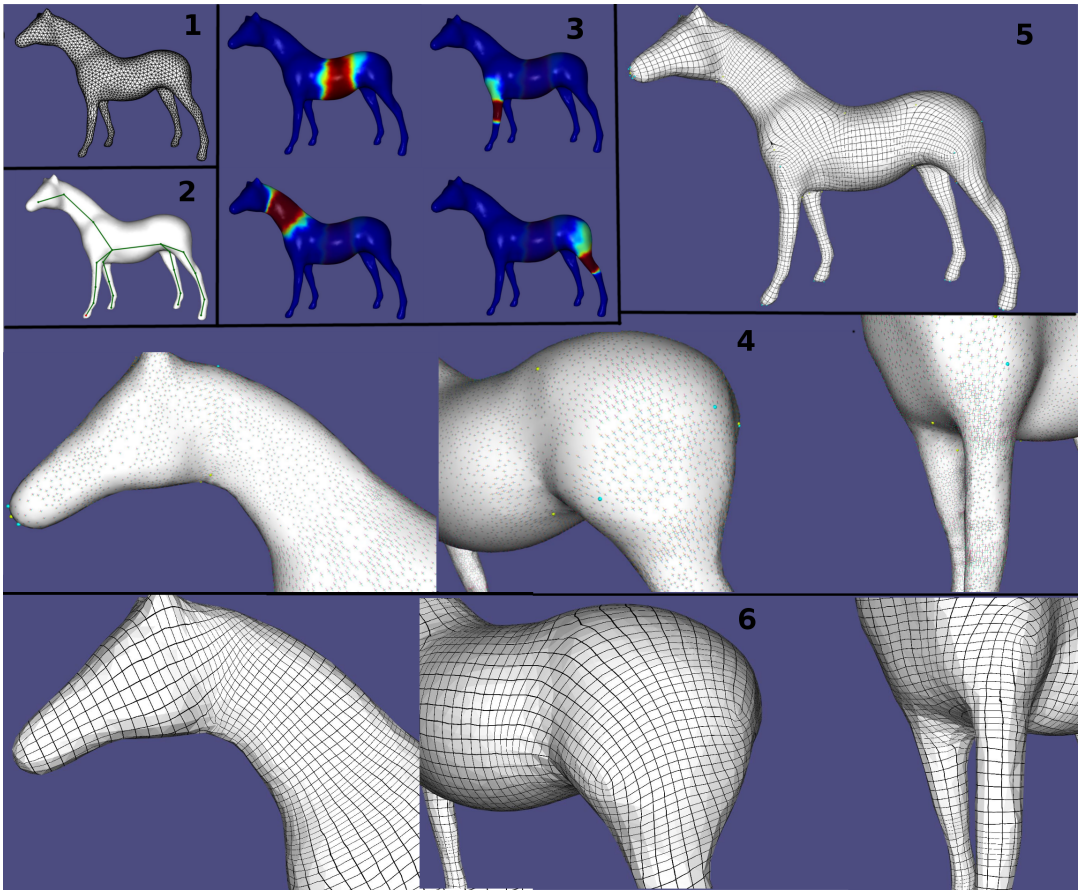


FIGURE 5.1: The complete pipeline of our method, from input to output. **1:** The input triangle mesh. **2:** The rig to be used for animating this mesh. **3:** Highlighted GVB skinning weights for selected bones, indicating key areas. **4:** Key areas of the PolyVector field computed from the input in steps 1, 2, and 3. **5:** The whole resulting quad mesh parametrization. **6:** Close-ups on the key areas of the parametrization, to notice the alignment along the rig/animation, and the adapted density.

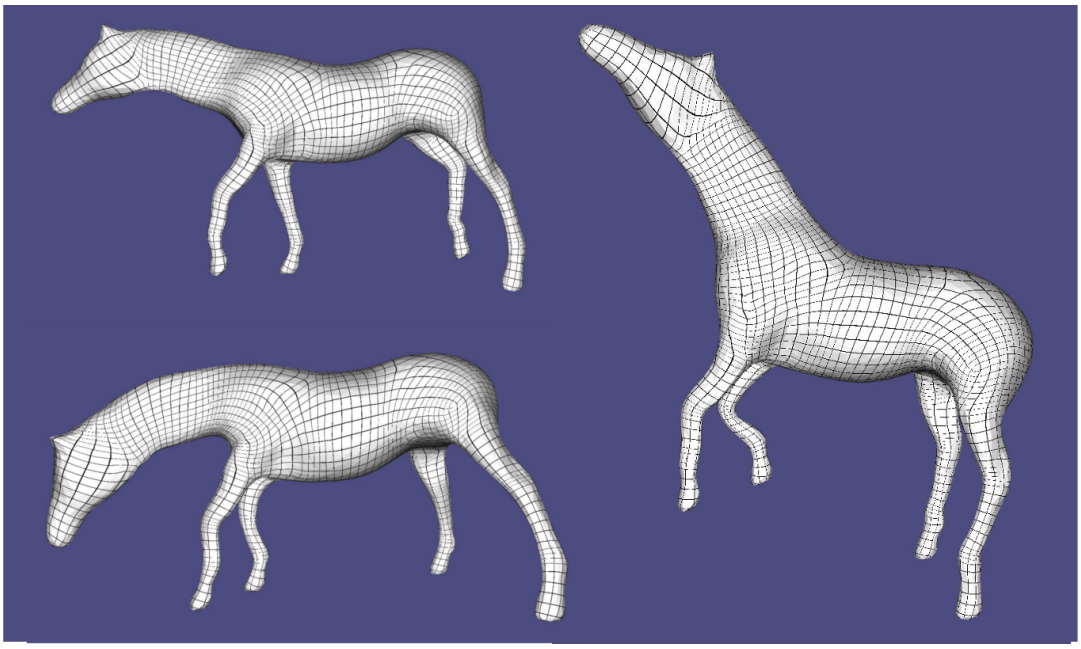

FIGURE 5.2: Several deformations of our resulting parametrization. The parametrization is aligned to the animation of the mesh, and it exhibits higher density near joints. We can see that the mesh deforms mostly around the areas that have higher density.

and representative skinning weights, that for these specific cases we have computed using BBW. 5.4 shows the fields and parametrizations resulted from applying our method on the presented input. In the case of the pear, the field is smooth and aligned to the bones throughout the entirety of the mesh, with singularities appearing at the tips of the bones. Whereas in the case of the box torus, the field is aligned evenly around the closed loop of the torus, with singularities appearing only exactly where the bones meet. We can also observe the directionals displaying alignment to both of the bones in the areas near the singularities, where the influence of the two closest bones becomes almost even.

Other than the alignment to the rig and the weights, our method also has the property or displaying higher quad density on the areas of the mesh that are near the joints (as defined by the skinning weights), that are going to be deformed the most. Our density can be further split into to two types: *along* the bones, and *perpendicular* to the bones. The first and more intuitive case is shown in 5.6, where by observing the knee or upper thigh joints, we can see that the quad density is higher near the joints of the skeleton. 5.2 can also be used as an example to observe the varying quad density, mainly for this type of density. The second type of density is observed when the two most dominant bones (not necessarily connected bones) of a region of the mesh are more parallel than aligned, which results in perpendicular alignment. An example of this can be seen in 5.5, on the wing of the displayed model. By observing the wing, we can easily distinguish the higher density on the folds that the rig would cause by animating the wing.

A key part of our method's input are the skinning weights used, so as mentioned in the beginning of the section, we will compare different skinning methods and evaluate: a) how different skinning methods affect our results, and b) how dependent the method is to good quality weights. Figure 5.7 shows a rigged mesh, along with

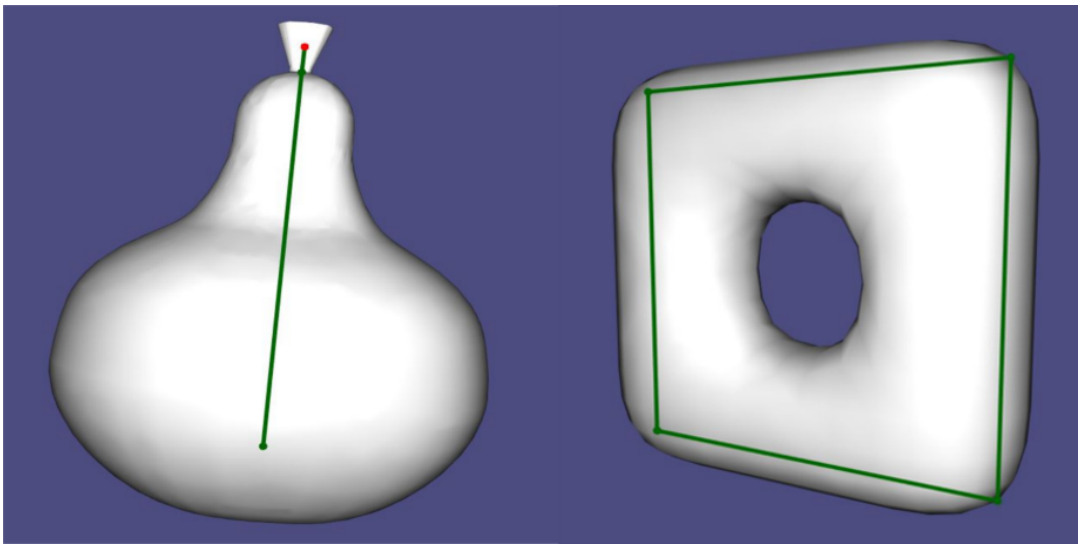

FIGURE 5.3: Trivial cases. The rigs used to create 5.4. **Left:** A 2-bone rig to represent the pear's body and tip. **Right:** A 4-bone square shaped rig to represent the box torus.

three difference skinning methods, computed on given rig: a) Bounded Biharmonic Weights, b) Geodesic Voxel Binding, and c) Piece-wise constant weights. Piece-wise constant weights represent the simple notion of each vertex is only deformed by a single bone. In this case, they are computed by taking the closest bone of each vertex, and these are only here to provide insight on how important quality is. In 5.7 we can clearly see that GVB has the better quality weights for this example. BBW tends to be very dependent and sensitive on the placement of the bones, as they have to be carefully spaced within the tetmesh. The quality and resolution of the tetmesh itself is also a dependency. It can be often tedious to get a good result, since the slightest change on the skeleton can have big impact on the resulting weights. On the other hand, GVB is dependent on joint placement, but in contrast, it is much more consistent in terms of quality. From our experiments, the best case of BBW does not surpass GVB. Moving on to 5.8, we can easily observe that piece-wise constant weights barely provide any density variance along the leg, which was expected as they don't have any smoothness in transitioning. BBW and GVB provide better results, but GVB has the best result in terms of displaying higher density exactly near the joints. BBW also displays that on the lower part of the leg, but the higher one suffered due to worse skinning weights. In general, good quality skinning weights, like GVB in this example, naturally display better results and especially in terms of higher density near joints. Despite that, from observing the results that BBW gave here, the method can still produce viable results from lower quality weights.

The second key component of the input is the rig used. Our method. much like actual animations, is especially dependent on the quality of the rig, because of the direct relation it has with our alignment of the directional field, as opposed to the skinning weights quality, which had a more indirect effect. We will examine two characteristics of a skeleton: a) resolution, and b) bone/joint placement. 5.9 shows two different rigs for the displayed mesh, with their obvious difference of the top one utilizing a minimal number of bones to represent the mesh, while the bottom one has more bones to better approximate the general shape. These resolution cases may be extreme, and one can easily create rigs of intermediate resolution as well, but these are used to

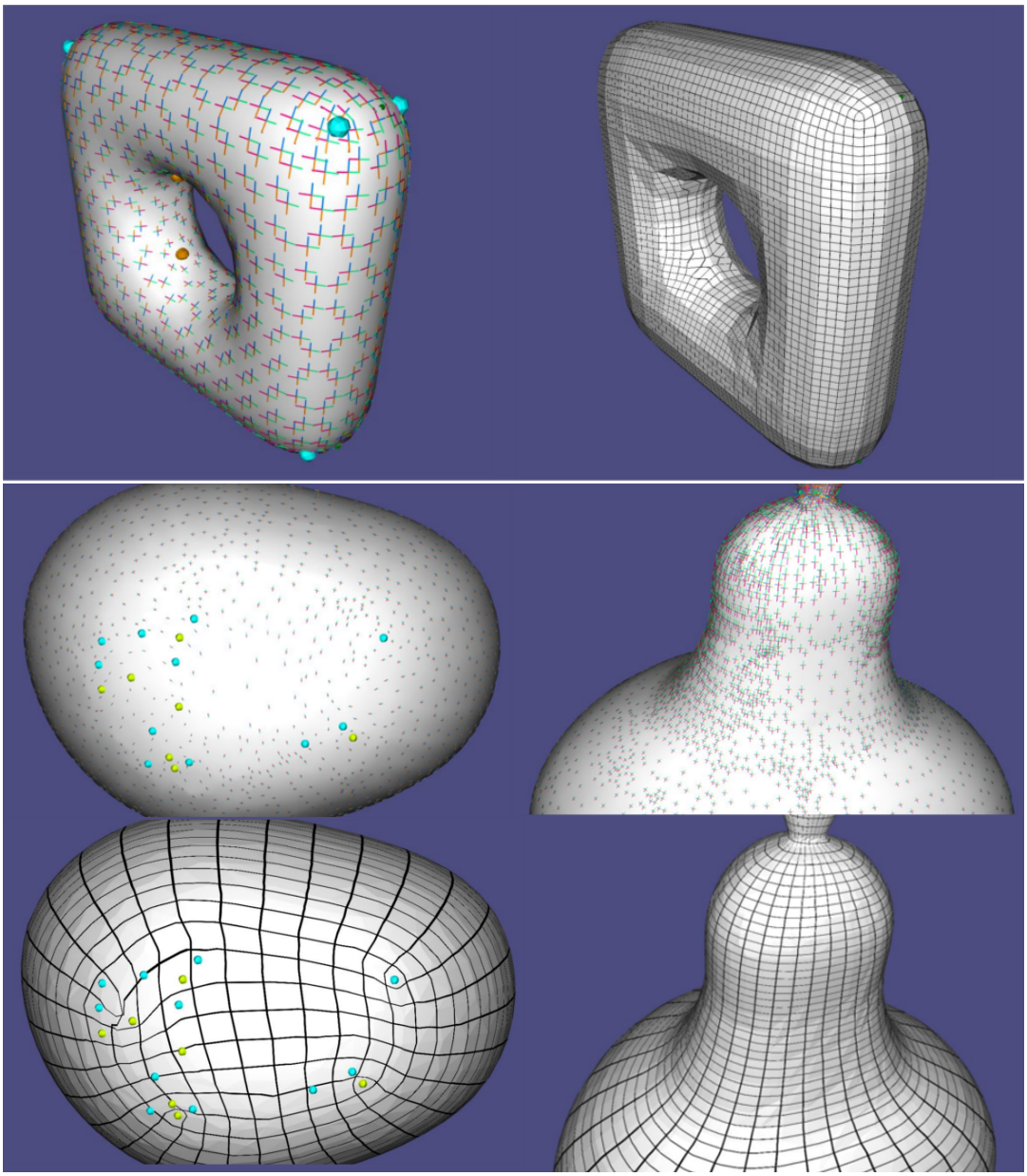

FIGURE 5.4: Trivial cases. Directional fields of the input illustrated in 5.3 and their respective parametrizations.

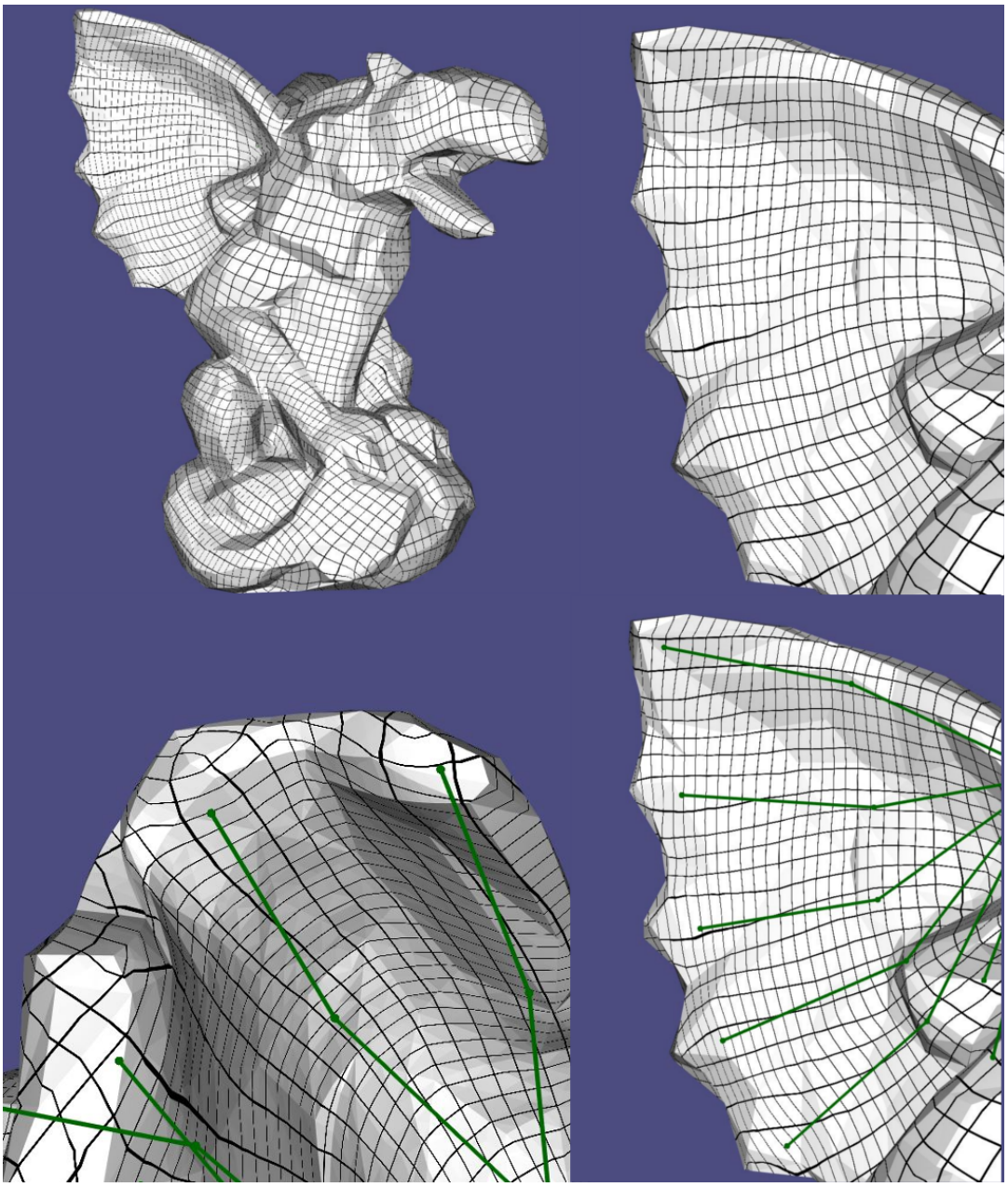

FIGURE 5.5: Showcase of the varying density between *'parallel' bones*. **Top:** The resulting parametrization, highlighting the wing. **Bottom right:** The bones composing the wing. **Bottom left:** Sample deformation on the wing, showing how the higher density is where the deformation happens.

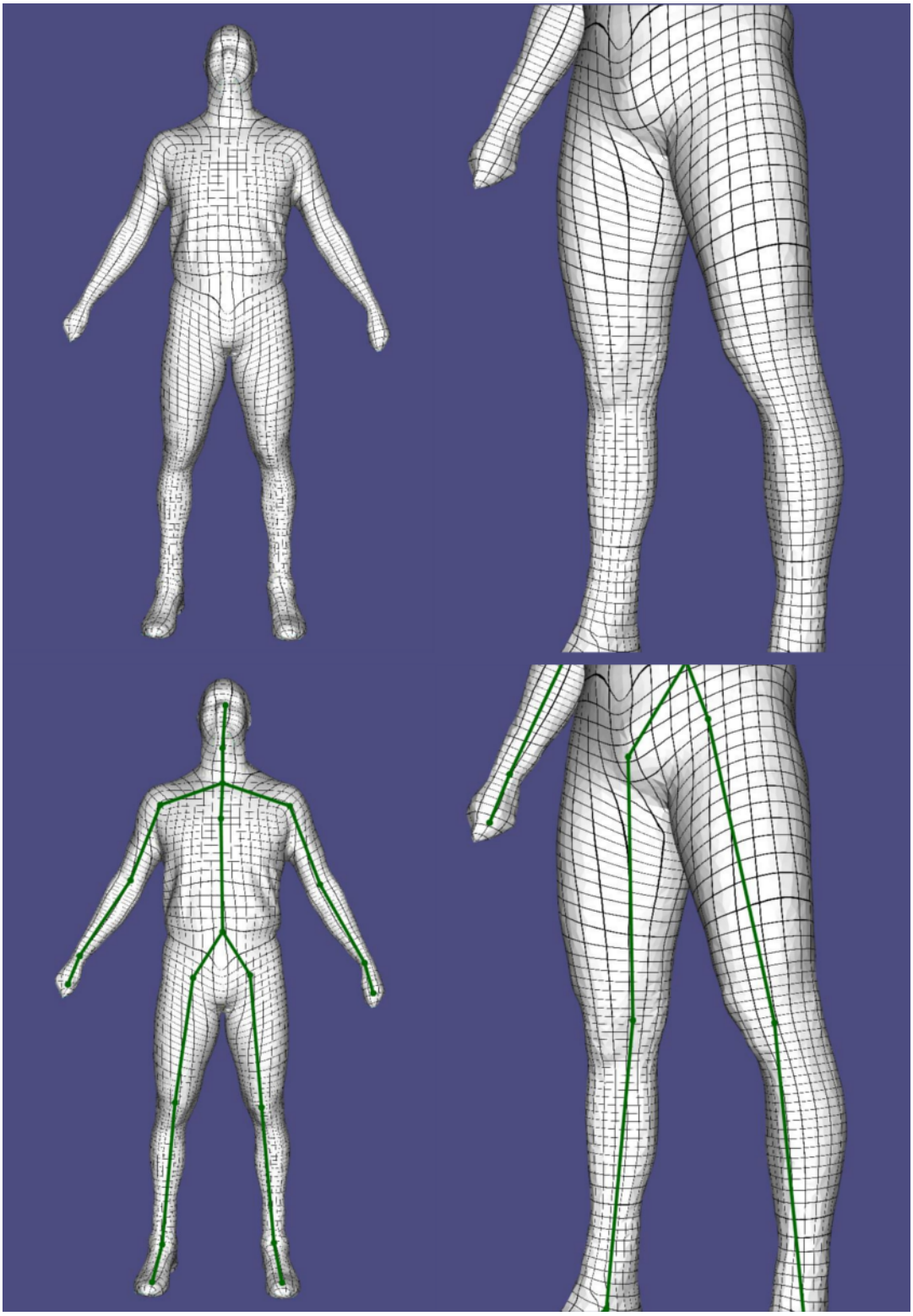

FIGURE 5.6: Showcase of the varying density on joints between *consecutive bones*. **Top:** The resulting parametrization. **Bottom:** Highlighting the higher density along the legs.

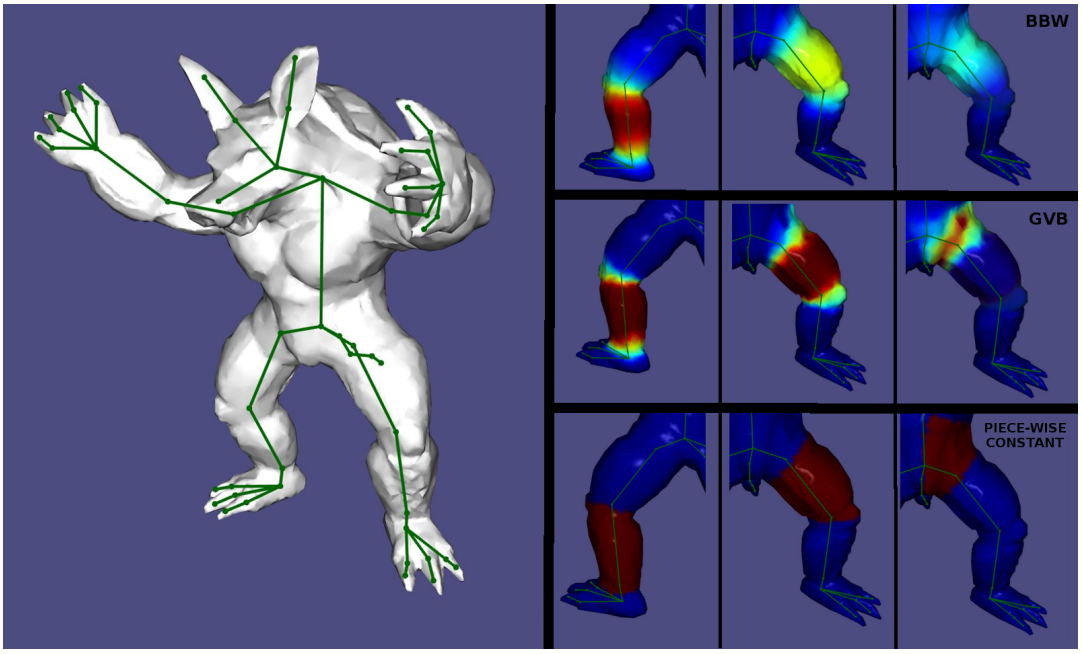


FIGURE 5.7: Three different types of skinning weights, and the rig used to compute them. **Bounded Biharmonic Weights, Geodesic Voxel Binding, and Piece-wise constant weights.**

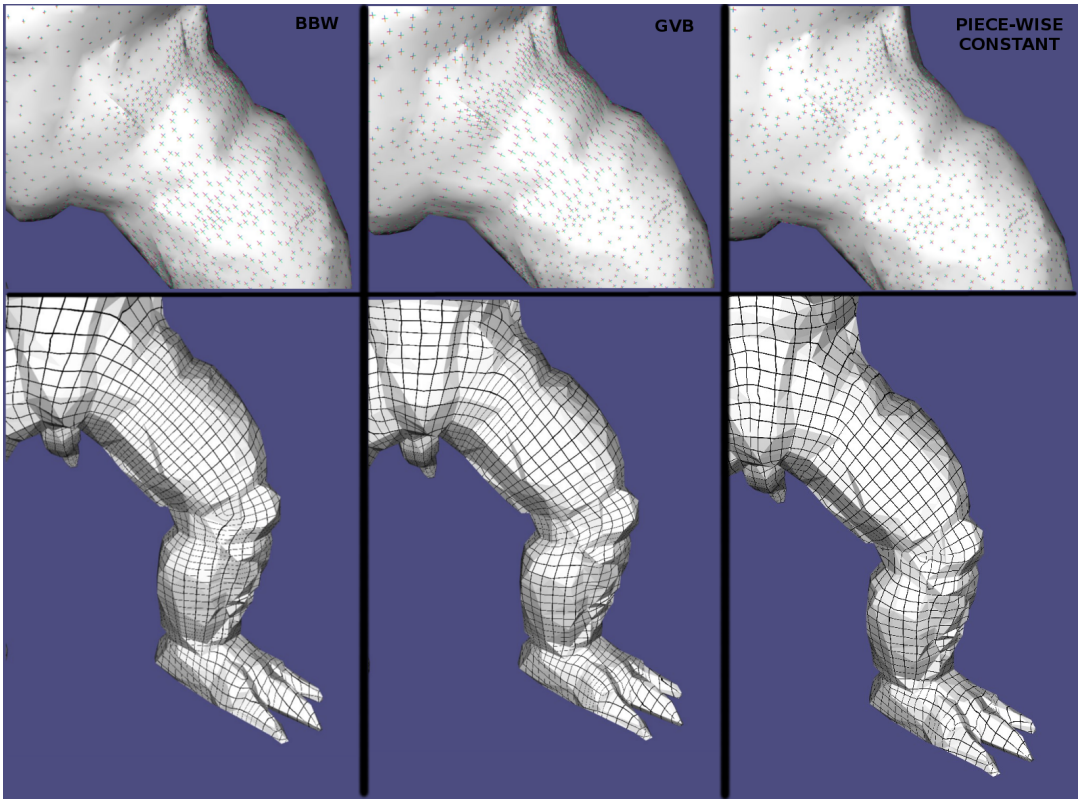


FIGURE 5.8: **Top:** The directional field computed based on the rig shown 5.7, for each of the skinning methods described. **Bottom:** The resulting quad parametrization.

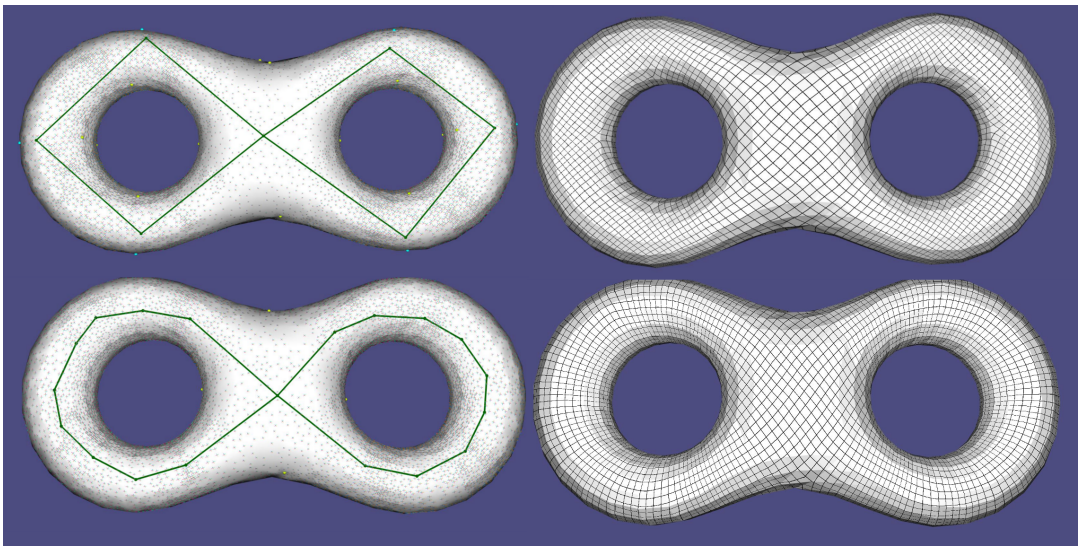

FIGURE 5.9: **Top:** Low rig resolution, with minimal bones and parametrization that barely approximate the mesh. **Bottom:** Higher rig resolution for better approximation and improved parametrization.

highlight the effect it can have. For this example, one is not necessarily objectively better than the other, but the bottom parametrization much better aligns with the overall shape of the mesh. The next comparison is shown in 5.10, indicating bad bone placement throughout the mesh, that does not approximate its topology well enough, in the sense that there are joints where there should not have been, and the bones don't properly follow the mesh. The difference in result is obvious on the right hand side of the figure.

The third discrete component of the input is a composite one: the input rest pose. This is the pose that both the mesh and rig are in when they are being used as input to the method. Needless to say, they have to be in the same rest pose, since the rig should be based on the mesh after all. In our case, given a single bone, a good rest transformation for that bone would be about the center/average of all the intended possible transformations that it is supposed to undergo. An example can be seen in 5.11, where at the top we can see an example of a good rest pose regarding the arms, while at the bottom is a worse one. An arm of that composition as in the figure has essentially an entire hemisphere as a movement space. Therefore, a good rest pose for it would be to be extended completely to its respective side (left or right). As a result, we can see on the right hand side of the figure that parametrizing with a better rest pose allows us to create the desired pose with a much better quad alignment around the arm joint. Whereas the bottom example used that pose as input instead, and the result does not align as intuitively to that joint. Even this slight deviation is proven to have significant impact on the result.

When it comes to the input triangle mesh itself, assuming a generally good triangulation, an important factor is the polygon count. Our method's results are not directly affected by the polygon count. However, in principle, significantly low triangle resolution implies less detail and therefore less information extracted from the mesh. This is apparent in the top example of figure 5.12, and specifically the hands of the model. Since the polygon count was not high enough, the directional field was not smooth enough around certain areas. Moving to the higher polygon count examples below, we can see improvement in the area of the hand. In the cases of areas where

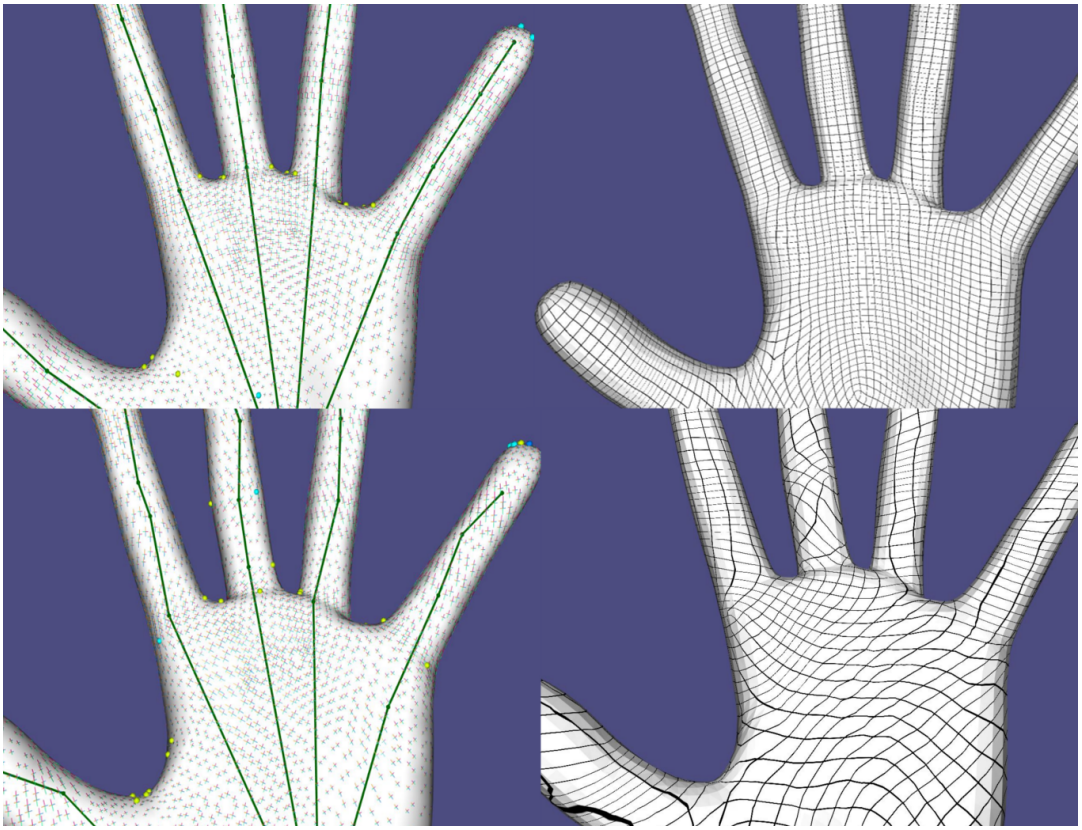

FIGURE 5.10: **Top:** Bad rig placement/transformations, not capturing the topology well enough. **Bottom:** Quality rig that results in a much better parametrization.

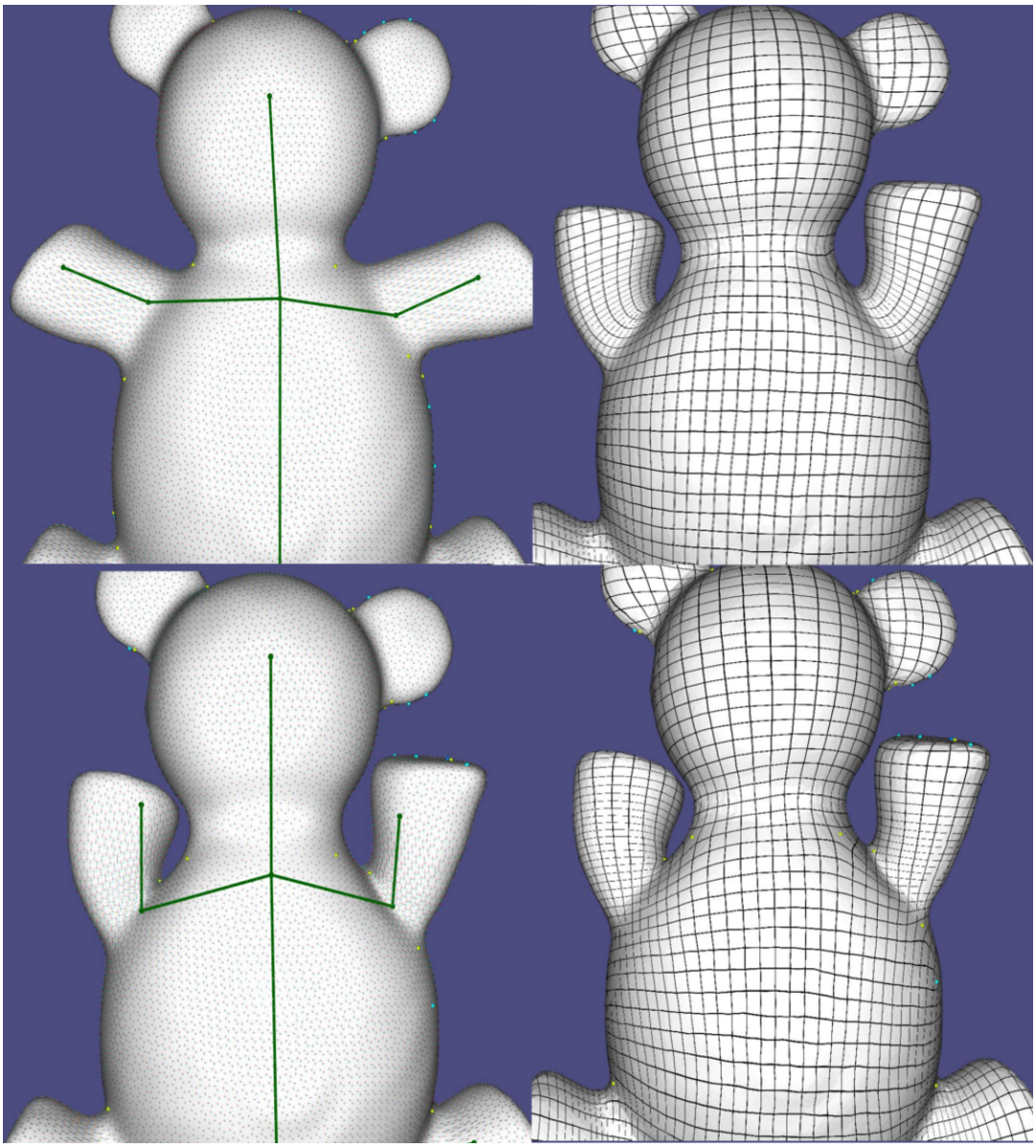

FIGURE 5.11: Different rest pose example. **Top:** Field created with a proper rest pose, and then deformed to obtain the configuration on the right. **Bottom:** Field created with a less good rest pose, and then parametrized. Notice the difference in quad alignment on the shoulders.

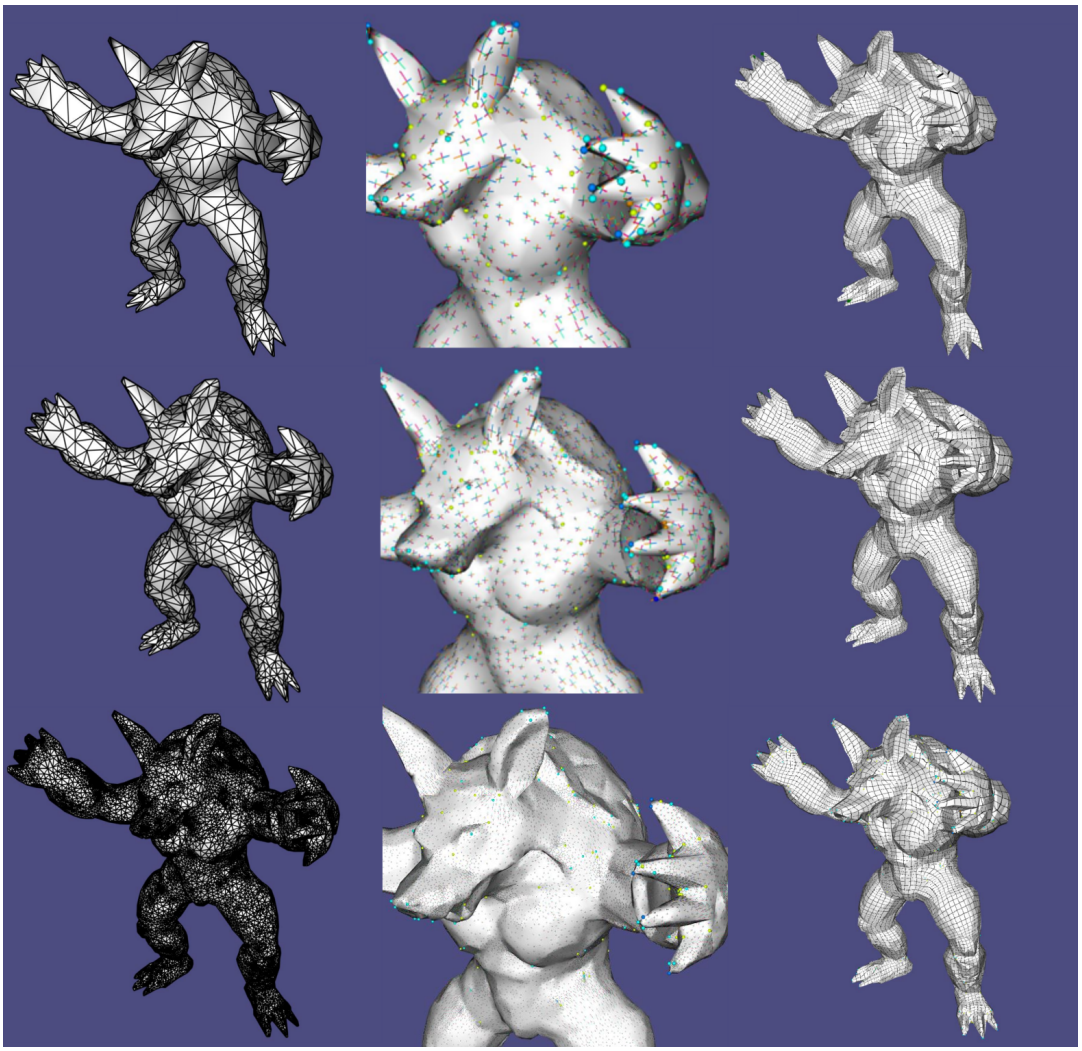

FIGURE 5.12: Three different triangle resolutions for input meshes, and the results produced. **Middle:** Low resolution. **Bottom:** Medium resolution. **Top:** High resolution.

there was enough information though, like the upper thigh, we can see that all three examples managed to capture both the alignment, and the density needed. Moreover, a mesh has the factor of size, which is being taken into account during parametrization, and is used a scaling for the quads. Throughout this section the meshes used for the example have been of varying size.

The examples we have seen so far have been, for the most part, in three categories: meshes that a) are suitable for animation, b) have intuitive and suitable topology for a rig, and c) do not have very and many sharp features along the surface of the mesh. An example doesn't necessarily exclusively belong to a single of these categories. Now we are going to examine certain cases, where our method is not very suitable by principle. One case is shown in figure 5.13, which is neither suitable for animation, nor does it have clear topology for a rig. Cases like this this block a lot of our method's benefits. Trying to go for a normal rig here can provide arbitrary results, as we can see in the figure. It would possibly be better if we tried to guide the field using a non-realistic rig, to obtain a better resulting quad mesh. Another example can be seen in 5.14, which is not suitable for animation, and it has a lot of sharp features. Our method cannot capture these features and does not enforce alignment to such

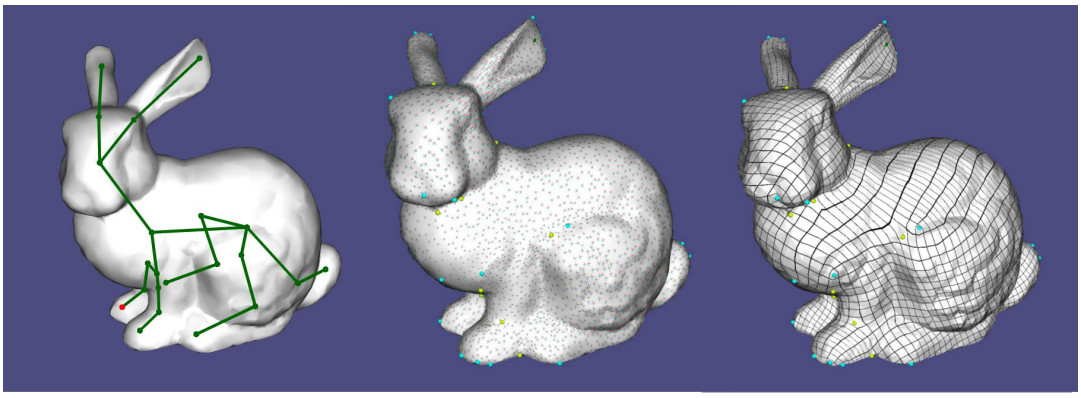

FIGURE 5.13: Example process on a non-suitable mesh, which has neither clear features that can guide a skeleton, nor is it round enough on the surface.

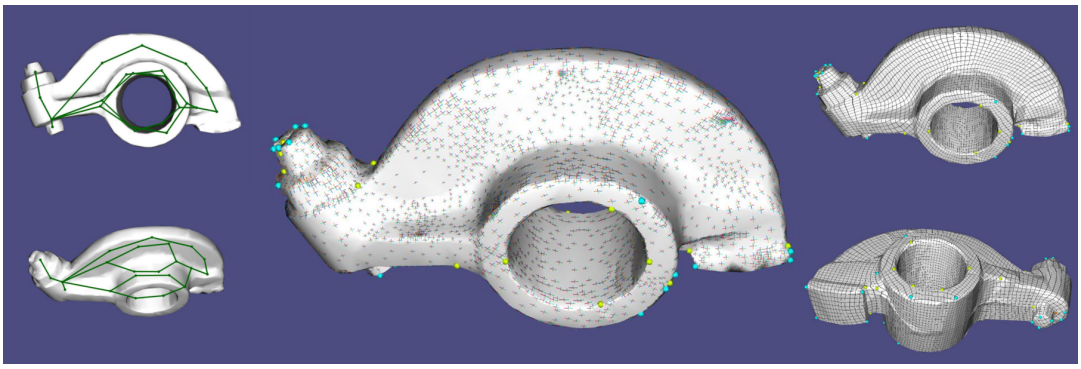

FIGURE 5.14: Example process on a non-suitable mesh, which consists of a lot of sharp features, and no features that can guide a skeleton.

sharp features, since it doesn't explicitly consider surface features.

After examining the method within its own context, we will proceed to compare it with a static meshing method, as figure 5.15 shows. This parametrization was obtained by applying curl reduction on the input directional field. The aim is to make the field as integrable as possible, which results in the parametrization approximating the input field as much as possible. This is a property we don't enforce. As we can see from the figure, the parametrization on the right has fewer singularities, as the field was specifically optimized for that. Whereas our method displays clear alignment to the mesh and higher density near the joint areas.

Lastly, figures 5.16 and 5.17 illustrate a quad mesh obtained after remeshing using our parametrization. The deformation shown in 5.17 was done before the remeshing, on the triangulated input mesh, but the parametrization used for both cases was the same.

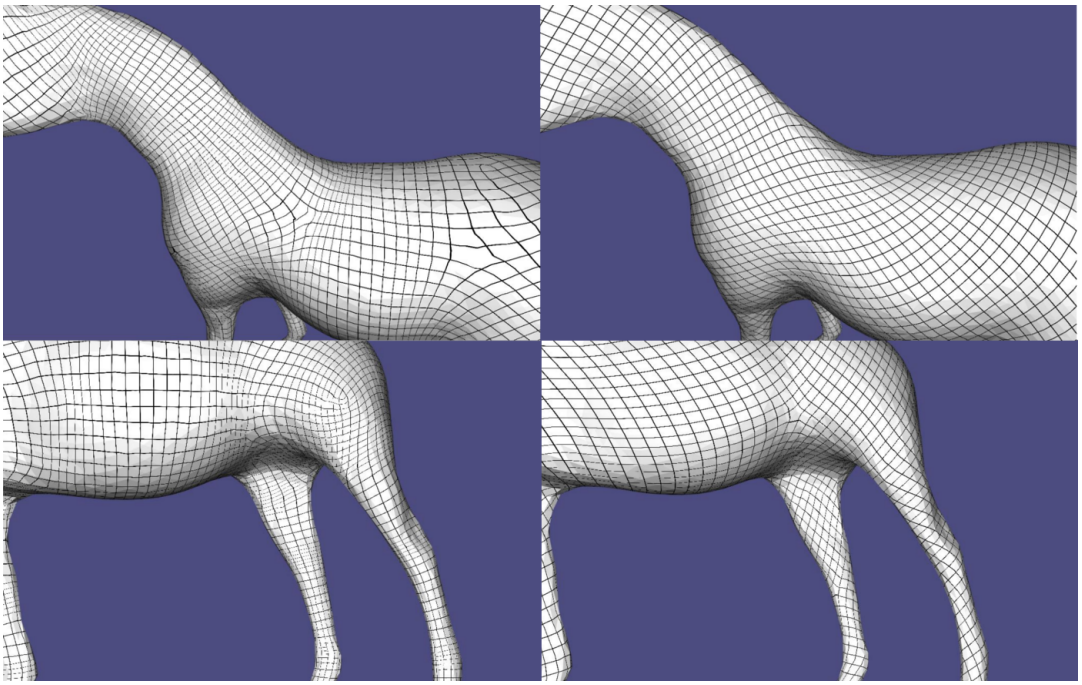

FIGURE 5.15: Comparison between our method and a static meshing method. **Left:** Our method's parametrization. **Right:** Parametrization obtained from [Vax+].

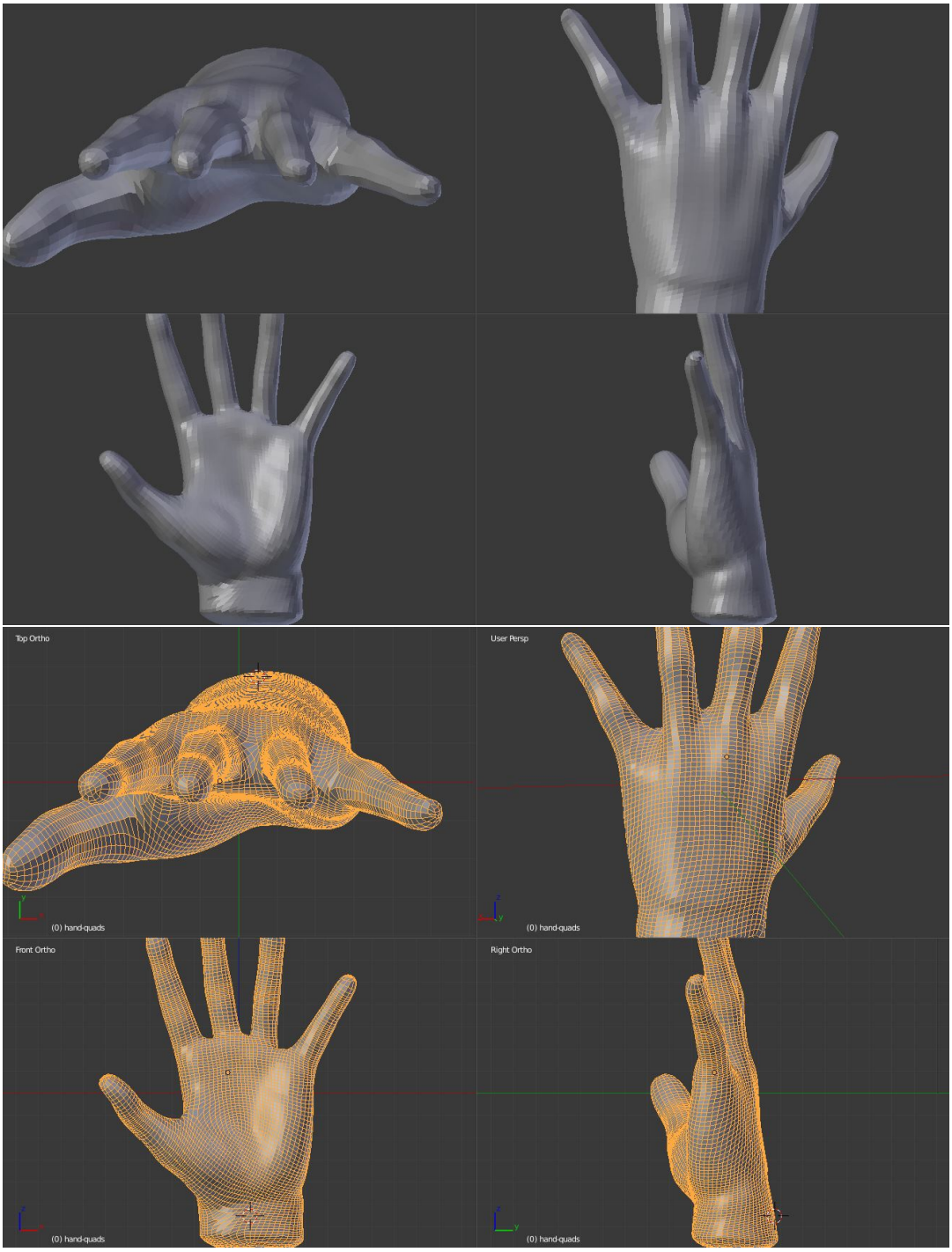


FIGURE 5.16: Quad mesh after performing remeshing on the triangulated input mesh using our parametrization. Scaled jacobian [Knu00a; Knu00b] average: 0.969; minimum: 0.024.

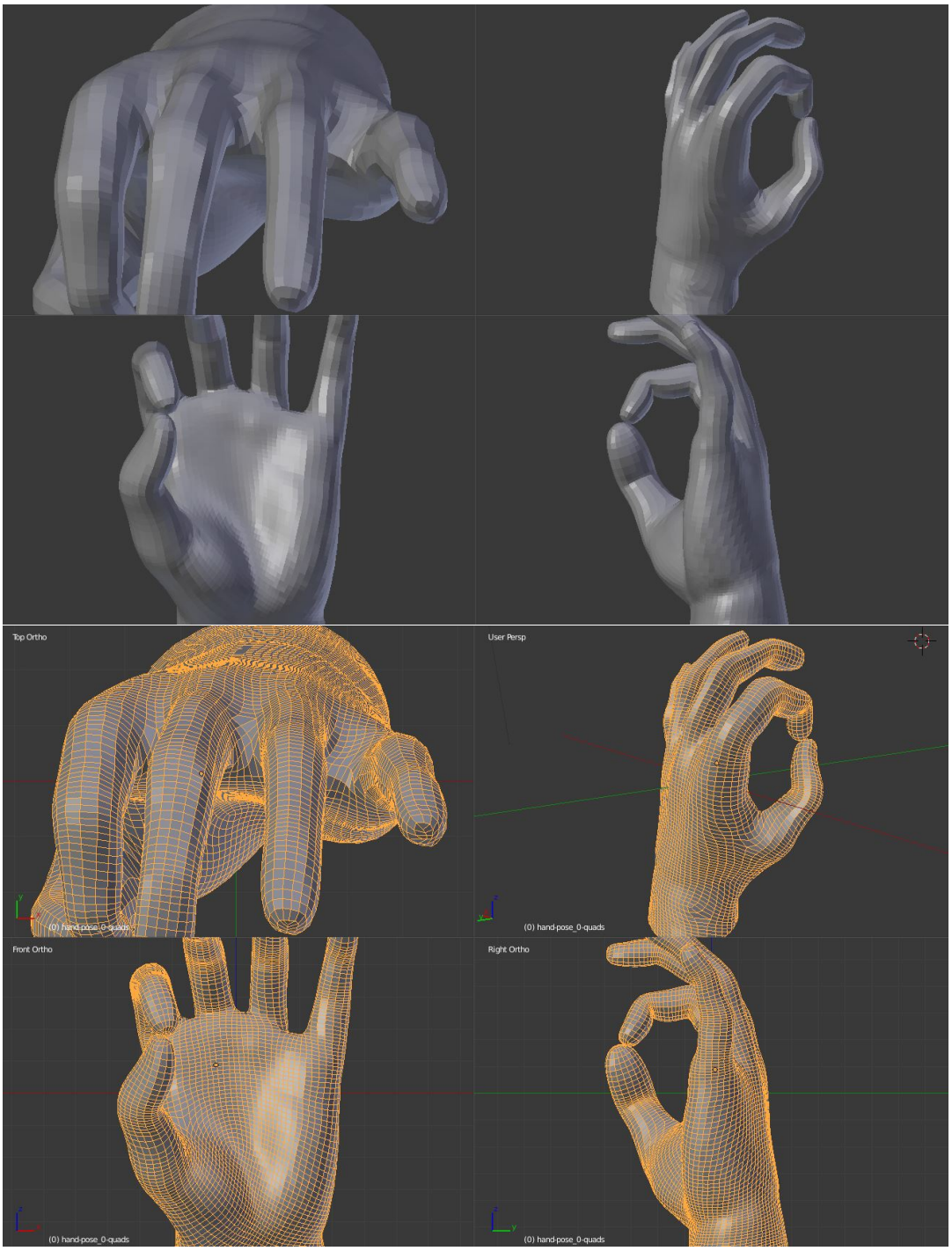


FIGURE 5.17: Quad mesh after performing remeshing on the deformed triangulated input mesh using our parametrization. Scaled jacobian average: 0.964; minimum: 0.019.

# Chapter 6

# Discussion

Both by *design*, and after evaluating the performance of our method in the previous section, certain limitations have become apparent. There is definitely room for improvement, especially in the areas where the method fails by design.

The first noticeable issue is the lack of curl optimization, a core element that helps reducing the seams on the parametrization [Dia+15]. Moreover, reducing curl can also improve the positioning of singularities, making them more natural; depending on the initial field of course. Lower curl also means more integrable directional field, and therefore a more representative quad mesh. Introducing curl reduction in our system could either be incorporated in our optimization step, or be applied as a post-process step, as described in [Dia+15].

The method additionally does not perform well on sharp features, such as sharp angles along a surface, as observed in 5.14. This can be a deal-breaker when it comes to such cases, as it is very important. Methods that design directional fields based on curvature directions usually do a better job at capturing such sharp features. It is also valid for such traits to appear on animatable meshes. A way to counter this would be to specifically adjust input rigs to try to capture sharp features themselves, by guiding them along sharp areas. Another solution could be to combine our method with already existing ways to detect sharp features and use them as constraints.

Lastly, as observed from the previous section during testing, our method can be sensitive to certain elements of its already specific input. A great example would be the skeleton that accompanies the given mesh, which was further explained in the previous section, since some rig mesh combinations will not work very well, and may need tweaking before being used as input.

# Bibliography


[Ale02] Marc Alexa. "Linear Combination of Transformations". In: *ACM Trans. Graph.* 21.3 (July 2002), pp. 380–387. ISSN: 0730-0301 (cit. on p. 3).

[Aze+17] Omri Azencot, Etienne Corman, Mirela Ben-Chen, and Maks Ovsjanikov. "Consistent Functional Cross Field Design for Mesh Quadrangulation". In: *ACM Trans. Graph.* 36.4 (July 2017), 92:1–92:13. ISSN: 0730-0301 (cit. on p. 6).

[Bom+13] David Bommes, Bruno Lévy, Nico Pietroni, Enrico Puppo, Claudio Silva, Marco Tarini, and Denis Zorin. "Quad-Mesh Generation and Processing: A Survey". In: *Comput. Graph. Forum* 32.6 (Sept. 2013), pp. 51–76. ISSN: 0167-7055 (cit. on p. 4).

[BP07] Ilya Baran and Jovan Popović. "Automatic Rigging and Animation of 3D Characters". In: *ACM Trans. Graph.* 26.3 (July 2007). ISSN: 0730-0301 (cit. on pp. 3, 10).

[BS08] Mario Botsch and Olga Sorkine. "On Linear Variational Surface Deformation Methods". In: *IEEE Transactions on Visualization and Computer Graphics* 14.1 (Jan. 2008), pp. 213–230. ISSN: 1077-2626 (cit. on p. 4).

[BZK09] David Bommes, Henrik Zimmer, and Leif Kobbelt. "Mixed-integer Quadrangulation". In: *ACM Trans. Graph.* 28.3 (July 2009), 77:1–77:10. ISSN: 0730-0301 (cit. on pp. 5, 13).

[CDS10] Keenan Crane, Mathieu Desbrun, and Peter Schröder. "Trivial Connections on Discrete Surfaces". In: *Computer Graphics Forum (SGP)* 29.5 (2010), pp. 1525–1533 (cit. on pp. 6, 11).

[DDL14] Olivier Dionne and Martin De Lasa. "Geodesic binding for degenerate character geometry using sparse voxelization". In: *IEEE transactions on visualization and computer graphics* 20.10 (2014), pp. 1367–1378 (cit. on pp. 3, 21).

[Dia+14] Olga Diamanti, Amir Vaxman, Daniele Panozzo, and Olga Sorkine-Hornung. "Designing $N$-PolyVector Fields with Complex Polynomials". In: *Computer Graphics Forum (proceedings of EUROGRAPHICS Symposium on Geometry Processing)* 33.5 (2014), pp. 1–11 (cit. on pp. 7, 12).

[Dia+15] Olga Diamanti, Amir Vaxman, Daniele Panozzo, and Olga Sorkine-Hornung. "Integrable PolyVector Fields". In: *ACM Trans. Graph.* 34.4 (July 2015), 38:1–38:12. ISSN: 0730-0301 (cit. on pp. 5, 37).

[For+07] Sven Forstmann, Jun Ohya, Artus Krohn-Grimberghe, and Ryan McDougall. "Deformation Styles for Spline-based Skeletal Animation". In: *Eurographics/SIGGRAPH Symposium on Computer Animation*. Ed. by Dimitris Metaxas and Jovan Popovic. The Eurographics Association, 2007. ISBN: 978-3-905673-44-9 (cit. on p. 3).

[IBB15] Emmanuel Iarussi, David Bommes, and Adrien Bousseau. "BendFields: Regularized Curvature Fields from Rough Concept Sketches". In: *ACM Transactions on Graphics* (2015) (cit. on p. 6).

[Jac+11] Alec Jacobson, Ilya Baran, Jovan Popović, and Olga Sorkine. "Bounded Biharmonic Weights for Real-Time Deformation". In: *ACM Transactions on Graphics (proceedings of ACM SIGGRAPH)* 30.4 (2011), 78:1–78:8 (cit. on pp. 3, 21).

[Jac+12] Alec Jacobson, Ilya Baran, Ladislav Kavan, Jovan Popović, and Olga Sorkine. "Fast Automatic Skinning Transformations". In: *ACM Transactions on Graphics (proceedings of ACM SIGGRAPH)* 31.4 (2012), 77:1–77:10 (cit. on p. 4).

[Jac+14] Alec Jacobson, Zhigang Deng, Ladislav Kavan, and JP Lewis. "Skinning: Real-time Shape Deformation". In: *ACM SIGGRAPH 2014 Courses*. 2014 (cit. on p. 3).

[Jak+15] Wenzel Jakob, Marco Tarini, Daniele Panozzo, and Olga Sorkine-Hornung. "Instant Field-Aligned Meshes". In: *ACM Transactions on Graphics (Proceedings of SIGGRAPH ASIA)* 34.6 (Nov. 2015) (cit. on p. 5).

[JP+18] Alec Jacobson, Daniele Panozzo, et al. *libigl: A simple C++ geometry processing library*. 2018 (cit. on p. 21).

[JSW05] Tao Ju, Scott Schaefer, and Joe Warren. "Mean Value Coordinates for Closed Triangular Meshes". In: *ACM Trans. Graph.* 24.3 (July 2005), pp. 561–566. ISSN: 0730-0301 (cit. on p. 3).

[Kav+08] Ladislav Kavan, Steven Collins, Jiří Žára, and Carol O'Sullivan. "Geometric Skinning with Approximate Dual Quaternion Blending". In: *ACM Trans. Graph.* 27.4 (Nov. 2008), 105:1–105:23. ISSN: 0730-0301 (cit. on p. 3).

[KH14] YoungBeom Kim and JungHyun Han. "Bulging-free Dual Quaternion Skinning". In: *Comput. Animat. Virtual Worlds* 25.3-4 (May 2014), pp. 323–331. ISSN: 1546-4261 (cit. on p. 4).

[Kn13] Felix Knöppel, Keenan Crane, Ulrich Pinkall, and Peter Schröder. "Globally Optimal Direction Fields". In: *ACM Trans. Graph.* 32.4 (July 2013), 59:1–59:10. ISSN: 0730-0301 (cit. on p. 7).

[KNP07] Felix Kaelberer, Matthias Nieser, and Konrad Polthier. "QuadCover - Surface Parameterization using Branched Coverings". In: *Computer Graphics Forum* (2007). ISSN: 1467-8659 (cit. on p. 5).

[Knu00a] Patrick Knupp. "Achieving finite element mesh quality via optimization of the Jacobian matrix norm and associated quantities. Part I—A framework for surface mesh optimization". In: *International Journal for Numerical Methods in Engineering Int. J. Numer. Meth. Engng* 48 (May 2000), pp. 401–420 (cit. on p. 35).

[Knu00b] Patrick Knupp. "Achieving finite element mesh quality via optimization of the Jacobian matrix norm and associated quantities. Part II - A framework for volume mesh optimization and the condition number of the Jacobian matrix". In: *International Journal for Numerical Methods in Engineering* 48 (July 2000) (cit. on p. 35).

[Mar+13] Giorgio Marcias, Nico Pietroni, Daniele Panozzo, Enrico Puppo, and Olga Sorkine-Hornung. "Animation-Aware Quadrangulation". In: *Computer Graphics Forum (proceedings of EUROGRAPHICS/ACM SIGGRAPH Symposium on Geometry Processing)* 32.5 (2013), pp. 167–175 (cit. on p. 6).

[Mar+15] Giorgio Marcias, Kenshi Takayama, Nico Pietroni, Daniele Panozzo, Olga Sorkine, Enrico Puppo, and Paolo Cignoni. "Data-Driven Interactive Quadrangulation". In: *ACM Trans. on Graphics - Siggraph 2015* 34.65 (2015) (cit. on p. 5).

[MH16] Min Meng and Ying He. "Consistent Quadrangulation for Shape Collections via Feature Line Co-extraction". In: *Comput. Aided Des.* 70.C (Jan. 2016), pp. 78–88. ISSN: 0010-4485 (cit. on p. 6).

[MTLT88] N. Magnenat-Thalmann, A. Laperrire, and D. Thalmann. "Joint-Dependent Local Deformations for Hand Animation and Object Grasping". In: *Proceedings of Graphics Interface '88.* GI '88. Edmonton, Alberta, Canada: Canadian Man-Computer Communications Society, 1988, pp. 26–33 (cit. on pp. 3, 9).

[SA07] Olga Sorkine and Marc Alexa. "As-rigid-as-possible Surface Modeling". In: *Proceedings of the Fifth Eurographics Symposium on Geometry Processing.* SGP '07. Barcelona, Spain: Eurographics Association, 2007, pp. 109–116. ISBN: 978-3-905673-46-3 (cit. on p. 4).

[Vax+] Amir Vaxman et al. *Directional: A library for Directional Field Synthesis, Design, and Processing* (cit. on pp. 21, 34).

[Vax+16] 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). ISSN: 1467-8659 (cit. on p. 6).

[Yao+09] Chih-Yuan Yao, Hung-Kuo Chu, Tao Ju, and Tong-Yee Lee. "Compatible quadrangulation by sketching". In: *Journal of Visualization and Computer Animation* 20 (2009), pp. 101–109 (cit. on p. 5).

[YSZ06] Xiaosong Yang, Arun Somasekharan, and Jian J. Zhang. "Curve Skeleton Skinning for Human and Creature Characters: Research Articles". In: *Comput. Animat. Virtual Worlds* 17.3-4 (July 2006), pp. 281–292. ISSN: 1546-4261 (cit. on p. 3).