Utrecht University

Master of Science

Directional Field Optimization guided by Rigging and Skinning

---

Author: **Thijs Ratsma**

Supervisor: **Dr. Amir Vaxman**

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

Department of Information and Computing Sciences
Utrecht University
The Netherlands
July 5, 2022

Abstract

We design a directional field guided by a rig and skinning weights belonging to a mesh. We optimize the directional field to align to the direction of the rig, while regulating for the smoothness and the orthogonality of the directional objects. Like this, the field is a good fit for downstream applications like quad meshing for animation. We explore how different parameters affect the performance and outcome of the algorithm. We conclude that higher quality input results in better directional fields at the cost of computation time. We propose a number of improvements for future exploration.

Contents

Chapter 1

Introduction

In the field of animation, quadrangular meshes offer several advantages over triangular meshes and are therefore more desirable to use. An existing triangle mesh may be reconstructed into a quad mesh using a quad meshing pipeline, where a directional field of the original triangle mesh is parametrized and then used to construct a new quad mesh. During this process, however, animation data such as degrees of freedom and placement of the rig joints are not taken into consideration. The resulting mesh is not properly fit to the intended movement possibilities, which leads to artifacts on sensitive areas of the model.
Several methods aim to alleviate these artifacts, either by post-processing or by taking a completely different approach for the quad meshing pipeline. Some of them are more successful than others, but nearly all of them lack a common quality: the rig is not taken into consideration, which results in a quad mesh that is not properly aligned with the rig and therefore suffers from artifacts.
We present a method that properly integrates not only the rig, but also the skinning weights of a mesh to generate a directional field, which can then be used in quadrangulation to create a quad mesh that adheres to the movement ranges made possible by the rig. Given a triangle mesh, its skinning weights and a rig, we produce an optimized directional field to fit as closely as possible to the smoothness of the surface and the alignment of the rig, while keeping the individual directional objects as orthogonal as possible. Our method only requires an existing triangle mesh, assigned skinning weights and a rig and can therefore be applied to many existing methods.
More formally, our proposal offers the following: Design a directional field on the surface of the mesh, optimized for curvature alignment, smoothness and orthogonality while taking skinning weights and rigging into account.
This thesis addresses the following topics. In chapter 2, we discuss the literature relevant to the various topics regarding our solution. In chapter 3, we provide an overview of the background required for our solution. In chapter 4, we define the methodology used in our solution. In chapter 5, we explain the solution process and show the results. We discuss these results in chapter 6, as well as possible improvements and opportunities for future work.

Chapter 2

Related Work

2.1 Skinning

Skinning methods are the part of the computer animation pipeline that controls how the mesh of the animated model deforms, according to a rig defined and controlled by the animator. Geometric skinning methods deform the model mesh using the current pose of the rig at any given time by calculating where each vertex should be. This quality makes the method generally offer high performance and makes it a suitable candidate for real-time applications. We distinguish three different types of geometric methods: linear direct, non-linear direct and variational.

2.1.1 Linear direct methods

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 direct skinning method and the first skinning method in general is (Magnenat-Thalmann et al., 1988), which introduces the Linear Blend Skinning (LBS) method. After assigning joint weights to vertices, LBS uses operators based on the angles of the joints to determine the mesh deformation of the affected vertices. Despite its age, this method is still one of the most used methods for skinning because of its computational efficiency and simple GPU implementation. However, even though this method is still widely used today, it is prone to two different types of artifacts: the elbow collapsing effect, where the mesh collapses into itself when joints are bent ~ 180 degrees, and the candy-wrapper effect, where the mesh deforms around a rotated joint. To fix these artifacts, most skinning methods had to adapt to a non-linear algorithm, which we discuss later. As an alternative to the interior rig, (Ju et al., 2005) introduces an exterior cage to deform the mesh. Both methods still require manual weight assignment, which is a tedious process. (Jacobson et al., 2014) improves upon both methods by automating the weight assignment process using Bounded Biharmonic Weights (BBW) and allowing free use of different handles (points, bones and cages) to deform 2D and 3D models. (Baran & Popović, 2007) and (Dionne & de Lasa, 2014) both describe other methods of automatic weight assignment, with the former adapting the rig to the mesh to assign automatic weights and the latter using voxel rasterization to optimize skinning weights.
Of the discussed methods, we use BBW to generate our own skinning weights. The process of doing so is described in chapter 5.

2.1.2 Non-linear direct methods

Non-linear methods were originally developed to alleviate the mesh artifacts that naturally occur when using LBS. Although non-linearity is generally best avoided, the quality of the resulting skinning weights often makes up for it. (Alexa, 2002) deviates from LBS by describing mesh deformation as a combination of transformations. By not utilizing the LBS framework, the corresponding artifacts do not occur either. (Yang et al., 2006) and (Forstmann et al., 2007) both adapt LBS, but utilize curves to develop non-linear methods that aim to get rid of these artifacts. A non-linear alternative to LBS called Dual Quaternion Skinning (DQS) is introduced in (Kavan et al., 2008). The intent of this method is to challenge the simplicity and efficiency of LBS while simultaneously addressing the artifacts it produces. This is done by expressing transformations in the form of dual quaternions rather than rotations. While this method alleviates the artifacts seen in LBS, it introduces new ones, such as bulging at joints and distorted normals. These artifacts can be manually resolved, but (Kim & Han, 2014) proposes a post-processing pipeline that removes these artifacts and significantly improves the skinning animation quality.

2.1.3 Variational methods

Variational methods differ from direct methods in that they interpret skinning as an optimization problem, in which the desired deformation is expressed as an energy function. As such, variational methods are not required to use a rig. (Botsch & Sorkine, 2008) presents various linear variational methods and compares their qualities. The problem with most of these methods is that the optimization problem is inherently non-linear and as such, these methods can at most approximate a solution. (Sorkine & Alexa, 2007) and (Jacobson et al., 2012) present non-linear methods that are more accurate, but are also slower to optimize due to their non-linearity.

2.2 Meshing

Meshing is the process of creating a mesh suited for the desired task based on an existing representation of the object, such as a 3D surface, point cloud or pre-existing mesh. Quadrangular meshes are most often used in the field of animation due to their ease of use, mesh quality and quad characteristics. In the standard meshing pipeline, a directional field is constructed from the existing object representation, followed by parametrization and finally the construction of the quadrangular mesh. (Bommes et al., 2009) and (Bommes et al., 2013) describe five aspects that can be used to assess the quality of the resulting mesh: individual element quality, orientation, alignment, global structure and semantics. In most cases, these aspects cannot be satisfied equally and depending on the application of the mesh, certain aspects may be more desirable to optimize for than others. We define and cover two categories of meshing techniques: static and dynamic. Since designing a directional field is a part of the meshing pipeline, we also review papers that solely describe directional fields and considering how they may be relevant to our problem.

2.2.1 Static methods

Static meshing methods utilize a single pose for the construction of the new mesh. This makes them suited for many different applications and as such, there are many more works discussing static methods than dynamic methods. (Kälberer et al., 2007) is a good example of an all-purpose quad meshing pipeline. It converts any given frame field into a single vector field, which is then parametrized to obtain a quad mesh. Because of its generality, however, it is not able to optimize for certain desired qualities. (Bommes et al., 2009) formulates the optimization problem as a mixed-integer problem. Meshing is executed in segmented patches of the mesh, of which the borders are defined by placement of singularities. This patch-based approach proves to be efficient, but requires proper singularity placement, either predetermined or automatic. The importance of singularity placement is further explored in (Crane et al., 2010), which proposes an intuitive algorithm that creates connections on surfaces while keeping specific singularities in mind. (Knöppel et al., 2013) expands upon this method and presents an n-directional field construction algorithm that automatically determines the optimal configuration of singularities without sacrificing performance. (Jakob et al., 2015) describes a quad meshing algorithm that uses an N-RoSy field which is then parametrized on a local level. This method has been proven to scale extremely well linearly, making it able to effectively deal with large meshes. An alternative to N-RoSy fields is to use PolyVector fields, first introduced in (Diamanti et al., 2014). These fields are more flexible in their use than N-RoSy fields, making them more suited for quad meshing, as demonstrated in (Diamanti et al., 2015). In chapter 3.3, we show how we integrate the PolyVector algorithm into our application.

2.2.2 Dynamic methods

In contrast to static methods, dynamic meshing methods operate on mesh sequences or a moving rig. The benefit of this is that information such as movement ranges of the rig can be taken into consideration when constructing the new mesh. The difference between static and dynamic meshing methods is immediately apparent through (Yao et al., 2009). The method takes multiple meshes with rigs and approximates a general rig. Then, poly-pipes are constructed around certain bones and joints, after which parametrization takes place by partitioning the surface into patches based on the connectivity of the poly-pipes. (Marcias et al., 2013) uses a sequence of meshes to generate a uniform animation-ready mesh. Emphasis is placed on explicit introduction of singularities to achieve a level of detail close to manual artists. (Marcias et al., 2015) describes an interactive meshing tool that applies tessellation patterns learned through machine learning to user-defined patches. Quadrangulation happens in user-defined patches based on the patterns previously learned. The method described in (Meng & He, 2016) extracts feature lines from individual models in a set, creates directional constraints from these lines and then creates universal patch-based quad meshes for each model. The directional field generated is bound to the directional constraints defined earlier. (Azencot et al., 2017) creates a mapping between two models without relying on point-to-point connectivity which is then used for coherent quadrangulation, focusing on consistent cross-field design between models. Lastly, (Zhou et al., 2018) is likely the algorithm that comes closest to solving our problem. Given a set of key poses of a model, this method generates a mesh that takes possible deformations into account. The pipeline takes the orientation field and metric of each pose into account to generate a universal cross field, which is then used in parametrization.

Chapter 3

Background

In this chapter, we define the full theoretical background necessary to express our methodology later.

3.1 Overall mesh structure

A mesh consists of vertices, edges and faces, denoted as $M = (V, E, F)$. Vertices are points located in 3D-space, edges are lines connecting vertices and faces are planes bordered by edges. From these three sets of different mesh properties, we can extract data such as edge length, a list of boundary edges, the normals of the faces and so forth.

3.2 Directional fields

A vital part of the quadrangulation process is the directional field, which is a collection of directional objects, each located on a face of the triangular mesh. We establish an algorithm that balances alignment to bones, smoothness along all faces and orthogonality of the directional objects in order to obtain a cohesive directional field.
To properly represent directional objects, a base axis system is established per face. The exact orientation of the axis system is arbitrary, so long as it lies along the surface of the face. We establish one of the three edges of the face as the base vector $B_1(f)$. By using the cross product of the face normal and the edge, we can then establish the resulting vector as the base vector $B_2(f)$. The newly established axis system describes a complex space, meaning that the directional vectors of the face are represented by a complex number. To express a directional object V in complex space c, we use the following formula:

$$v_c = (V \cdot B_1) + i * (V \cdot B_2)$$

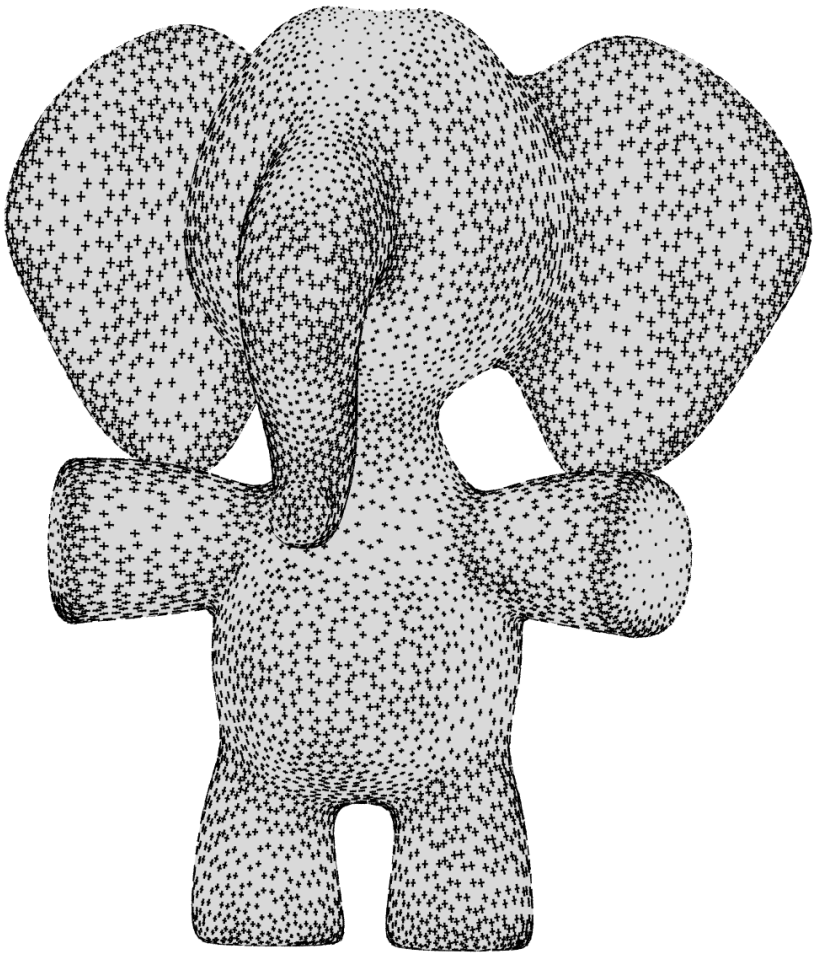


Figure 3.1: The directional field of a mesh.

3.2.1 Connections

The local basis of each face is a complex plane, in which the directional vectors are expressed as complex numbers. Since each face has its own complex space, we cannot compare two vectors from different faces in their local bases. To compare spaces and transform objects from one space to another, we need *connections*. By expressing the common edge between two neighboring faces in both complex spaces as a complex number, we can construct a transformation of the tangent space from one face to another, thereby establishing a connection. By calculating the connections between all neighbouring pairs of faces, all tangent spaces can be expressed in the same form, effectively creating one uniform tangent space. This is then used formulate the smoothness of the directional field as a measurable energy for which we can optimize.
To establish a connection, we need to take the common edge between faces f and g and express it in both complex spaces, which we call the edge representations e_f and e_g respectively. The transportation of vector u_g to face f is calculated as follows:

$$u_f = u_g * \frac{\overline{e_f}}{\overline{e_g}}$$

Where $\overline{e_f}$ and $\overline{e_g}$ are the conjugates of the respective vectors. When two vectors hold this equality, they are considered to be parallel.

3.3 PolyVector algorithm

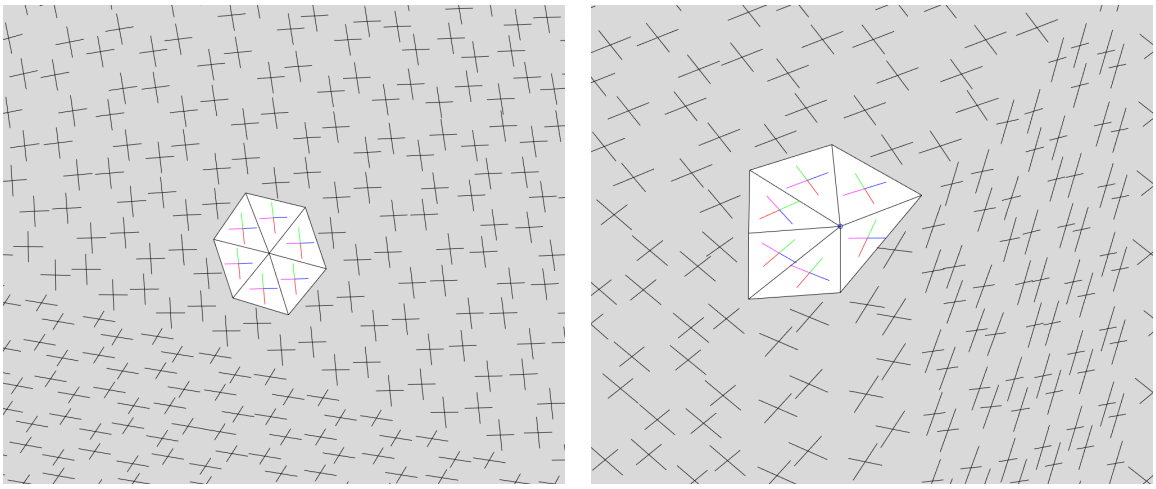
Compared to traditional directional field algorithms, the PolyVector algorithm encodes directional objects differently. Rather than interpolating each individual direction, the whole directional object is encoded in the form of a polynomial. The individual factors of the polynomial, which we call the *coefficients*, are then used as variables to optimize an energy formula for. We further discuss how this is done in chapter 4. When an optimal configuration of coefficients has been found, we can derive the complex directional vectors back from those coefficients to obtain a directional field.
The biggest advantages that the PolyVector algorithm offers are ease of use and efficiency. Most directional fields need to carefully take magnitude and order into account when transporting directional objects, but since the PolyVector algorithm encodes the whole object into a polynomial, this is no longer an issue.
The behaviour of the directional objects determines how the polynomial is formed. In our case, we use two 2-RoSy fields, meaning we have two vectors u and v that make up the directional object, along with their negatives. The whole object thus consists of the vectors $u, v, -u$ and $-v$. Writing this as a polynomial results in the following formula:

$$P(z) = (z-u)(z-v)(z+u)(z+v) = z^4 - z^2(u^2+v^2) + u^2v^2$$

The parameters of the polynomial that contain either u or v are the coefficients we interpolate along the mesh surface. In our case, we have two coefficients: $-(u^2+v^2)$ and u^2v^2. The order of the coefficient is determined by how many powers it contains. This means the first coefficient, henceforth referred to as X_2, has an order of 2 and the other coefficient, which we name X_0, has an order of 4. In our case of a two 2-RoSy field, these coefficients also have an inherent meaning: X_0 represents "...the cross of bisectors between u and v by its root set:$\{\sqrt[4]{|x_0|}exp(i\frac{k\pi}{2})|0 \leq k \leq 3\}$", while X_2 represents "...the deviation of u, v from forming a perfect cross field." (Diamanti et al., 2014)

3.4 Singularities

Another important attribute of connections is their matchings, which describe the rotation a directional object undergoes when transported by the connection. By adding the rotation deficit of all matchings around a single vertex, we can obtain its singularity index. For most vertices this sums up to zero, but a few vertices have a non-zero index, which classifies them as a singularity. Singularities indicate points in the mesh where the general meshing structure gets disrupted. An uneven directional field is also more likely to produce singularities, and as such singularities can be seen as an objective measurement of the quality of a directional field.



(a) Matching around a regular vertex (b) Matching around a singularity

Figure 3.2: Difference in matching between a regular vertex and a singularity. The directions around the regular vertex line up neatly, while after a full rotation around the singularity, a mismatch happens.

Since we are trying to match specific vectors in different directional objects to one another, all vectors are indexed in a counterclockwise (CCW) order. Since the order of the vectors does not change, we only need to find the best matching for one vector of face f with any vector of face g, which is also the best matching for the other vectors. This means that if $u_{f,m} = u_{g,n}$, then $u_{f,m+1} = u_{g,n+1}$ as well.
After an order has been established, we check for each vector in face g which one has the smallest rotation compared to a set vector in face f when transported via the edge connection. More specifically, we are looking for the principal matching, which is the best fitting matching between $-\pi$ and π. We also calculate the effort of the total rotation, which is, similar to the directional field construction, a formulation of the energy each vector requires to rotate from face g to face f.
After every connection has their principal matching assigned, we can check per vertex whether they are a singularity or not. This is done by summing up all efforts of the connections containing the vertex, as well as looking at the angle deficit of the vertex itself. If the result is a non-zero output, the vertex has an index of $\frac{k}{N}$, where k is the rotation deficit and N is the order of the directional field.

3.5 Rigging and skinning

A common technique used to animate a mesh is through use of rigging. By using a skeleton-like representation of the mesh, it is possible to control the deformation of individual vertices. The rig is defined as a hierarchy of joints and bones, in which joints connect bones to one another. By rotating the joints, the corresponding bones in the hierarchy move along with it.
The rig can deform the mesh by assigning weights to vertices to indicate the degree of deformation when a bone moves. This process is called skinning. Vertices can get weights assigned by multiple bones and in different ratios.
The quality of the resulting animation greatly depends on how accurately the rig represents the movement options of the mesh and how the mesh deforms along with the rig. Skinning is a tedious process and the slightest error in assigning weights can make the animation seem unnatural. In our case, skinning weights are generated using a Bounded Biharmonic Weight (BBW) function (Jacobson et al., 2014). By converting a surface mesh into a tetrahedral volumetric mesh after inserting control points along the rig, we can obtain skinning weights which can then be used for the original surface mesh.

3.6 Quadrangulation

Quadrangulation is the process of creating a quad mesh from an existing object. Although it is possible to directly convert a triangle mesh into a quad mesh, a more sophisticated method is to create a directional field of the triangle mesh and use this as data for the quadrangulation. Specifically, a frame field, i.e. a 4-directional orthogonal field, is perfectly suited for quadrangulation. The quality of the directional field greatly affects the resulting quad mesh. As such it is imperative that the directional field properly represents the rig and skinning weights.

Chapter 4

Methodology

The method we propose requires an existing mesh with a rig and skinning weights as input. We design a directional field according to the PolyVector algorithm. The directional field is optimized for alignment to the rig, smoothness along all faces and orthogonality of the directional objects. Each of these optimizations is formulated in a separate energy. We then minimize the total energy such that an optimal directional field that best fits our wishes is produced.
To see how changes in inputs and variables affect the directional field, we conduct multiple tests with different parameters. Using objective data such as computation time and sum of energies as well as subjective data such as visuals depicting the projected directional field, we will observe the different effects these parameters have on the resulting directional field. From this, we can then make suggestions on how the input should be prepared to obtain a directional field that is best suited for quadrangulation to a quad mesh fit for animation.

4.1 Input

Our input is a set of bones $B = \{b\}$ given as literal segments in 3D. These are joined by a set of joints $J = \{j\}$. Furthermore we are working of a triangle mesh $M = (V, E, F)$, where we are trying to compute a PolyVector field with one element per face $f \in F$.
Each bone b can be projected to a vector $v_{b,f}$ per face $f \in F$ which is computed as follows:

$$v_{b,f} = \frac{\hat{b}_f - \left(\hat{b}_f \cdot \hat{n}_f\right) \hat{n}_f}{\left|\hat{b}_f - \left(\hat{b}_f \cdot \hat{n}_f\right) \hat{n}_f\right|}$$

Next we solve the basic problem: compute PolyVector per face $X_{f,i}$ so that:

$$X_{f,i} = argmin(\lambda_a E_a(X) + \lambda_s E_s(X) + \lambda_o E_o(X))$$

Where a, s and o represent the alignment, smoothness and orthogonality components of the directional field. The following sections describe how we produce these energies and what components play a role in the construction of their formulae.

4.2 Alignment energy

The alignment energy describes how well the directional field aligns to the rig and skinning weights. We previously established that each bone of the rig can be projected to a vector $v_{b,f}$ per face. The challenge lies in formulating a directional object per face that best aligns with all projected bone vectors. For this, we instead first describe the directional object per face for only one projected bone vector. (Meekes & Vaxman, 2021) describes in its appendix a method for finding a PolyVector $\overline{X}$ closest to a given PolyVector X using partial constraints. We can describe this as an energy by taking the difference between X and its partially constrained relative $\overline{X}$, resulting in the following formula:

$$E_{a(f,b)}(X_f) = \left|X_f - \overline{X_f}\right|^2 = \left|X_f - A_{f,b} \cdot A_{f,b}^{-1}(X_f - r_{f,b}) - r_{f,b}\right|^2$$

Since this only describes the energy of one projected bone vector on one face, we take the sum of all projected bone vectors for all faces. We also have skinning weights per face $w_{b,f}$ and the area of the face Area_f to serve as scaling factors, since size and weight should affect the outcome of the directional field. Lastly, we design a confidence function $c_{b,f}$ that determines which faces are qualified to be aligned to. With this, we arrive at the following formula:

$$E_a(X) = \sum_{f,b} c_{f,b} w_{f,b} \text{Area}_f \left|X_f - A_{f,b} A_{f,b}^{-1}(X_f - r_{f,b}) - r_{f,b}\right|^2$$

Which can be further expanded to:

$$E_a(X) = \sum_{f,b} c_{f,b} w_{f,b} \text{Area}_f \left|(I_{2\times 2} - A_{f,b} A_{f,b}^{-1})X_f + A_{f,b} A_{f,b}^{-1} r_{f,b} - r_{f,b}\right|^2 \tag{4.1}$$

For which $A_{f,b} = \begin{pmatrix} -v_{f,b}^2 & 0 \\ -1 & -v_{f,b}^2 \end{pmatrix}$ and $r_{f,b} = \begin{pmatrix} 0 \\ -v_{f,b}^2 \end{pmatrix}$.
The alignment energy essentially describes the optimal PolyVector X_f such that the resulting directional object of each face aligns as cohesively with the collection of scaled bone vectors $v_{b,f}(c_{b,f} w_{b,f})$ as possible.

4.2.1 Confidence function

The confidence value is assigned to bone vectors $v_{b,f}$ on top of the skinning weights $w_{b,f}$. As such, the function should be constructed such that bone vectors that are favorable to be aligned to are assigned a value $c_{b,f} \in [0, 1]$. We determine how favorable each bone vector is based on two criteria. The first criterion is how perpendicular face f is to bone b, which we can simply measure by taking the dot product of the normalized face normal $\hat{n}_f$ and normalized bone vector $\hat{b}$. Faces that are more parallel to bone vectors are desired for the confidence function, which means the dot product of the bone vector and face normal should be as close to 0 as possible. As such, we can describe the perpendicularity factor $p_{b,f}$ as follows:

$$p_{b,f} = 1 - \left(\frac{\hat{b}}{|\hat{b}|} \cdot \frac{\hat{n}_f}{|\hat{n}_f|}\right)$$

The second criterion is how much $w_{b,f}$ is dominated by the bone with the maximum skinning weight assigned to it compared to the second largest skinning weight, which we call bone dominance. In our case, faces are more favorable if there are multiple bones with skinning weights of roughly equal size. As such, the bone dominance factor $w_{max,f}$ is described as:

$$w_{max,f} = 1 - (max(w_{b,f}) - max(max(w_{b,f}) \notin w_{b,f}))$$

Due to the nature of the input, both the perpendicularity factor and bone dominance factor have a value $p_{b,f} \in [0, 1]$ and $w_{max,f} \in [0, 1]$. However, to further increase the quality of our selection of favorable faces, any faces for which any of the individual factors have a value lower than 0.5 are set to 0, which means they are not included in the calculation for the alignment energy. With this, we can finish the formal description of the confidence function:

$$c_{b,f} = \begin{cases} p_{b,f} w_{max,f} & \text{if } p_{b,f} \geq 0.5 \wedge w_{max,f} \geq 0.5 \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

4.3 Smoothness energy

Smoothness was previously touched upon in chapter 3.2.1 when discussing connections. By using connections, we can formulate the difference between PolyVector coefficients across all faces as a measure of smoothness. Since the coefficients are complex numbers just like the complex representations of the directional vectors are, they can be transported along the connections in the same way. Ideally, all faces would simply get the same directional objects and coefficients, resulting in a perfectly smooth directional field. While this is possible on a simple plane mesh, the complexity of 3D meshes results in conflicting neighboring faces with different directional objects. As such, we aim to make the directional field as smooth as possible. This is done by formulating the connection as a smoothness energy and minimizing this energy along the whole mesh, which results in the following formula:

$$E_s = \sum_{m=0}^{N} \sum_{(f,g)\equiv e\in\epsilon} \mathrm{Diam}_{f,g} |x_{f,m}(e_f)^m - x_{g,m}(e_g)^m|^2 \tag{4.3}$$

With this formula, we calculate the squared sum difference of every PolyVector coefficient x_m along every connection $e_{f,g}$. In the alignment energy, we use the area of a face as a scaling factor to take the size of the mesh into consideration, but since this energy is described in connections, both faces need to be taken into consideration. $\mathrm{Diam}_{f,g}$ represents the diamond area described by the face centers of f and g and the connection edge e.

4.4 Orthogonality energy

In chapter 3.3 we explained the construction and function of the PolyVector format, and how the X_2 coefficient represents the orthogonality of the directional field objects. As such, we can simply describe the orthogonality energy as follows:

$$E_o(X) = \sum_f \mathrm{Area}_f |X_{2,f}|^2 \tag{4.4}$$

Since we are trying to form a directional field that is as orthogonal as possible, we want to optimize for X_2 to be as close to 0 as possible.

4.5 Putting the energies together

Now that we have a formal definition of all three energies, we construct the energy minimization formula as described previously:

$$X_{f,i} = argmin(\lambda_a E_a(X) + \lambda_s E_s(X) + \lambda_o E_o(X)) \tag{4.5}$$

Where λ_a, λ_s and λ_o are scaling factors for alignment, smoothness and orthogonality respectively. By default, these parameters are set to the following values:

$$\lambda_a = 0.65$$

$$\lambda_s = 0.2$$

$$\lambda_o = 0.15$$

These values were chosen with a purpose. Since the aim of this experiment is to see how variable inputs affect the resulting directional field, we wanted the majority of the directional field to be guided by the rig and the confidence function, two important components both represented properly in the alignment energy. However, we also conduct separate experiments with varying parameters to see how they influence the resulting directional field.

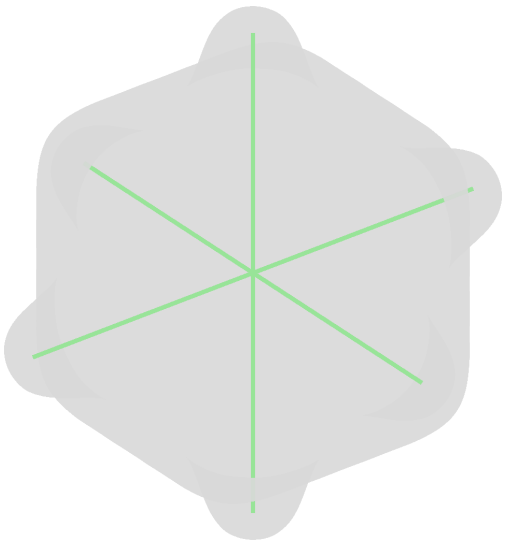
Chapter 5

Results

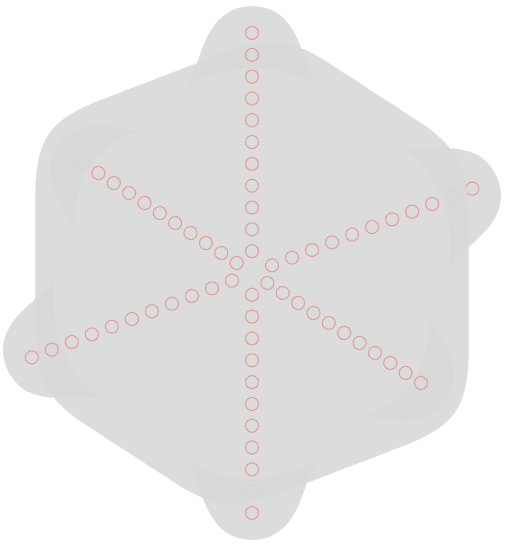
Before we take a look at how different inputs affect the directional field that results from our algorithm, we show the step-by-step process on a general case mesh, which is shown in figure 5.1. Next, we show how rig resolution, mesh resolution and extreme parameters affect the resulting directional field in figures 5.2, 5.5 and 5.7 and 5.9 respectively. Lastly, we explore some unorthodox applications of the algorithm, such as a rig designed for animation applied to a rigid mesh in figure 5.10 and 5.11 and a simplified rig applied to a mesh designed for animation in figure 5.12.
While our algorithm is mainly written in MATLAB (MATLAB, 2022), part of the pre-processing is done using libigl (Jacobson, Panozzo, et al., 2018). We represent rigs as a graph of nodes and edges in a libigl-native format. Given an input surface mesh and rig, we insert vertices along the rig bones into the surface mesh, which we call the control mesh. These vertices serve as control points, which we need in order to generate skinning weights using the Bounded Biharmonic Weights (BBW) function supported by libigl. The control mesh is then used as input in a tetrahedralization function native to the tetgen (Si, 2015) library, which creates a tetrahedral volumetric mesh. Since we used the control mesh, the control vertices are still present in this tet-mesh. The tet-mesh and rig are then used as input for the BBW-function, after which we get skinning weights for the tet-mesh. Since the tetrahedralization function only inserts new vertices into the control mesh to create a tet-mesh, we can take the indices of the surface vertices to obtain skinning weights for our original surface mesh. Lastly, the surface mesh and accompanying rig are scaled to the dimensions of a unit cube, where the scales of the three principal axes are kept consistent. This is to make sure that mesh size does not unexpectedly affect the results when comparing meshes.

5.1 General case

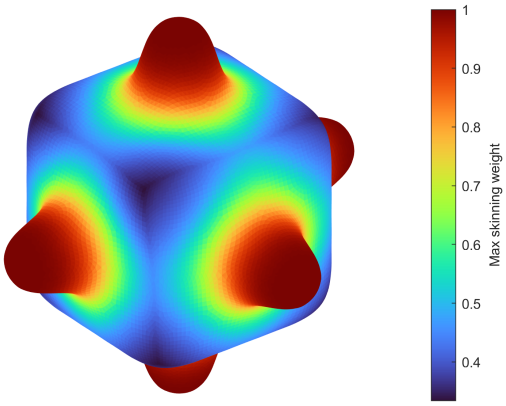
Now that all inputs are properly prepared, we move on to our main algorithm in MATLAB. The algorithm takes the input mesh, rig and skinning weights and uses the methodology described in chapter 4 to generate a directional field. This field describes directional objects containing four directions for each face of the mesh, which are optimized to align as closely as possible to the rig based on skinning weights and the confidence function, form a cohesive and smooth field that closely fit their neighbors, and are individually as orthogonal as possible. Alongside the directional field, other variables are calculated, such as the individual energies used to determine the coefficients, the number of singularities of the directional field and computation time. While the energies and singularities serve as objective measurements of quality, the visualizations of directional field and bone confidence heatmaps offer a more subjective view.



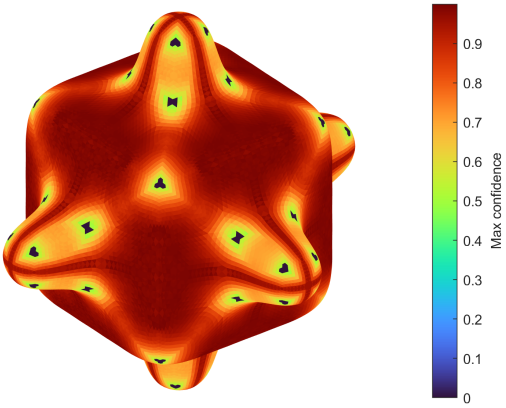
(a) Rig inside the mesh.



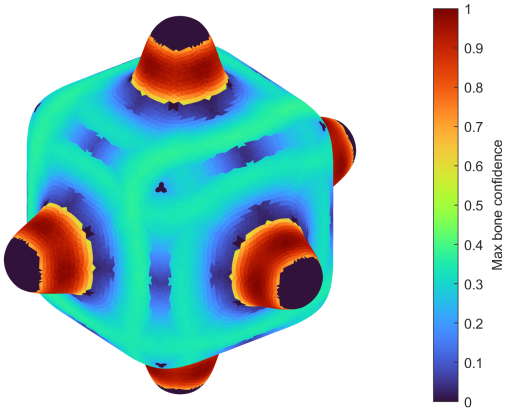
(b) Control points placed along the rig.



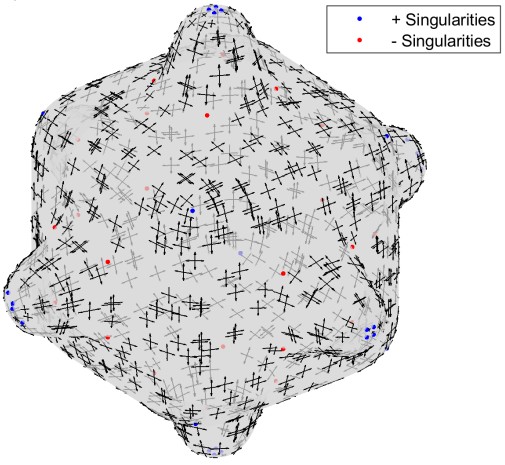
(c) Heatmap of the maximum skinning weights w_f per face.



(d) Heatmap of the maximum confidence weights c_f per face.



(e) Heatmap of maximum bone confidence weights $w_{b,f}c_{b,f}$ per face.



(f) Resulting directional field with singularities of the mesh.

Figure 5.1: Visualization of the full directional field generation pipeline: a surface mesh and rig are used as input to generate a mesh with control vertices along the rig. These are then used to create a tetrahedral volumetric mesh and generate skinning weights. Besides skinning weights, confidence values are calculated for all faces, which are combined to create a bone confidence heatmap. Lastly, energy minimization is used to calculate PolyVector coefficients for every face, which results in a directional field. Singularities are determined using matching.

Mesh	Nr of vertices / faces	Nr of rig nodes / bones	Computation time	Alignment energy	Smoothness energy	Orthogonality energy	Total energy	Nr of Singularities
Armadillo5k	2502 / 5000	80 / 79	8.3649368	1.95E-04	2.00E-03	2.60E-05	2.22E-03	160
Armadillo20k	10002 / 20000	80 / 79	31.5630117	4.53E-05	4.75E-04	9.36E-06	5.30E-04	201
Armadillo	42483 / 86482	80 / 79	130.612402	9.08E-06	9.88E-05	3.33E-06	1.11E-04	223
Bumpy-cube	19970 / 39936	7 / 6	7.6961043	7.44E-06	5.63E-05	1.50E-06	6.53E-05	56
Eight	5544 / 11092	19 / 20	4.866766	1.12E-05	1.40E-04	1.36E-05	1.65E-04	16
Eight	5544 / 11092	7 / 8	2.5267569	1.29E-05	1.65E-04	8.89E-06	1.87E-04	32
Horsers	3002 / 6000	21 / 20	2.5806152	2.93E-05	3.49E-04	1.10E-05	3.90E-04	47
Horsers (alignment dominant)	3002 / 6000	21 / 20	2.5779465	2.82E-06	5.70E-03	1.33E-07	5.70E-03	110
Horsers (smoothness dominant)	3002 / 6000	21 / 20	2.5881349	4.42E-08	2.00E-04	3.28E-08	2.00E-04	45
Horsers (orthogonality dominant)	3002 / 6000	21 / 20	2.577371	2.00E-02	1.73E-01	5.28E-05	1.93E-01	45
Hand	4780 / 9556	21 / 20	4.7393813	2.69E-05	3.48E-04	4.09E-05	4.16E-04	39
Hand (alignment dominant)	4780 / 9556	21 / 20	4.7869717	2.04E-06	4.10E-03	2.14E-07	4.10E-03	61
Hand (smoothness dominant)	4780 / 9556	21 / 20	4.8112322	8.86E-08	3.96E-04	4.29E-07	3.97E-04	38
Hand (orthogonality dominant)	4780 / 9556	21 / 20	4.6290809	7.79E-02	2.77E-01	1.19E-04	3.55E-01	39
Bunny	3485 / 6966	21 / 20	3.3245889	1.96E-04	2.10E-03	2.06E-04	2.50E-03	53
Rocker-arm2500	2500 / 5000	22 / 25	2.9253377	1.33E-04	1.60E-03	6.39E-05	1.80E-03	50
Teddy	14905 / 29806	11 / 10	7.5934827	1.20E-05	1.02E-04	1.15E-05	1.26E-04	51

Table 5.1: Table containing data of mesh and rig input as well as time, energy and singularity output for all meshes and experiments.

The visuals in figure 5.1 and data in table 5.1 show the output of our algorithm. For the sake of visual clarity, we have limited the number of directional objects in the visuals to a maximum of 1000, since meshes with a high number of faces would make it difficult to make out the individual elements otherwise. We have explicitly chosen for a simplistic mesh and rig in this case, since we can easily verify what the directional field should look like and how other aspects such as bone and confidence weights should be visualized. We can also see where complications start to occur, such as faces far removed from any bones or the tip of mesh segments, where one bone dominates the skinning weights and the faces are perpendicular to the bone.

5.2 Rig detail

First we take a look how the quality of the rig affects the directional field. While it is hard to objectively assess if one rig is of higher quality than another, we can say something out the level of detail. For instance, a low quality humanoid rig may articulate the arms, but may not have joints and bones for the fingers.
We apply two different rigs to a double torus mesh. Both describe two loops around the toruses, connecting in one node in the center of the mesh. The high detail rig has a total of 19 nodes and 20 bones (10 bones per loop), while the low detail rig has a total of 7 nodes and 8 bones (4 bones per loop).

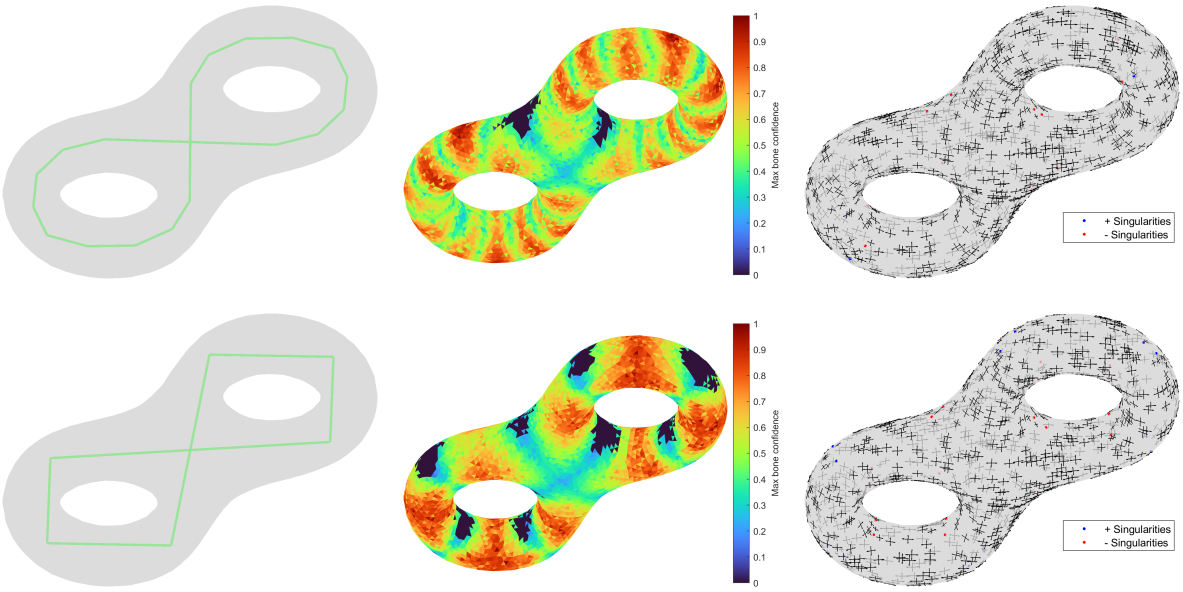


Figure 5.2: Rig, bone confidence heatmap and directional field + singularities of a mesh with a high detail rig (top) and a low detail rig (bottom).

5.3 Mesh resolution

Next, we investigate the resolution of the mesh affects the directional field. By this we mean the number of vertices and faces that make up a surface mesh. More faces means a higher resolution and therefore a surface that expresses more detail, but usually also comes at the cost of longer computation times. Less faces means a lower resolution and quicker computation times at the cost of loss of detail of the surface mesh.
The original surface mesh, *armadillo*, contains 43,243 vertices and 86,482 faces. Using MeshLab (Cignoni et al., 2008), we used a Quatric Edge Collapse Decimation function to simplify the full resolution mesh to a medium resolution mesh of 20,000 faces and a low resolution mesh of 5,000 faces. Then, using the same rig for all three meshes, we calculated skinning weights for all three meshes using the aforementioned BBW pipeline and calculated three different directional fields using our algorithm.

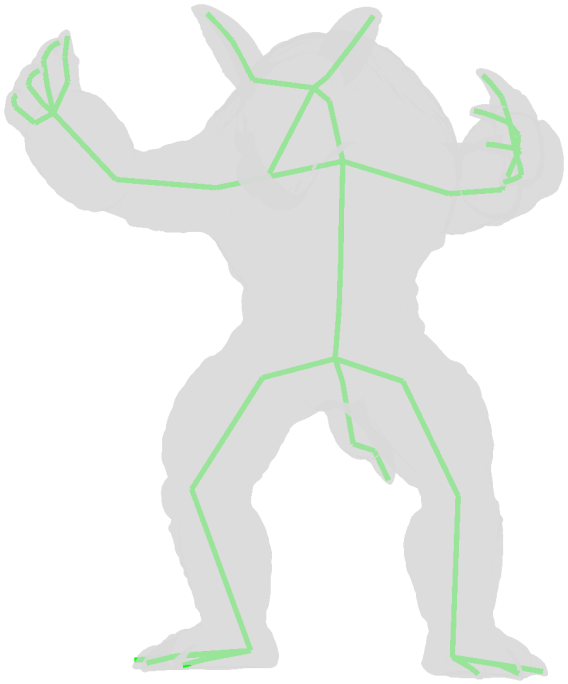
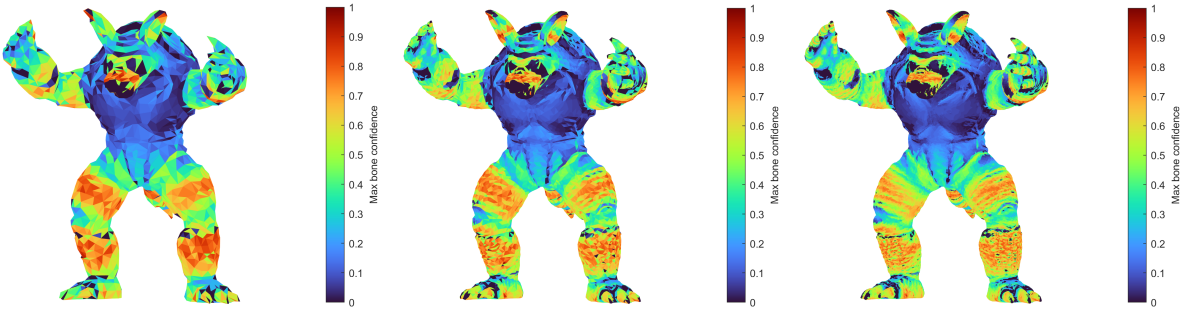
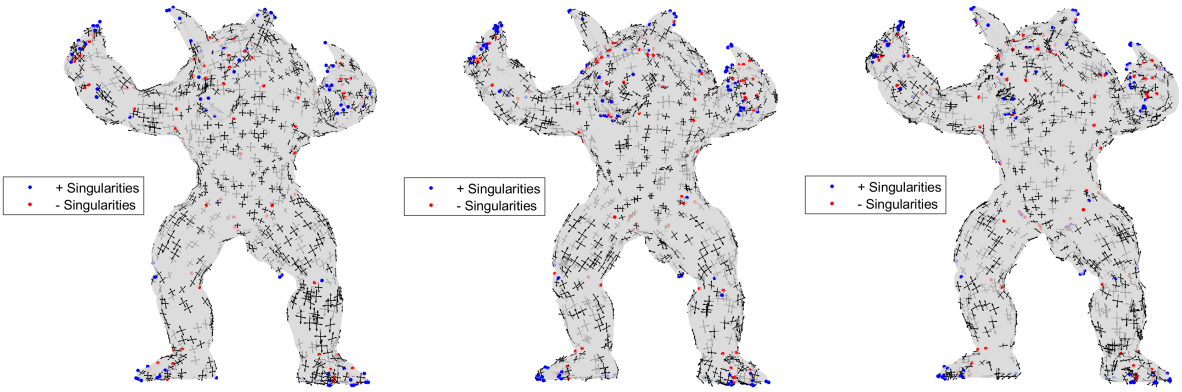


Figure 5.3: Rig used for a mesh with three different levels of mesh quality.



(a) 5,000 faces (b) 20,000 faces (c) 86,482 faces

Figure 5.4: Bone confidence heatmap of a mesh with three different levels of mesh quality.



(a) 5,000 faces (b) 20,000 faces (c) 86,482 faces

Figure 5.5: Directional field + singularities of a mesh with three different levels of mesh quality.

5.4 Parameters

For this variable, we use two meshes with corresponding rigs and skinning weights. Rather than change anything about the input files, we instead tweak the parameters of the energy minimization formula to see how they individually affect the resulting directional field.
We start with the default alignment, smoothness and orthogonality parameters:

$$\text{Default parameters: } \lambda_a = 0.65, \lambda_s = 0.2, \lambda_o = 0.15$$

Which we have determined to be best suited for our purpose, as discussed in chapter 4.5. We then take one of the parameters to a value of 0.999 and set the remaining two to a value of 0.0005. This means we get three additional distributions:

$$\text{Alignment dominant: } \lambda_a = 0.999, \lambda_s = 0.0005, \lambda_o = 0.0005$$

$$\text{Smoothness dominant: } \lambda_a = 0.0.0005, \lambda_s = 0.999, \lambda_o = 0.0005$$

$$\text{Orthogonality dominant: } \lambda_a = 0.0005, \lambda_s = 0.0005, \lambda_o = 0.999$$

We have also experimented with setting one value to 1.0 and the remaining two to 0.0, but found that the energy formula was unable to process this, resulting in NaN answers.

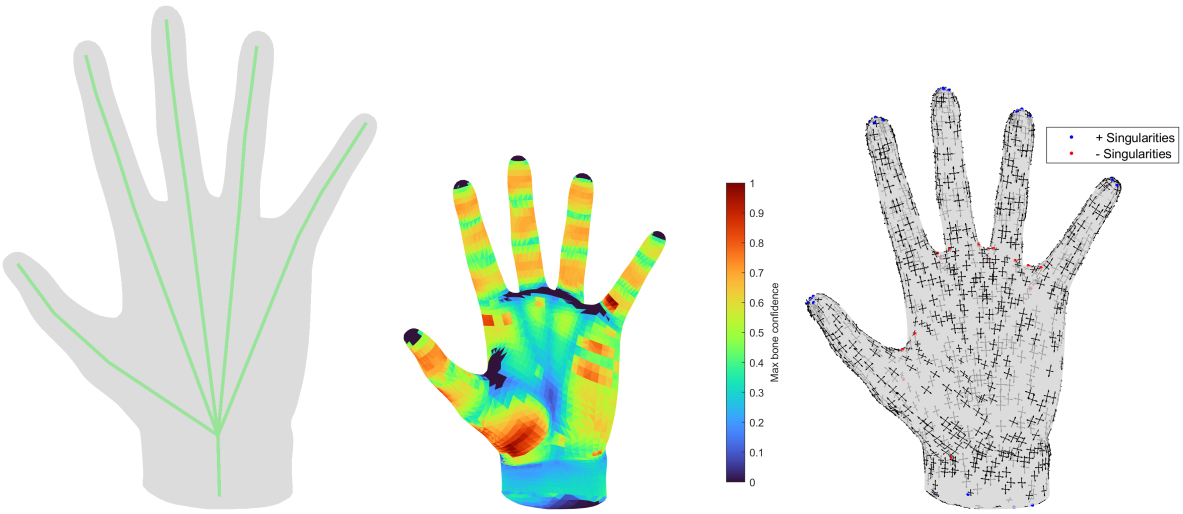
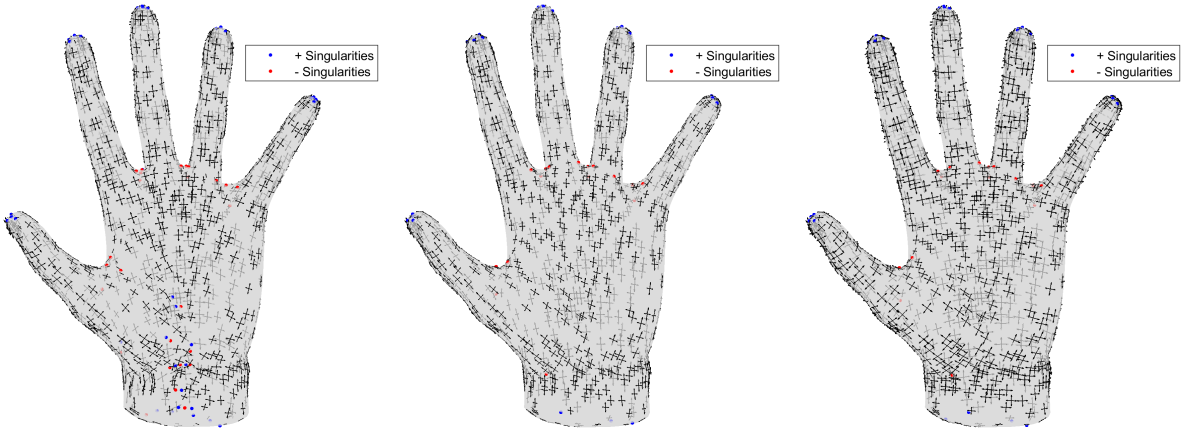


Figure 5.6: Rig, bone confidence heatmap and directional field + singularities of a mesh with the default parameters.



(a) Alignment dominant (b) Smoothness dominant (c) Orthogonality dominant

Figure 5.7: Directional field + singularities for three different configurations of parameters.

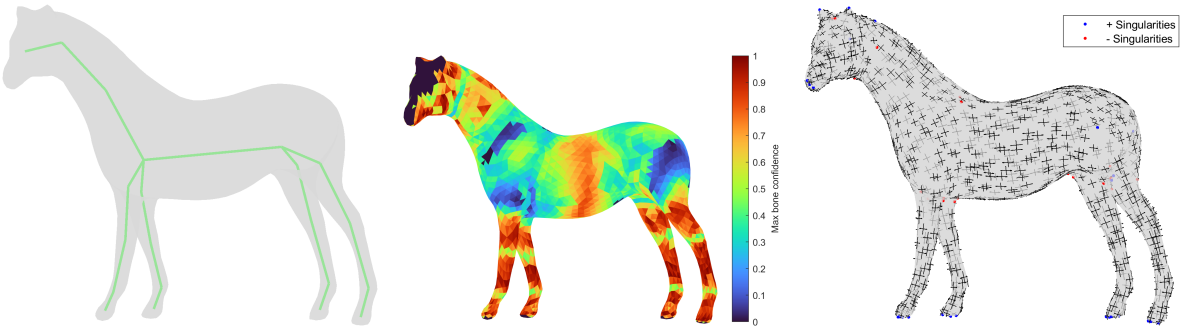
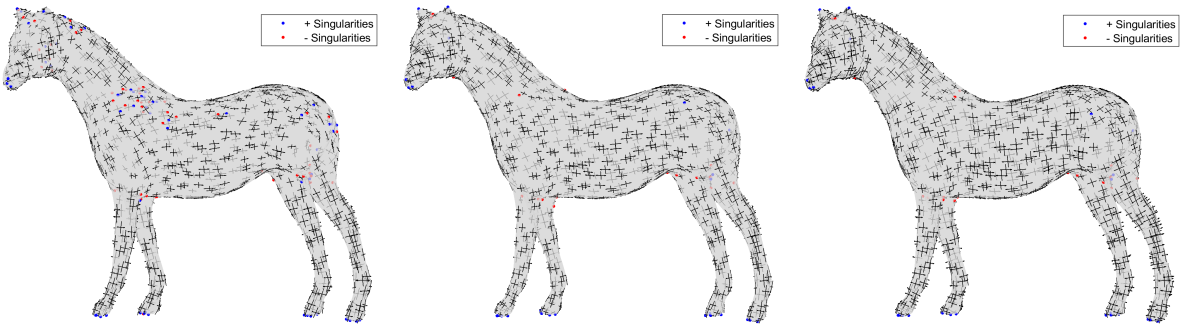


Figure 5.8: Rig, bone confidence heatmap and directional field + singularities of a mesh with the default parameters.



(a) Alignment dominant (b) Smoothness dominant (c) Orthogonality dominant

Figure 5.9: Directional field + singularities for three different configurations of parameters.

5.5 Unique cases

Lastly, we explore some unconventional inputs and see how the algorithm handles these requests. While there is not any variance in input variables, there is a certain type of mismatch between the mesh and the rig. In the first two cases, the mesh is a rigid body, unfit to be used for animation, while the accompanying rig is articulated and detailed enough to be used for animation. In the last case, we use a mesh and a rig that highlight the importance of the quality of the input. The mesh is technically fit for animation, but the lack of detail in the limbs lends itself to the low quality rig that is applied. This combination then results in large empty patches in the bone confidence heatmap, which lead to a poorly defined directional field in those areas.

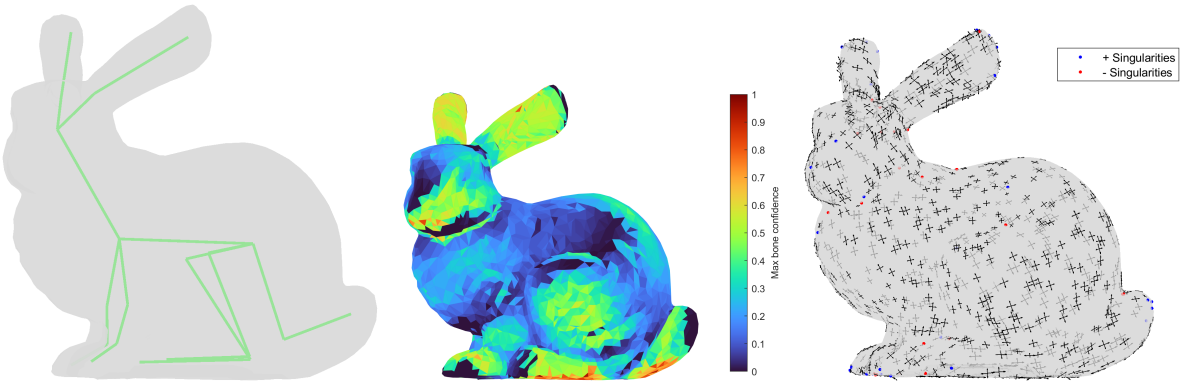


Figure 5.10: Rig, bone confidence heatmap and directional field + singularities of a rigid mesh unfit for animation.

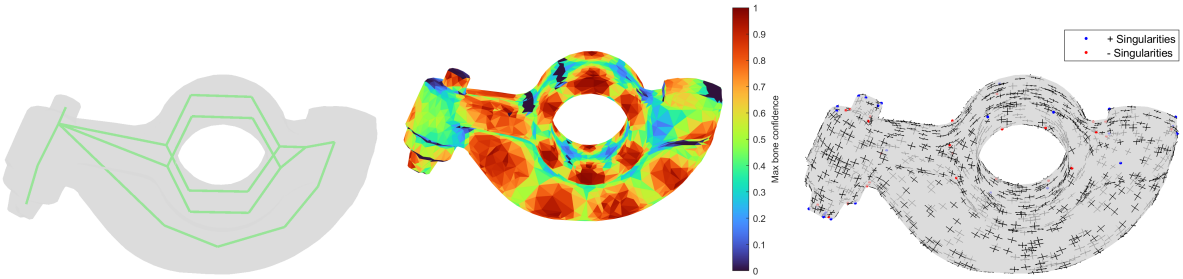


Figure 5.11: Rig, bone confidence heatmap and directional field + singularities of a rigid mesh unfit for animation.

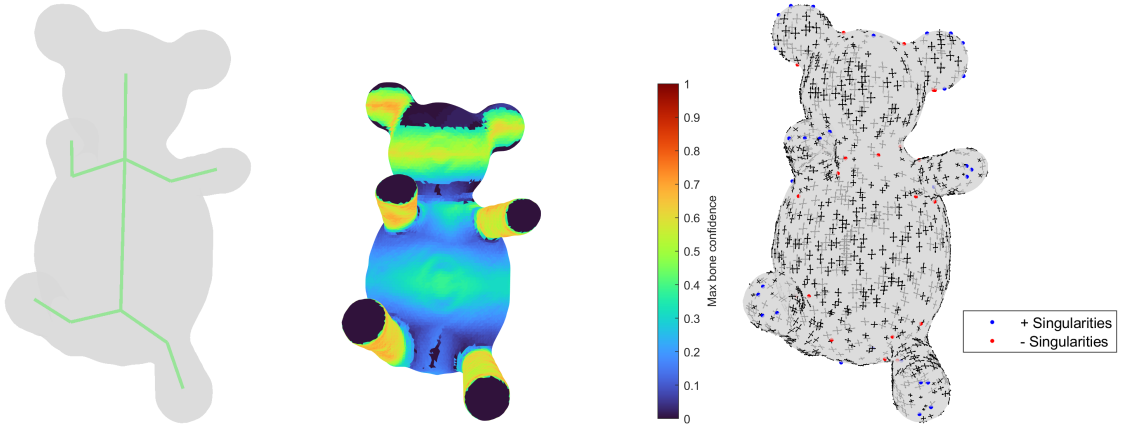


Figure 5.12: Rig, bone confidence heatmap and directional field + singularities of a mesh with a bad rig.

Chapter 6

Discussion

The quality of the input seems to affect the objective results fairly straightforward. A higher resolution rig and higher resolution mesh result in a lower total energy across both experiments, at the cost of more computation time. The only difference lies in the number of singularities: although the higher resolution rig has fewer singularities than a low resolution rig, a higher resolution mesh results in more singularities. This is likely an unavoidable consequence of increasing the number of vertices. Note that while there is an absolute increase in the number of singularities, the relative number compared to the amount of vertices / faces shows a decrease: 6.4% of vertices were singularities for the lowest resolution mesh, while only 0.52% of vertices were singularities for the full resolution mesh. If resources permit it, it definitely seems worth ensuring the input is of high quality.
One thing that immediately becomes clear and is consistent across all experiments is that meshes with large empty patches in the bone confidence heatmap are more prone to clusters of singularities forming there. These patches are formed by faces if none of the bones are parallel to the face or if the skinning weights of the faces are largely dominated by a single bone, resulting in a bone confidence value of 0. This also means that there is no inherent alignment energy and the directional object is mostly constructed through smoothness and orthogonality energies. Increasing the smoothness parameter seems to result in the lowest number of singularities for the two meshes we experimented with, although this is unnecessary for the rest of the mesh. A possible improvement could therefore be made by adapting the parameters locally depending on the bone confidence heatmap.
Another interesting observation regarding smoothness energy is that the objective results seem to favor the smoothness dominant parameter setting for both meshes. At first glance the resulting directional fields also do not seem to show any clear malfunctions. In order to really validate whether the directional field designed by the smoothness dominant parameter setting is the best fit for quadrangulation, we would have to compare the resulting quad mesh against one produced by default parameters.
Lastly, the directional fields of the rigid meshes do not seem to have any noteworthy defects. It would be interesting to see how the mesh deforms after quadrangulation and deformation of the rig.

6.1 Future work

There are a lot of opportunities to improve upon this model and pursue future work. For starters, the lack of verification through finishing the quadrangulation pipeline resulting in a quad mesh is a huge shortcoming which should be relatively easy to implement.
Furthermore, experimentation with input types could be extended to multiple skinning methods to obtain different skinning weights. Since improvements in the quality of the rig and mesh provide better directional fields, perhaps the same can be said for more advanced skinning methods.
Lastly, more types of measuring results could be added. The only energy that is currently properly visualized is the alignment energy through the bone confidence heatmap. A similar visual could be added for the orthogonality of the faces and the smoothness of the surface. More objective measurements could also be introduced in the form of other statistics.

Bibliography

Alexa, M. (2002). Linear combination of transformations. *ACM Trans. Graph.*, *21*(3), 380–387. https://doi.org/10.1145/566654.566592

Azencot, O., Corman, E., Ben-Chen, M., & Ovsjanikov, M. (2017). Consistent functional cross field design for mesh quadrangulation. *ACM Trans. Graph.*, *36*(4). https://doi.org/10.1145/3072959.3073696

Baran, I., & Popović, J. (2007). Automatic rigging and animation of 3d characters. *ACM Trans. Graph.*, *26*(3), 72–es. https://doi.org/10.1145/1276377.1276467

Bommes, D., Lévy, B., Pietroni, N., Puppo, E., Silva, C., Tarini, M., & Zorin, D. (2013). Quad-mesh generation and processing: A survey. *Computer Graphics Forum*, *32*(6), 51–76.

Bommes, D., Zimmer, H., & Kobbelt, L. (2009). Mixed-integer quadrangulation. *ACM Trans. Graph.*, *28*(3). https://doi.org/10.1145/1531326.1531383

Botsch, M., & Sorkine, O. (2008). On linear variational surface deformation methods. *IEEE Transactions on Visualization and Computer Graphics*, *14*(1), 213–230. https://doi.org/10.1109/TVCG.2007.1054

Cignoni, P., Callieri, M., Corsini, M., Dellepiane, M., Ganovelli, F., & Ranzuglia, G. (2008). MeshLab: an Open-Source Mesh Processing Tool. In V. Scarano, R. D. Chiara, & U. Erra (Eds.), *Eurographics italian chapter conference*. The Eurographics Association. https://doi.org/10.2312/LocalChapterEvents/ItalChap/ItalianChapConf2008/129-136

Crane, K., Desbrun, M., & Schröder, P. (2010). Trivial connections on discrete surfaces. *Computer Graphics Forum*, *29*(5), 1525–1533. http://search.ebscohost.com.proxy.library.uu.nl/login.aspx?direct=true&db=aph&AN=63527603&site=ehost-live

Diamanti, O., Vaxman, A., Panozzo, D., & Sorkine-Hornung, O. (2014). Designing n-polyvector fields with complex polynomials. *Computer Graphics Forum*, *33*(5), 1–11. http://search.ebscohost.com.proxy.library.uu.nl/login.aspx?direct=true&db=aph&AN=97620216&site=ehost-live

Diamanti, O., Vaxman, A., Panozzo, D., & Sorkine-Hornung, O. (2015). Integrable polyvector fields. *ACM Trans. Graph.*, *34*(4). https://doi.org/10.1145/2766906

Dionne, O., & de Lasa, M. (2014). Geodesic binding for degenerate character geometry using sparse voxelization. *IEEE Transactions on Visualization and Computer Graphics*, *20*(10), 1367–1378. https://doi.org/10.1109/TVCG.2014.2321563

Forstmann, S., Ohya, J., Krohn-Grimberghe, A., & McDougall, R. (2007). Deformation styles for spline-based skeletal animation. *Symposium on Computer Animation*, 141–150.

Jacobson, A., Baran, I., Kavan, L., Popović, J., & Sorkine, O. (2012). Fast automatic skinning transformations. *ACM Trans. Graph.*, *31*(4). https://doi.org/10.1145/2185520.2185573

Jacobson, A., Baran, I., Popović, J., & Sorkine-Hornung, O. (2014). Bounded biharmonic weights for real-time deformation. *Commun. ACM*, *57*(4), 99–106. https://doi.org/10.1145/2578850

Jacobson, A., Panozzo, D. et al. (2018). libigl: A simple C++ geometry processing library [https://libigl.github.io/].

Jakob, W., Tarini, M., Panozzo, D., & Sorkine-Hornung, O. (2015). Instant field-aligned meshes. *ACM Trans. Graph.*, *34*(6), 189–1.

Ju, T., Schaefer, S., & Warren, J. (2005). Mean value coordinates for closed triangular meshes. *ACM SIGGRAPH 2005 Papers*, 561–566. https://doi.org/10.1145/1186822.1073229

Kälberer, F., Nieser, M., & Polthier, K. (2007). Quadcover - surface parameterization using branched coverings. *Computer Graphics Forum*, *26*(3), 375–384. http://search.ebscohost.com.proxy.library.uu.nl/login.aspx?direct=true&db=aph&AN=27092453&site=ehost-live

Kavan, L., Collins, S., Žára, J., & O'Sullivan, C. (2008). Geometric skinning with approximate dual quaternion blending. *ACM Trans. Graph.*, *27*(4). https://doi.org/10.1145/1409625.1409627

Kim, Y., & Han, J. (2014). Bulging-free dual quaternion skinning. *Computer Animation and Virtual Worlds*, *25*(3-4), 321–329.

Knöppel, F., Crane, K., Pinkall, U., & Schröder, P. (2013). Globally optimal direction fields. *ACM Trans. Graph.*, *32*(4). https://doi.org/10.1145/2461912.2462005

Magnenat-Thalmann, N., Laperrire, R., & Thalmann, D. (1988). Joint-dependent local deformations for hand animation and object grasping. *In Proceedings on Graphics interface'88*.

Marcias, G., Pietroni, N., Panozzo, D., Puppo, E., & Sorkine-Hornung, O. (2013). Animation-aware quadrangulation. *Computer Graphics Forum*, *32*(5), 167–175.

Marcias, G., Takayama, K., Pietroni, N., Panozzo, D., Sorkine-Hornung, O., Puppo, E., & Cignoni, P. (2015). Data-driven interactive quadrangulation. *ACM Trans. Graph.*, *34*(4). https://doi.org/10.1145/2766964

MATLAB. (2022). *Version 9.10.0 (r2021a)*. The MathWorks Inc.

Meekes, M., & Vaxman, A. (2021). Unconventional patterns on surfaces. *ACM Trans. Graph.*, *40*(4). https://doi.org/10.1145/3450626.3459933

Meng, M., & He, Y. (2016). Consistent quadrangulation for shape collections via feature line co-extraction. *Computer-Aided Design*, *70*, 78–88.

Si, H. (2015). Tetgen, a delaunay-based quality tetrahedral mesh generator. *ACM Trans. Math. Softw.*, *41*(2). https://doi.org/10.1145/2629697

Sorkine, O., & Alexa, M. (2007). As-rigid-as-possible surface modeling. *Symposium on Geometry processing*, *4*, 109–116.

Yang, X., Somasekharan, A., & Zhang, J. J. (2006). Curve skeleton skinning for human and creature characters. *Computer Animation and Virtual Worlds*, *17*(3-4), 281–292.

Yao, C.-Y., Chu, H.-K., Ju, T., & Lee, T.-Y. (2009). Compatible quadrangulation by sketching. *Computer Animation and Virtual Worlds*, *20*(2-3), 101–109.

Zhou, J., Campen, M., Zorin, D., Tu, C., & Silva, C. T. (2018). Quadrangulation of non-rigid objects using deformation metrics. *Computer Aided Geometric Design*, *62*, 3–15. https://doi.org/https://doi.org/10.1016/j.cagd.2018.03.003