FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Fast Rotationally Symmetric Direction Fields on 3D Surfaces

Using a Globally Optimal Approach

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Medieninformatik und Visual Computing

eingereicht von

Christian Clemenz
Matrikelnummer 01226279

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Associate Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer
Mitwirkung: Dipl.-Ing. Philipp Erler

Wien, 25. Juni 2019

Christian Clemenz	Michael Wimmer

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Fast Rotationally Symmetric Direction Fields on 3D Surfaces

Using a Globally Optimal Approach

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Media Informatics and Visual Computing

by

Christian Clemenz
Registration Number 01226279

to the Faculty of Informatics

at the TU Wien

Advisor: Associate Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer
Assistance: Dipl.-Ing. Philipp Erler

Vienna, 25th June, 2019

Christian Clemenz	Michael Wimmer

Erklärung zur Verfassung der Arbeit

Christian Clemenz
Address

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 25. Juni 2019

Christian Clemenz

Danksagung

An dieser Stelle möchte ich mich zuerst herzlich bei meiner geliebten Freundin Alex bedanken, die mir in der Zeit meines Studiums und dieser Arbeit immer zur Seite stand und für mich ein großes Vorbild ist.

Meinen Eltern möchte ich dafür danken, dass sie mir die Möglichkeit gegeben haben, meinem Studium nachzugehen und meine Interessen zu finden.

Besonderer Dank gilt auch meinem Betreuer Philipp Erler, der mir immer wertvollen Input gab.

Außerdem bedanke ich mich bei InstaLOD für die Zusammenarbeit im Zuge meiner Bachelorarbeit und dafür, dass sie mir das Framework ihrer Software zur Verfügung gestellt haben.

Acknowledgements

At this point, I would like to first sincerely thank my beloved girlfriend Alex, who was always at my side during my studies and this work and who is a great role model for me.

I would like to thank my parents for giving me the opportunity to study and find my interests.

Special thanks also go to my supervisor Philipp Erler, who always gave me valuable input.

Additionally, I thank InstaLOD for the cooperation in the course of my bachelor thesis and for providing me the framework of their Software.

Kurzfassung

Wir beschreiben die Implementierung des Globally Optimal Direction Field Algorithmus von Knöppel et al. als Plug-in für ein Geometrieverarbeitungsprogramm. Das Plug-in erzeugt N-RoSy Felder, beliebigen Grades, durch das Lösen eines kleinsten Eigenwert Problems. Dafür benutzen wir einen sparsen Cholesky Solver und die Inverse Potenz Methode. Das Feld kann optional an der Hauptkrümmung, die durch die Geometrie erzeugt wird, ausgerichtet werden. Wir haben außerdem die Möglichkeit hinzugefügt, die Verbesserung, die von Pellenard et al. vorgeschlagen wurden zu benutzen. Diese Verbesserungen umfassen Constraints für bestimmte Stellen des Meshes. Ein linearer Ansatz der kleinsten Quadrate wird dann benutzt, um das überbestimmte Gleichungssystem zu lösen. Unser Hauptbeitrag besteht darin, Unklarheiten in diesen Werken zu klären, besonders in Bezug auf die Constraints.

Wir haben den Algorithmus an Meshes von unterschiedlichen gängigen Größen, die in der 3D Modellierung üblich sind, auf Laufzeit und Nutzbarkeit getestet. Obwohl der Algorithmus sehr schnell ist, verschlechtert sich die Reaktionsfähigkeit ab etwa $6 * 10^4$ Polygonen. Wir empfehlen ihn nicht auf sehr großen Meshes oder detailierten 3D Scans anzuwenden, wenn schnelle Ergebnisse notwendig sind. Anzupassen, wie stark die Ausrichtung an die Oberflächenkrümmung sein soll, ist schwierig. Zusammen mit den schnellen Ergebnissen, können die Parameter jedoch relativ schnell ausprobiert werden.

Die Ergebnisse sehen sehr gleichmäßig aus und Singularitäten liegen häufig bei geometrischen Merkmalen. Die Verwendung von Constraints hilft dabei, das Feld an Meshgrenzen, scharfen Kanten oder, falls es verzerrt ist, an Hauptkrümmungsrichtungen auszurichten. Ihre Verwendung ist sehr einfach, da die Ergebnisse vorhersehbar sind. Nur Krümmungsconstraints können manchmal schwer vorhersagbar sein und werden am besten in Verbindung mit anderen Beschränkungen verwendet.

Abstract

We demonstrate the implementation of the Globally Optimal Direction Field algorithm by Knöppel et al. as a plugin for a geometry processing software. The plugin constructs N-RoSy fields of arbitrary degree by solving a smallest eigenvalue problem. For that, we use a sparse Cholesky solver and the Inverse Power Method. The field can optionally be aligned to the principal curvature induced by the geometry. We also added the option to use the improvements proposed by Pellenard et al. These improvements contain constraints imposed on certain areas of the mesh. A linear least squares approach is then used for solving the over-constrained system. Our main contribution is to clarify ambiguities we found in these papers, especially regarding the constraints.

We tested the algorithm using meshes of different common sizes used in 3D modeling for the computation time and ease of usage. Although the algorithm is very fast the responsiveness starts to decline at about $6 * 10^4$ polygons. We recommend not to use it on huge meshes or detailed 3D scans if fast results are important. The degree of curvature alignment can be difficult to adjust. However, together with fast results, different parameter settings can be tested relatively easy.

The results look very smooth and singularities are often located at geometric features. Using constraints helps to align the field to mesh boundaries, sharp edges or, if it is warped, to the principal curvature directions. Their use is very easy because the results are predictable. Only curvature constraints can sometimes be hard to predict and are best used in conjunction with other constraints.

Contents

CHAPTER 1

Introduction

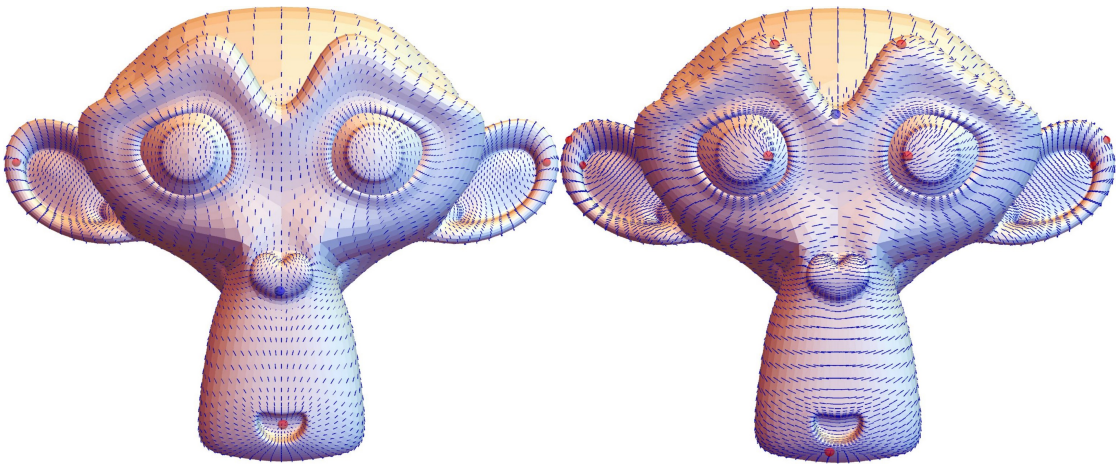

Figure 1.1: Different N-Symmetry Direction fields
Two smooth direction fields produced by our implementation. Singularities are located on triangle faces, field directions on vertices. The left one has one direction per vertex, the right one has two that are pointing in opposite directions. Singularities are highlighted by red (positive Singularities) and blue (negative Singularities) spheres.

A great amount of geometry processing and computer graphics methods rely on direction fields defined over the surface of arbitrary meshes. The specific requirements for such a field highly depend on the application. On top of that, there are multiple ways of defining a direction field, each with its own pros and cons. This leads to a great variety of algorithms that have been developed over the years.

In many cases, the main goal is to generate a smooth field that may also respect geometric features of the input mesh in terms of alignment. Since many geometry processing routines

are used by highly interactive computer-assisted design software, short computation times are needed to deliver rapid feedback to the user. In addition, it should be easy to use but also flexible enough to allow the user to modify the result according to his or her needs.

In recent years, the globally optimal direction field algorithm by Knöppel et al. [KCPS13] stood out by being able to meet the above requirements with an elegant solution. It is able to produce fields of any rotational symmetry and the result is either optimally smooth or aligned to principal curvature directions.

1.1 Problem statement

The main problem is that not on every surface a completely smooth direction field can be defined without any visible seams or jumps. They need points from where directions emanate or where they coincide. Those points are called singularities. In an automatic algorithm, the optimal number of singularities, as well as their locations, have to be computed. This looks like a difficult combinatorial problem as Knöppel et al. [KCPS13] state in their introduction. However, they also show that it can be reduced to a quadratic energy problem where the energy can be minimized and an optimal solution can be found. Another problem is which energy function to use, as some of the previously used functions tend to have local minima where the minimization could get stuck and not end up delivering a global optimum. The resulting field should also follow the natural curvature of the input mesh, which can be crucial for some applications like remeshing. All of these requirements need to be fulfilled while the number of adjustable parameters should stay minimal for it to be easy to use.

1.2 Aim of this work

The goal of this work is to implement the method by Knöppel et al. [KCPS13] as a plugin for the geometry processing software made by InstaLOD [Ins] and use the constraint extensions for curvature alignment proposed in the work of Pellenard et al. [POC+14]. The produced direction fields can then be used by further methods, especially remeshing methods. Our main contribution is to fill in the gaps and clarify ambiguities we found in those papers, particularly regarding the constraints. We will also provide a brief evaluation of the usability, the computation times and the overall look of the resulting fields. Specifically, we want to find out if the algorithm is able to be used interactively in a dedicated geometry processing software.

CHAPTER 2

Related Work

This chapter describes which work and technologies build up the basis of this thesis and will not only cover alternative algorithms but also fundamental definitions. Also, some examples of current applications for direction fields will be given afterward.

2.1 Definition of Direction Fields

As mentioned in the introduction, numerous methods for constructing direction fields have been developed and the terminology that previous literature used for them was often unclear and ambiguous up until recently. A survey by Vaxman et al. [VCD+16] not only listed the most important methods but also tried to unify the definitions and terminology as well. They refer to a *direction field* if the magnitudes of the used vectors do not matter. Otherwise, they use the term *vector field*. Of special interest to them are rotationally-symmetrical fields which are also called RoSy fields. These are used in most applications. The common types of N-RoSy fields are the N = 1,2,4 or 6, where N denotes the number of directions defined for each point on the surface. Those are also the fields that Knöppel et al. [KCPS13] produced and this work will focus on.

An important choice when constructing a direction field is the type of representation. There are currently three options according to De Goes et al. [dGDT16]: vertex-, edge- and face-based representation. The advantages each one provides differ dramatically. They even compare it to the choice of surface representation in Computer Aided Geometric Design. Face based fields have the advantage of being very simple and easy to evaluate in triangles while also having the disadvantage of being only piece-wise constant which means that their derivatives are ill-defined. It is also why computing a smooth field with user-specified constraints is not feasible according to De Goes et al [dGDT16]. The edge-based fields also tend to be very simple in construction and do not need any reference frame per simplex, as they can be represented without coordinates. However, a generalization to n-vector fields has currently not been studied. The used operators

only generate 1-direction fields. They also do not have a direction vector at vertices, making them not suitable for some use cases like vertex deformation for example. The vertex based fields are quite different from the other two types. They are continuous between vertices and therefore can be interpolated. Additionally, any n-vector field can be defined. On the other hand, they need to have a reference frame for each tangent space, although it can be arbitrarily chosen. This also leads to the need for a concept for simplicial connection, like parallel transport, which will be explained in section 3.3.

2.2 Smoothness Energy Function

An early use for direction fields can be found in the work by Hertzmann and Zorin [HZ00]. In order to generate a smooth cross field, they introduced the energy function

$$-\sum_{e_{ij}\in E} cos(4((\theta_i - \varphi_{ij}) - (\theta_j - \varphi_{ji}))),$$

where θ_i and θ_j denote the angles between one of the crossfield directions and a fixed tangent direction at vertices i and j respectively. E is the set of all edges of the mesh and φ_{ij} is the direction of the projection of the edge e_{ij} into the tangent plane, that was chosen for vertex i. This energy has then to be minimized.

Ray et al.[RVLL08] defined an alternative energy function that includes an integer variable called *period jump*. They explain that it is used to solve ambiguity in the interpolation of angles between two tangent spaces. The system to solve after a period jump p_{ij} is chosen for each dual edge and amounts to

$$\sum_{e_{ij}\in E} (\theta_j - \theta_i + \kappa_0(e_{ij}) + \frac{2\pi p_{ij}}{N})^2,$$

where $\kappa_0(e_{ij})$ resembles the angle between basis directions of the two tangent spaces adjacent to the dual edge e_{ij}, θ_i here resembles the angle between the field direction and the fixed tangent basis direction at tangent space of vertex i and N denotes the field symmetry number as described in section 2.1. Knöppel et al.[KCPS13] state that this function resembles a mixed-integer problem and therefore its optimization is NP-hard. Ray et al.[RVLL08] try to solve that by first treating the period jumps as real-valued variables before rounding them. Bommes et al.[BZK09] improved upon the rounding by a greedy rounding strategy, but both methods cannot guarantee a global optimum.

2.3 Alternative methods

An obvious main alternative to the globally optimal solution of Knöppel et al. [KCPS13] is the algorithm by Bommes et al. [BZK09] with their mixed-integer solver. But there are other methods, with more specific use cases.

For example, if the input mesh has a general symmetry to it, then the algorithm by Panozzo et al. [PLPZ12] is a good alternative. The resulting field is aligned to a

symmetry axis of the mesh. It mainly focuses on reflections as they are responsible for most symmetries of real objects. For the overall smoothness of the field, they use an energy function akin to Ray et al. [RVLL08] and minimize it with the mixed integer solver of Bommes et al.[BZK09]. The main difference to these two methods, in terms of direction field synthesis, is that Panozzo et al. [PLPZ12] formulate constraints along the symmetry line. Their algorithm is also fully automatic and the user has only one parameter for adjusting the importance of symmetry over smoothness.

If it is not necessary to automatically produce the singularities, i.e. the sources and sinks of a vector or direction field, then a very fast alternative would be the algorithm, that was introduced by Crane et al.[CDS10]. By constructing a set of basis cycles around the mesh, which are loops of dual edges, and by calculating the angle defect within such a cycle, they build a linear system and solve it for adjustment angles. After that, their algorithm starts at an arbitrary face and an arbitrary starting field direction, to traverse the other faces and compute their direction with the previously found adjustment angles. Crane et al. [CDS10] showed that their algorithm is fast enough, to achieve real-time intractability, which is certainly advantageous if singularities have to be picked manually. The advantage of this method is direct control over the location and number of singularities. On the downside, it might be tedious or even difficult for a user to create a field, that suits his or her needs with this algorithm. This is especially the case for users with little experience.

2.4 Applications

Many different types of applications have made use of direction fields. Most of them are methods for generating meshes, but there have also been different uses as well. Example applications can be found in the survey by Vaxman et al.[VCD+16] and include illustrative rendering, architectural geometry, cultural heritage, deformation, mesh segmentation, procedural modeling, urban planning and many more.

Let us now focus on some remeshing methods, as they are the predominant use case for direction fields. For example, an application that uses a crossfield for mesh generation, specifically one produced by the globally optimal direction field method of Knöppel et al. [KCPS13], can be found in the work of Pellenard et al.[POC+14]. They use the direction field for a quad dominant remeshing algorithm they call QMCF, which is a modification of the original QMorph algorithm introduced by Owen et al.[OSCS99]. Advancing from an edge front, new vertices are created by the guidance of the crossfield directions. To find the new vertex positions, the field lines are projected onto the triangle surface of the background mesh. After connecting them to the front edge, forming a quad, the encased old vertices can be removed. In order to guarantee a good quality quad mesh, they modified the crossfield algorithm as well. Pellenard et al. introduced alignment constraints on certain vertices. These constraints will be looked at in detail in Chapter 3, as they are part of the implementation in this thesis. They do not process singularities specifically and only the field directions are important, although they mention that

certain patterns arise around singularities.

Jakob et al. [JTPSH15] developed a different kind of quad remeshing method than the one mentioned above, although they are also using direction fields as guidance for edge alignment. To produce the needed field they have two different approaches they call intrinsic and extrinsic smoothness. Their extrinsic smoothness formulation is different in that it does not use any curvature based heuristics. The intrinsic approach they mention is similar to the one of Ray et al. [RVLL08], although they specifically mention that the method of Knöppel et al. [KCPS13] is also usable for the intrinsic smoothness. The direction field their method produces is vertex based. An interesting note is that this algorithm also works for meshes represented by point clouds.

CHAPTER 3

Method

In this chapter, we will discuss the theory behind the algorithm of Knöppel et al.[KCPS13], which makes up the most part of our algorithm, and the extension by Pellenard et al. [POC+14]. We start with a brief overview of the main steps of our algorithm. After that, each section will cover one of the main concepts that are part of it.

3.1 Overview

The input mesh is required to have two-manifold edges, meaning that each edge is incident to exactly two faces. Boundary edges are allowed.

Our algorithm consists of five major steps:

1. setup
2. matrix construction
3. solving for the smooth direction field
4. setting alignment constraints
5. solving for curvature alignment

In the setup, the preliminary parameters are calculated. These include the angle sum around vertices, the basis directions (Section 3.2) for each tangent space and parallel transport coefficients (Section 3.3). Afterward, the mass and energy matrices are computed (Section 3.4). This step also includes the calculation of the holonomy (Section 3.3) for each surface triangle, as they are needed for the matrices. In the third, step we solve the linear system, that is represented by the matrices for the globally optimal smooth solution. Here, we make use of the Inverse Power Method (Section 3.5). Step four is

responsible for constraining vertices, where the field should align with specific directions (Section 3.8). In the final step, we solve the constrained system from step four for the final solution (Sections 3.7 and 3.8). Steps 4 and 5 are completely optional if only the globally smooth solution is needed or one does not want to use constraints.

3.2 Complex Representation Vectors

In this algorithm, the direction field is represented by complex numbers. Because of this, we want to clarify that in any case we use the symbol i in a formula, we mean the imaginary number, except for index descriptions like v_i.

Any vector in a given tangent field can be expressed by the product of a complex number with a given basis direction. An example Knöppel et al. [KCPS13] give is that any point in the complex plane can be expressed by multiplying the real unit vector with a complex number. This gives us the following expression:

$$Z = zX,$$

where Z is a vector field described by the complex multiple z of a basis vector X. Since the goal is to produce N-RoSy fields, the complex representation also comes in handy. Any rotation by the angle ϕ on the complex plane can be accomplished by multiplying a complex number with $e^{i\phi}$. If we use this for our rotational symmetry, the field can be expressed by

$$e^{\frac{2k\pi}{N}},$$

where k =1,2,...,N-1. If those vectors are now raised to the N^{th} power, they become indistinguishable, forming an *n-vector* u. Two of those representation vectors can now be easily compared if they are within the same tangent plane, which is the main advantage of this representation according to Knöppel et al. [KCPS13]. If the individual vectors are needed again, they can be extracted by computing the N^{th} root which will be used for visualization. But first, a way of mapping one vector into the tangent space of another one is needed. This operation will be explained in the next section.

3.3 Parallel Transport and Holonomy

Imagine a curved object like in Figure 3.2 where one can define a tangent vector at each point of the surface. If one of these vectors is now to be moved along the surface from one point to another it either stops being tangential or, if it is forced to stay tangential, it changes direction. The latter case can be used for comparing two of our representation vectors. Only the angle between the vector that is going to be moved and a common geodesic, i.e. the shortest path between two points of a curved surface, is needed to correctly map it to the new tangent space. This process is called *parallel transport*.

In the discrete setting, an edge between two neighboring vertices resembles a common geodesic. For each edge, we can now measure the angles between it and the corresponding

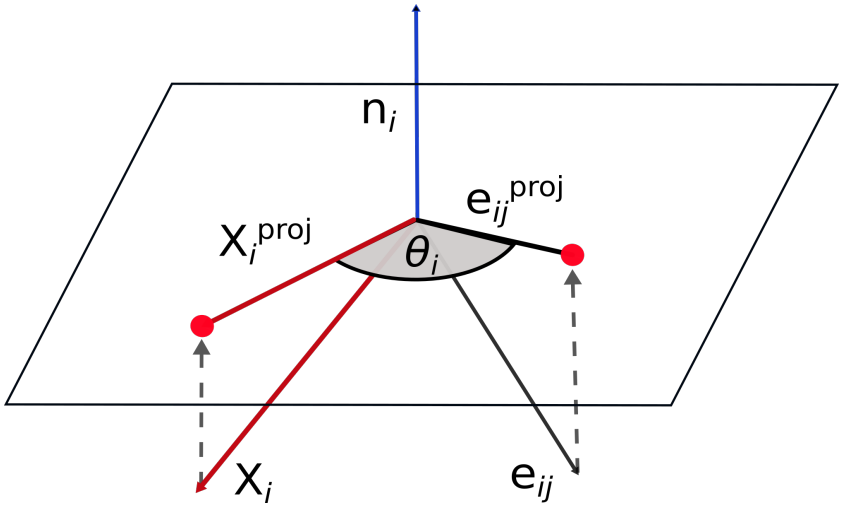

Figure 3.1: Projection onto Tangent Plane
Basis direction X_i and edge e_{ij} are projected onto the tangent plane of vertex i using the vertex normal n_i to measure the angle θ_i between the projections.

arbitrarily chosen basis vectors at the connected vertices. These angles are measured on the individual tangent spaces of the vertices, i.e. the edge and the basis vector are projected onto the tangent plane described by the vertex normal before the angle is calculated. Knöppel et al. [KCPS13] do not go into detail on how they do this but we calculate $v^{proj} = v - \frac{v*n_i}{||n_i||^2} n_i$, where n_i here denotes the vertex normal at vertex i and the vector v stands either for edge e_{ij} or the basis X_i, depending on which one of them is currently projected. A visualization of the projection can be found in Figure 3.1. The difference of these angles $\theta_{pq} = \theta_q - \theta_p$ now maps the tangent space from a vertex p to a vertex q. If we translate this definition to our representation where we use complex representation vectors, the mapping from tangent space T_p to T_q is done by

$$T_p \to T_q = e^{i\theta_{pq}} z_p X_q.$$

The difference between two vectors can now be defined as

$$|e^{i\theta_{pq}} z_p - z_q|^2.$$

Or in the case of n-vectors,

$$|r_{pq} u_p - u_q|^2,$$

where $r_{pq} = e^{in\theta_{pq}}$ and $u_p = z_p^n$.

If we trace the Parallel Transport around a closed loop over the surface, the change of the original vector can be measured. This measure is called Holonomy and is a result of the surface curvature. Back in the discrete setting, the curvature can only be observed on vertices, but according to Knöppel et al. [KCPS13] it can be pushed into the surrounding triangles by normalizing the angle sums around each vertex to be 2π. Subsequently,

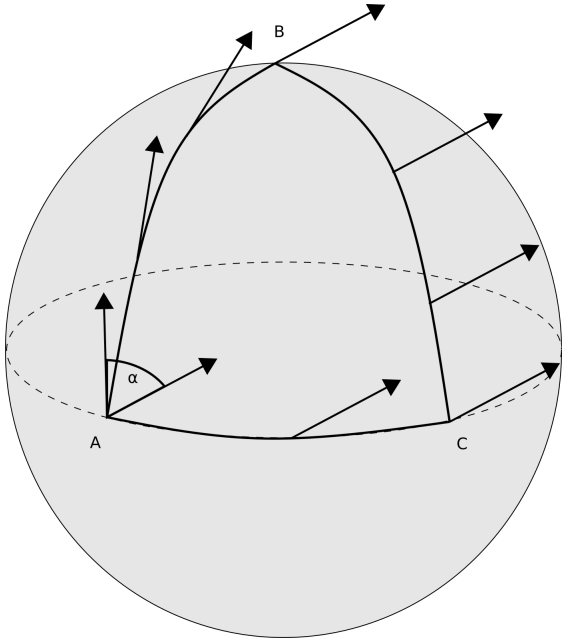

Figure 3.2: Parallel Transport over a Surface
A tangent vector at point A traveling around a closed loop over the surface. The difference in angle α is a result of the curvature.

whenever an angle, that relates to a specific vertex, for example for computing the parallel transport, is measured it has to be multiplied with the normalization factor $s_i = \frac{2\pi}{\sum_{t_{ijk} \ni i} \alpha_i}$, where the denominator is the angle sum around the vertex i. When we compute the holonomy Ω of a triangular face on the mesh, the mapping coefficients r_{ij} are used. Since the holonomy is the rotation around the closed loop, i.e. following the edges of the face, it amounts to the product

$$e^{\Omega_{ijk}} = r_{ij} r_{jk} r_{ki}$$

and further to

$$\Omega_{ijk} = arg(r_{ij} r_{jk} r_{ki}),$$

where $arg(...)$ is the angle between a complex number and the real axis.

3.4 System To Solve

We now discuss the energy function that will define the field. A detailed derivation and explanation of the formulas that follow in this section can be found in the appendix of Knöppel et al. [KCPS13].

Like the methods stated in Chapter 2, the algorithm by Knöppel et al. [KCPS13] also tries to minimize an energy function to get an optimal solution. The overall smoothness

of a function can usually be evaluated by the Dirichlet energy. We can apply it to our field and get the expression

$$E_D(\psi) = \frac{1}{2}\int_M |\nabla\psi|^2 dA.$$

In geometry evaluation, the Dirichlet energy can measure the covariant derivative of a field ψ over a surface M with a scalar value. To get an optimal field this energy, therefore, needs to be minimized. On a direction field, however, where the field vectors have unit length and singularities are present, there is a problem with this type of energy. As Knöppel et al. [KCPS13] show, the Dirichlet energy is infinite at singularities and a smoothest field cannot be found. They also prove that it is finite in a discrete setting. But the result depends on the resolution around singularities and "smoother" fields end up having higher smoothness energy if the mesh is more refined around singularities. Because of this, fields can end up being less optimal numerically, even if they look more desirable. In their explanation they also show that the Dirichlet energy can be written as

$$\frac{1}{2}\int_M |\nabla a\varphi|^2 dA = \frac{1}{2}\int_M |\nabla a|^2 + a^2|\omega|^2 dA = \frac{1}{2}\langle\langle(\Delta + |\omega|^2)a, a\rangle\rangle,$$

where φ is a unit vector field, a is a re-scaling of φ so that $\psi = a\varphi$, Δ denotes the Laplace-Beltrami operator and $\langle\langle ., .\rangle\rangle$ the L_2 inner product. In this equation, ω is the rotation speed of the field φ, which indicates how fast the unit field rotates along the surface. This is important because they then state, that for a fixed unit field φ an optimal scaling $a \geq 0$ has to be found to evaluate its energy:

$$\hat{E}(\varphi) = \min_{a\geq 0, ||a||=1} \int_M |\nabla(a\varphi)|^2 dA,$$

enforcing $||a|| = 1$ to exclude the solution $a \equiv 0$.

The expression $\langle\langle(\Delta + |\omega|^2)a, a\rangle\rangle$ can be minimized by solving the eigenvalue problem

$$(\Delta + |\omega|^2)a = \lambda a$$

for the smallest eigenvalue λ. The globally optimal solution to the original problem can, therefore, be found by solving $\Delta\psi = \lambda\psi$. Knöppel et al. emphasize that in practice it is not needed to explicitly minimize the energy $\hat{E}(\varphi)$ or build the Schrödinger operator $\Delta + |\omega|^2$ and only the Laplacian Δ is needed for the final solution.

To get more control over the field the covariant derivative in the Dirichlet energy can be split up into a sum of Cauchy-Riemann derivatives:

$$\nabla\psi = \bar{\partial}\psi + \partial\psi$$

and

$$\bar{\partial}\psi := \frac{1}{2}(\nabla_z\psi + J\nabla_{JZ}\psi),\ \partial\psi := \frac{1}{2}(\nabla_z\psi - J\nabla_{JZ}\psi),$$

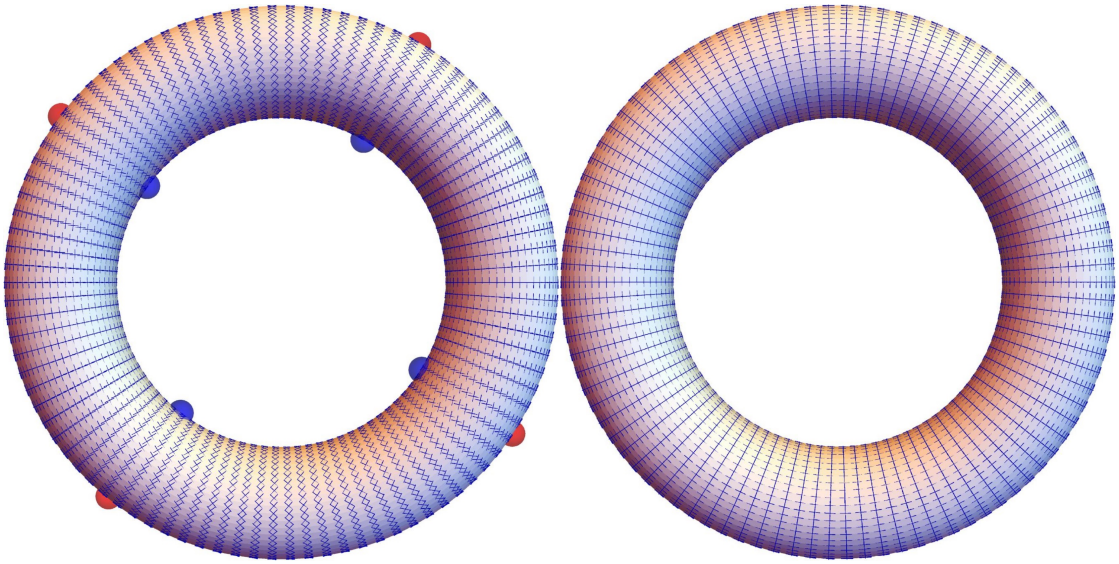

Figure 3.3: Different Energy Types
On the left, the energy parameter $s = 0$ was used, resulting in evenly spaced singularities and relatively straight lines. On the right, $s = 1$ was used which produces no singularities at all.

where Z is a vector field and J is a 90 degree rotation in the tangent space. An n-vector field can be holomorphic if $\bar{\partial}\psi = 0$ or anti-holomorphic if $\partial\psi = 0$. The Dirichlet energy $E_D(\psi)$ now consists of two terms $E_H(\psi)$ for the holomorphic part and $E_A(\psi)$ for the anti-holomorphic part, so that

$$E_D(\psi) = E_H(\psi) + E_A(\psi) = \frac{1}{2}\int_M |\bar{\partial}\psi|^2 dA + \frac{1}{2}\int_M |\partial\psi|^2 dA.$$

In practice, the equation above is used for the smoothness energy

$$E_S = (1+s)E_H + (1-s)E_A,$$

where s is a variable to shift the energy towards the holomorphic (if $s = 1$), the anti-holomorphic (if $s = -1$) or towards the standard Dirichlet energy (if $s = 0$). The variable s is also a way of controlling the geometry awareness of the algorithm, as singularities are either preferably placed in areas with high or low Gaussian curvature for $s < 0$ and $s > 0$ respectively. It also controls the number of singularities and straightness of field lines, where the holomorphic energy produces fewer singularities, the Dirichlet energy provides straighter lines, and the anti-holomorphic energy is a trade-off between the aforementioned two. An example of the difference between Dirichlet and holomorphic energy can be seen in Figure 3.3.

After discussing the theory behind the smoothness energy it is time to turn to the practical use. The field over the discrete surface is computed using the following *finite*

element method. As we have previously looked at the case of a continuous function over the whole surface domain, we now need to discretize the formulation. In other words, we need to approximate the continuous energy function by a set of basis sections Ψ. These basis sections represent the extension of the basis vectors X (see Section 3.2), located on the tangent planes, into the adjacent triangles via the hat function. The linear combination of these basis sections

$$\psi = \sum_{v_i \in V} u_i \Psi_i,$$

gives an approximation to the integral of the continuous formulation. Here u is a vector containing the coefficients, associated with each vertex, and eventually will contain the solution to our direction field. Typically in the finite element method, the problem is expressed with matrices, building up a linear system. In our case, the *energy matrix* A, in literature often called *stiffness matrix*, represents the system the resulting field should follow, i.e. the energy function. The *mass matrix* describes the integrals over the individual triangles via the hat function. For a comprehensive introduction to the finite element method in general, we recommend the work of Larson and Bengzon [LB10].

The problem that now needs to be solved amounts to

$$Au = \lambda M u,$$

where A is the Hermitian energy matrix of size $|V| \times |V|$, that represents the smoothness energy E_s and is connected to the piece wise linear basis sections with

$$A_{ij} = \langle\langle A\Psi_i, \Psi_j \rangle\rangle$$

and M is the Hermitian mass matrix defined by

$$M_{ij} = \langle\langle \Psi_i, \Psi_j \rangle\rangle.$$

These two matrices can be built by first computing the local 3×3 mass and energy matrices for each triangle t_{ijk} and then adding their entries into the global matrices via summation. This also accounts for boundary conditions.

The entries for local mass matrix M^l are defined by

$$M^l_{ii} = \langle\langle \Psi_i, \Psi_i \rangle\rangle_{ijk} = \frac{1}{6}|t_{ijk}|$$

and

$$M^l_{jk} = \langle\langle \Psi_j, \Psi_k \rangle\rangle_{ijk} = \bar{r}_{jk}|t_{ijk}| \frac{6e^{i\Omega_{ijk}} - 6 - 6i\Omega_{ijk} + 3\Omega^2_{ijk} + i\Omega^3_{ijk}}{3\Omega^4_{ijk}},$$

where $\bar{r}_{jk}$ is the conjugated complex of the parallel transport coefficient from j to k and $|t_{ijk}|$ denotes the triangle area.

The terms for the local energy matrix A^l are a bit more complex.

$$A^l_{ii} = \Delta_{ii} - s\frac{\Omega_{ijk}}{|t_{ijk}|}|M_{ii}$$

$$A^l_{jk} = \Delta_{jk} - s(\frac{\Omega_{ijk}}{|t_{ijk}|} - \epsilon_{jk}\frac{i\bar{r}_{jk}}{2}),$$

where Δ in this context denote the Dirichlet terms and $\epsilon_{jk} = \pm 1$, depending on the orientation of edge e_{jk} in t_{ijk}. The Dirichlet terms can be computed with

$$\Delta_{ii} = \langle\langle \Delta\Psi_i, \Delta\Psi_i\rangle\rangle_{ijk} = \frac{1}{4|t_{ijk}|}(|p_{jk}|^2 + \Omega^2_{ijk} + \frac{|p_{ij}|^2 + \langle p_{ij}, p_{ik}\rangle + |p_{ki}|^2}{90})$$

$$\Delta_{jk} = \langle\langle \Delta\Psi_j, \Delta\Psi_k\rangle\rangle_{ijk} = \frac{\bar{r}_{jk}}{|t_{ijk}|}[(|p_{ij}|^2 + |p_{ki}|^2)f_1(\Omega_{ijk}) + \langle p_{ij}, p_{ik}\rangle f_2(\Omega_{ijk})]$$

where $p_{ij} = p_j - p_i$ is the vector of edge e_{ij} and f_1 and f_2 are the following functions

$$f_1(\Omega) = \frac{1}{\Omega^4}(3 + i\Omega + \frac{\Omega^4}{24} - \frac{i\Omega^5}{60} + (-3 + 2i\Omega + \frac{\Omega^2}{2})e^{i\Omega})$$

$$f_2(\Omega) = \frac{1}{\Omega^4}(4 + i\Omega - \frac{i\Omega^3}{6} - \frac{\Omega^4}{12} + \frac{i\Omega^5}{30} + (-4 + 3i\Omega + \Omega^2)e^{i\Omega}).$$

The expressions for the off-diagonal matrix entries have singularities for $\Omega_{ijk} \to 0$ that can be removed. Knöppel et al. [KCPS13] used Chebyshev expansion to ensure precise evaluation for these expressions and included usable C++ code in the ancillary material of their paper for calculation. This code, contained in the file *SectionIntegrals.cpp*, was adjusted to work with complex numbers from the C++ standard library and was used in our implementation as well.

3.5 Solving the System

Finding the optimal solution, in this case, is finding the smallest eigenvector to a given linear system. In theory, one could just compute the inverse matrix A^{-1} and solve the expression for x. In practice though, this is not desirable, like Crane et al. [CDGDS13] describe. The inverse matrix A^{-1} might be very dense even if A is very sparse. For large systems, this calculation becomes numerically unstable and the memory usage drastically increases. Because of this, an alternative route is chosen that composes of two steps. First, a matrix factorization has to be computed. In our case, a Cholesky decomposition is used for factorization. This operation splits a matrix into the product $A = LDL^T$, where L is a lower diagonal matrix, L^T its conjugate transpose and D a diagonal matrix. Now those matrix parts are used individually for solving the linear system to improve efficiency. In the second, the system is iteratively solved with an algorithm called Inverse Power Method to approximate the smallest eigenvector. The basic idea for this algorithm

is to solve the system $Ax = Mu$ for x multiple times while back-substituting the resulting vector to u until the change becomes very small. The Inverse Power Method was also used by Knöppel et al.[KCPS13].

We start with a random initialization for u with floating point numbers for each coefficient in the range of $[-1, 1]$. Then we decompose matrix A using Cholesky factorization to efficiently solve the linear system. After each iteration, we back-substitute the solution with $u = \frac{x}{\sqrt{x^T Mx}}$. The solution for u will converge quickly towards the smallest eigenvector and a fixed number of iterations is sufficient. Knöppel et al.[KCPS13] used 20 iterations for their results. Pseudo code for reference can be found in Algorithm 3.1.

For solving linear systems, we use the Eigen library [GJ+10] for linear algebra in our implementation. Specifically, the *SimplicialLDLT* solver for sparse systems is used. It handles both the Cholesky decomposition and the solving step in Algorithm 3.1.

Algorithm 3.1: Inverse Power Method

Input: A,M
Output: u
1 factorize(A);
2 $u \leftarrow random()$;
3 **for** $i \leftarrow 1$ **to** *maximum iterations* **do**
4 $\quad x \leftarrow solve(Ax = Mu)$;
5 $\quad u \leftarrow \frac{x}{\sqrt{x^T Mx}}$
6 **end**

3.6 Triangle Index

To determine if a triangle contains a singularity, a form of label is needed. This label is typically known as an *index* as described by Vaxman et al. [VCD+16]. It measures how much the field rotates along a closed loop around a singularity. Points whose index is not 0 can be considered singular according to them. This definition does not directly apply to surfaces in general, however, the index of a singular point p can still be calculated via an arbitrary chart around p. As they further describe, a vector field can not have an arbitrary number of singularities. The Poincaré-Hopf theorem, as described in Vaxman et al. [VCD+16], dictates that the sum of all indices of a vector field have to sum up to $2 - 2g = \chi$, where g is the genus of the surface and χ is the Euler characteristic.

Knöppel et al. [KCPS13] et al. prove a discrete version of the Poincaré-Hopf theorem and formulate a method for calculating the index of any triangle via the rotational angles ω_{ij} located on each edge. The rotational angles are defined such that

$$u_j = e^{i\omega_{ij}} r_{ij} u_i,$$

where r_{ij} is the parallel transport coefficient. Together with the holonomy Ω_{ijk}, the index of each triangle is given as the integer

$$index = \frac{1}{2\pi}(\omega_{ij} + \omega_{jk} + \omega_{ki} + \Omega_{ijk}) \in \{-1, 0, 1\}.$$

3.7 Principal Curvature Alignment

Up until now, the field does not necessarily follow the inherent curvature of the mesh. Knöppel et al.[KCPS13] show that the algorithm is capable of smoothly interpolating between the smooth setting and curvature alignment with the interpolating function

$$E_{s,t} = (1 - t)E_s(\psi) - tE_l(\psi),$$

where $E_s(\psi)$ is the smoothness energy from section 3.4, $t \in [0, 1]$ and

$$E_l = \int_M Re(\langle\phi, \psi\rangle)dA = Re(\langle\langle\phi, \psi\rangle\rangle),$$

ϕ representing the field we want ours to align with.

Minimizing $E_{s,t}$ can be done by solving

$$(A - \lambda_t I)\tilde{\psi} = \phi,$$

where $\lambda_t \in (-\infty, \lambda_1)$ and λ_1 being the smallest eigenvalue of A. In their proof, they show an underlying connection between the interpolation factor t and the eigenvalue λ_t. If λ_t is set to the smallest eigenvalue of A, no alignment will be performed. Knöppel et al.[KCPS13] suggest that setting $\lambda_t = 0$ is often a good starting point for initial alignment and can afterward be adjusted. An alternative we found can be the approximation of the smallest eigenvalue of the smooth solution u via the Rayleigh quotient $R(A, u) = \frac{u^T Au}{u^T u}$, where u^T is the conjugate transpose. Then the deviation from that eigenvalue towards $-\infty$ can be set as a parameter. We chose to implement this option to allow the user to change the alignment strength.

The alignment field that was chosen by Knöppel et al.[KCPS13] is a piece wise linear approximation of the Hopf differential

$$\phi = \sum_{v_i \in V} q_i \Psi_i.$$

The Hopf differential is part of the shape operator and contains information about the principal curvature directions, from which the directions of maximal curvature can be found.

They argue that in the discrete setting this differential is only obtainable as a concentration on the edges and that the coefficients in q have to be approximated with

$$Mq = \tilde{q}.$$

Together with the basis sections Ψ_i from Section 3.4 the coefficients for $\tilde{q}$ can be calculated with

$$\tilde{q}_i := q(\Psi_i) = \sum_{e \ni i} q_e(\Psi_i) = -\frac{1}{4} \sum_{e \ni i} r_{ie} \beta_e |p_e|,$$

where r_{ie} is the transport coefficient $e^{i2\theta_i(X_i,e)}$, $\theta_i(X_i, e)$ here being the rescaled angle between the basis vector of vertex i and the edge e, β_e resembles the dihedral angle between faces adjacent to e and $|p_e|$ is the length of e. They suggest to use q^2 for 4-direction fields. Finally, to get the aligned field the linear system

$$(A - \lambda_t M)\tilde{u} = Mq$$

needs to be solved, where $\tilde{u}$ holds the result. Examples of different alignment weights λ_t are shown in Figure 3.4.

Detailed derivations of the formulas above can again be found in the appendix of Knöppel et al.[KCPS13].

3.8 Alignment Constraints

Like mentioned in Chapter 2, a quad-dominant remeshing method was introduced by Pellenard et al. [POC+14]. It uses the algorithm by Knöppel et al.[KCPS13] and is a variant of the QMorph algorithm, but it also uses a crossfield for guidance when adding new vertices. To use it for their purposes, Pellenard et al. first addressed a few limitations of the original globally optimal direction field synthesis. The first one is that the direction field does not line up with the boundary of the mesh, making the result of an advancing front algorithm like QMorph, less optimal. The second one is distortion that sometimes arises in areas with low curvature. It would be desirable if the flatter sections of the mesh align better with principal curvature. A third problem is that the field should align with sharp edges so that they can be kept intact during remeshing. The last limitation they addressed is that it cannot properly handle non-manifold edges. In order to overcome these limitations, they first split the mesh on edges that are part of more than 2 faces. The affected vertices now have more than one cross assigned to them. Before solving for the alignment, a constraint vector is defined for each vertex that falls under one of the following conditions: a) v is on the boundary, b) v is part of an edge whose dihedral angle is greater than 30 degrees, c) v has a mean- or Gaussian curvature that differs from 0 or d) v is part of a non-manifold edge. We did not implement condition d) due to the framework we are using, which does not handle the splitting of non-manifolds into multiple meshes.

Pellenard et al. [POC+14] do not mention how they calculated the mean- and Gaussian curvature. We chose to use the discrete Gaussian curvature operator i.e. the angle defect

$$K_G(v_i) = \frac{2\pi - \sum_j \theta_j}{A_{v_i}},$$

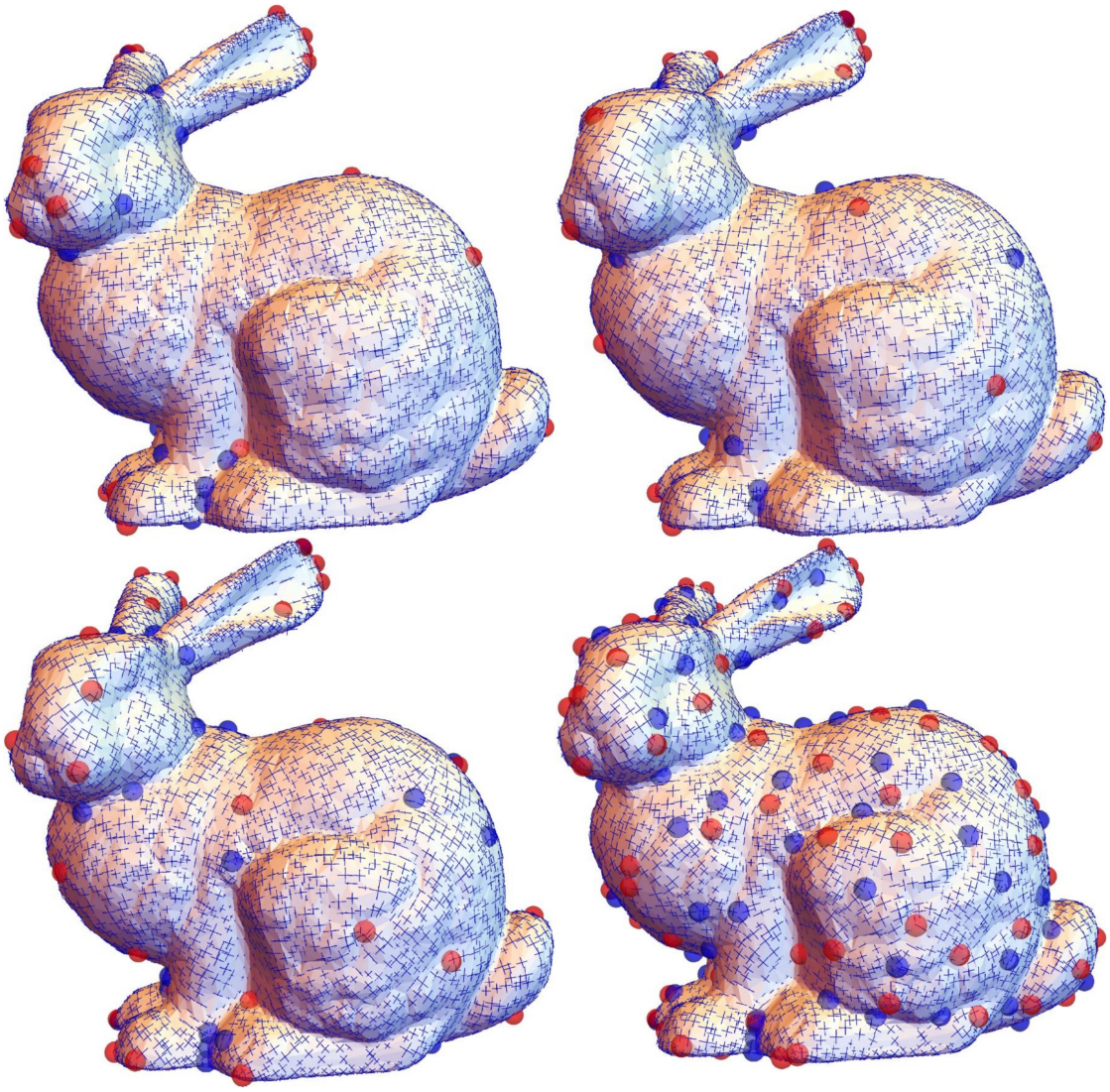

Figure 3.4: Different Alignment Weights
The upper left bunny used $\lambda_t = \lambda_1 - 5$, the top right one $\lambda_t = \lambda_1 - 15$, the lower left $\lambda_t = \lambda_1 - 30$ and the lower right one $\lambda_t = \lambda_1 - 100$. The eigenvalue λ_1 was approximated with the Rayleigh coefficient and q^2 was used for alignment. We can see that the further we deviate from λ_1 the more aligned to local curvature the field gets, but on the other hand the number of singularities rises.

where $\sum_j \theta_j$ is the sum of all angles around the vertex i, and the discrete Laplace-Beltrami operator

$$K_M(v_i) = \frac{1}{2A_{v_i}} \sum_{v_j \in N_1(i)} (\cot \alpha_{ij} + \cot \beta_{ij})(v_j - v_i),$$

where $v_j \in N_1(i)$ is the 1-ring neighborhood of vertex v_i, α_{ij} and β_{ij} are the angles opposite of edge e_{ij} in the faces connected by e_{ij}. The final value for the mean curvature of each vertex can be found with $\frac{1}{2}||K_M(v_i)||$. In both formulas above, A_{v_i} is $\frac{1}{3}$ of the triangle areas that are part of $N_1(i)$. A detailed description of these operators can be found in Meyer et al. [MDSB03].

One ambiguity we want to clarify is how to get the constraint vectors. We choose a direction according to the categories mentioned above:

a) On the boundary, each vertex has two boundary edges. We choose the vector $\frac{e_1}{|e_1|} - \frac{e_2}{|e_2|}$, where e_1 and e_2 are the two boundary edges of a vertex. If those two edges are close to a right angle, i.e. the dot product is almost zero, we choose one of them directly as the constraint direction instead, to preserve corners.

b) Vertices that are part of at least one sharp edge get one of those incident edges assigned as the constraint direction.

c) Constraint directions for vertices that have non-zero mean- or Gaussian curvature are set to their principal curvature direction $\tilde{q}_i$. We found that sometimes, we also want to align vertices that are not completely flat but are very close to zero curvature to their principal curvature direction. For example, if the mesh is very noisy. For these cases, we implemented a slider to adjust the tolerance towards the mean- and Gaussian curvature.

If a vertex qualifies for more than one constraint, we prioritize boundary over sharp edge and sharp edge over curvature constraints. Our reasoning behind this is that constraint types a) and b) are somewhat "hard" and more clearly defined, whereas curvature constraints are primarily there to prevent distortions that occur because of the other two types. Pellenard et al. [POC$^+$14] do not mention any prioritization.

After setting the constraints for vertices, Pellenard et al.[POC$^+$14] mention that they remove constraints from vertices whose crosses are inconsistent. They remove constraints from triangles whose *index* is non-zero and from vertices whose directions differ too much from their neighbor ones. We chose to only remove the curvature constraints. Otherwise, boundary and sharp edge constraints would get removed very often because they tend to differ more significantly. Additionally, we chose not to remove constraints from triangles whose index is non-zero because most of the time this case is also covered by differing neighbor constraint directions. As a threshold for when to remove the constraints if neighbors differ, we chose a difference of 30 degrees which in our opinion is very liberal but Pellenard et al [POC$^+$14] do not go into detail at which difference they remove the constraints. Otherwise, a lot of the time most of the curvature constraints get removed. This happens especially if the mesh is not very well tessellated and principal curvature directions are very inconsistent.

The linear system for the alignment is now adjusted to incorporate the constraints

$$Au = \alpha \bar{*} Mq,$$

where α is a vector of length $|V|$ whose coefficients are 0 for constrained vertices and their neighbors and 1 otherwise, $\bar{*}$ describes an element-wise multiplication. The vector α controls which vertices should influence the alignment process with their principal curvature directions. Constrained vertices should not influence it because they should only contribute by adding their constraint direction to the alignment. Vector u now contains the constraint directions as components for constrained vertices. Those directions are again expressed as complex numbers describing them in tangent space.

The problem with this adjusted linear system is that it is over-constrained and cannot be solved like before. Instead, a linear least squares approach is used to solve it. First, A is split up into A_f, containing only columns of the free vertices, and A_c, containing only the constrained columns. Likewise, the vector u is split up into u_f and u_c as well, where u_c contains the constraint directions that were previously computed and u_f will contain the result for the free vertices. Therefore let

$$Au = A_f u_f + A_c u_c$$

and

$$A_f u_f = \alpha \bar{*} Mq - A_c u_c.$$

For convenience the right side of the above expression gets shortened to $b = \alpha \bar{*} Mq - A_c u_c$ Now we build the conjugate transpose matrix A_f^T and let

$$A_f^T A_f u_f = A_f^T b.$$

Now

$$Bu_f = b',$$

where $B = A_f^T A_f$ and $b' = A_f^T b$, can be solved.

CHAPTER 4

Results

All visual representations for direction fields in this thesis were made by converting the directions to world coordinates and displaying them on the respective tangent planes. For this, we used Mathematica. The machine we used was an Intel Core i7-7500 Processor at 2.9 GHz running Windows 10 Pro. Our implementation was tested with varying types of geometries and different parameter settings.

4.1 Visuals

One key observation we made was that fields produced by the core algorithm by Knöppel et al.[KCPS13] do not necessarily align with the mesh boundary. This is one of the main limitations Pellenard et al. [POC+14] mention. Although this solution looks optimally smooth, it may cause problems in remeshing algorithms where alignment with the boundary produces the best results. For example the advancing front algorithm of Pellenard et al. [POC+14]. An example of a flat mesh can be seen in Figure 4.1. Using the boundary constraints alleviates those problems but at the cost of potentially additional singularities. Figure 4.2 shows the same mesh as Figure 4.1 with boundary constraints applied.

The results produced by using sharp edge constraints highly depend on the input mesh. If the number of sharp edges is very high and the surface is very noisy, the field can look very uneven with a lot of singularities. Figure 4.3 shows a very noisy mesh. If used on more regular meshes, where sharp edges are more aligned to each other, the constraints produce results where the number of singularities stays almost the same but with field directions that are well aligned to the sharp edges. A comparison between a result with and without the edge constraints can be seen in Figure 4.4.

The curvature constraints can be very useful if the field gets distorted by enforcing other constraints. They restrict the field to the principal curvature direction in curved

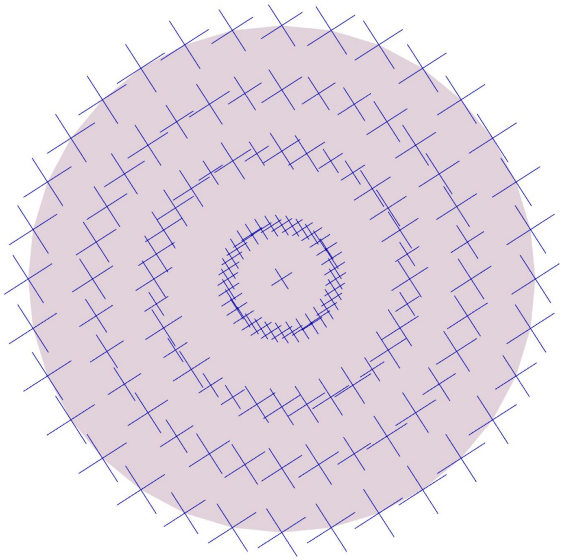

Figure 4.1: Field on a Disk
Crossfield directions on a flat disc. Notice how the field does not align with the boundary.

areas and align the flat parts to them. Boundary constraints, in particular, have a very high tendency to warp the direction field, even on distant parts of the mesh. This is especially true if the boundary is curved. Figure 4.5 shows an example of how the curvature constraints affect the result. We do not recommend using the curvature constraints if no other constraints are enforced as well. The reason for this is that in most cases the curvature alignment produces better-looking results than just with added curvature constraints. Also, if the mesh is not tessellated very well, the computed principal curvature directions are not always very accurate. Therefore they disturb the field when using curvature constraints. Increasing the tolerance at which point we do not constrain vertices can be very useful as noisy regions also get the chance to align better with directions from vertices with stronger curvature.

We noticed, that for some meshes the number of singularities does not follow the discrete Poincaré-Hopf theorem of Knöppel et al.[KCPS13]. The exact reason is not known to us. We believe it may be due to the input mesh not being n-smooth i.e. the total curvature pushed into each triangle does not follow $|\sigma_t| < \frac{\pi}{n}$, where σ_t is the curvature pushed into triangle t and n is the rotational symmetry, see Appendix B of Knöppel et al. [KCPS13] for reference.

4.2 Computation times

We tested the implementation with meshes, that resemble models produced by 3D artists and did not test 3D scans which tend to either be non-manifolds or have extremely high

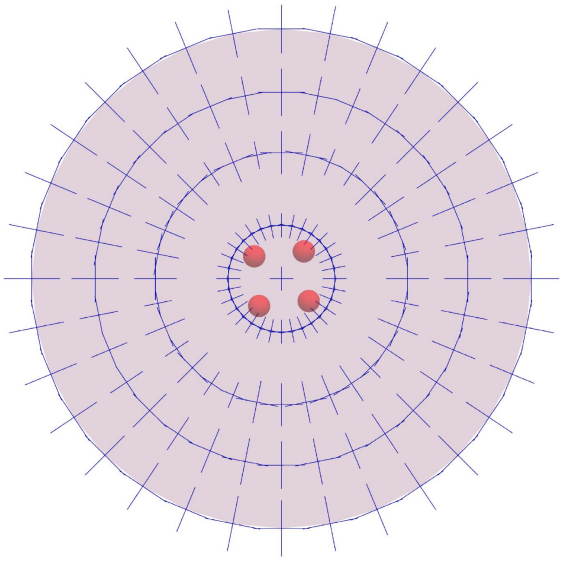

Figure 4.2: Field on a Disk using Boundary Constraints
Crossfield directions on a flat disc aligned with the border. New singularities arise but the field looks smooth and symmetric.

polygon counts. We also set the number of solver iterations to 30. If faster results are needed, fewer iteration can be used, although the result might not be as smooth because the inverse power method might not have converged. Figure 4.6 shows a comparison of computation time and the number of polygons. At about 6×10^4 polygons, the increased computation time becomes noticeable. Larger meshes are not guaranteed to be processed fast enough for real-time interactivity, which we assume lies below the 1-second mark. Curvature alignment increases computation time as expected because principal curvature directions have to be approximated. Afterward, it takes another solver iteration for the alignment. Using the constraints adds the computation time for the constraint directions and for splitting the energy matrix. Staying under one second can then only be achieved by reducing the polygon count. 4.1 shows the used data for Figure 4.6.

4.3 Usability

In the case of the smooth setting without alignment, we found that the algorithm is simple to use. This is because only the type of energy has to be chosen. The fast computation time helps immensely because the three different results can be compared relatively easy. When we look at the case of curvature alignment, on the other hand, it can be tedious to find the best-suited solution. Like before, there are three different energy types to compare, but the degree of alignment is very hard to estimate in advance. We included a slider in the user interface to let the user control how much he or she wants to deviate from the eigenvalue of the smooth solution, controlling the strength of alignment. In theory there are infinite different possibilities for this value because $\lambda_t \in (-\infty, \lambda_1)$ (see

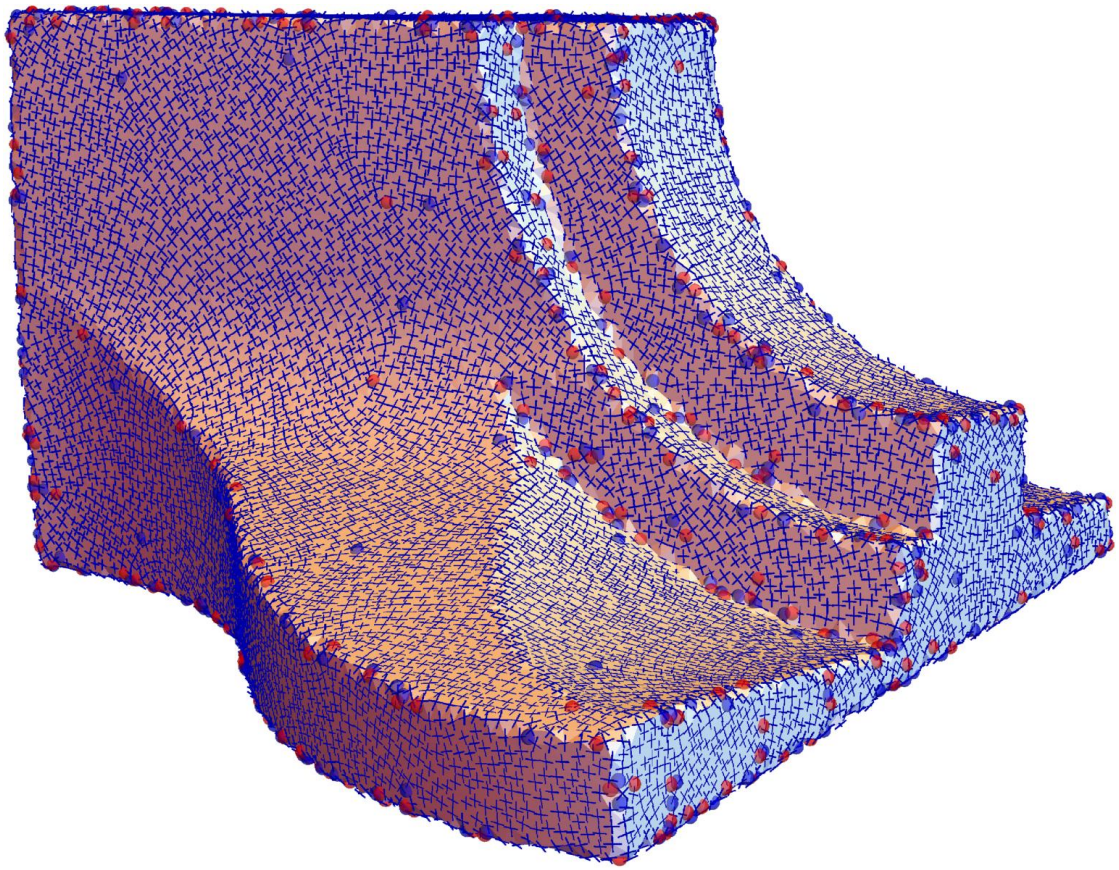

Figure 4.3: Sharp Edge Constraints used on a noisy Fandisk Mesh
This fandisk model has noisy edges that are not well aligned. Using the sharp edge constraints adds a great amount of singularities and disturbs the field.

Section 3.7). For this reason, we have to limit it to an arbitrary maximum deviation and we chose to set this limit to $\lambda_1 - 100$. From our experience, any deviation beyond that increases the number of singularities too much and the field loses its overall smoothness. A further improvement may be to introduce a logarithmic slider to give more control closer to the eigenvalue while also increasing the limit. We leave this improvement up to future work on our implementation. Using the constraints seems straightforward to us. In our opinion, it is easy to estimate how the result may change when using them. The exception being the curvature constraints which can sometimes be less predictable. The slider for adjusting the tolerance can be a bit confusing but we think that users might get used to it after experimenting with it. A nice addition would be a parameter to set at which angle an edge is considered sharp.

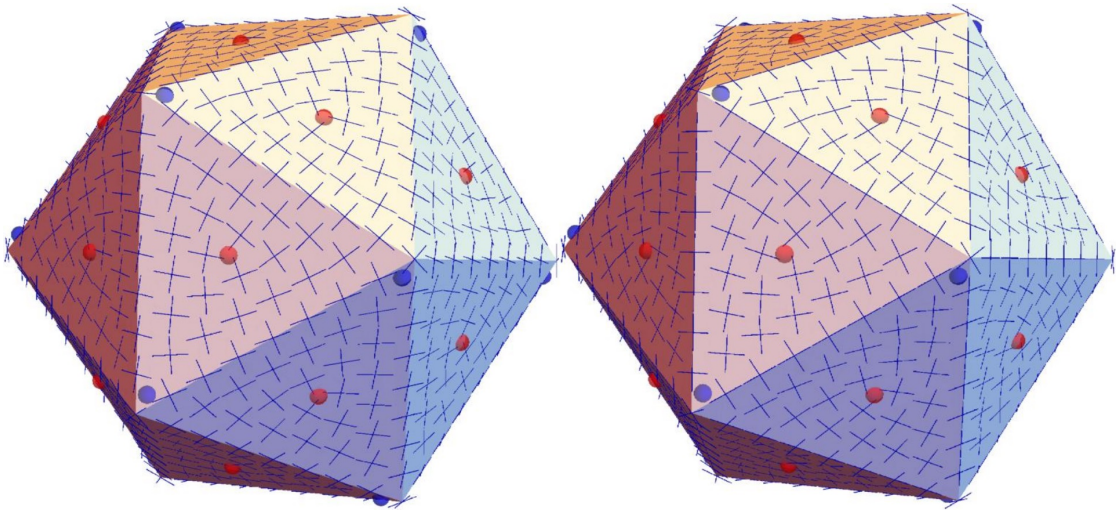

Figure 4.4: Sharp Edge Constraints used on an Icosahedron
Edges on this icosahedron are well aligned to each other. On the left, no constraints where used and λ_t is set to 0. On the right, sharp edge constraints force the field to align with the edges.

Mesh	Polygons	Vertices	Smooth time	Alignment time	Constraint time
Teapot	15704	8435	0,14	0,24	0,32
Monkey	15744	7958	0,13	0,26	0,38
Cow	22620	11339	0,19	0,35	0,55
Space Suit	30888	18153	0,25	0,44	0,6
Fandisk	31682	16784	0,41	0,75	1,06
Horse	40826	81666	0,41	0,77	1,39
High Resolution Rounded Cube	49152	24578	0,64	1,35	2,71
Bunny	69630	34817	0,99	1,96	3,22
Teddy	101018	50548	1,02	1,94	3,76
Hand	133692	66848	1,6	2,87	4,91
Deer	192350	96320	2,15	4,16	7,5
High Poly Torso with Head	234496	117250	4,78	10,78	19,03

Table 4.1: Description of used meshes and their computation times in seconds.

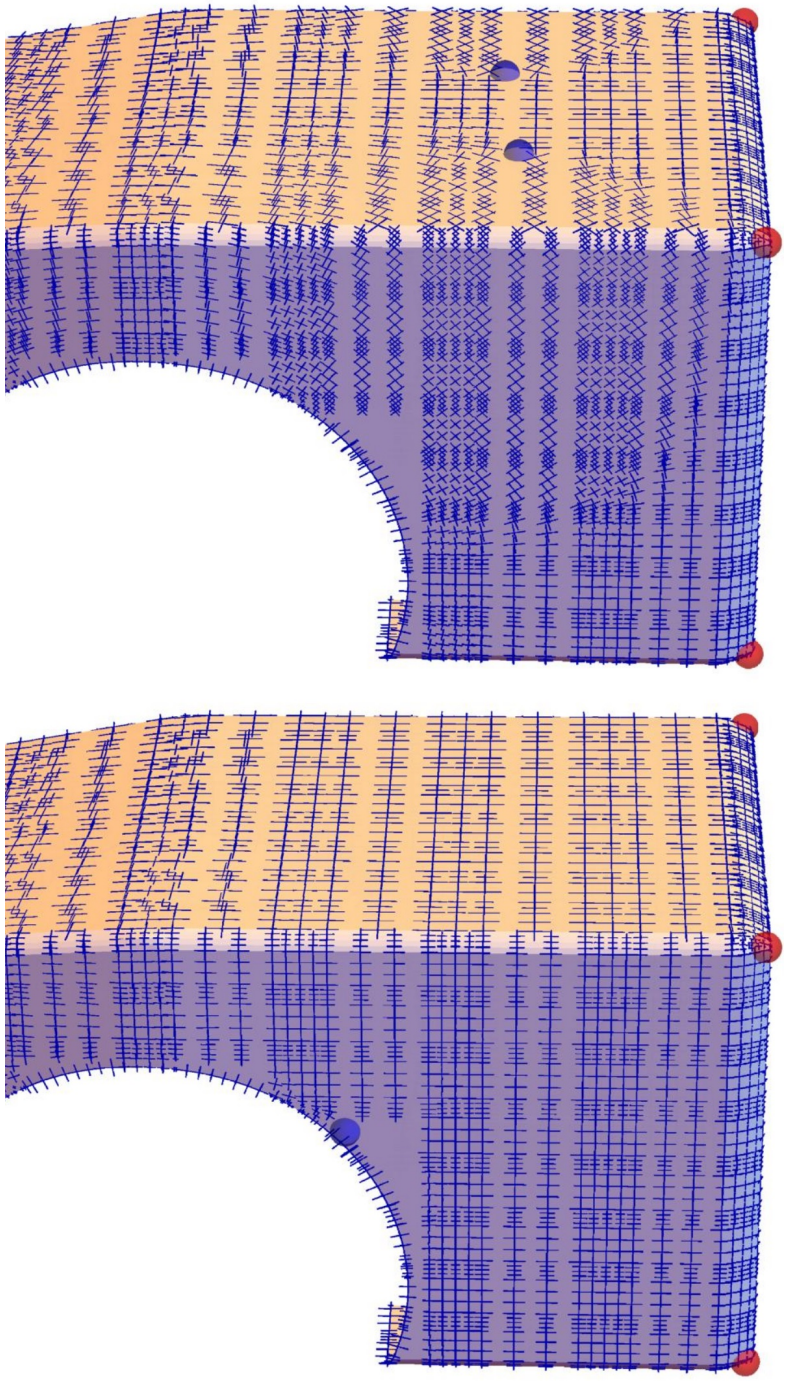

Figure 4.5: Curvature Constraints used in addition to Boundary Constraints
This mesh has a large circular boundary. Using boundary constraints warps the field across the whole mesh on the top. On the bottom, curvature constraints are added which forces the field to align with the beveled edges and straightens it up on the flat areas.

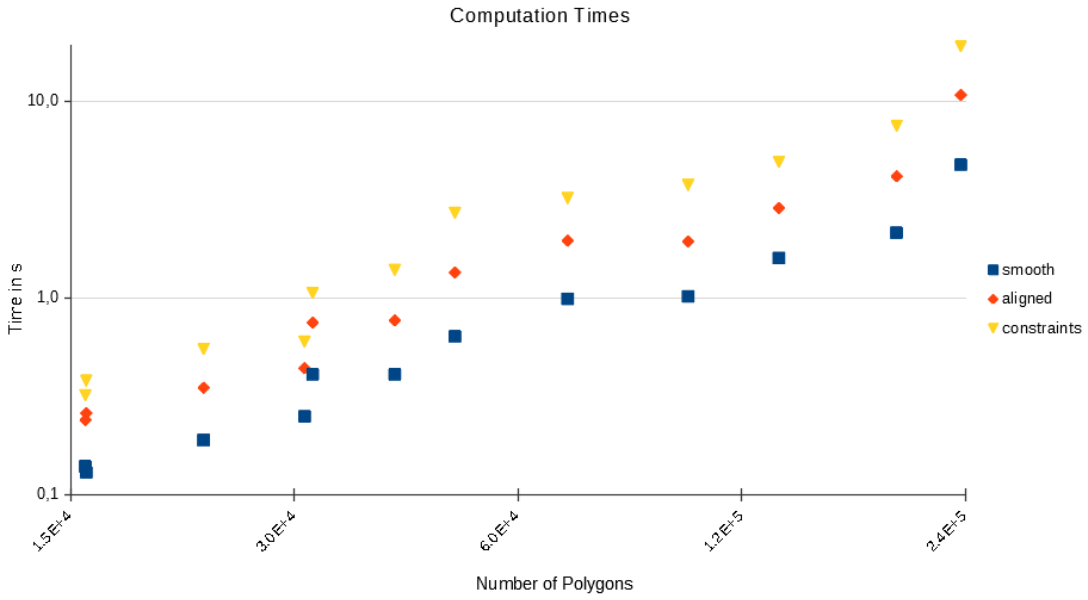

Figure 4.6: Computation Times
A log-log plot showing computation time compared to the number of polygons for smooth, curvature aligned and constrained fields. On average the additional computation time from curvature alignment makes up about 47,5% of the total computation time. Using all constraints adds a similar amount of computation time on top of that. The data we used for this figure can be found in Table 4.1.

CHAPTER 5

Conclusion and Future Work

We demonstrated the implementation of the globally optimal direction fields algorithm by Knöppel et al. [KCPS13] using the constraints introduced by Pellenard et al [POC+14], clarified ambiguities in the explanation and have shown that it is a powerful tool for constructing direction fields of arbitrary rotational symmetry over a mesh. It computes either an optimally smooth or curvature aligned direction field with optional constraints and it generates singularities automatically. The core of the algorithm is a smallest eigenvalue problem which can be solved efficiently with a solver for sparse linear systems. We have found that for meshes of common size in artistic modelling it is fast enough to finish in under 1 second and stay interactive. The limit for 1 second computation time is at roughly 6×10^4 triangles. Larger meshes than that, like 3D scans, may take significantly more time. The added computation time for constraints further decreases the number of triangles per second.

The algorithm is easy to use but determining how strong the field should align to the curvature is hard to estimate beforehand. Combined with the longer computation time for large meshes, interactive changes to the parameters are not feasible. Some computation time would be saved if transport coefficients and the system matrices could be saved between parameter changes. However, as our implementation is a plugin, the setup must be recomputed each time.

The results look very smooth and singularities are commonly located on geometric features. The aligned field directions follow the natural curvature of the mesh. The changeable deviation from the smallest eigenvalue can control the strength of alignment and the number of singularities. Adding constraints to the field alignment gives better control and helps aligning it with features such as the border, sharp edges or principal curvature.

Future work on our algorithm may include its separation into a standalone application or direct implementation into a framework to facilitate faster user interaction. The addition

of a changeable parameter for the number of solver iterations could be a solution to allow for larger meshes. With this, the user can decide if he or she wants faster computation or more accuracy. A drawback may be that inexperienced users do not find a middle ground. The addition of a remeshing algorithm to our plugin would also be of high interest in future work.

List of Figures

Bibliography

[BZK09] David Bommes, Henrik Zimmer, and Leif Kobbelt. Mixed-integer quadrangulation. *ACM Transactions On Graphics (TOG)*, 28(3):77, 2009.

[CDGDS13] Keenan Crane, Fernando De Goes, Mathieu Desbrun, and Peter Schröder. Digital geometry processing with discrete exterior calculus. In *ACM SIGGRAPH 2013 courses*. ACM, 2013.

[CDS10] Keenan Crane, Mathieu Desbrun, and Peter Schröder. Trivial connections on discrete surfaces. *Computer Graphics Forum (SGP)*, 29(5):1525–1533, 2010.

[dGDT16] Fernando de Goes, Mathieu Desbrun, and Yiying Tong. Vector field processing on triangle meshes. In *ACM SIGGRAPH 2016 Courses*. ACM, 2016.

[GJ+10] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[HZ00] Aaron Hertzmann and Denis Zorin. Illustrating smooth surfaces. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 517–526. ACM Press/Addison-Wesley Publishing Co., 2000.

[Ins] InstaLOD. Studio xl. https://instalod.com.

[JTPSH15] Wenzel Jakob, Marco Tarini, Daniele Panozzo, and Olga Sorkine-Hornung. Instant field-aligned meshes. *ACM Transactions on Graphics (TOG)*, 34(6):189, 2015.

[KCPS13] Felix Knöppel, Keenan Crane, Ulrich Pinkall, and Peter Schröder. Globally optimal direction fields. *ACM Transactions on Graphics (TOG)*, 32(4):59, 2013.

[LB10] Mats G Larson and Fredrik Bengzon. The finite element method: theory, implementation, and practice. *Texts in Computational Science and Engineering*, 10, 2010.

[MDSB03] Mark Meyer, Mathieu Desbrun, Peter Schröder, and Alan H Barr. Discrete differential-geometry operators for triangulated 2-manifolds. In *Visualization and mathematics III*, pages 35–57. Springer, 2003.

[OSCS99] Steven J Owen, Matthew L Staten, Scott A Canann, and Sunil Saigal. Q-morph: an indirect approach to advancing front quad meshing. *International Journal for Numerical Methods in Engineering*, 44(9):1317–1340, 1999.

[PLPZ12] Daniele Panozzo, Yaron Lipman, Enrico Puppo, and Denis Zorin. Fields on symmetric surfaces. *ACM Transactions on Graphics (TOG)*, 31(4):111, 2012.

[POC+14] Bertrand Pellenard, Gunay Orbay, James Chen, Shailendra Sohan, Wa Kwok, and Joseph R Tristano. Qmcf: Qmorph cross field-driven quad-dominant meshing algorithm. *Procedia Engineering*, 82:338–350, 2014.

[RVLL08] Nicolas Ray, Bruno Vallet, Wan Chiu Li, and Bruno Lévy. N-symmetry direction field design. *ACM Transactions on Graphics (TOG)*, 27(2):10, 2008.

[VCD+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*, volume 35, pages 545–572, 2016.