b
UNIVERSITÄT
BERN

3D Metric Fields

Optimizing Frame Fields in a new Metric

Bachelor Thesis

Florin Luc Achermann

from

Bern, Switzerland

Faculty of Science, University of Bern

October 2023

Prof. Dr. David Bommes
Denis Kalmykov
Computer Graphics Group
Institute of Computer Science
University of Bern, Switzerland

Abstract

Automatic generation of volumetric hexahedral meshes continues to be a hard challenge. Directly optimizing for a decomposition of an object into cube-like elements is infeasible with today's available optimization techniques. Thus, this optimization problem is often relaxed through frame fields. By adding a covering metric field on the object to mesh and optimizing the frame field in this new metric, more control of the frame field is achieved. In this work, it is shown how to optimize a frame field in a new, user-defined metric. A full explanation is given of how to pull back the user-defined metric such that already available Euclidean metric frame field optimization techniques can be applied. The metric field is piecewise linearly discretized, for which an exact and robust algorithm is formulated that finds all intersected tetrahedra between two points in space. We test the generation of frame fields on some custom metric fields on cubes, where the resulting frame fields exhibit the expected singularity structure.

Contents

Chapter 1

Introduction

In many applications of computer graphics or engineering, the geometry of objects must be described in some way. Finite elements encode various information such as physical properties to be used in e.g., stress simulations. While it is possible for simple objects such as spheres or planes to be described by analytical formulas, this is infeasible for arbitrary objects of any shape. To describe any geometric object, it is discretized and encoded as a mesh. Volumetric meshes are used to encode the surface and the interior of an object. The most common types of volumetric meshes are tetrahedral and hexahedral meshes. Hexahedral meshes, that is a decomposition into cube-like elements, are usually preferred due to better numerical features. However, automatically generating high-quality hexahedral meshes remains a hard challenge in computer graphics. A promising research direction is through the use of frame fields [Pietroni et al., 2022].

A *frame field* can be seen as a generalization of vector fields. They prescribe a *frame* to each point on an object, that is, three linearly independent vectors. Each frame locally represents the orientation and deformation of a cube. Consequently, frame fields must be so called boundary aligned: At the boundary of an object, one vector of the frame must be aligned with the surface normal as hexahedral elements (*short: hex elements*) should align at the boundary. Then, the objective is to find the "best" frame field. Usually, "best" is measured in the smoothness of the frame field. Smoothness is measured with the Dirichlet energy $\int_\Omega ||\nabla\phi||^2 dx$, where ϕ is a frame, and we measure how much these frames twist and rotate within the object.

To allow for non-uniform and anisotropic hex elements, we introduce *metric fields*. Optimizing the frame field under another metric (instead of the usual Euclidean metric) then results in elements that are uniform and isotropic in this new metric. To this end, the expression for the Dirichlet energy must be changed, as Euclidean derivatives assume non-curved space. We will show how to calculate a special rotation matrix R, with which we can measure smoothness of frames in the new metric.

In this thesis, we rederive how to pull back the metric with the rotation coefficient R such that we can measure the Dirichlet energy in the usual cartesian coordinates as described in *Metric-Driven 3D Frame Field Generation* [Fang et al., 2023]. The necessary mathematical background is introduced in Section 2. In Section 3, by focusing on integrability of the frame field, we derive the connection ω induced by the metric field and how to evaluate it. Section 4 discusses how the metric field is piecewise linearly discretized on a tetrahedral mesh and how the rotation coefficient R to align frames in the metric is calculated within a specific tetrahedron (*short: tet*). Because in general, R is needed between arbitrary points in the metric field and the metric field changes from one tet to another, a robust tet finder algorithm is presented in Section 5 to calculate R in the correct metric field. We briefly explain how the actual frame field optimization works in Section 6. The whole frame field generation is tested with some experiments in Section 7. We conclude in Section 8.

Chapter 2

Mathematical Background

A solid mathematical introduction is key to understanding how and why the approach in this thesis works. This chapter serves as a primer to differential geometry and frame fields.

2.1 Differential Geometry

We will make heavy use of differential geometry in Chapter 3. I introduce the basic concepts that we will use, but I refrain from giving any proofs. The following is an incomplete summary of concepts that we need presented in *Introduction to Smooth Manifolds* [Lee, 2000].

2.1.1 Manifold

A manifold $\mathcal{M}$ is a space that locally looks like Euclidean space. More exactly, a n-manifold is a topological space, where each point on the manifold has an open neighborhood that is locally homeomorphic to an open subset of Euclidean space $\mathbb{R}^n$. A manifold can be equipped with additional structure. For example, we can work on *smooth manifolds*. In simple terms, a manifold is *smooth* if it is similar enough to $\mathbb{R}^n$ that we can do Calculus like differentiation or integration on it. For this, each point on the manifold must be locally *diffeomorphic* to an open subset of $\mathbb{R}^n$ space. In this thesis, we will mostly work with 3-manifolds. For easier illustration, some examples are in 2 dimensions.

2.1.2 Tangent space, Tangent bundle

There are many equivalent definitions for the tangent space. The tangent space is a vector space which has the same dimension as its manifold, which is 3 in our case. One definition is for each point p in the manifold $\mathcal{M}$, the tangent space $T_p\mathcal{M}$ consists of $\gamma'(0)$ for all differentiable paths $\gamma : (-\varepsilon, \varepsilon) \to \mathcal{M}, \varepsilon > 0$ with $p = \gamma(0)$. These tangent spaces can be "glued" together to form the *tangent bundle* $T\mathcal{M} = \sqcup_{p\in\mathcal{M}} T_p\mathcal{M}$, which itself is a manifold of dimension $2n$. An element of $T\mathcal{M}$ can be written as (p, v) with $p \in \mathcal{M}$ and $v \in T_p\mathcal{M}$. This admits a natural projection $\pi : T\mathcal{M} \to \mathcal{M}$, which sends each vector $v \in T_p\mathcal{M}$ to the point p where it is tangent: $\pi(p, v) = p$. A *section* $\sigma : \mathcal{M} \to T\mathcal{M}$ is a continuous map, with $\pi \circ \sigma = \mathrm{Id}_{\mathcal{M}}$. Sections of $T\mathcal{M}$ are tangent vector fields on $\mathcal{M}$. An ordered k-tuple $(X_1, \ldots, X_k)$ of vector fields is linearly independent, if at each point p, the k-tuple $(X_1|_p, \ldots, X_k|_p)$ is linearly independent. A *local frame for* $\mathcal{M}$ is an ordered n-tuple of vector fields $(E_1, ..., E_n)$ defined on an open subset $U \subseteq \mathcal{M}$ that spans the tangent bundle $T\mathcal{M}$. Thus, $(E_1|_p, \ldots, E_n|_p)$ form a basis for $T_p\mathcal{M}$ for each point $p \in \mathcal{M}$. A *frame* is called *smooth* if the underlying vector fields are smooth and *global* if $U = \mathcal{M}$.

2.1.3 Cotangent space, Cotangent bundle

The dual space V^* of a vector space V consists of all linear maps $\omega : V \to \mathbb{R}$. We call these functionals *covectors* on V. V^* is itself a vector space, with the same dimension as V and operations like addition

and scalar multiplication can be performed on its elements. Any element in a finite vector space can be expressed as a finite linear combination of its basis. The basis of a dual space is called the *dual basis*. Thus, we call the dual space of the vector space $T_p\mathcal{M}$ its *cotangent space*, denoted by $T_p^*\mathcal{M}$. As before, the disjoint union of $T_p^*\mathcal{M}$ forms the *cotangent bundle*: $T^*\mathcal{M} = \sqcup_{p\in\mathcal{M}} T_p^*\mathcal{M}$. Defined analogously from above, sections σ on $T^*\mathcal{M}$ define *covector fields* or *1-forms*. The standard basis of $T_p^*\mathcal{M}$ is given by $\mathbb{E}^3 = [dx, dy, dz]$. A *local coframe for* $\mathcal{M}$ is an ordered n-tuple of covector fields $(\mathcal{E}^1, \ldots, \mathcal{E}^n)$ such that $(\mathcal{E}^i|_p)$ form a basis of $T_p^*\mathcal{M}$ at each point $p \in \mathcal{M}$. This definition is analogous from before. For each local frame $(E_1, \ldots, E_n)$ for $T\mathcal{M}$, we get a uniquely determined local coframe $(\mathcal{E}^1, \ldots, \mathcal{E}^n)$ such that $\mathcal{E}^i(E_j) = \delta^i_j$, where δ^i_j is the Kronecker delta. We call this coframe the *dual coframe to* (E_i).

2.1.4 Tensors

Before we can introduce differential forms in the next paragraph, we need to go a little bit into *tensors*. In simple words, tensors are real-valued, multilinear functions. A map $F : V_1 \times \ldots V_k \to W$ is multilinear, if F is linear in each component. For example, consider the *Tensor Product of Covectors*: Let V be a vector space and take two covectors $\omega, \eta \in V^*$. Define the new function $\omega \otimes \eta : V \times V \to \mathbb{R}$ by $\omega \otimes \eta(v_1, v_2) = \omega(v_1)\eta(v_2)$. It is multilinear, because ω and η are linear.

We look at a special class of tensors, the *alternating tensors*. A tensor is alternating, if it changes sign whenever two arguments are interchanged, i.e. $\omega(v_1, v_2) = -\omega(v_2, v_1)$. A covariant tensor field over a manifold defines a covariant tensor at each point on the manifold, covariant because the tensor is over the cotangent space $T_p^*\mathcal{M}$. An alternating tensor field is called a *differential form*.

2.1.5 Differential Forms, Exterior Derivative

Recall that a section from $T^*\mathcal{M}$ is called a differential 1-form, or just 1-form. We define the *wedge product* (or *exterior product*) between two 1-forms:

$$(\omega \wedge \eta)_p = \omega_p \wedge \eta_p$$

Notice the similarity to the *Tensor Product of Covectors*: We get a new map, (a 2-form):

$$\omega \wedge \eta : T\mathcal{M} \times T\mathcal{M} \to \mathbb{R}$$

The wedge product is antisymmetric, therefore $\omega \wedge \eta = -\eta \wedge \omega$ for 1-forms ω and η. We denote by $\Omega^k(\mathcal{M})$ the space of differential k-forms on $\mathcal{M}$. There is a natural differential operator d on differential forms we call *exterior derivative*. It maps k-forms to $(k+1)$-forms, i.e. $d : \Omega^k(\mathcal{M}) \to \Omega^{k+1}(\mathcal{M})$ and has the following properties:

- df is the ordinary differential of a smooth function f. Smooth functions are 0-forms.
- $d(d\alpha) = 0$
- $d(\alpha \wedge \beta) = (d\alpha \wedge \beta) + (-1)^k(\alpha \wedge d\beta)$ for a k-form α (Leibnitz Rule).

In Section 3 we will need what $d\omega$ is for some smooth 1-form ω, the calculation will be done there. For now, just note that any smooth 1-form can be written as $\omega = Fdx + Gdy + Hdz$ for some appropriate smooth functions F, G, H with the standard basis $\mathbb{E}^3$.

2.1.6 Riemannian metric, g-orthonormality

Inner products $\langle \cdot, \cdot \rangle$ are examples of symmetric tensors. They allow us to define lengths and angles between vectors. We can apply this idea to manifolds. A Riemannian metric g is a symmetric positive-definite tensor field at each point. If $\mathcal{M}$ is a manifold, the pair $(\mathcal{M}, g)$ is called a *Riemannian manifold*. Let g be the Riemannian metric on $\mathcal{M}$ and $p \in \mathcal{M}$, then g_p is an inner product on $T_p\mathcal{M}$. We write

$\langle \cdot, \cdot \rangle_g$ to denote this inner product. Any Riemannian metric can be written as positive-definite symmetric matrix g, which allows for this simple form: $\langle v, w \rangle_g = v^\top g w$.

Such a new metric allows for the definition of *g-orthonormality*: A basis $[e_1, e_2, e_3]$ of $T_p\mathcal{M}$ is g-orthonormal if $\langle e_i, e_j \rangle_g = \delta_{ij}$.

2.1.7 Connection, Covariant derivative, Parallel transport

A connection defines how two different tangent spaces are connected to each other, such that tangent vector fields can be differentiated. There is an infinite amount of connections on a manifold. An *affine connection* ∇ is a bilinear map that takes two tangent vector fields $X, Y \in \mathcal{T}(\mathcal{M})$ and maps it to a new tangent vector field $\nabla_X Y \in \mathcal{T}(\mathcal{M})$ such that

- $\nabla_{fX} Y = f \nabla_X Y$, where $f \in C^\infty(\mathcal{M}, \mathbb{R})$
- $\nabla_X (fY) = f \nabla_X Y + (Xf)Y$ for $f \in C^\infty(\mathcal{M}, \mathbb{R})$, it satisfies the Leibnitz rule in the second variable

We call $\nabla_X Y$ the *covariant derivative of Y in the direction of X*. A connection ∇ defines the parallel transport of a vector along a curve. Given a curve $\gamma : [0, 1] \to \mathcal{M}$ and a vector $v_0 \in T_{\gamma(0)}\mathcal{M}$, there exists a unique parallel vector field $V : [0, 1] \to T\mathcal{M}$ along γ such that $V(0) = v_0$ [Lee, 1997]. Recall: a vector field V along γ means $\pi(V(t)) = \gamma(t)$. The uniqueness and existence is a consequence of the vector field V being the solution of $\nabla_{\dot\gamma(t)} V(t) = 0$ defining a linear ordinary differential equation with initial condition $V(0) = v_0$. This vector field $V(t)$ is called the *parallel transport* of v_0 along γ. It is "parallel" in the sense that the transported vector does not change within the tangent space. See Figure 2.1 for an illustration in 2D.

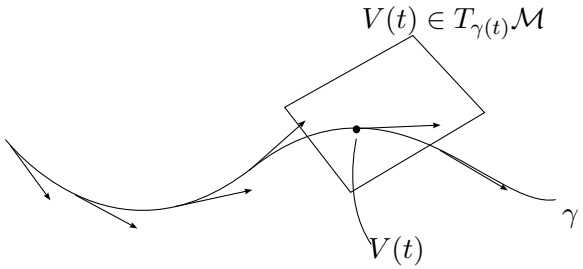


Figure 2.1. A parallel vector field $V(t)$ along a curve γ. Each $V(t) \in T_{\gamma(t)}\mathcal{M}$ and $\nabla_{\dot\gamma(t)} V(t) = 0$, so each vector that is drawn is parallel to each other.

In the following, we show how a connection can be written as a matrix 1-form: Let (E_1, E_2, E_3) be a smooth local frame for $T\mathcal{M}$ and $(\mathcal{E}^1, \mathcal{E}^2, \mathcal{E}^3)$ its dual coframe for $T^*\mathcal{M}$ i.e. $\mathcal{E}^i(E_j) = \delta^i_j$. Using the definition of an affine connection, we can write vector fields $X, Y \in \mathcal{T}(\mathcal{M})$ with the local frames, i.e.

$$\nabla_X Y = \nabla_{X^i E_i}(Y^i E_j) = X^i \nabla_{E_i}(Y^i E_j) = \underbrace{X^i Y^j}_{\in C^\infty(\mathcal{M},\mathbb{R})} \cdot \underbrace{\nabla_{E_i} E_j}_{\in \mathcal{T}(\mathcal{M})} + \underbrace{X^i E_i(Y^j)}_{\in C^\infty(\mathcal{M},\mathbb{R})} \cdot \underbrace{E_j}_{\in \mathcal{T}(\mathcal{M})} \in \mathcal{T}(\mathcal{M}) \quad (2.1)$$

Because $\nabla_{E_i} E_j \in \mathcal{T}(\mathcal{M})$, we can also write this in the local frame (E_1, E_2, E_3), that is, for all pairs of indices $i, j \in \{1, 2, 3\}$, we write

$$\nabla_{E_i} E_j = \Gamma^k_{ij} E_k$$

The Γ^k_{ij} are called *Christoffel symbols* or *connection coefficients*. With these coefficients, we can rewrite Equation 2.1 to

$$\nabla_X Y = X^i Y^j \Gamma^k_{ij} E_k + X^i E_i(Y^j) E_j = X^i (X^i Y^j \Gamma^k_{ij} + E_i(Y^k)) E_k \quad (2.2)$$

where the dummy index j was renamed to k in the last term (we are using Einstein summation convention, so the indices are independent). This derivation shows that the choice of Γ^k_{ij} fully determine the

connection ∇. We can now define another connection $\hat{\nabla}$ with 1-forms:

$$\hat{\nabla}_X(Y^j E_j) \triangleq X(Y^j)E_j + Y^j \hat{\nabla}_X E_j \qquad \text{where } \hat{\nabla}_X E_j = \omega_j^k(X)E_k \tag{2.3}$$

and ω_j^k are 1-forms. It thus satisfies the Leibnitz rule in the second variable. To show that it is indeed a connection, it must also satisfy C^∞ linearity in the direction of X, i.e.

$$\hat{\nabla}_X E_j = \hat{\nabla}_{X^i E_i} E_j = \omega_j^k(X^i E_i)E_k = X^i \omega_j^k(E_i)E_k = X^i \hat{\nabla}_{E_i} E_j \tag{2.4}$$

where we used C^∞ linearity of 1-forms. Notice the relation between the Christoffel symbols and the ω 1-form:

$$\hat{\nabla}_{E_i} E_j = \omega_j^k(E_i)E_j = \Gamma_{ij}^k E_k \qquad \text{with } \Gamma_{ij}^k = \omega_j^k(E_i)$$

Thus, the 3×3 1-forms ω_j^k define a connection and are called a connection 1-form, or a matrix-valued connection 1-form. In the coframe, these are written as $\omega_j^k = \Gamma_{ij}^k \mathcal{E}^i$.

2.2 Frame fields

Recall from the introduction that we are trying to find a hexahedral mesh for some geometric object. In the following sections, the key idea to tackle this problem through *seamless maps* and *Integer-Grid Maps* is discussed, and how the problem of finding such maps is relaxed through *frame fields*.

2.2.1 Integer-Grid Maps

In 2D, Integer-Grid Maps have proven to be reliable in generating the analogue of hexahedral meshes, quad meshes [Bommes et al., 2013]. The goal is to use the same approach, but in three dimensions. The objective is to find a seamless map $\phi : \mathcal{M} \to \mathbb{R}^3$, which smoothly maps the 3-dimensional object to be meshed into $\mathbb{R}^3$. In a later step, after finding a good seamless map ϕ, it is quantized to get an Integer-Grid Map f, where the hexahedral elements can be extracted from, see Figure 2.2 for an illustration. Directly

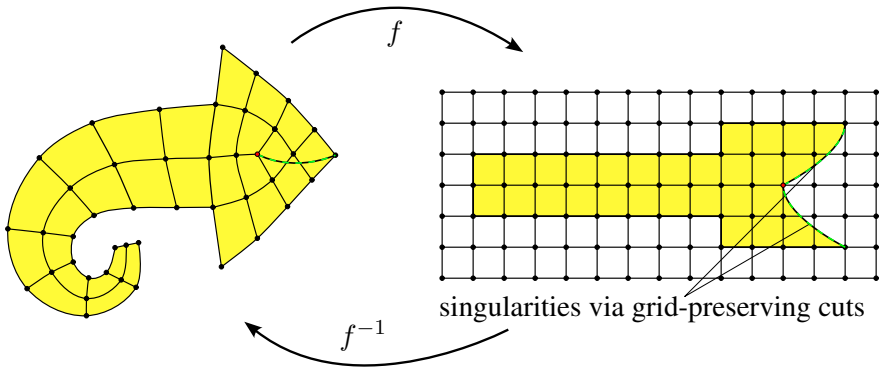


Figure 2.2. Idea of quadrangulation with Integer-Grid Maps in 2D (Figure from [Pietroni et al., 2022])

searching for good Integer-Grid Maps that minimize distortion and satisfy the boundary alignment condition is a hard mixed-integer and non-convex optimization problem, for which no current optimization technique is available that would result in an acceptable solution.

2.2.2 Frame fields as a relaxation

By taking the Jacobian of the seamless map, $\nabla\phi$, we get a mapping $\nabla\phi : \mathcal{M} \to \mathbb{R}^{3\times3}$ of *frames*, a *frame field*. The idea of frame fields is to then search for an approximation of $\nabla\phi$. If $F : \mathcal{M} \to \mathbb{R}^{3\times3}$ is the approximation of $\nabla\phi$, we can solve for $\phi : \mathcal{M} \to \mathbb{R}^3$ with

$$\min_{\phi} \int_{\mathcal{M}} ||\nabla\phi F - \mathrm{Id}\,||^2 \tag{2.5}$$

where $|| \cdot ||$ is the Frobenius norm. If this is sufficiently small, then $F^{-1} \approx \nabla\phi$ and the extracted hexahedral mesh closely follow the integer isolines. A frame locally represents the edges of a deformed cube. We can think of a frame field as the composition of three vector fields and as a relaxation of the original problem. However, frame fields can contain singularities that the underlying vector fields do not contain [Nieser et al., 2011] and contain types of singularities that cannot appear in hex meshes and are non-meshable [Liu et al., 2018, Liu and Bommes, 2023]. The problem of non-meshability is not discussed further here, but it is something to be aware of.

We treat a frame F as a set of 3 linearly independent vectors $\{F_1, F_2, F_3\}$ which we can collect into a matrix $F = (F_1, F_2, F_3) \in \mathbb{R}^{3\times3}$. Notice that many of these frames are equivalent. For example, the frames $(F_1, F_2, F_3) \sim (-F_1, F_2, F_3) \sim (F_2, F_1, F_3)$ all represent the same frame. There are 2^3 choices of the sign, and 3! possible permutations, which gives $2^3 \cdot 3! = 48$ equivalent frames. Since we want non-degenerate frames and the same orientation through the grid, the constraint $\det(F) > 0$ is imposed, which leaves 24 equivalent frames. Equivalence of frames is then defined as

$$F_u \sim F_v \iff \exists R \in \mathcal{O} : F_u = F_v R$$

where $\mathcal{O}$ is the *chiral cubical symmetry group* [Nieser et al., 2011]. These symmetries complicate the optimization process for frame fields. When optimizing the smoothness of a frame field, frames have to be compared. To compare frames, we need to check if the vectors making up the frames point in the same direction or if they have to be rotated by some matrix $M \in \mathrm{SO}(3)$ first, to make them point in the same direction. But now, the mentioned symmetry of representing the frames in $\mathbb{R}^3$ make M ambiguous ($M_1 = M_2 R, R \in \mathcal{O}$). Therefore, we instead use the spherical harmonics based approach to represent frames [Huang et al., 2011]. The idea is to represent frames as rotations of the polynomial $\mathbb{P} = x^4 + y^4 + z^4$. Under the restriction of the 4th degree polynomials to the sphere, that is $\mathbb{S}^2 \to \mathbb{R}$, the rotated polynomials are lifted to a 9D vector and expressed in the spherical harmonics basis. The restriction makes the rotated polynomials naturally invariant under $\mathcal{O}$. For example, the axis-aligned frame $F = (e_1, e_2, e_3)$ corresponds to $\mathbb{P}$, as $\mathbb{P}$ under the restriction to the sphere takes its maxima at $\pm e_i$. The action of rotating 4th degree polynomials can be represented by so called 9×9 ***Wigner D-matrices***. These Wigner D-matrices and rotation matrices in $\mathbb{R}^3$ both represent the same elements of the group $\mathrm{SO}(3)$, just act on different vector spaces. It is clear that this is a relaxation: Since rotations in 3D only exhibit 3 degrees of freedom, not all 9D spherical harmonic vectors are valid rotations of $x^4 + y^4 + z^4$.

While optimizing the frame field, some kind of "average" of two frames will be done. This averaging makes sense in the spherical harmonics space, but the result may be outside the valid space of frames. A projection from the spherical harmonics space back to the valid space of frames will be needed, see Figure 2.3 [Ray and Sokolov, 2015].

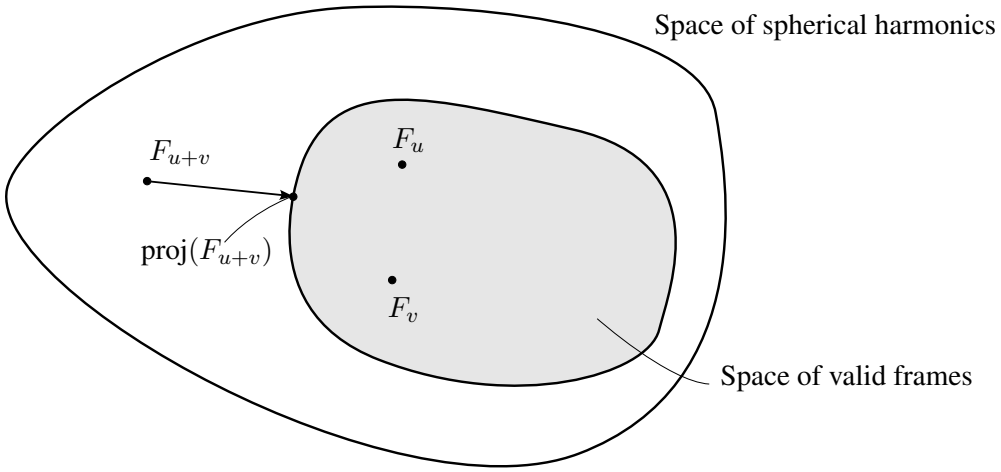


Figure 2.3. Combination of two valid frames leads to a resulting frame outside the valid frames. A projection to the nearest valid frame is done.

When a frame F has orthonormal columns, the resulting hex elements resemble unit cubes. By introducing a metric g, we can control the size and shear of the cubes by relaxing the orthonormality constraint to g-orthonormality when $F^\top gF = \mathrm{Id}$ holds. We can decompose the frame field F into a rotational part $R : \mathcal{M} \to \mathrm{SO}(3)$ and a symmetric metric part $g^{-1/2}$ (akin to the polar decomposition of linear transformations) [Panozzo et al., 2014], that is $F = g^{-1/2}R$, see Figure 2.4. Requirements for a

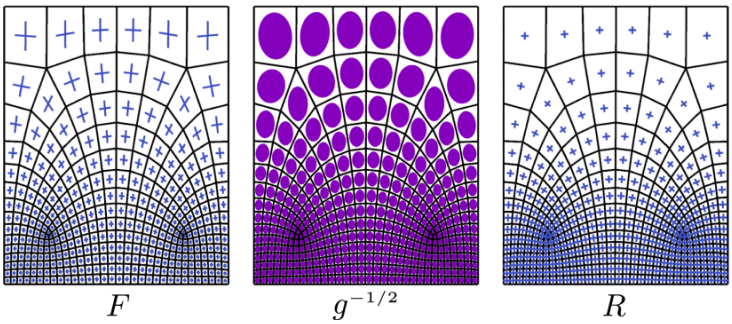


Figure 2.4. Factorization of a g-orthogonal frame field into a symmetric metric part $g^{-1/2}$ and a rotational part R (Figure from [Fang et al., 2023]).

frame field are usually formulated as

- Smoothness: by minimizing the Dirichlet energy $E(F) = \int_{\mathcal{M}} ||\nabla F||^2 dV$, the smoothness within the frame field is maximized which minimizes distortion of the seamless map;
- Boundary alignment: one of the columns of the frame field should match with the surface normal such that the hex elements accurately trace the surface of the object;
- Integrability: to find ϕ with Equation 2.5, the frame field must be as close to integrable as possible.

Chapter 3

Connection One-Form ω

Recall that our goal is to find a frame field $F : \mathcal{M} \to \mathbb{R}^{3\times 3}$, such that the target parametrization of ϕ can sufficiently reduce the difference between $\nabla\phi$ and F^{-1}. For this, F^{-1} needs to stay as close to being locally integrable as possible. In this chapter, by starting from the local integrability condition, we formulate a connection 1-form ω which is used to measure the Dirichlet energy in the new metric g, such that the frames stay close to being g-orthonormal ($F^{-1}gF = \mathrm{Id}$). We follow the steps in *Metric-Driven 3D Frame Field Generation* [Fang et al., 2023].

3.1 Local integrability

A vector field U is integrable if and only if $\nabla \times U = 0$, which means the vector field has vanishing curl [Papachristou, 2020]. Although in general it is more complicated [Nieser et al., 2011], we can think of a frame field F as the composition of 3 vector fields

$$F = \begin{bmatrix} | & | & | \\ F_1 & F_2 & F_3 \\ | & | & | \end{bmatrix}$$

where $F_i : \mathbb{R}^3 \to \mathbb{R}^3$ are vector fields. To achieve local integrability for F^{-1}, we therefore want

$$\nabla \times F^{-1} \stackrel{!}{=} 0$$

where the curl is applied to each column. We can express this more naturally with the language of differential forms by writing the curl as the exterior derivative d of a 1-form α. A 1-form (more generally, a differential form) is closed, if $d\alpha = 0$. We construct a vector-valued 1-form out of our frame field, given $\boldsymbol{p} = (x, y, z)^\top$ in Euclidean coordinates

$$\alpha \triangleq F^{-1}d\boldsymbol{p} = R^\top g^{1/2} d\boldsymbol{p}$$

where $d\boldsymbol{p} = (dx, dy, dz)^\top$ is the common orthonormal 1-form basis $\mathbb{E}^3$. Local integrability of F^{-1} is then formulated as the closedness of α, i.e.

$$F^{-1} \text{ locally integrable } \iff \mathbf{0} = d\alpha$$

Some reformulations yield:

$$\begin{aligned} \mathbf{0} = d\alpha &= d(F^{-1}d\boldsymbol{p}) = d(R^\top g^{1/2} d\boldsymbol{p}) \\ &= dR^\top \wedge (g^{1/2}d\boldsymbol{p}) + R^\top d(g^{1/2}d\boldsymbol{p}) \\ &= R^\top(\omega \wedge (g^{1/2}d\boldsymbol{p}) + d(g^{1/2}d\boldsymbol{p})) \end{aligned}$$

which we can simplify to

$$\mathbf{0} = \omega \wedge (g^{1/2} d\boldsymbol{p}) + d(g^{1/2} d\boldsymbol{p}) \tag{3.1}$$

where the Leibnitz Rule for the exterior derivative is applied and $\wedge$ is the exterior product between a matrix-valued 1-form and a vector-valued 1-form, i.e. 1-forms in each component and the matrix vector product uses $\wedge$ as the multiplication, see Section 3.3 for the evaluation. We further define

$$\omega = RdR^\top \in \mathfrak{so}(3)$$

which is an antisymmetric matrix-valued 1-form. To see this, we differentiate the orthogonality condition of the rotation matrix R (we assume R rotates about the axis a with angle θ):

$$\begin{aligned} \mathrm{Id} &= RR^\top \\ d(\mathrm{Id}) &= d(RR^\top) \\ \mathbf{0} &= dRR^\top + RdR^\top \\ \mathbf{0} &= (RdR^\top)^\top + RdR^\top \\ -(RdR^\top)^\top &= RdR^\top \end{aligned}$$

The Lie algebra $\mathfrak{so}(3)$ consists of all antisymmetric 3×3 matrices. Elements of $\mathfrak{so}(3)$ are infinitesimal rotations, that is, they are tangent to the manifold $\mathrm{SO}(3)$ at the element Id. Indeed, we characterised the elements of $\mathfrak{so}(3)$ by taking the derivative at Id, which is one definition to get to the tangent space. Thus, we can use ω as a connection 1-form to do the alignment of frames in the new metric g, i.e. the parallel transport is done with ω and then compared.

To solve for local integrability, we find ω such that the 1-form α is closed, then try to match the ω with R. This can be expressed as

$$\min_{R \in \mathrm{SO}(3)} ||RdR^\top - \omega||^2, \tag{3.2}$$

where ω is determined by g through a system of linear equations (see 3.3). In general, Equation 3.2 cannot be minimised to zero, therefore we solve for the nearly integrable 3D rotation field R.

3.2 Smoothness measure

Many frame field generation methods rely on maximising smoothness through the minimization of the Dirichlet energy. Here, we show that integrability through ω is closely related to the usual Dirichlet energy. We can take Equation 3.2 as a smoothness measure and reformulate, i.e.

$$||RdR^\top - \omega||^2 = || - dRR^\top - \omega||^2 = || - dRR^\top R - \omega R||^2 = ||dR + \omega R||^2. \tag{3.3}$$

By defining

$$\mathcal{D}R \triangleq dR + \omega R, \tag{3.4}$$

we can measure the smoothness and the integrability with a single energy $\int_{\mathcal{R}} ||\mathcal{D}R||^2$. Now, when g is constant in euclidean coordinates (x, y, z), Equation 3.1 tells us that $\omega = 0$, which reduces the smoothness measure $||\mathcal{D}R||$ to $||dR||^2$, which corresponds to the usual Dirichlet energy. In fact, $\mathcal{D}R$ corresponds to the covariant derivative of R under the connection ω, which shows that local integrability is related to the covariant-based Dirichlet energy. Special care must be taken as we abuse the notation of the covariant derivative. The covariant derivative $\mathcal{D}$ acts on each column of R separately. We define

$$\mathcal{D}R_i \triangleq \nabla_{\dot{\gamma}(t)} R_i \triangleq dR_i + \omega R_i$$

where dR_i is the derivative of each entry with respect to the angle θ and γ is some curve on the manifold (again, we assume R rotates about an axis a with angle θ).

3.3 Connection evaluation

To find ω, we use Equation 3.1

$$\mathbf{0} = \omega \wedge (g^{1/2} d\boldsymbol{p}) + d(g^{1/2} d\boldsymbol{p})$$

and reformulate into a linear system. We represent the antisymmetric matrix-valued 1-form ω

$$\omega = \begin{bmatrix} 0 & \omega_{12} & -\omega_{31} \\ -\omega_{12} & 0 & \omega_{23} \\ \omega_{31} & -\omega_{23} & 0 \end{bmatrix}$$

by $\begin{bmatrix} \omega_{23} & \omega_{31} & \omega_{12} \end{bmatrix} = \begin{bmatrix} dx & dy & dz \end{bmatrix} W$. We write $W = \begin{bmatrix} W_1, W_2, W_3 \end{bmatrix}$, $W_i \in \mathbb{R}^3$. That is, W is the matrix with the coefficients for the 1-forms, e.g. $W_1 = \begin{bmatrix} (\omega^{23})_1, (\omega^{23})_2, (\omega^{23})_3 \end{bmatrix}^\top$. Recall, a 1-form can be expressed as

$$\omega_{ij} = (\omega^{ij})_1 dx + (\omega^{ij})_2 dy + (\omega^{ij})_3 dz$$

So e.g. for ω_{23} we get

$$\omega_{23} = \begin{bmatrix} dx & dy & dz \end{bmatrix} W_1 = (\omega^{23})_1 dx + (\omega^{23})_2 dy + (\omega^{23})_3 dz.$$

We also write $A = g^{1/2} = \begin{bmatrix} A^1, A^2, A^3 \end{bmatrix}$. Starting with the first part of Equation 3.1, we get

$$\begin{aligned} \omega \wedge g^{1/2} d\boldsymbol{p} &= \begin{bmatrix} 0 & \omega_{12} & -\omega_{31} \\ -\omega_{12} & 0 & \omega_{23} \\ \omega_{31} & -\omega_{23} & 0 \end{bmatrix} \bigwedge \begin{bmatrix} A_1^1 & A_1^2 & A_1^3 \\ A_2^1 & A_2^2 & A_2^3 \\ A_3^1 & A_3^2 & A_3^3 \end{bmatrix} \begin{bmatrix} dx \\ dy \\ dz \end{bmatrix} \\ &= \begin{bmatrix} +\omega_{12} \wedge (A_2^1 dx + A_2^2 dy + A_2^3 dz) - \omega_{31} \wedge (A_3^1 dx + A_3^2 dy + A_3^3 dz) \\ -\omega_{12} \wedge (A_1^1 dx + A_1^2 dy + A_1^3 dz) + \omega_{23} \wedge (A_3^1 dx + A_3^2 dy + A_3^3 dz) \\ +\omega_{31} \wedge (A_1^1 dx + A_1^2 dy + A_1^3 dz) - \omega_{23} \wedge (A_2^1 dx + A_2^2 dy + A_2^3 dz) \end{bmatrix} \end{aligned}$$

It will get really messy if we calculate each component here, so let us calculate one component separately first:

$$\begin{aligned} \omega_{ij} \wedge (A_k^1 dx + A_k^2 dy + A_k^3 dz) &= ((\omega^{ij})_1 dx + (\omega^{ij})_2 dy + (\omega^{ij})_3 dz) \wedge (A_k^1 dx + A_k^2 dy + A_k^3 dz) \\ &= (\omega^{ij})_1 A_k^2 dx \wedge dy + (\omega^{ij})_1 A_k^3 dx \wedge dz \\ &+ (\omega^{ij})_2 A_k^1 dy \wedge dx + (\omega^{ij})_2 A_k^3 dy \wedge dz \\ &+ (\omega^{ij})_3 A_k^1 dz \wedge dx + (\omega^{ij})_3 A_k^2 dz \wedge dy \\ &= ((\omega^{ij})_1 A_k^2 - (\omega^{ij})_2 A_k^1) dx \wedge dy \\ &+ ((\omega^{ij})_2 A_k^3 - (\omega^{ij})_3 A_k^2) dy \wedge dz \\ &+ ((\omega^{ij})_3 A_k^1 - (\omega^{ij})_1 A_k^3) dz \wedge dx \end{aligned}$$

where we use the fact that $dx \wedge dx = 0$ and $dx \wedge dy = -dy \wedge dx$. We can clean up the above expression using the cross product:

$$\begin{bmatrix} ((\omega^{ij})_2 A_k^3 - (\omega^{ij})_3 A_k^2) \\ ((\omega^{ij})_3 A_k^1 - (\omega^{ij})_1 A_k^3) \\ ((\omega^{ij})_1 A_k^2 - (\omega^{ij})_2 A_k^1) \end{bmatrix}^\top \begin{bmatrix} dy \wedge dz \\ dz \wedge dx \\ dx \wedge dy \end{bmatrix} = \begin{bmatrix} W_i \times A^k \end{bmatrix}^\top \begin{bmatrix} dy \wedge dz \\ dz \wedge dx \\ dx \wedge dy \end{bmatrix}$$

We use the fact that $\begin{bmatrix} A_1, A_2, A_3 \end{bmatrix} = \begin{bmatrix} A^1, A^2, A^3 \end{bmatrix}$ because A is symmetric, and W_1 corresponds to ω_{23}, W_2 to ω_{31} and W_3 to ω_{12}. We continue with the second part of Equation 3.1:

$$d(g^{1/2} d\boldsymbol{p}) = d(A d\boldsymbol{p}) = d\left(\begin{bmatrix} A_1^1 & A_1^2 & A_1^3 \\ A_2^1 & A_2^2 & A_2^3 \\ A_3^1 & A_3^2 & A_3^3 \end{bmatrix} \begin{bmatrix} dx \\ dy \\ dz \end{bmatrix} \right)$$

Again, we can do this separately for each row (we use the fact that the exterior derivative d is the ordinary differential for a smooth function):

$$
\begin{aligned}
& d(A_k^1 dx + A_k^2 dy + A_k^3 dz) \\
&= dA_k^1 \wedge dx + dA_k^2 \wedge dy + dA_k^3 \wedge dz \\
&= \frac{\partial A_k^1}{\partial x} dx \wedge dx + \frac{\partial A_k^1}{\partial y} dy \wedge dx + \frac{\partial A_k^1}{\partial z} dz \wedge dx \\
&+ \frac{\partial A_k^2}{\partial x} dx \wedge dy + \frac{\partial A_k^2}{\partial y} dy \wedge dy + \frac{\partial A_k^2}{\partial z} dz \wedge dy \\
&+ \frac{\partial A_k^3}{\partial x} dx \wedge dz + \frac{\partial A_k^3}{\partial y} dy \wedge dz + \frac{\partial A_k^3}{\partial z} dz \wedge dz \\
&= \left(\frac{\partial A_k^3}{\partial y} - \frac{\partial A_k^2}{\partial z} \right) dy \wedge dz + \left(\frac{\partial A_k^1}{\partial z} - \frac{\partial A_k^3}{\partial x} \right) dz \wedge dx + \left(\frac{\partial A_k^2}{\partial x} - \frac{\partial A_k^1}{\partial y} \right) dx \wedge dy \\
&= (\nabla \times A_k)^\top \begin{bmatrix} dy \wedge dz \\ dz \wedge dx \\ dx \wedge dy \end{bmatrix}
\end{aligned}
$$

Finally, we can put everything together:

$$
\begin{aligned}
\mathbf{0} &= \begin{bmatrix} (W_3 \times A^2 - W_2 \times A^3 + \nabla \times A^1)^\top \\ (W_1 \times A^3 - W_3 \times A^1 + \nabla \times A^2)^\top \\ (W_2 \times A^1 - W_1 \times A^2 + \nabla \times A^3)^\top \end{bmatrix} \begin{bmatrix} dy \wedge dz \\ dz \wedge dx \\ dx \wedge dy \end{bmatrix} \\
\iff & \begin{bmatrix} (W_2 \times A^3 - W_3 \times A^2)^\top \\ (W_3 \times A^1 - W_1 \times A^3)^\top \\ (W_1 \times A^2 - W_2 \times A^1)^\top \end{bmatrix} \begin{bmatrix} dy \wedge dz \\ dz \wedge dx \\ dx \wedge dy \end{bmatrix} = \begin{bmatrix} (\nabla \times A^1)^\top \\ (\nabla \times A^2)^\top \\ (\nabla \times A^3)^\top \end{bmatrix} \begin{bmatrix} dy \wedge dz \\ dz \wedge dx \\ dx \wedge dy \end{bmatrix}
\end{aligned}
$$

We take the curl to the other side and switch order on the left-hand side to cancel the -1. As we are only interested in the 9 components of W, we omit the two-form basis and transform into a 9×9 linear system for W. We define $A_\times$ and $\mathrm{vec}(\cdot)$ as

$$
A_\times = \begin{bmatrix} 0 & -A_\times^3 & A_\times^2 \\ A_\times^3 & 0 & -A_\times^1 \\ -A_\times^2 & A_\times^1 & 0 \end{bmatrix}, \mathrm{vec}(W) = \begin{bmatrix} W_1 \\ W_2 \\ W_3 \end{bmatrix}
$$

with $A_\times^i$ defined as

$$
A_\times^i = \begin{bmatrix} A_1^i & A_2^i & A_3^i \end{bmatrix}_\times = \begin{bmatrix} 0 & -A_3^i & A_2^i \\ A_3^i & 0 & -A_1^i \\ -A_2^i & A_1^i & 0 \end{bmatrix}
$$

and $\mathrm{vec}(\cdot)$ turns a 3×3-matrix into a 9×1-vector by stacking the columns. With these two definitions, we can transform the above equality into a linear system

$$
A_\times \, \mathrm{vec}(W) = \mathrm{vec}(\nabla \times A) \tag{3.5}
$$

where $\nabla \times A$ is just the curl applied to each column. This transformation can be checked by laboriously plugging in the definitions and comparing the coefficients. With tedious brute-force calculations, one can show that $\det(A_\times) = -2 \det(A)^3 = -2 \det(g)^{3/2} < 0$, which means this is a linear system that is solvable and can be used to calculate W at a point.

Chapter 4

Alignment of frames

When the metric g is globally flat with no additional constraints like boundary alignment, a globally integrable frame field is achievable by parallel transport. No singularities in the frame field would be present. However, this is not possible in general. In this section, by starting from the assumption that the metric is flat, we recover the rotation happening by parallel transport under the connection ω which allows us to pull back the covariant derivative based Dirichlet energy and minimize as if we were minimizing the Dirichlet energy in Euclidean coordinates.

The smoothness measure $\mathcal{D}R$ is zero if all frames are parallel to each other under the connection ω, that is, if all frames are the same when parallel transported to the same point. This is deliberately kept vague as the parallel transport is path dependent in general (parallel transport is path independent if the metric is flat). To get a feeling for the problem statement, see Figure 4.1.

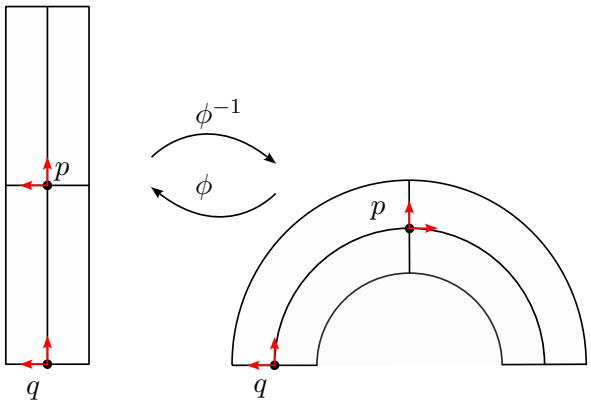


Figure 4.1. Under the metric induced by ϕ, the frame at q is parallel to p. To compare them, we need to recover how p is rotated under ω.

As we want parallelness under ω along a path γ, we set $\mathcal{D}R \stackrel{!}{=} 0$ and treat it as a set of differential equations. Again, as R essentially consists of three rotation fields, what is meant that $\mathcal{D}R_i = 0$ for each field.

Let p, q be points on the manifold and $\ell : [0,1] \to \mathcal{M}$ a path connecting them, with $\ell(0) = q$ and $\ell(1) = p$. Our differential equation is then given by

$$\mathbf{0} = dR_i(\ell(t)) + \omega(\ell(t))R_i(\ell(t)) \tag{4.1}$$

which we rearrange to

$$\underbrace{-\omega(\ell(t))}_{\boldsymbol{A}(t)}\underbrace{R_i(\ell(t))}_{\boldsymbol{\alpha}(t)} = \underbrace{dR_i(\ell(t))}_{\dot{\boldsymbol{\alpha}}(t)} \tag{4.2}$$

where we explicitly indicate that ω depends on the point evaluated and is not constant. Do not get

confused what we are differentiating: The differential equation consists of mappings

$$\omega : \mathcal{M} \to \mathbb{R}^{3\times 3} \text{ and } R_i : \mathcal{M} \to \mathbb{R}^3.$$

Thus, the concatenation with $\ell : [0,1] \to \mathcal{M}$ gives us mappings $\boldsymbol{A} : [0,1] \to \mathbb{R}^{3\times 3}$ and $\boldsymbol{\alpha} : [0,1] \to \mathbb{R}^3$ where it makes sense to take the derivative with respect only to t. This differential equation has the form of a *matrix differential equation*, for which a general closed form solution is hard to find. However, when ω and its integral $\int_0^t -\omega(\ell(s))ds$ commute, the general solution $\boldsymbol{\alpha}(t) = e^{\int_0^t \boldsymbol{A}(s)ds}\boldsymbol{\alpha}(0)$ exists, which is the case when ω along ℓ rotates around the same axis (the same way rotation matrices in 3D only commute when they rotate around the same axis). Plugging in values 0 and 1 for t and assuming ω commutes with its integral gives the solution

$$\boldsymbol{\alpha}(1) = R(\ell(1)) = R(p) = e^{\int_0^1 \boldsymbol{A}(s)ds}R(\ell(0)) = e^{\int_0^1 \boldsymbol{A}(s)ds}R(q),$$

which shows that we can align frames with a single rotation matrix R_{qp}, i.e.

$$R(p) = R_{qp}R(q)$$

which corresponds to the parallel transport under the connection ω of the frame along ℓ. Discretization of the covariant based Dirichlet energy is then given by

$$||\mathcal{D}R||_{\text{discretized}} = ||R_{qp}R(q) - R(p)||^2. \tag{4.3}$$

We parametrize the path by $\ell(0) = a, \ell(1) = b$ and use numerical integration for R_{ab}. For this, the path is cut into n small segments, i.e.

$$R_{ab} = R_n R_{n-1} \cdots R_1$$

where $R_i = \exp(-\omega(\dot{\ell}(i\gamma))\gamma)$, γ is the length of a segment and $\dot{\ell}(s) = \frac{\partial \ell}{\partial s}(s)$. By cutting up the path into small segments, ω should rotate around the same axis along ℓ, which as described above allows us to use the general solution $R_i = \exp((-\int_\gamma \omega d\boldsymbol{p})_\times)$. Calculating the exponential map of an antisymmetric matrix (which $\omega \in \mathfrak{so}(3)$ is) yields a rotation matrix $R \in \mathrm{SO}(3)$ and can be done with Rodrigues' formula:

$$\exp(u_\times) = \mathrm{Id} + \sin(\theta)\hat{u}_\times + (1 - \cos(\theta))\hat{u}_\times^2$$

where $\theta = ||u||_2$ is the rotation angle and $\hat{u} = u/\theta$ is the rotation axis. We use the trapezoidal rule to evaluate the short interval of the integral of W, which is given by

$$R_{ab} = \exp\left(\left(\frac{1}{2}(W_a + W_b)^\top (b - a)\right)_\times\right)$$

where W_a, W_b are solved for by the 9×9 linear system given by $A_\times$ and $\nabla \times A$ from Equation 3.3.

4.1 Piecewise linear discretization

We discretize our metric field with a tetrahedral mesh $\mathcal{T}$. At each vertex, we attach a metric and linearly interpolate with barycentric coordinates within a tet.

Let $A_i \in \mathbb{R}^{3x3}, i \in \{1,2,3,4\}$ represent the square root matrices of the metric at the vertices $v_i \in \mathbb{R}^3$ of a tet, i.e., such that $A_i^2 = g(v_i)$. We represent a tet by its four vertices $\{v_1, v_2, v_3, v_4\}$. With barycentric coordinates, any point p within the tet can then be represented as

$$p = \alpha v_1 + \beta v_2 + \gamma v_3 + \delta v_4$$

with $\alpha, \beta, \gamma, \delta \geq 0$ and $\alpha + \beta + \gamma + \delta = 1$. This is a linear transformation between two coordinate systems, which we can write in matrix form as

$$\underbrace{\begin{pmatrix} | & | & | & | \\ v_1 & v_2 & v_3 & v_4 \\ | & | & | & | \\ 1 & 1 & 1 & 1 \end{pmatrix}}_{T^{-1}} \underbrace{\begin{pmatrix} \alpha \\ \beta \\ \gamma \\ \delta \end{pmatrix}}_{\lambda} = \underbrace{\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}}_{p} \iff T^{-1}\boldsymbol{\lambda} = \boldsymbol{p} \iff \boldsymbol{\lambda} = T\boldsymbol{p}.$$

The matrix T always exists because $v_1, \ldots, v_4$ are linearly independent, else it would not be a valid tetrahedron. By denoting $T = \{t_{ij}\}_{i,j \in \{1,\ldots,4\}}$, we can write our barycentric functions as

$$\begin{aligned} \alpha(x, y, z) &= t_{11}x + t_{12}y + t_{13}z + t_{14} \\ \beta(x, y, z) &= t_{21}x + t_{22}y + t_{23}z + t_{24} \\ \gamma(x, y, z) &= t_{31}x + t_{32}y + t_{33}z + t_{34} \\ \delta(x, y, z) &= t_{41}x + t_{42}y + t_{43}z + t_{44} \end{aligned}$$

The convex combination

$$A(x, y, z) = \alpha A_1 + \beta A_2 + \gamma A_3 + \delta A_4$$

is the metric prescribed in the tet. To find $\nabla \times A$, let $A = (A^1, A^2, A^3)$. We will need the derivatives for the curl, so let

$$(A^i_j)_x \triangleq \frac{\partial A^i_j}{\partial x}$$

be the derivative with respect to x of entry i, j. For example, $(A^i_j)_x$ is given by

$$(A^i_j)_x = \alpha_x (A_1)^i_j + \beta_x (A_2)^i_j + \gamma_x (A_3)^i_j + \delta_x (A_4)^i_j = t_{11}(A_1)^i_j + t_{21}(A_2)^i_j + t_{31}(A_3)^i_j + t_{41}(A_4)^i_j$$

If we write $T = (T^1, T^2, T^3, T^4)$ and collect $(A_k)^i_j$ into a vector

$$\bar{A^i_j} = \begin{pmatrix} (A_1)^i_j \\ (A_2)^i_j \\ (A_3)^i_j \\ (A_4)^i_j \end{pmatrix}$$

this can be shortened to $\bar{A^i_j}^\top T^1 = (A^i_j)_x$. Analogously, we get

$$(A^i_j)_y = \bar{A^i_j}^\top T^2 \text{ and } (A^i_j)_z = \bar{A^i_j}^\top T^3$$

The curl is then given by

$$\nabla \times A^i = \begin{pmatrix} (A^i_3)_y - (A^i_2)_z \\ (A^i_1)_z - (A^i_3)_x \\ (A^i_2)_x - (A^i_1)_y \end{pmatrix} = \begin{pmatrix} \bar{A^i_3}^\top T^2 - \bar{A^i_2}^\top T^3 \\ \bar{A^i_1}^\top T^3 - \bar{A^i_3}^\top T^1 \\ \bar{A^i_2}^\top T^1 - \bar{A^i_1}^\top T^2 \end{pmatrix}$$

and $\nabla \times A = \nabla \times (A^1, A^2, A^3)$. Notice that the curl is constant within a tetrahedron.

4.2 Recursive subdivision

Whenever the rotation coefficient R_{qp} is needed between some points p and q, it is unclear ahead of time how many sampling points on the line ℓ are needed such that R_{qp} accurately describes how the frame rotates along ℓ. We apply a recursive subdivision scheme to recursively sample more points on ℓ only

where is needed until sampling more points leads to no noticable improvement anymore (see Figure 4.2). We begin by calculating R_{ab} with just the endpoints. This coefficient is then compared to the result if the midpoint was sampled aswell, so if

$$\frac{||R_{ab} - R_{am} \cdot R_{mb}||_2^2}{\ell^2} < \varepsilon$$

then no measurable improvement happened. We divide by the length $\ell = ||a - b||_2$ of the segment $\vec{ab}$, because the segments get smaller and we want the tolerance ε to remain the same. This approach has the advantage of only sampling points where there is improvement. The algorithm for this simple recursive

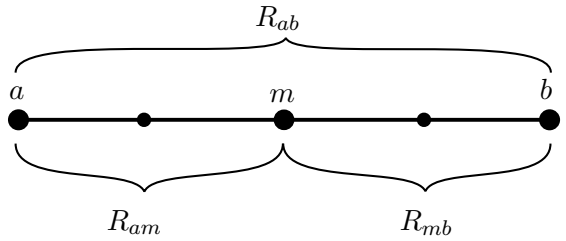


Figure 4.2. Points are recursively sampled as midpoints between the line $\vec{ab}$. If sampling more points within some line segment leads to noticable improvement, more points are sampled.

scheme is shown below.

Algorithm 1 Recursive Subdivision

recursiveDivide (a, b)
    $R_1 \leftarrow$ *Rotation(a,b)*
    $\ell \leftarrow$ *length*$(a - b)$
    midpoint $\leftarrow \frac{a+b}{2}$
    if $\frac{||R_1 - \text{Rotation}(a,\text{midpoint}) \cdot \text{Rotation}(\text{midpoint},b)||_2^2}{\ell^2} < \varepsilon$
        return R_1
    else
        return *recursiveDivide*$(a, \text{midpoint}) \cdot$ *recursiveDivide*$(\text{midpoint}, b)$

Chapter 5

Algorithm for R between two arbitrary points in a mesh

To measure the modified Dirichlet energy from Equation 4.3, we need to calculate the rotation coefficient R_{qp} between two arbitrary points q and p that do not necessarily lie within the same tetrahedron. Since the metric and curl are different in each tet, we need to be able to efficiently determine all tets that get intersected by the straight line from q to p and use the correct metric for each corresponding line segment. The calculation for the coefficient then works in the following way:

Algorithm 2 Rotation coefficient R between q and p

Input (q, p)
    //returns all tets intersected by the line $\vec{qp}$ with the line segments within them
    LINESEGMENTS $\leftarrow$ *tetFinder*(q, p)
    $R \leftarrow \mathrm{Id}$
    for each SEGMENT **in** LINESEGMENTS
        $R \leftarrow R \cdot$ *calcCoeff*(SEGMENT)
return R

The missing component here is how to efficiently find all tetrahedra that get intersected. One possibility would be to use ray-triangle intersection and test against the whole mesh, but this is not practical as we have local information that we can exploit.

We use the idea of the straight walk from *Walking in a Triangulation* [Devillers et al., 2001], which we modify and relies only on so called *orientation tests* to determine which triangles we traverse. This chapter covers how the *tetFinder* algorithm works, how it is made robust against degenerate cases, and a full description of the algorithm is given, more than what is described in the reference.

5.1 Framework

Let $\mathcal{T}$ be a triangulation of a domain Ω that, for simplicity, is convex. In principle, the domain does not need to be convex. It suffices that for all requested paths from q to p, it is contained in Ω. The straight walk traverses all triangles that get intersected by the line segment from q to p. The algorithm first makes an initialization step to get into a valid state, then the straight walk can start. To get a feeling of how the algorithm works, let us go through an example in 2D. If the algorithm starts in a valid state, the line from q to p intersects with some edge $\vec{lr}$. Two triangles share this edge. We test on which side point p lies of this edge to decide whether the walk continues. If the walk continues, we jump through the edge to hop from the old triangle to a new one. This triangle is defined by three vertices (l, r, s). We decide if the new candidate point s lies on the left side or right side of the line from q to p. If s lies on the left, point l is moved, else point r is moved. A new edge intersected with the ray $\vec{qp}$ is found and the walk

repeats. This process is illustrated in Figure 5.1 Notice how the ray $\vec{qp}$ always intersects the edge $\vec{lr}$ at

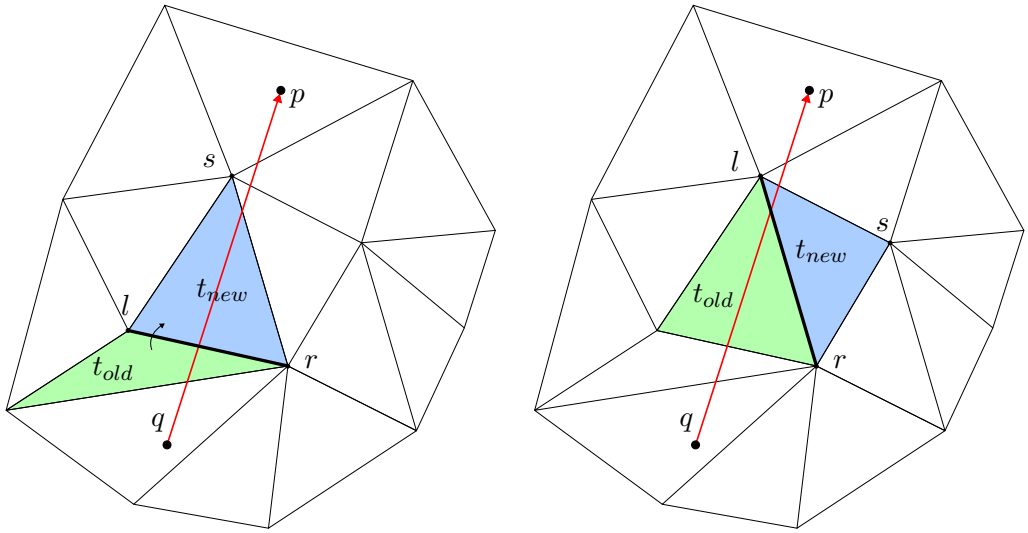


Figure 5.1. Straight walk step

each update step. We can use this observation to add each edge at every update step to a list. When the algorithm terminates, we can iterate over this list, find the intersection point of the ray $\vec{qp}$ with the edge, and calculate the rotation coefficient for this segment. The straight walk in 3D works similarly, but edges are now faces and triangles are tets. The initialization step consists of finding a starting tet t where q is contained. Then, we find the face of the tet t that gets intersected by the ray $\vec{qp}$. Again, at each step, we know that the ray goes out of our current tet t through some face f defined by vertices uvw. We decide if the walk continues by checking on which side p lies relative to f. If the walk should continue, we hop through f to a new tet t_{new}. With two 3D orientation tests, we decide which of the vertices u, v, w gets moved to the new candidate point s. This defines a new face f_{new} with which our ray intersects and the walk repeats. Degenerate cases such as when the ray $\vec{qp}$ goes exactly through a vertex or when the ray lies within a face may get the algorithm into an invalid state, where the algorithm may then traverse through cells that do not get intersected by the ray. We need to detect and escape those degenerate cases through some additional checks. How we handle these degenerate cases is described in Section 5.2.4.

5.1.1 A note on orientation tests

To determine on which side some point s lies relative to two other points q and p (that represent a line) in 2D, the geometric *orientation* predicate is used. It corresponds to evaluating the sign of a determinant. Analogously in 3D, the orientation predicate tests whether a fourth point lies above or below a plane defined by three other points. How "above" the plane is defined depends on the ordering within the determinant. In the case here, it is above if point a sees the triangle bcd in counterclockwise order, see Figure 5.2.

$$\text{orientation}(\alpha, \beta, \gamma) = sign\left(\begin{vmatrix} \beta_x - \alpha_x & \gamma_x - \alpha_x \\ \beta_y - \alpha_y & \gamma_y - \alpha_y \end{vmatrix}\right)$$

$$\text{orientation}(\alpha, \beta, \gamma, \delta) = sign\left(\begin{vmatrix} \beta_x - \alpha_x & \gamma_x - \alpha_x & \delta_x - \alpha_x \\ \beta_y - \alpha_y & \gamma_y - \alpha_y & \delta_y - \alpha_y \\ \beta_z - \alpha_z & \gamma_z - \alpha_z & \delta_z - \alpha_z \end{vmatrix}\right)$$

It is important that the sign of the determinant is evaluated exactly. If geometric predicates are implemented with floating point arithmetic, the answers may be inconsistent and wrong. Because of the finite mantissa in floating point representation, many numbers cannot be represented exactly. The machine needs to round a number x to its nearest number that is exactly representable and roundoff errors occurs.

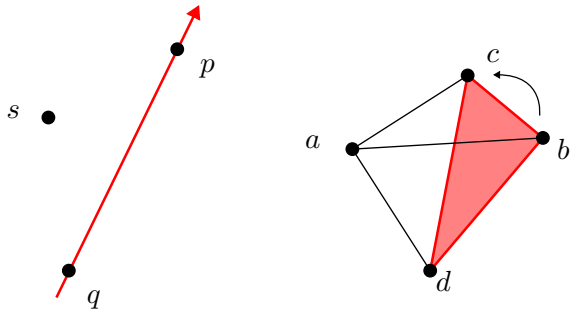


Figure 5.2. Orientation predicate: Point a lies above the plane bcd because it sees the points in counter-clockwise order, $orientation(a, b, c, d) > 0$

A famous example of this phenomenon is $0.1 + 0.2 = 0.30000000000000004$. We call the distance between two exactly representable floating point numbers *machine epsilon* ε. See Figure 5.3 for how this machine epsilon can cause trouble (Example from [Teichmann, 2009]). Because of the imprecision of

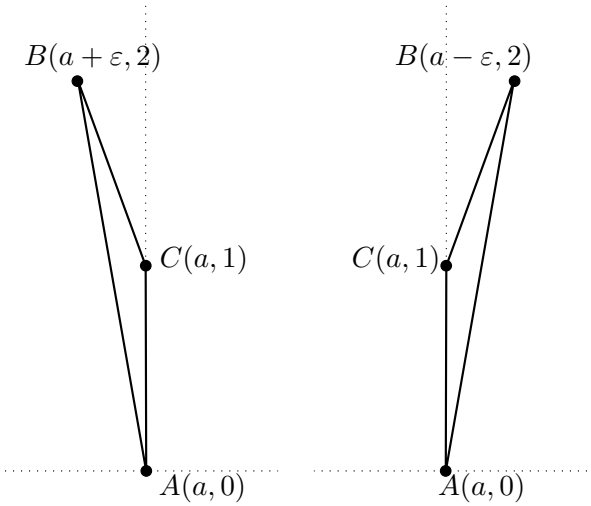


Figure 5.3. The orientation predicate implemented with floating point arithmetic cannot correctly distinguish these two cases. The orientation test calculates $(a + \varepsilon) - a$. If $a = 0$, the orientation test is strictly positive. If $a = 1$, then the result is 0, because during the calculation the machine had to round.

floating-point arithmetic (FP), logical decisions based on FP should be avoided. To fix this issue, we make use of exact predicates that are implemented with arbitrary precision [Richard Shewchuk, 1997]. It is assumed that inputs to the orientation test subroutines are exact and during the calculation, precision of the number representation is extended as needed such that we can be sure that the result has the correct sign.

5.2 Algorithm *tetFinder*

As described in Section 5.1, the straight walk needs different components to work. These components are described in the following subsections. The final algorithm to find all intersected tetrahedra between two points is in Section 5.2.5.

5.2.1 Starting tet

To start, the algorithm needs to know in which tet the starting point q is located. With a simple linear nearest neighbor search of point q, we find a good heuristic starting vertex s. From s, every tet incident to s is checked if it contains q. Most of the times, this method works for locating which tet contains q.

In case it fails, an exhaustive search through all the tets of the mesh is done to search which tet contains q. Figure 5.4 shows a constellation where the nearest neighbor approach fails. Checking if a point q is

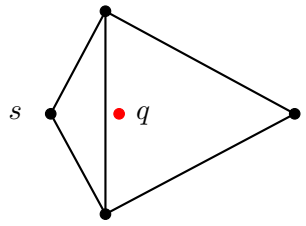


Figure 5.4. The point s is the nearest neighbor to q. However, searching all incident triangles to s, point q cannot be located. A similar example can be constructed in 3D.

within some tet $\mathcal{T} = [a, b, c, d]$ can be done with the orientation predicate, i.e.

$$q \in \mathcal{T} \iff \begin{array}{c} \textit{sameSign}(\textit{orientation}(a,b,c,d),\textit{orientation}(a,b,c,q))\&\& \\ \textit{sameSign}(\textit{orientation}(b,c,d,a),\textit{orientation}(b,c,d,q))\&\& \\ \textit{sameSign}(\textit{orientation}(c,d,a,b),\textit{orientation}(c,d,a,q))\&\& \\ \textit{sameSign}(\textit{orientation}(d,a,b,c),\textit{orientation}(d,a,b,q)) \end{array}$$

We check for each plane of the tet if the point q is on the same side of the plane (characterised by the same orientation sign) as the remaining vertex.

5.2.2 Valid state and initialization

To walk in the tet mesh, the algorithm needs to be in a state where everything is as expected. In 2D, we define this valid state as

- the line $\vec{qp}$ intersects the edge defined by vertices $\vec{lr}$ and this edge lies between q and p
- vertex l lies on the left of the line defined by $\vec{qp}$
- vertex r lies on the right of the line defined by $\vec{qp}$

If the algorithm starts with these configurations, then in each step the valid configuration will be preserved and all edges intersected are traversed. Analogously in 3D, we define the valid state such that

- the line $\vec{qp}$ intersects the face/triangle defined by vertices $\triangle uvw$
- vertices of the triangle $\triangle uvw$ are ordered in the way that

$$\textit{orientation}(w, v, q, p) > 0, \textit{orientation}(v, u, q, p) > 0, \textit{orientation}(u, w, q, p) > 0$$

Again, by starting in this configuration, all faces $\triangle uvw$ traversed get intersected by $\vec{qp}$. Then, we find each intersection point with the ray, add the intersection points to a list with the current tet t and finally use this as input for calculating the rotation coefficient as described in Chapter 4. To get into a valid state, an initialization needs to be performed. In *Walking in a triangulation* [Devillers et al., 2001], they assume that q is a vertex in the tet mesh. Thus, their approach of turning around q is not applicable in our case.

For simplicity, we solve this problem with a brute-force approach. First, we find which tet t contains q as described in Section 5.2.1. From there, we go through the four faces of the tet and check which one gets intersected by the ray $\vec{qp}$. When the face which gets intersected was found, we check the 6 permutations of the three vertices (u, v, w) until the configuration is found, where the orientations are positive.

5.2.3 Robust ray-triangle intersection

When doing the initialization, we check the four faces of the tet to test which face intersects with $\vec{qp}$. This ray-triangle intersection check must be robust, meaning that when the segment $\vec{qp}$ goes exactly through an edge or vertex, it must also be detected. Again, we can use the orientation predicate from before to do this robustly. Let $\triangle uvw$ be a triangle in $\mathbb{R}^3$ and q, p the two points that determine a line segment. If u, v, w is ordered in the way that $\mathit{orientation}(q, u, v, w) > 0$, then the segment determined by $\vec{qp}$ cuts the triangle if and only if [Segura and Feito, 1998]:

$$\mathit{orient}(p, u, q, v) \geq 0 \wedge \mathit{orient}(p, w, v, q) \geq 0 \wedge \mathit{orient}(p, u, w, q) \geq 0 \tag{5.1}$$

If $\mathit{orientation}(q, u, v, w) < 0$, then the orientation tests in Equation 5.1 must be less or equal to zero. Before checking for intersection with Equation 5.1, we must ensure that p and q lie on opposite sides of the triangle $\triangle uvw$, i.e.

$$(\mathit{orient}(q, u, v, w) < 0 \wedge \mathit{orient}(p, u, v, w) > 0) \vee (\mathit{orient}(q, u, v, w) > 0 \wedge \mathit{orient}(p, u, v, w) < 0) \tag{5.2}$$

If all 5 points are coplanar, then the ray-triangle intersection reduces to a 2D case. However, this case is given the default value of NOINTERSECTION, as the intersection point can be calculated by the other faces of the tet.

5.2.4 Robustness

The *tetFinder* algorithm is designed to be robust and exact. Exact means no logical decisions based on FP, because this can lead to unexpected and wrong behaviour. Exactness is handled already, because all decisions are made based on the orientation predicate, which is made exact through arbitrary precision arithmetic. Robust in our context means that it is able to handle degenerate cases, for example when the ray goes exactly through an edge or vertex of a tet. The first case to handle is when the starting point q is in multiple tets. Since the mesh is not a disjoint union of tets as they share their boundaries, a point can be in multiple tets at once. This can lead to problems in the initialization: If a random tet which contains q is chosen, a valid state must not necessarily be found, see Figure 5.5. To fix this, we add all tets to a list that contain q while searching for the start tet. While initializing, we go through the tets one by one until a valid configuration is found. The degenerate cases where the ray lies exactly on a face or goes exactly

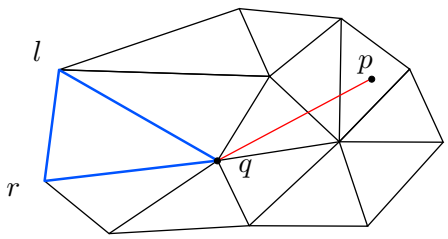


Figure 5.5. As q is exactly on a vertex and the chosen triangle (in bold blue) does have an edge that intersects with $\vec{qp}$, it does not matter how l and r are chosen on this triangle, no valid state can be found. The same thing happens in 3D, where no valid configuration for u, v, w can be constructed given a bad tet.

through an edge are all handled similarly. The problem in these cases is that while walking, the next step (reassigning u, v or w to the next vertex) cannot make a valid state, e.g. $\triangle uvw$ do not intersect with $\vec{qp}$ anymore. We detect this by checking if we are still in a valid state and that the end tet does not match with the currently visited tet before the next iteration of the walk. If the walk has not finished yet and we are not in a valid state, we reinitialize from the current tet and use the brute-force approach to search for a valid state again. When found, the walk can continue as normal. Because degenerate cases can lead to invalid states, faces that do not get intersected by $\vec{qp}$ may be highlighted by $\triangle uvw$ while walking. Thus, before adding the face $\triangle uvw$ to the list to calculate the intersection points, we quickly check if the face to be added actually intersects with the robust ray-triangle subroutine described before.

5.2.5 Formulation of algorithm

The final algorithm *tetFinder* is presented in pseudocode and uses operations such as:

- `neighbor(t through` uvw`)`, returns the tetrahedron sharing face uvw with tetrahedron t
- `s=vertex of` $t, s \neq u, s \neq v, s \neq w$, returns the remaining vertex of a tetrahedron whose other three vertices are known
- `intersection(ray, triangle)`, returns the intersection point of the ray with the triangle
- `validState(q,p,u,v,w)`, checks if the current configuration is valid as in Section 5.2.2
- `initialization(t,q,p,u,v,w)`, tries to initialize by assigning vertices u, v, w from tet t as described in Section 5.2.2. Returns `true` if successful
- `locateTets(q)`, returns all tets that contain q

Algorithm 3 tetFinder

Input (q, p)
    STARTS = *locateTets(q)*
    LINESEGMENTS = [ ]
    t = NIL // working variables
    u, v, w, s = NIL // working variables
    for $\hat{t} \in$ STARTS
        if ($p \in t$) //p in same tet as q, we are done
            return LINESEGMENTS.*add*($[t, q, p]$)
        if (*initialization*(t, q, p, u, v, w))
            $t = \hat{t}$
    // qp intersects triangle uvw
    // $wvqp, vuqp, uwqp$ are positively oriented
    PREV = q
    CURR = *intersection*(qp, uvw)
    LINESEGMENTS.*add*($[t$, PREV, CURR$]$)
    while $orientation(u, w, v, p) > 0$ {
        if (*!validState(q,p,u,v,w)*) // degenerate cases can lead to invalid states
            initialization(t, q, p, u, v, w)
        t = neighbor(t through uvw)
        s = vertex of $t, s \neq u, s \neq v, s \neq w$
        PREV = CURR
        if $orientation(u, s, q, p) > 0$ // qp does not intersect triangle usw
            if $orientation(v, s, q, p) > 0$ // qp intersects triangle vsw
                u = s
            else // qp intersects triangle usv
                w = s
        else // qp does not intersect usv
            if $orientation(w, s, q, p) > 0$ // qp intersects triangle usw
                v = s
            else // qp intersects triangle vsw
                u = s
        if (*robustRayTriangle(q,p,u,v,w)*)
            CURR = *intersection*(qp, uvw)
            LINESEGMENTS.*add*($[t$, PREV, CURR$]$)
    } // t contains p
    return LINESEGMENTS.*add*($[t$, PREV, $p]$)

Chapter 6

Frame Field optimization

Until now, we have only covered how we measure the Dirichlet energy $||\mathcal{D}R||$ in the new metric g. This chapter covers how we minimize

$$E(\mathcal{M}) = \int_{\mathcal{M}} ||DR||^2.$$

We use an optimization scheme based on the PhD work of Simone Raimondi[1], which in turn is based on the Merriman-Bence-Osher (MBO) algorithm. In principle, any frame field optimization scheme that works based on optimizing the Dirichlet energy (or its discretized version $||R(q) - R(p)||^2$) can be modified with the rotation coefficient R_{qp} to optimize in a new metric.

6.1 Optimization Algorithm

The algorithm works by dividing the manifold into cubes, and optimizing the frames per cube. The algorithm works based on an adaptive grid, refining the grid and optimizing where more resolution is needed, see Figure 6.1 From a given voxel with coordinates $v = (x, y, z)$, the rotation coefficients are

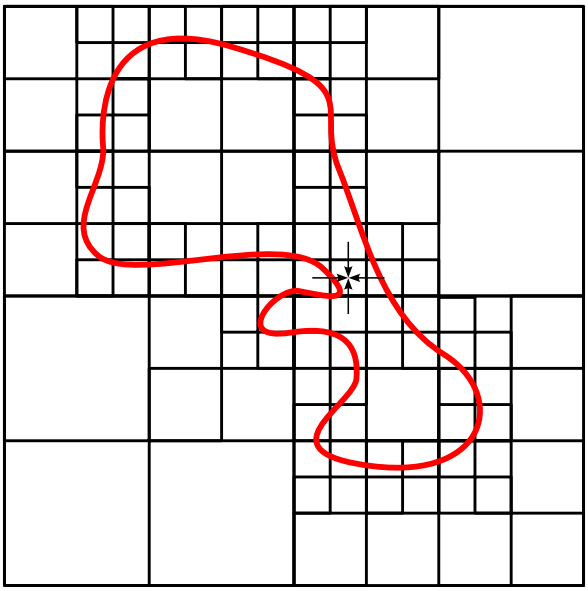


Figure 6.1. An adaptive grid is formed around the manifold (bold red), with increased resolution where needed. The rotation coefficients for a given voxel are calculated from its neighbours (arrows). In 3D, the cubes get divided into eight smaller cubes.

[1] `https://cgg.unibe.ch/person/1105/ - Simone Raimondi`

calculated from their neighbors $v_n = (x \pm 1, y \pm 1, z \pm 1)$ to v, i.e.

$$||[R_{v_n \rightsquigarrow v}]v_n - v||^2.$$

The MBO algorithm is based on repeated diffusion in the spherical harmonics (as averaging frames makes sense in the spherical harmonics), with a projection back to the manifold of valid frames (see Figure 6.2).

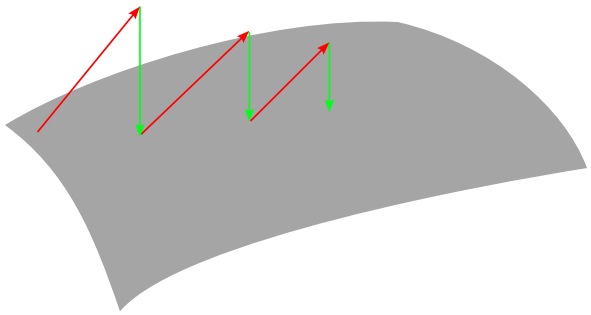


Figure 6.2. Repeated diffusion of frames in spherical harmonics with a projection back to the valid space of frames is used to minimize the energy. Figure inspired from [Palmer et al., 2020].

Because the implementation for the adaptivity of the grid was not finished at the time of this thesis, the optimization is done on a regular grid where all voxels have the same size. Two parameters depth d and number of diffusion steps n are present. The depth d is how many times the grid gets subdivided, with depth 0 corresponding to a cube that is subdivided once. The resulting frame fields are expected to be the same, it just does it slower.

6.2 Caching of coefficients

Without any optimization, the algorithms time complexity is exponential with the depth. At depth n, there are 8^{n+1} voxels making up the enclosing cube. For every voxel, 6 rotation coefficients are calculated from its neighbors to itself and diffused d times. Because the underlying metric field does not change while optimizing, we can cache these rotation coefficients to not recalculate R from scratch at every diffusion step. We initialize a list of length $8^{n+1} \cdot 6$ at depth n, which in the first run at each depth gets filled with the corresponding rotation coefficients. Afterwards, the corresponding rotation coefficient for the voxel can get looked up at each diffusion step.

Chapter 7

Experiments

In this section, the frame field generation applied to a cube mesh with some different metrics is discussed. *Setup:* If not said otherwise, we run the frame field optimization to depth 3 with 300 diffusion steps. We store the associated matrices of the metric field at each vertex of the tet mesh. The cube has corners at $(0, 0, 0)$ and $(1, 1, 1)$.

To start, we apply the constant metric $g^{1/2} = \mathrm{diag}(1, 1, 1)$ everywhere. The rotation coefficients R are the identity matrix everywhere, as the metric does not twist or squish the space within the field. The result is the boundary aligned frame field with no singularities, as the energy can be minimised to zero if the frames are constant. Figure 7.1 shows the mesh used to store the metric and the resulting constant frame field.

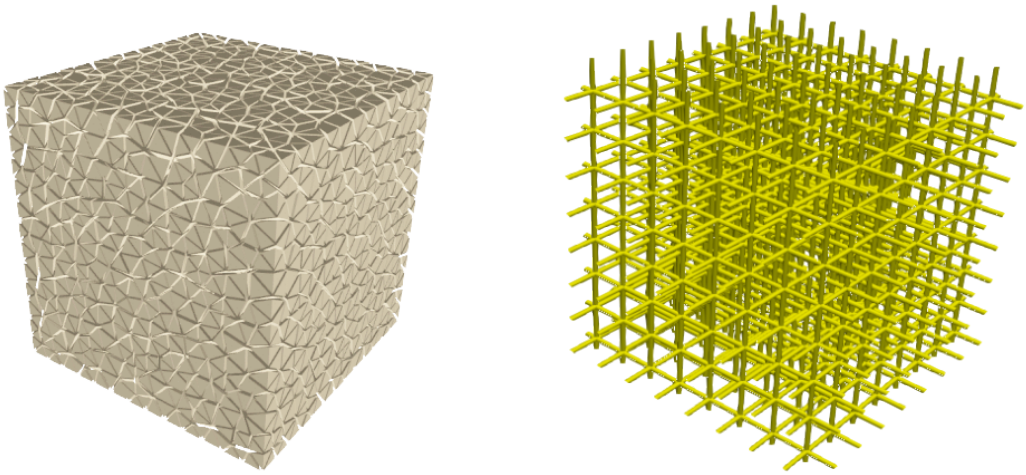


Figure 7.1. Left: The mesh used to store the metric field. Right: Constant metric everywhere which gives the boundary aligned constant frame field with no singularities.

To further validate the frame field generation, we test a 2D analogue setup. We divide the cube along the z-axis into three equally sized parts. Then, the metric varies only in the x, z-dimension. The function to attach the metric to the vertices is defined as

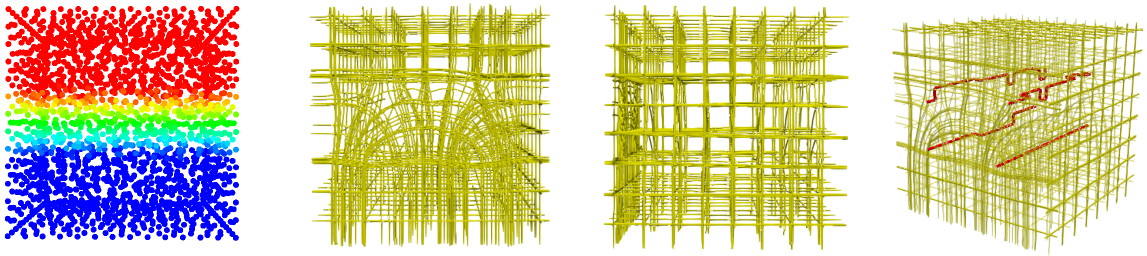
$$g^{1/2}(z) = \begin{cases} \mathrm{diag}(1, 1, 1) & 0 < z < 1/3 \\ \mathrm{diag}(h_k(z), 1, h_k(z)) & 1/3 < z < 2/3 \\ \mathrm{diag}(k, 1, k) & 2/3 < z < 1 \end{cases} \tag{7.1}$$

with $h_k(z) = 3z(k - 1) - k + 2$. The metric is constant in the y-axis and constant-linear-constant in the

x, z-axis. Thus, the rotation coefficients are only of the form

$$\begin{pmatrix} \cos(\alpha) & 0 & \sin(\alpha) \\ 0 & 1 & 0 \\ -\sin(\alpha) & 0 & \cos(\alpha) \end{pmatrix}$$

for some angle α. The result for a factor $k = 10$ is depicted in Figure 7.2. For a factor of $k = 10$,



(a) (b) (c) (d)

Figure 7.2. (a) The varying $g_{00}^{1/2}$ component along the z-axis for $k = 10$. Visible is how the streamlines of the frame field only change in the middle third (b) and how the frames do not change along the y-axis (c). The singularities (d) are only points when taking a slice along the y-axis.

4 singularities are present. If we decrease the metric change in Equation 7.1 by lowering the factor to $k = 4$, less singularities are present (see Figure 7.3). These results match with what previous works have

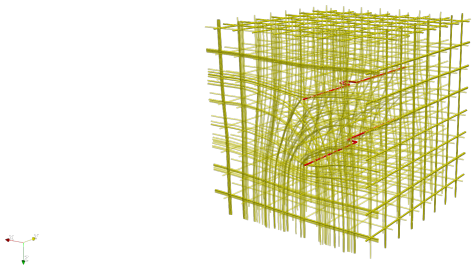


Figure 7.3. With a lower factor of $k = 4$, the metric change is not as extreme and less singularities are present.

done in 2D. The third experiment shows how important higher depths (more subdivision of voxels) are. Again, we divide the cube into three parts along the z-axis. This time, we isotropically scale the metric, i.e. the metric is defined as

$$g^{1/2}(z) = \begin{cases} \mathrm{diag}(1,1,1) & 0 < z < 1/3 \\ \mathrm{diag}(h_k(z), h_k(z), h_k(z)) & 1/3 < z < 2/3 \\ \mathrm{diag}(k,k,k) & 2/3 < z < 1 \end{cases}. \tag{7.2}$$

We expect rotational symmetry (because the metric field is symmetric around the z-axis) of the singularity graph around the z-axis, with the singularity structure going from side to side of the surface. We run the frame field optimization to different depths, the result is pictured in Figure 7.4. At depth 3 the singularities cancel out before reaching the surface of the cube and do not connect in the middle. At depth 4, the singularity structure looks better, but the arcs still cancel out before reaching the surface. At depth 5, it looks the most reasonable, with the arcs reaching the surface and being connected in the

middle. However, it is still not perfect as we still expect a 90 degree rotational symmetry of the arcs. An even higher depth is required, but due to the exponential time growth, this is not feasible with the naive optimization approach. This fact further justifies only adaptively increasing resolution where is needed. As a last example, we design a metric where we expect larger cubes at the surface of the cube and smaller

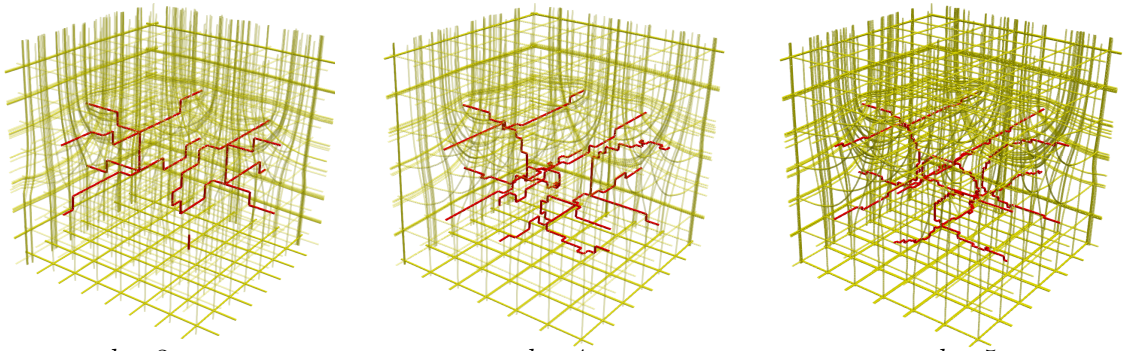


Figure 7.4. The isotropic metric scaling from Equation 7.2 is run to different optimization depths. Depth 3 is lacking significant features of what we expect. At depth 4, the structure begins to look as expected, but e.g. still is not connected in the middle. At depth 5, it looks almost as expected. It does not quite show full 90 degree symmetry yet, as e.g. there is only one "hook-arcs" on the left, but two on the right of the cube.

in the interior. We divide the cube into 3 parts, but this time from the center out. We measure the radius to the center of the cube with the infinity-norm. In the inner third of the cube, we set $g^{1/2} = \mathrm{diag}(1, 1, 1)$. The metric is linearly increasing to $k = 10$ in the second third and in the outside third, it is a constant metric again at $g^{1/2} = \mathrm{diag}(10, 10, 10)$. We expect pointwise symmetry of the singularity graph around the center of the cube. The result is shown in Figure 7.5.

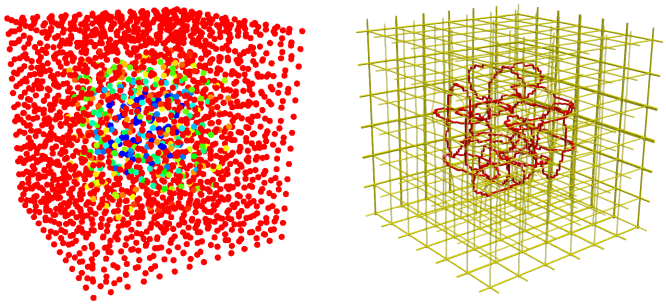


Figure 7.5. Left: The nonuniform metric, visualized is the $g_{00}^{1/2}$ component. Right: The singularity graph. The optimization is run to depth 5. Lower depths displayed similar defects as described in Figure 7.4, where the singularity structure was not fully formed yet.

Chapter 8

Conclusion

In this work, we showed how smoothness measured with covariant Dirichlet energy is directly related to local integrability. In fact, we showed that when the underlying metric field is flat, that local integrability is actually equivalent to zero covariant derivative under the connection induced by the metric field. This justifies the application of metric fields to the frame field optimization. A beneficial aspect of the metric change approach through the rotation coefficients is that it is independent of the Dirichlet energy minimizer, which makes it an independent building block to control the size and shear of hexahedral elements. The approach is tested on cubes, where the resulting frame fields look as expected.

To store the metric field, we used a tetrahedral mesh and piecewise linearly discretized the metric field. This is different compared to the constant discretization done in *Metric-Driven 3D Frame Field Generation* [Fang et al., 2023]. To calculate the rotation coefficients on the piecewise linear metric field, an exact and robust algorithm was needed to locate all tets between two points. The formulation relies only on exact, arbitrary precision arithmetic to make decisions. It fully handles degeneracies, unlike the algorithm described in *Walking on a Triangulation* [Devillers et al., 2001]. To calculate the rotation coefficient, points are sampled recursively until there is no noticable improvement. There are minor things that impact speed but not correctness which could be improved. Namely, the point localisation in the *tetFinder* could be changed to use e.g. k-d trees for finding the nearest neighbor.

Future work. As an extension and to make this work more complete, several things could be investigated. First of, the frame fields get more accurate the more the voxels are subdivided. It is thus natural to run the frame field optimization to deeper depths. However, the current implementation on a standard office machine gets to depth 5 in about 16 minutes, which is pretty long for such a simple mesh like a cube. As explained already, the original frame field optimization works by adaptively refining voxels for higher resolution only where is needed. Thus, the caching of the rotation coefficients in an Octree instead of a regular grid would be appropriate.

Currently, the metric fields have to be designed by hand and defined at all vertices of the covering tetrahedral mesh everywhere. This is a tedious process and not easy to do. An automatic metric field design process where the user only inputs some intuitive metric constraints at some vertices (e.g. “larger” metric at boundary, “smaller” metric in the interior) and the vertices where no metric is defined would get automatically assigned a metric that makes the metric field as easy to optimize as possible would be of practical use.

Lastly, it is still unknown how much the piecewise linear discretization improved the accuracy of the computation of the rotation coefficients. A thorough comparison of results on the same models between the constant discretization [Fang et al., 2023] and linear discretization would paint a clearer picture if the extra effort is worth it.

Bibliography

[Bommes et al., 2013] Bommes, D., Campen, M., Ebke, H.-C., Alliez, P., and Kobbelt, L. (2013). Integer-grid maps for reliable quad meshing. *ACM Trans. Graph.*, 32(4).

[Devillers et al., 2001] Devillers, O., Pion, S., Teillaud, M., Logiciel, T., and Prisme, P. (2001). Walking in a triangulation. *International Journal of Foundations of Computer Science*.

[Fang et al., 2023] Fang, X., Huang, J., Tong, Y., and Bao, H. (2023). Metric-driven 3d frame field generation. *IEEE Transactions on Visualization and Computer Graphics*, 29(4):1964–1976.

[Huang et al., 2011] Huang, J., Tong, Y., Wei, H., and Bao, H. (2011). Boundary aligned smooth 3d cross-frame field. *ACM Trans. Graph.*, 30(6):1–8.

[Lee, 1997] Lee, J. M. (1997). *Riemannian manifolds : : an introduction to curvature*. Graduate Texts in Mathematics ; 176. Springer, New York, 1st ed. 1997. edition.

[Lee, 2000] Lee, J. M. (2000). *Introduction to Smooth Manifolds*. Springer.

[Liu and Bommes, 2023] Liu, H. and Bommes, D. (2023). Locally meshable frame fields. *ACM Trans. Graph.*, 42(4).

[Liu et al., 2018] Liu, H., Zhang, P., Chien, E., Solomon, J., and Bommes, D. (2018). Singularity-constrained octahedral fields for hexahedral meshing. *ACM Trans. Graph.*, 37(4).

[Nieser et al., 2011] Nieser, M., Reitebuch, U., and Polthier, K. (2011). Cubecover- parameterization of 3d volumes. *Computer Graphics Forum*, 30(5):1397–1406.

[Palmer et al., 2020] Palmer, D., Bommes, D., and Solomon, J. (2020). Algebraic representations for volumetric frame fields. *ACM Trans. Graph.*, 39(2).

[Panozzo et al., 2014] Panozzo, D., Puppo, E., Tarini, M., and Sorkine-Hornung, O. (2014). Frame fields: Anisotropic and non-orthogonal cross fields. *ACM Trans. Graph.*, 33(4).

[Papachristou, 2020] Papachristou, C. (2020). *Aspects of Integrability of Differential Systems and Fields: A Mathematical Primer for Physicists*. Springer.

[Pietroni et al., 2022] Pietroni, N., Campen, M., Sheffer, A., Cherchi, G., Bommes, D., Gao, X., Scateni, R., Ledoux, F., Remacle, J., and Livesu, M. (2022). Hex-mesh generation and processing: A survey. *ACM Trans. Graph.*, 42(2).

[Ray and Sokolov, 2015] Ray, N. and Sokolov, D. (2015). On smooth 3d frame field design. *CoRR*, abs/1507.03351.

[Richard Shewchuk, 1997] Richard Shewchuk, J. (1997). Adaptive precision floating-point arithmetic and fast robust geometric predicates. *Discrete & Computational Geometry*, 18(3):305–363.

[Segura and Feito, 1998] Segura, R. J. and Feito, F. R. (1998). An algorithm for determining intersection segment-polygon in 3d. *Computers & Graphics*, 22(5):587–592.

[Teichmann, 2009] Teichmann, M. (2009). Robust predicates and degeneracy.

[Vaxman et al., 2017] Vaxman, A., Campen, M., Diamanti, O., Bommes, D., Hildebrandt, K., Technion, M. B.-C., and Panozzo, D. (2017). Directional field synthesis, design, and processing. In *ACM SIGGRAPH 2017 Courses*, SIGGRAPH '17, New York, NY, USA. Association for Computing Machinery.

Erklärung

Erklärung gemäss Art. 30 RSL Phil.-nat. 18

Ich erkläre hiermit, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen benutzt habe. Alle Stellen, die wörtlich oder sinngemäss aus Quellen entnommen wurden, habe ich als solche gekennzeichnet. Mir ist bekannt, dass andernfalls der Senat gemäss Artikel 36 Absatz 1 Buchstabe r des Gesetzes vom 5. September 1996 über die Universität zum Entzug des auf Grund dieser Arbeit verliehenen Titels berechtigt ist.

Für die Zwecke der Begutachtung und der Überprüfung der Einhaltung der Selbständigkeitserklärung bzw. der Reglemente betreffend Plagiate erteile ich der Universität Bern das Recht, die dazu erforderlichen Personendaten zu bearbeiten und Nutzungshandlungen vorzunehmen, insbesondere die schriftliche Arbeit zu vervielfältigen und dauerhaft in einer Datenbank zu speichern sowie diese zur Überprüfung von Arbeiten Dritter zu verwenden oder hierzu zur Verfügung zu stellen.

Ort/Datum

Unterschrift