b
UNIVERSITÄT
BERN

3D Metric Fields

A Novel Approach to a New Idea

Bachelor Thesis

Florin Achermann
from
Bern, Switzerland

Faculty of Science, University of Bern

15. September 2023

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

Abstract

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Contents

Chapter 1

Introduction

- What are Frame Fields?
- Why are they important?
- How do *we* generate them?

A frame field
A vector field is locally integrable, if and only if $\nabla \times A = 0$. [5]

Chapter 2

Mathematical Background

We will make heavy use of differential geometry in the following sections. To get us all on the same page, I introduce the basic concepts what we will use, but I refrain from giving any proofs. I will give definitions only as far as we need it. These definitions will by no means be exhaustive. The following is an incomplete summary of what we need presented in “Introduction to smooth manifolds” [4]

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.

Tangent space, Tangent bundle There are many equivalent definitions for the tangent space. 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 $\gamma : (-\varepsilon, \varepsilon) \to \mathcal{M}$ with $p = \gamma(0)$. The tangent space is a vector space which has the same dimension as its manifold, which is 3 in our case. 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 continuos map, with $\pi \circ \sigma = \mathrm{Id}_{\mathcal{M}}$. Sections of $T\mathcal{M}$ are vector fields on $\mathcal{M}$.

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 vector space can be expressed as a finite linear combination of its basis. This basis 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*.

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, the dot product in $\mathbb{R}^n$ is a tensor. It takes two vectors and is linear in each component - bilinear. Another example is 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*.

Differential Forms, Exterior Derivative Recall that a section from $T^*\mathcal{M}$ is called a differential 1-form, or just 1-form. 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. There is a natural differential operator d on differential forms we call *exterior derivative*. The exterior derivative is a generalization of the differential of a function. In particular, a smooth function f (a 0-form) has the derivative df which is a 1-form. The exact definition is not important for us, so let us just look at some properties so we can work with it. If ω is a k-form, $d\omega$ is a (k+1)-form. TODO

Riemannian metric, g-orthonormality Inner products 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, 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 $\langle e_i, e_j \rangle_g = \delta_{ij}$.

connection, covariant derivative

lie algebra so(3)

Frame field, vector field, integrability

A frame F is a set of 6 vectors $\{\pm F_0, \pm F_1, \pm F_2\}$. We can represent such a frame F as a 3×3 matrix F, where the ith-column is F_i. A frame field then maps to every point in 3D-space such a frame, i.e. $F : \mathbb{R}^3 \to \mathbb{R}^{3\times3}$. Usually, we work on a 3-manifold $\mathcal{M}$ and a positively oriented frame field, i.e. $F|_{\mathcal{M}} : \mathcal{M} \to \mathbb{R}^{3\times3}$, where $\det(F) > 0$. To allow for anisotropic, nonuniform meshes, we generalize orthonormality of frames to g-orthonormal frames. Orthonormality is measured in some metric g, and a frame F satifisfies the condition $\langle F_i, F_j\rangle_g = \delta_{ij}$. Any frame field with $\det(F) > 0$ naturally defines a metric $g = (FF^\top)^{-1}$, where F is g-orthonormal

$$F^\top g F = Id.$$

We can factor the frame field F into a symmetric part $g^{1/2}$ and a rotational part R

$$F = g^{-1/2} R$$

The symmetric part $g^{-1/2}$ keeps F g-orthonormal

$$\implies F^\top g F = (g^{-1/2}R)^\top g g^{-1/2} R) = R^\top g^{-1/2} g g^{-1/2} R = Id.$$

and R represents a rotational field $R : \mathcal{M} \to SO(3)$. The requirements for our frame field are:

- Smoothness
- Integrability
- Metric consistency: $g = (FF^\top)^{-1}$

Chapter 3

Connection One-Form ω

A vector field U is integrable, if and only if $\nabla \times U = 0$, which means the vector field has vanishing curl everywhere. We can express this more naturally with the language of differential forms: The curl can be written as the exterior derivative d of a one-form α. A one-form (more generally, a differential form) is closed, if $d\alpha = 0$. Therefore, the local integrability can be expressed as the closedness of a one-form. We want F^{-1} (TODO: why F^{-1}) to be integrable. To achieve local integrability for, it suffices to make R locally integrable. We can think of a rotation field R as the composition of 3 vector fields

$$R = \begin{bmatrix} | & | & | \\ R_1 & R_2 & R_3 \\ | & | & | \end{bmatrix}$$

where $R_i : \mathbb{R}^3 \to \mathbb{R}^3$ is a vector field. We can therefore construct a vector-valued one-form, given $p = (x, y, z)^\top$ in Euclidean coordinates

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

where $dp = (dx, dy, dz)^\top$ is the common orthonormal one-form basis of $\Omega^1(\mathcal{M})$.

$$R \text{ locally integrable} \iff \mathbf{0} = d\alpha$$

Some reformulations yield:

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

where for (1), the Leibnitz Rule for the exterior derivative is applied, and we define

$$\omega = R dR^\top \in \mathfrak{so}(3).$$

Remark. *ω is an antisymmetric matrix-valued one-form (every element is a one-form).*

Proof. We differentiate the orthogonality condition of the rotation matrix R (taking the derivative w.r.t. every element):

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

$\square$

Elements of the Lie algebra $\mathfrak{so}(3)$ can be thought as infinitesimal rotations and ω can be used as a connection one-form. The Lie algebra $\mathfrak{so}(3)$ has manifold structure, but for our purposes, it suffices to know that elements are antisymmetric matrices. To make R locally integrable, we find ω such that the one-form α is closed ($d\alpha = 0$, curl-free), then try to match the ω with R, which can be expressed as

$$\min_{R \in SO(3)} ||RdR^T - \omega||^2.$$

Connection evaluation To find ω, we use the above equation

$$\mathbf{0} = R^\top(\omega \wedge (g^{1/2}dp) + d(g^{1/2}dp)) \iff \mathbf{0} = \omega \wedge (g^{1/2}dp) + d(g^{1/2}dp)$$

and reformulate into a linear system. We represent the antisymmetric matrix-valued one-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 one-forms, e.g. $W_1 = \begin{bmatrix}(\omega^{23})_1, (\omega^{23})_2, (\omega^{23})_3\end{bmatrix}^\top$. Recall, a one-form can be expressed as

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

So e.g.

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

We also write $A = g^{1/2} = \begin{bmatrix}A^1, A^2, A^3\end{bmatrix}$. We use the equation $\mathbf{0} = \omega \wedge (g^{1/2}dp) + d(g^{1/2}dp)$. Let us start with the first part:

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

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 $\left[A_1, A_2, A_3\right] = \left[A^1, A^2, A^3\right]$ because A is symmetric, and W_1 corresponds to ω_{23}, W_2 to ω_{31} and W_3 to ω_{12}. Second part:

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

Again, we can do this separately for each row:

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

Finally, we can put everything together:

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

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 9x9 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 3x3-matrix into a 9x1-vector by stacking the columns. With these two definitions, we can transform the above equality into a linear system

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

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 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

Calculation of rotation coefficient

We are interested in the rotation. Let p, q be points on the manifold and ℓ a path connecting them. Let $R : \mathcal{M} \to SO(3)$ be a rotation field, i.e. $R(p), R(q)$ are orthogonal frames. Under the initial condition that $R(p) = \mathrm{Id}$, the equation

$$R(q) = \exp(R_{pq}(\omega))R(p)$$

is a differential equation that corresponds to the parallel transport under the connection ω of the frame along ℓ. To recover this rotation R_{pq}, we integrate ω along ℓ.

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

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

where $R_i = \exp(-\omega(\dot{\ell}(i\gamma))\gamma) = \exp((\int W^\top dp)_\times)$, and γ is the length of a segment and $\dot{\ell}(s) = \frac{\partial \ell}{\partial s}(s)$. Calculating the exponential map of an antisymmetric matrix (which $\omega \in \mathfrak{so}(3)$ is) 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 a 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 is solved for by the 9x9 linear system given by A_x and $\nabla \times A$.

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 $A_i \in \mathbb{R}^{3x3}, i \in \{1, 2, 3, 4\}$ be the square root metrics at the vertices $v_i \in \mathbb{R}^3$ of a tet, such that $A_i^2 = g(v_i)$. We represent a tet given by its four vertices by a 3x4 matrix, i.e.

$$\begin{pmatrix} | & | & | & | \\ v_1 & v_2 & v_3 & v_4 \\ | & | & | & | \end{pmatrix} \in \mathbb{R}^{3\times 4}.$$

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

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

T always exists because $v_1, \ldots, v_4$ are linearly independent, else it would not be a 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. E.g. $(A^i_j)_x$ 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.

- Calculation of R
- Piecewise linear discretization

Chapter 5

Algorithm for R between two arbitrary points in a mesh

To measure the Dirichlet energy, we need to calculate the rotation coefficient between to arbitrary points q and p that do not necessarily lie within the same tetrahedron. Since the metric and curl is 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 1 Rotation coefficient R between q and p

Input (q, p)
  LINESEGMENTS $\leftarrow$ *tetFinder*(q, p) //returns all tets intersected by the line $\vec{pq}$ with the line segments within them
  $R \leftarrow \mathrm{Id}$
  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 have local information that we can exploit.

We use the idea of the straight walk from *Walking in a Triangulation*[2], which relies only on on so called *orientation tests* to determine which triangles we traverse.

Framework Let $\mathcal{T}$ be a triangulation of a domain Ω that is convex. 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 how the algorithm works, let us go through an example in 2D. If the algorithm was in a valid state before, 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 each update step. We can use this observation to add each edge at each update step to a list. When the algorithm terminates, we can just 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. 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

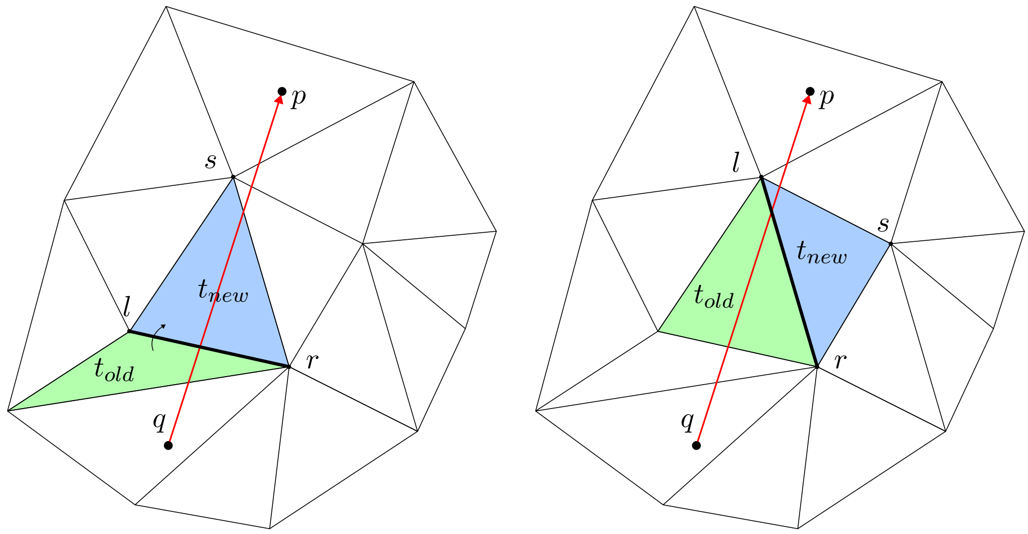

Figure 5.1. Straight walk step

step, we know that the ray goes out of our current tet t through some face e defined by vertices uvw. We decide if the walk continues by checking on which side p lies relative to e. If the walk should continue, we hop through e to a new tet t_{new}. With two orientation tests we decide which of the vertices u, v, w gets moved to the new candidate point s. This defines a new face e_{new} where 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, the algorithm may get 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. These are not described in the paper[2]. In most cases, a simple reinitialization is enough when the invalid state is detected.

A note on orientation tests To determine on which side some point s lies relative to two other points q, 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 this case, it is above if point a sees the triangle bcd when turning counterclockwise (see figure 5.2).

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

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 [1], many numbers cannot be represented exactly. The machine needs to round a number x to its nearest number that is exactly representable. Many times, roundoff errors occur, a famous example of this 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 http://groups.csail.mit.edu/graphics/classes/6.838/S98/meetings/m12/pred/m12.ht Logical decision based on floating point arithmetic is correct most of the times, but not always. To fix this issue, we make use of exact predicates that are implemented with arbitrary precision [7]. Inputs to the orientation test subroutines are assumed to be exact, and we can be sure that the result has the correct sign.

Algorithm 2 tetFinder

Input (q, p)
    $t =$ *LocateTet(q)*
    initialization(t)

    // qp intersects triangle uvw
    // $wvqp, vuqp, uwqp$ are positively oriented
    LINESEGMENTS $= [\,]$
    PREV $= q$
    CURR $=$ *intersection*(qp, uvw)

    LINESEGMENTS.*add*$([t,$ PREV, CURR$])$

    while $orientation(u, w, v, p) > 0$ {
        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
        CURR $=$ *intersection*(qp, uvw)
        LINESEGMENTS.*add*$([t,$ PREV, CURR$])$
    } // t contains p
    LINESEGMENTS.*add*$([t,$ PREV$, p])$

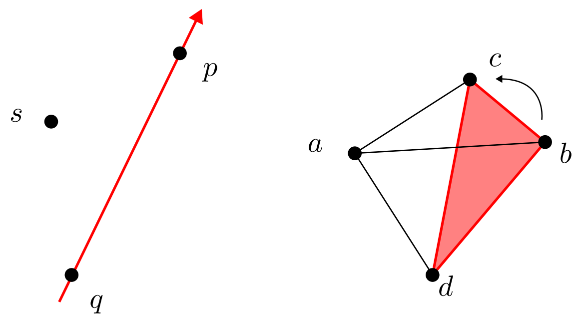

Figure 5.2. Orientation predicate: Point a lies above the plane bcd because it sees the points in counter-clockwise order, $orientation(a, b, c, d) > 0$

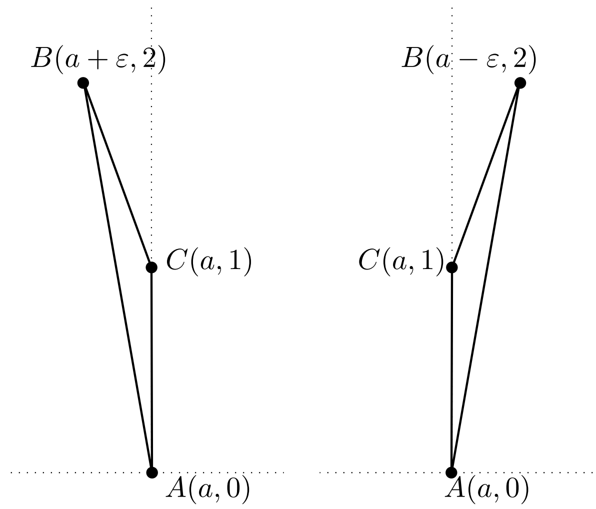

Figure 5.3. The orientation predicate implemented with floating point arithmetic cannot distinguish these two cases. The orientation test calculates $(a + \varepsilon) - a$. If $a = 0$, the orientation test is strictly positive. If $a = 1$, then the result is 0, because during the calculation the machine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

The final algorithm to find all intersected tetrahedra between two points is as follows.
The algorithm uses several subroutines which are described here:

- `neighbor(t through` uvw`)` returns the tetrahedron sharing face uvw with tetrahedron t
- `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
- `initialization()`

Chapter 6

Conclusion

The conclusion looks back at the entire work, gives a critical look, summarizes, and discusses extensions and future work.

Appendix A

Extra material

Extra material may be placed in an appendix that appears after the conclusion.

Bibliography

[1] *IEEE standard for binary floating-point arithmetic*. New York: Institute of Electrical and Electronics Engineers, 1985. Note: Standard 754–1985.

[2] O. Devillers, S. Pion, M. Teillaud, T. Logiciel, and P. Prisme, "Walking in a triangulation," *International Journal of Foundations of Computer Science*, 03 2001.

[3] X. Fang, J. Huang, Y. Tong, and H. Bao, "Metric-driven 3d frame field generation," *IEEE Transactions on Visualization and Computer Graphics*, vol. 29, no. 4, pp. 1964–1976, 2023.

[4] J. M. Lee, *Introduction to Smooth Manifolds*. 2000.

[5] C. Papachristou, *Aspects of Integrability of Differential Systems and Fields: A Mathematical Primer for Physicists*. 01 2020.

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

[7] J. Richard Shewchuk, "Adaptive precision floating-point arithmetic and fast robust geometric predicates," *Discrete & Computational Geometry*, vol. 18, pp. 305–363, Oct 1997.

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

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