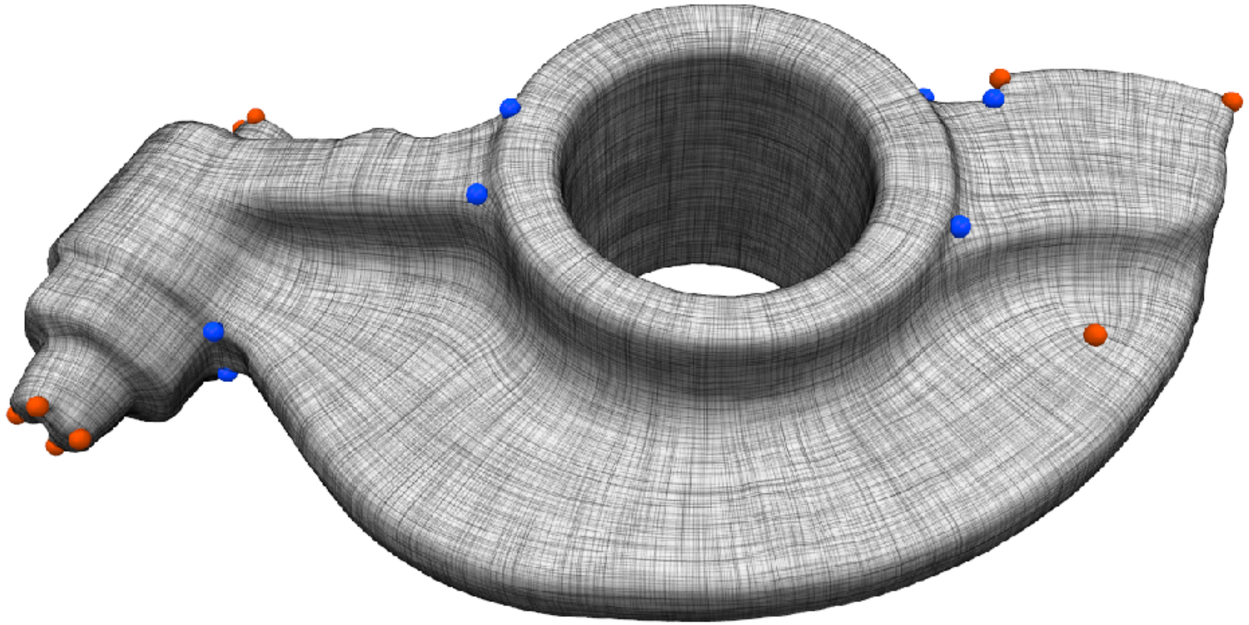
Directional Fields

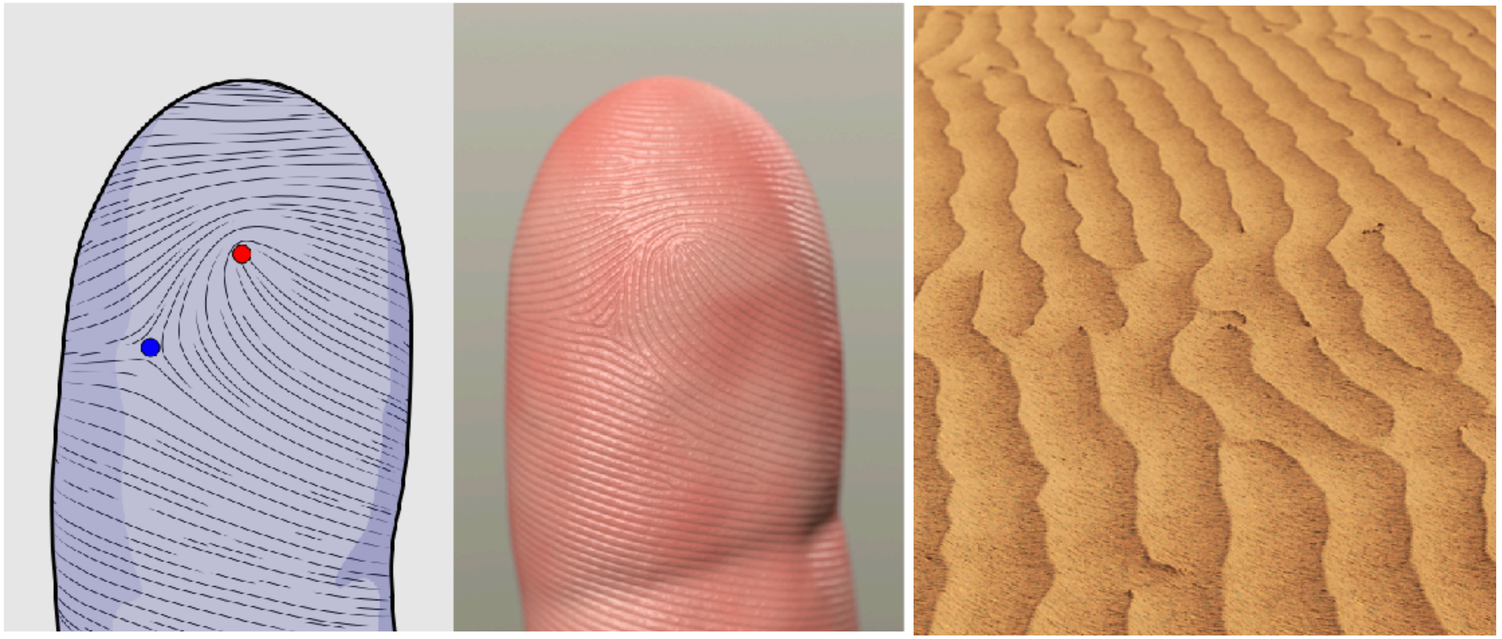
Extended version: github.com/avaxman/DirectionalFieldSynthesis

Directional Fields

- Directional information per point of a domain
- Various types
  - vector vs. direction (with or without magnitude)
  - one or *N* per point
  - symmetric vs. non-symmetric



Directional Fields in Nature



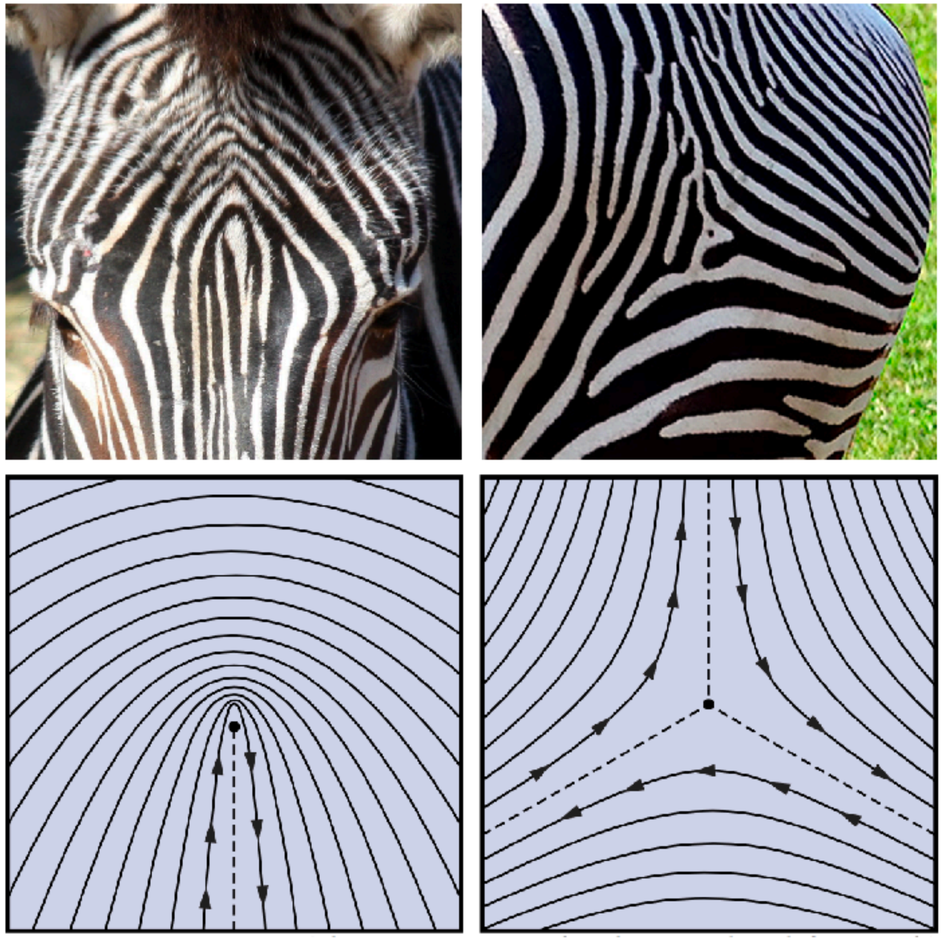


Photo Copyright: Richard Wheeler, Keenan Crane, Trisha Shears, and André Karwath

Directional Fields

- We are here ***not*** concerned with measuring or analyzing them
- We are interested in the ***synthesis*** of directional fields
  - reproduction/simulation/modeling of natural phenomena
  - abstract mathematical tool for diverse applications
    - meshing
    - deformation
    - fabrication
    - data analysis
    - …

Directional Field Synthesis

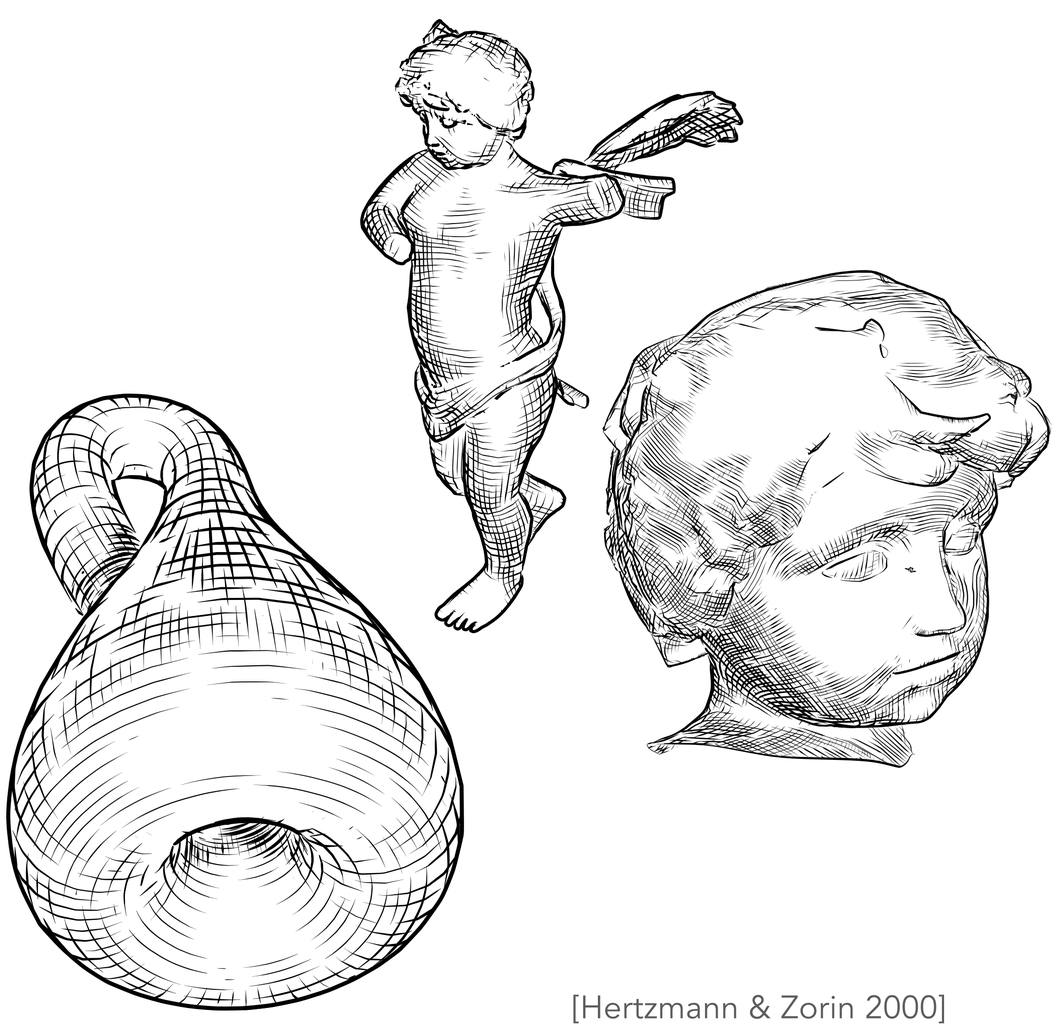
- This area of research has undergone significant development in past 10 years
  - technical novelties introduced in ~50 papers
- Numerous competing approaches for
  - representation
  - discretization
  - optimization
  - formulation of objectives
  - constraining

I will give a high-level overview,

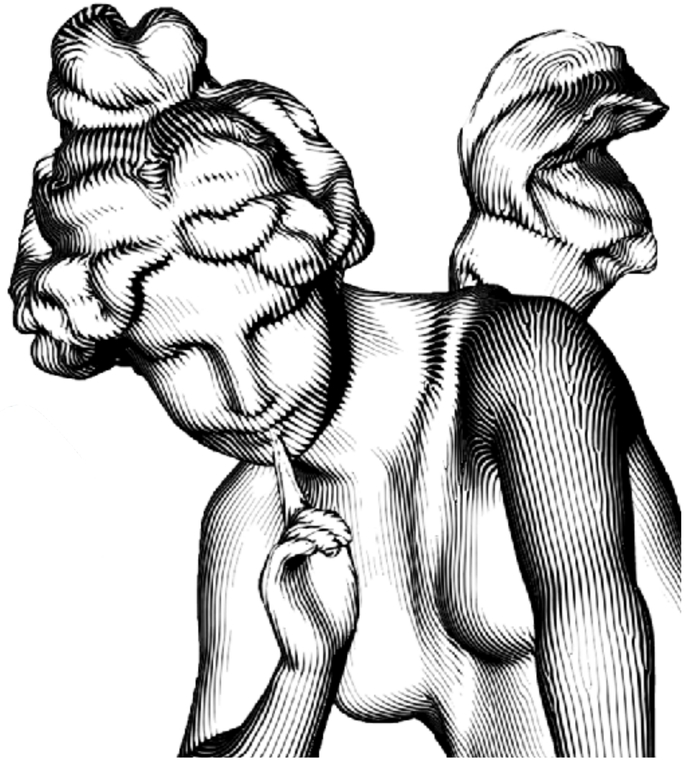
then focus on a specific case that happens often

in geometry processing

Non-Photorealistic Rendering

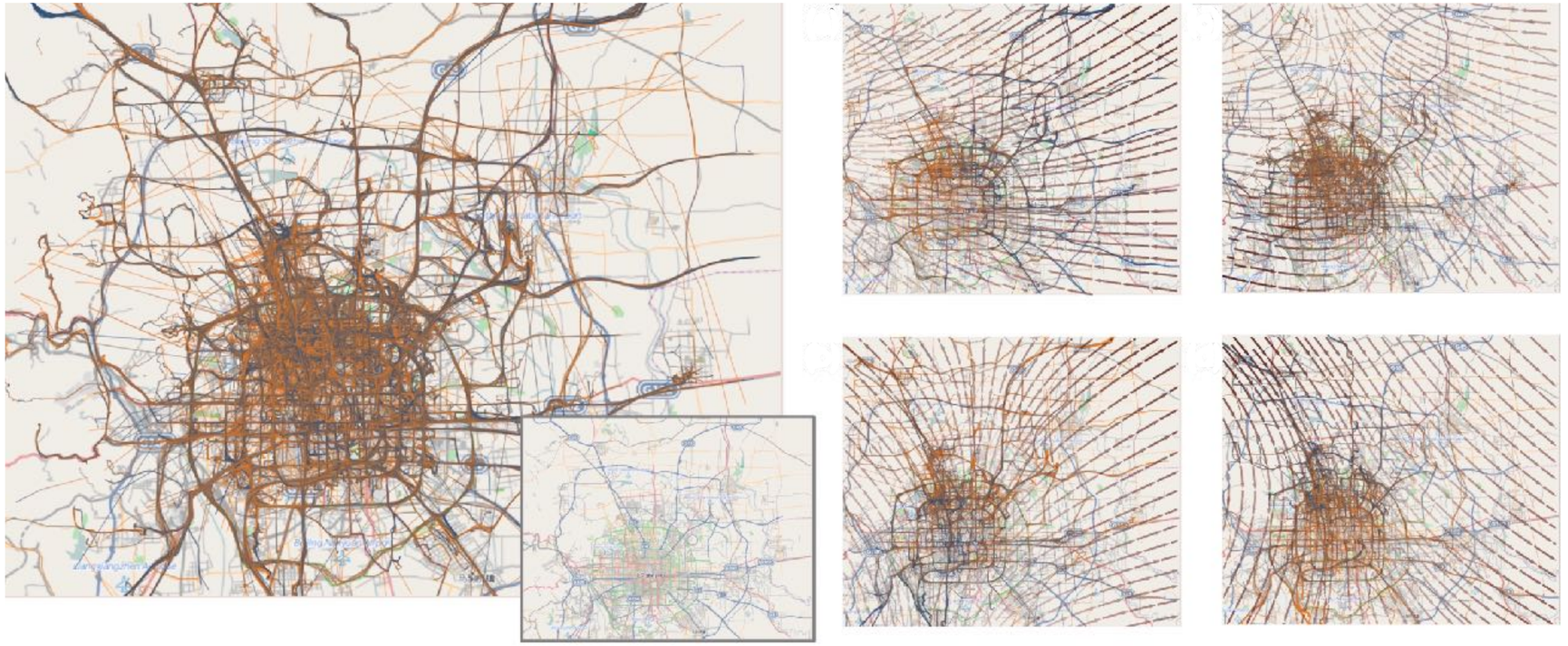


[Hertzmann & Zorin 2000]



[Knöppel et al. 2015]

Data Analysis

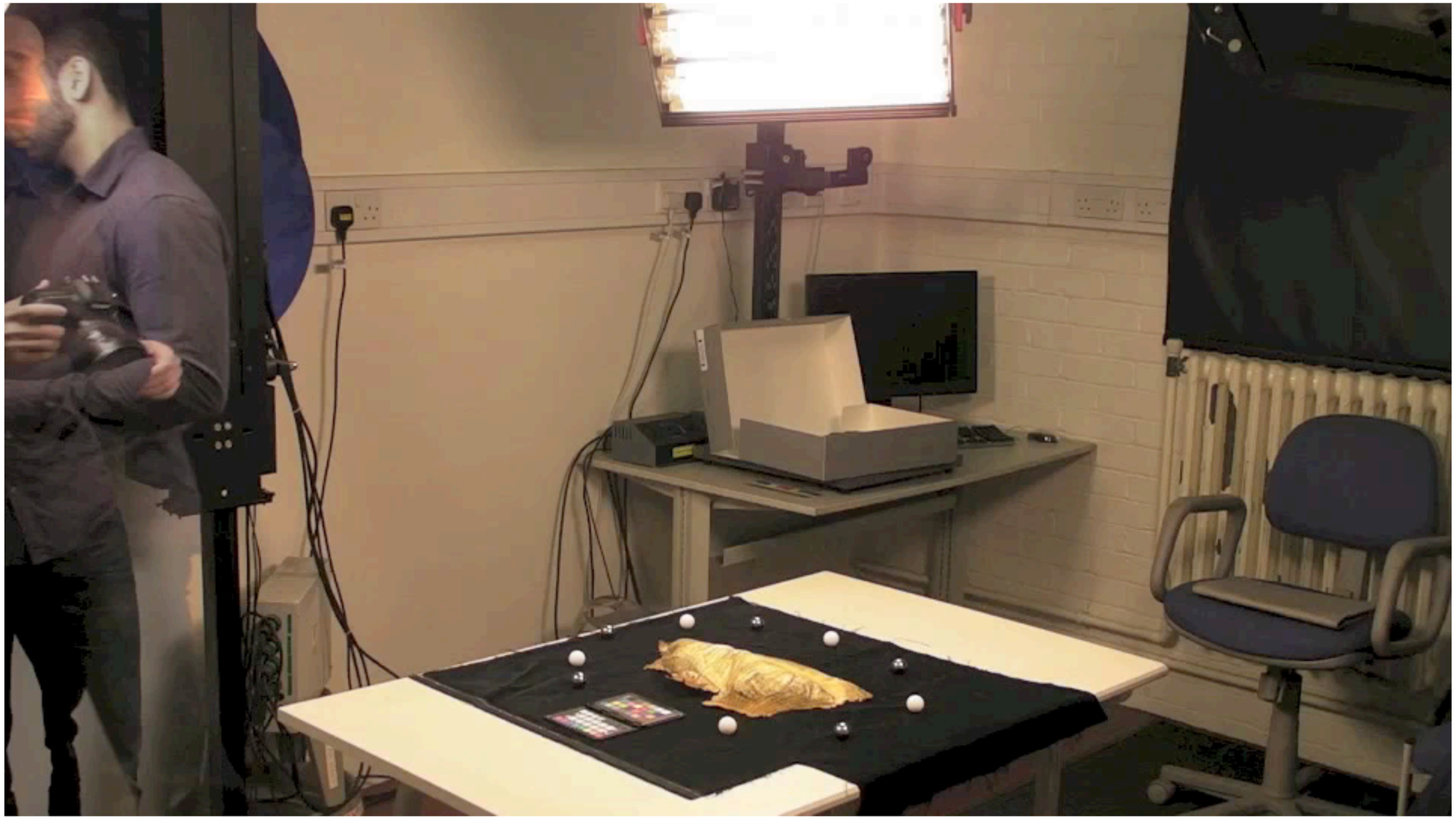


[Ferreira et al., 2013]

Cultural heritage

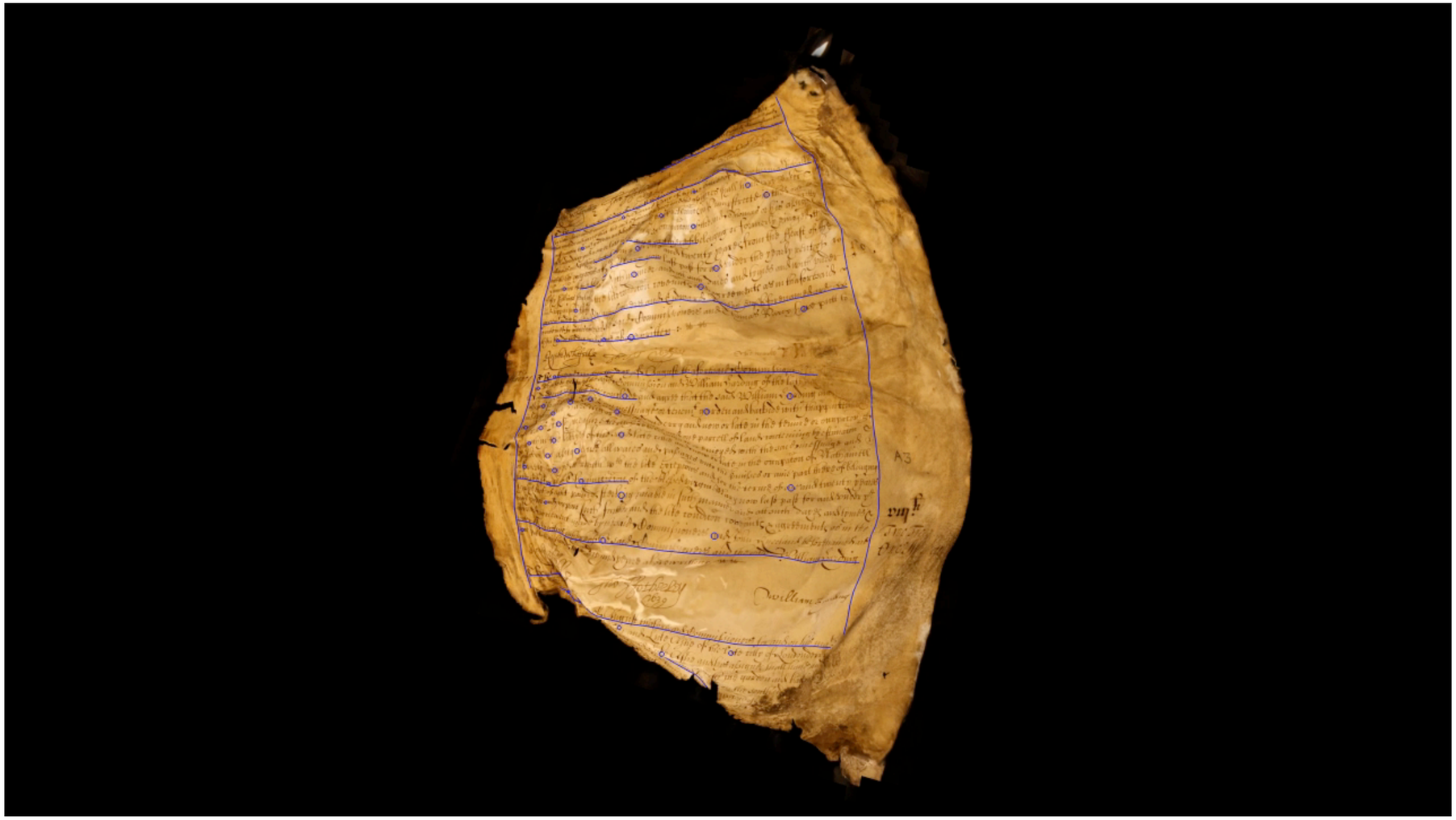
[Pal et al. 2014]

Cultural heritage



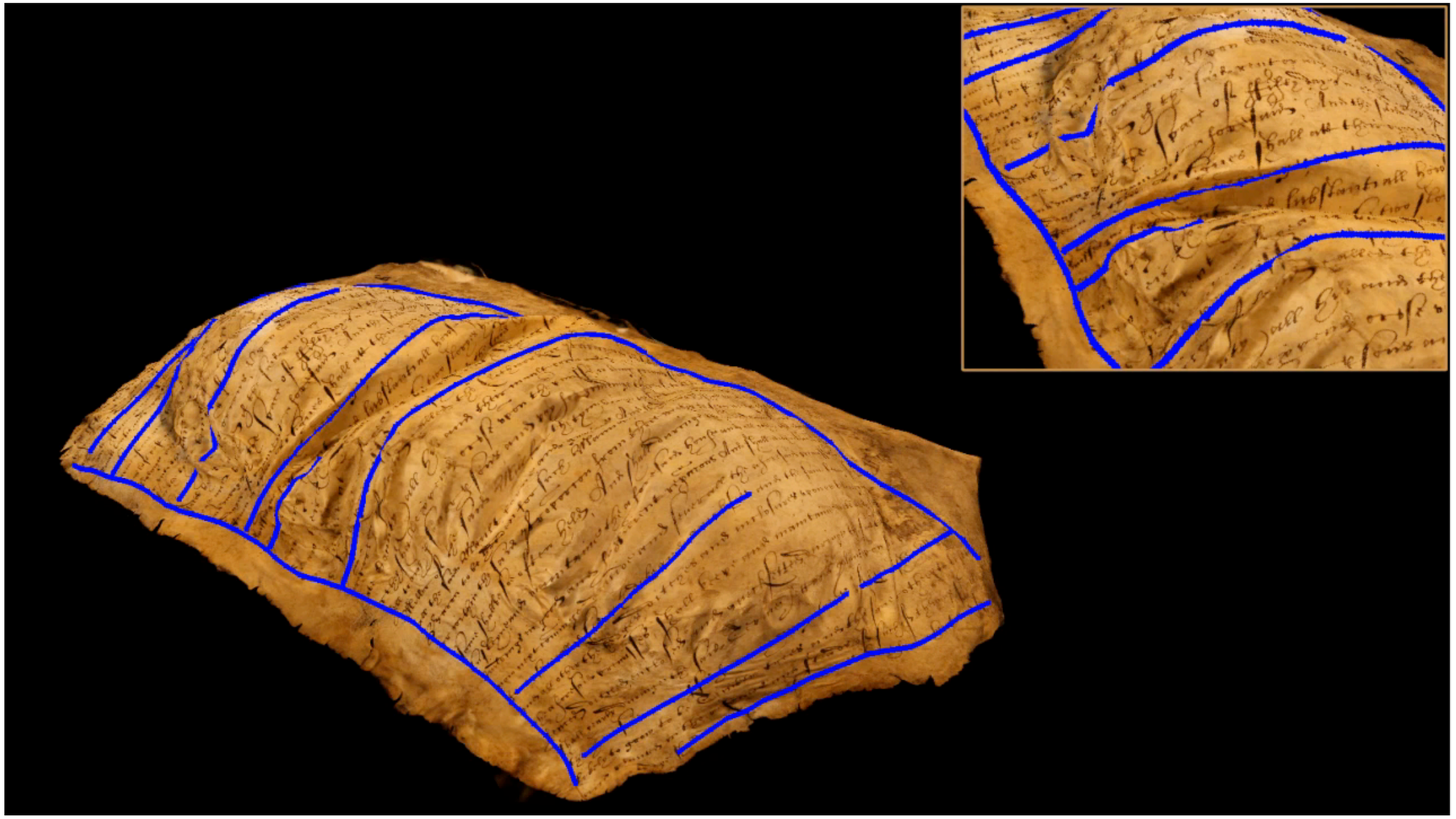
[Pal et al. 2014]

Cultural heritage



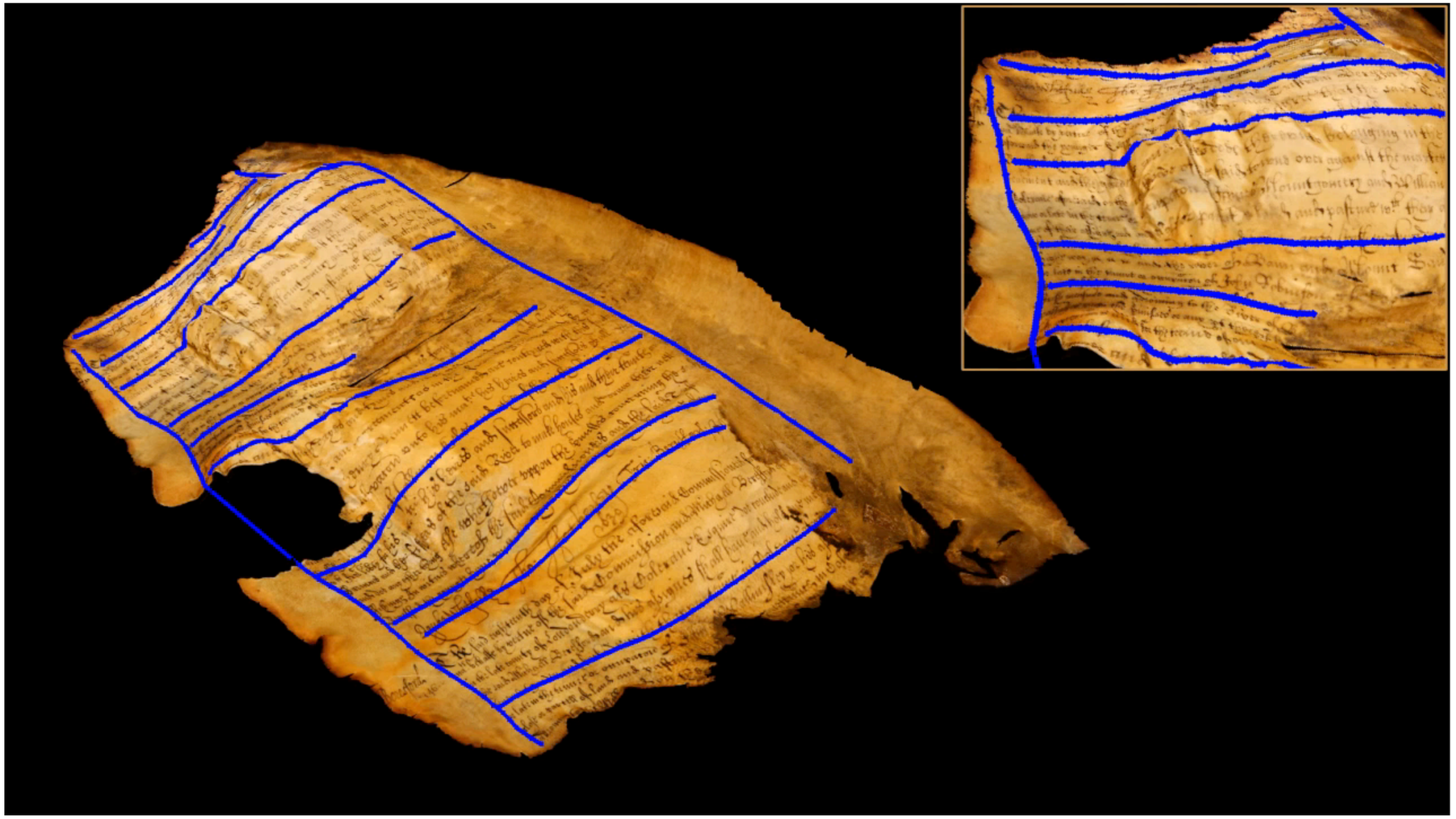
[Pal et al. 2014]

Cultural heritage



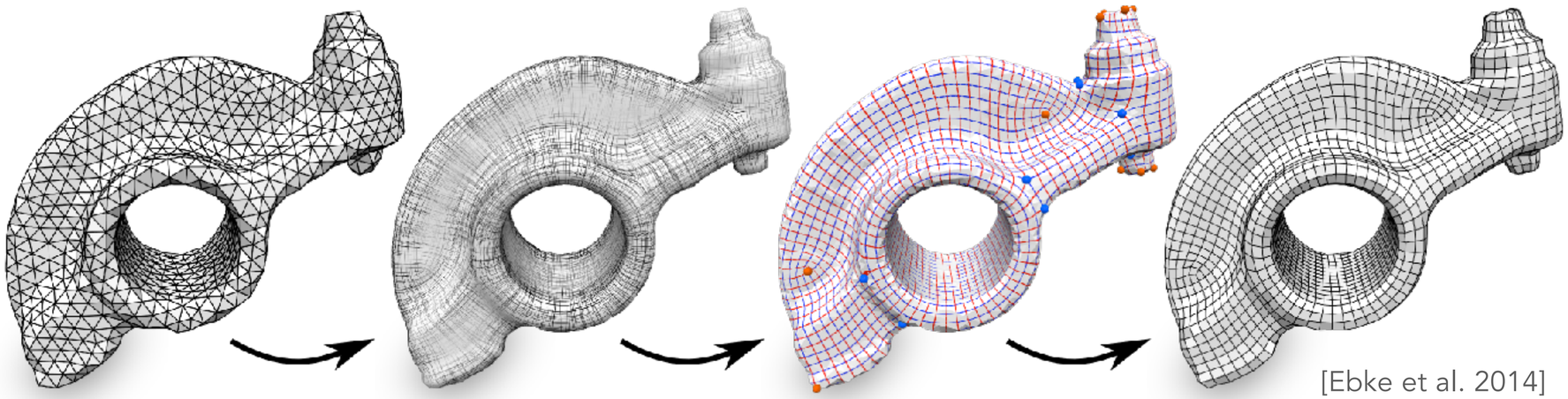
[Pal et al. 2014]

Cultural heritage

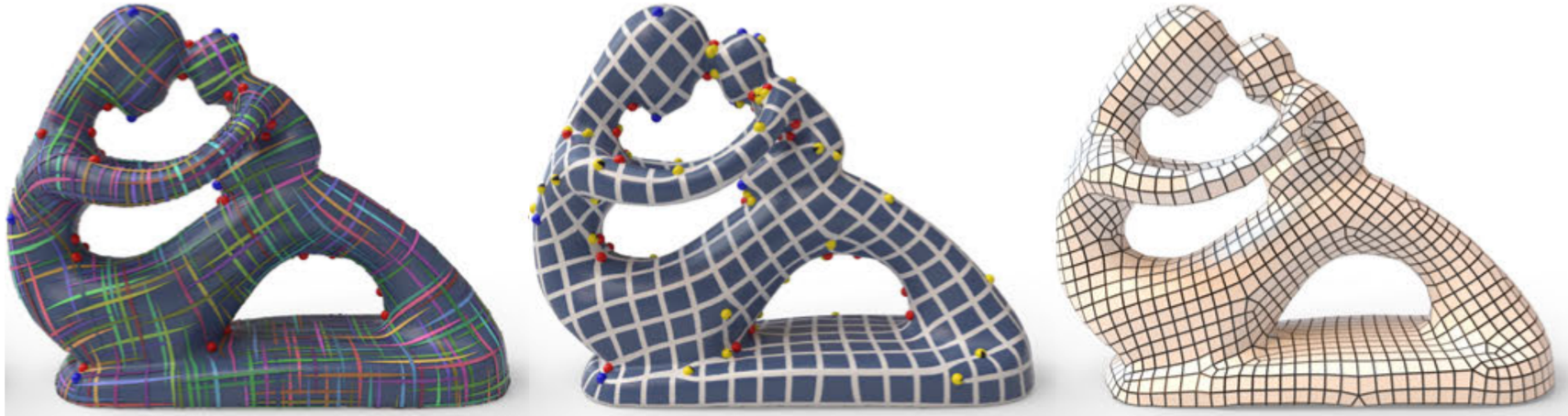


[Pal et al. 2014]

Meshing

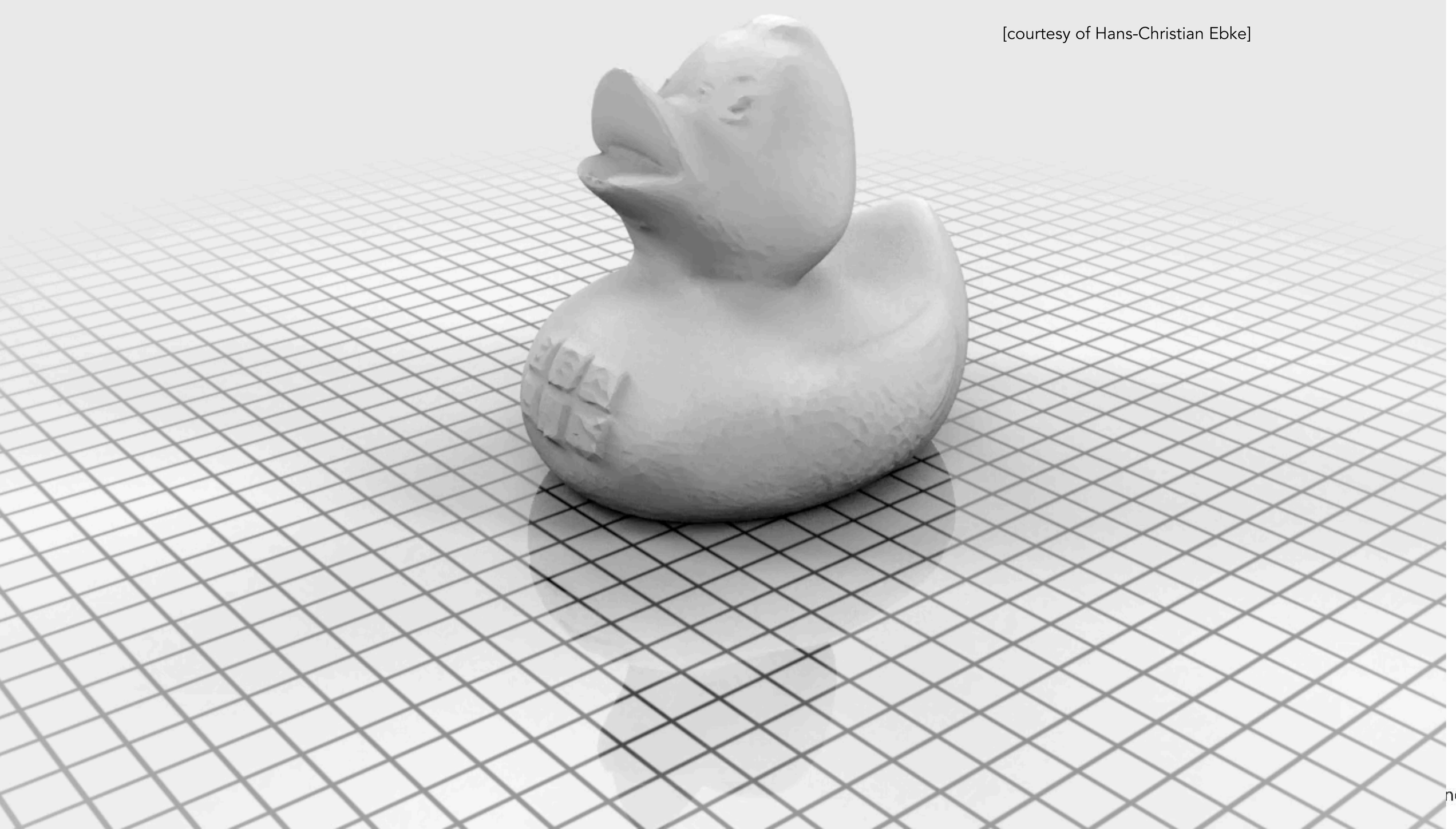


[Ebke et al. 2014]

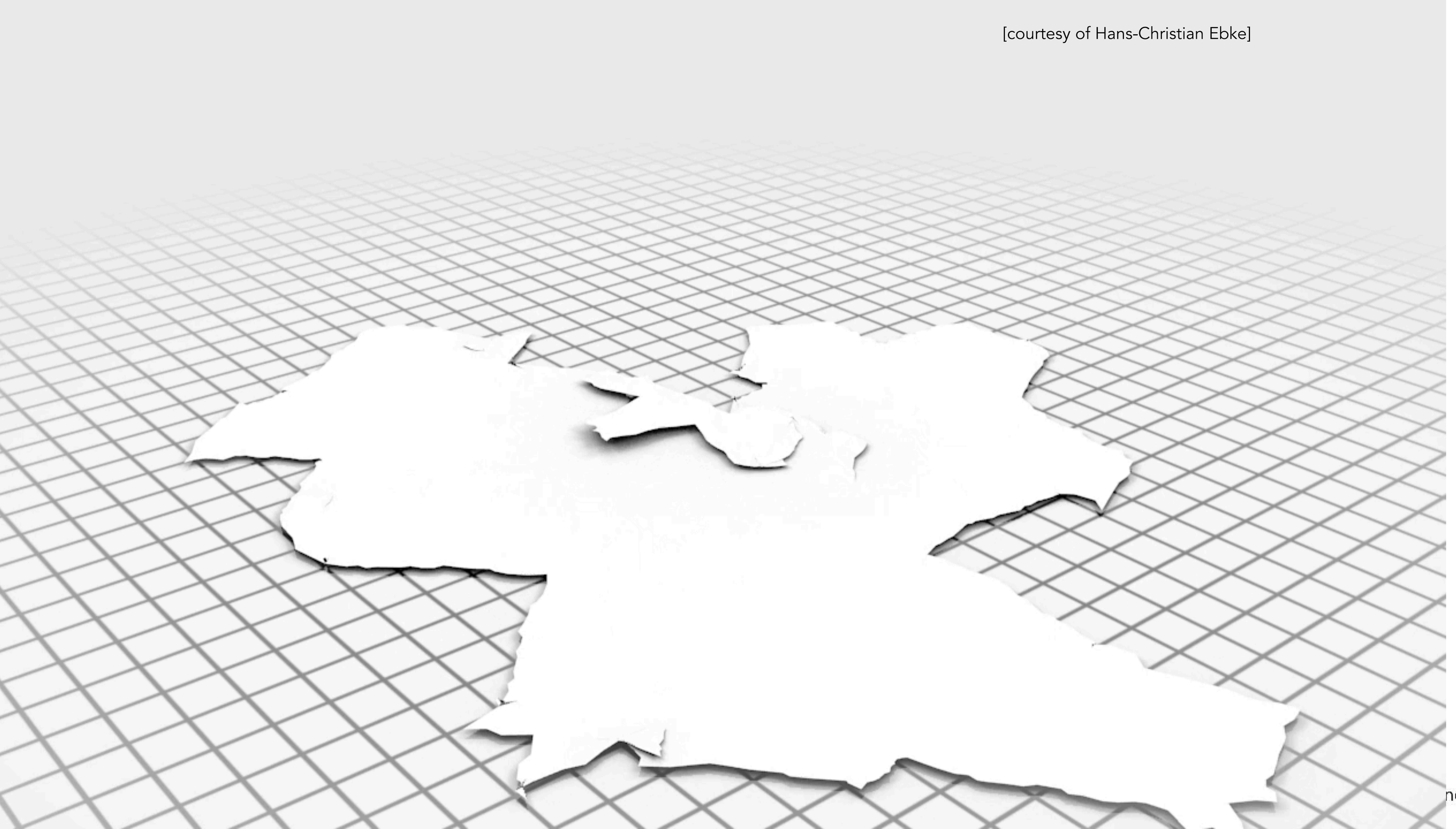


[Jakob et al. 2015]

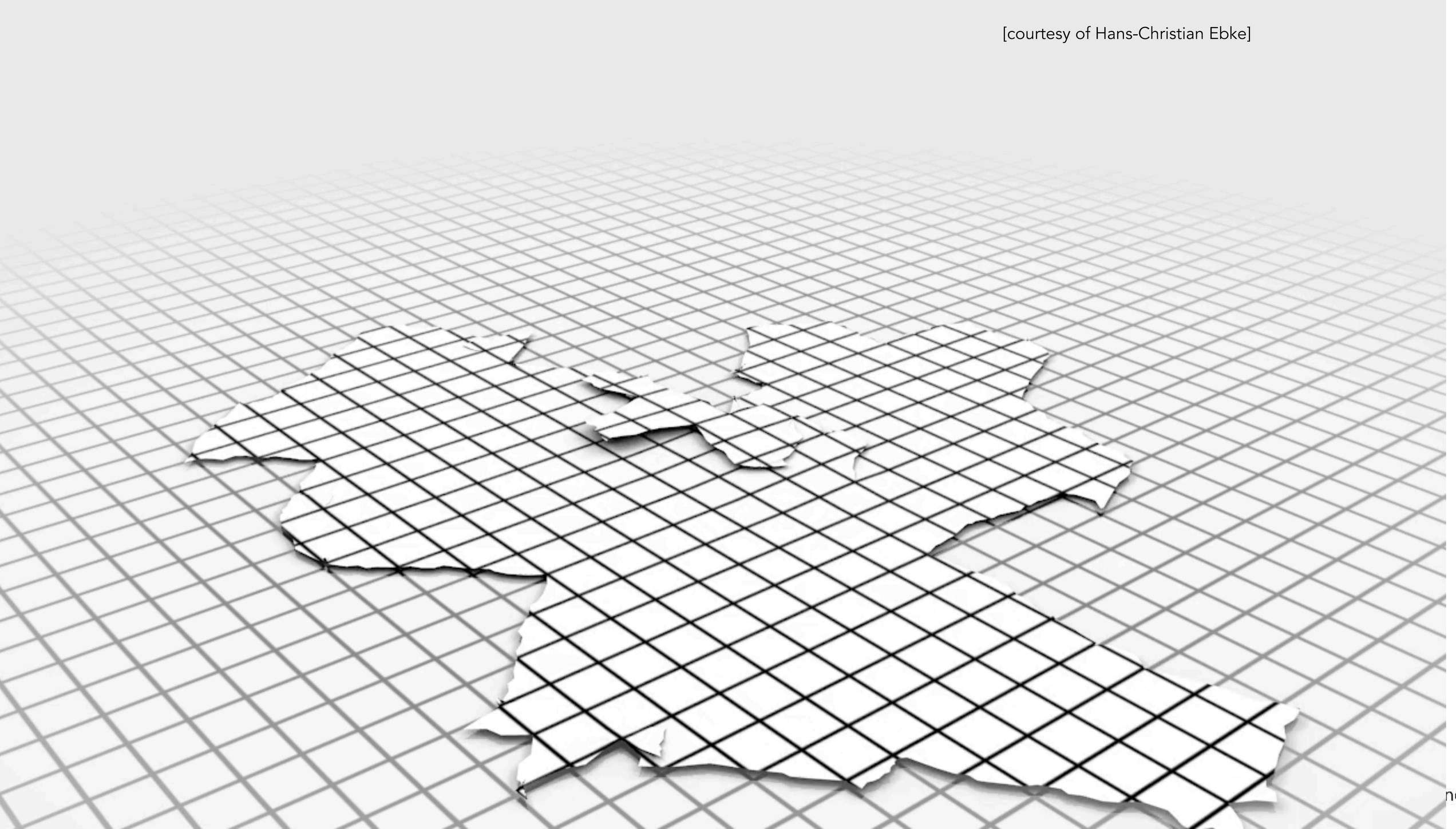
[courtesy of Hans-Christian Ebke]



[courtesy of Hans-Christian Ebke]

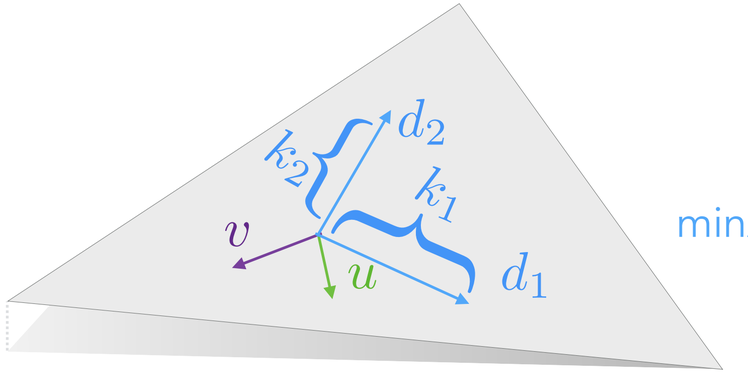


[courtesy of Hans-Christian Ebke]



[courtesy of Hans-Christian Ebke]

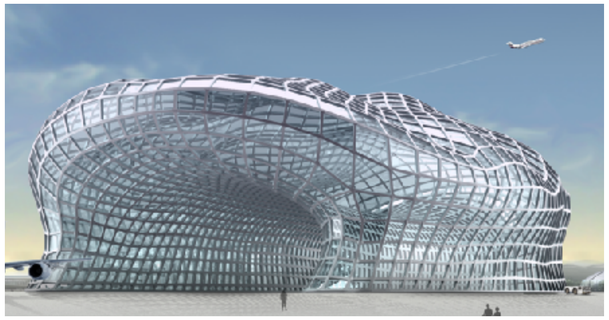
Conjugate Direction Fields - Planar Meshing



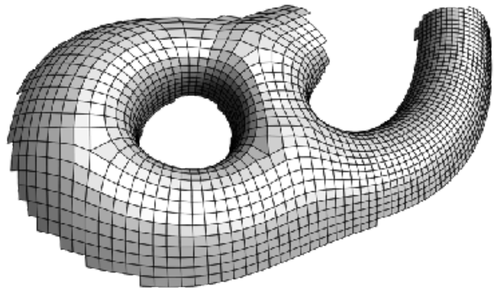
$$\kappa_1(u^T d_1)(v^T d_1) + \kappa_2(u^T d_2)(v^T d_2) = 0$$

min/max curvatures

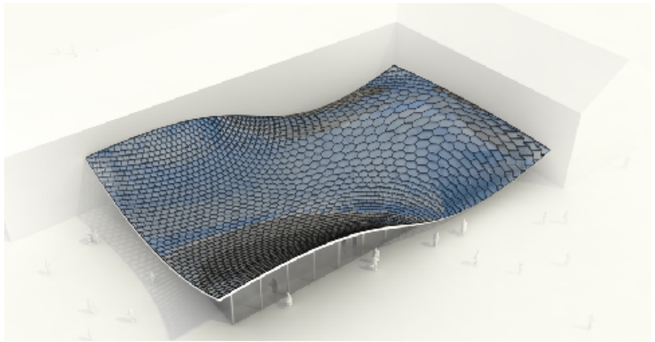
principal directions



[Liu et al. 2011]

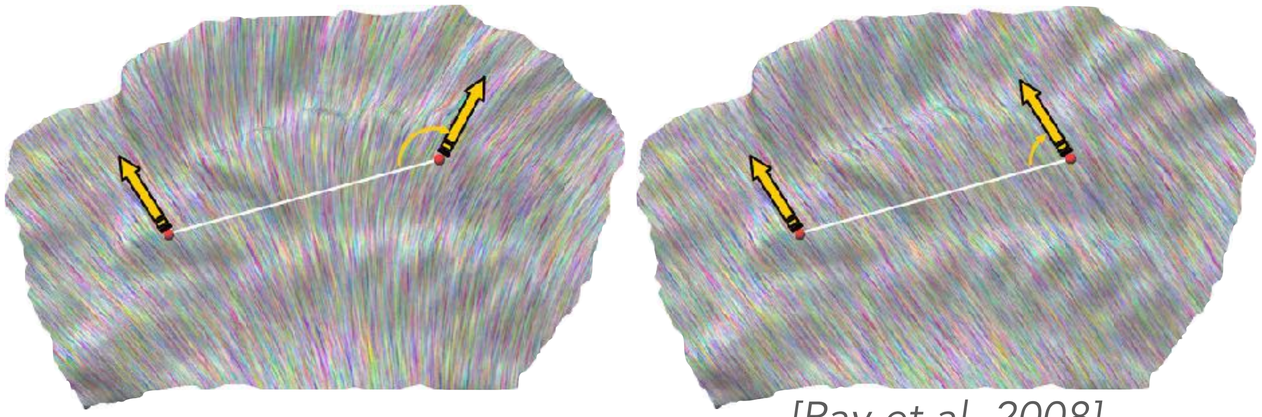


[Diamanti et al. 2015]



[Vaxman et al. 2015]

Vector Field Design



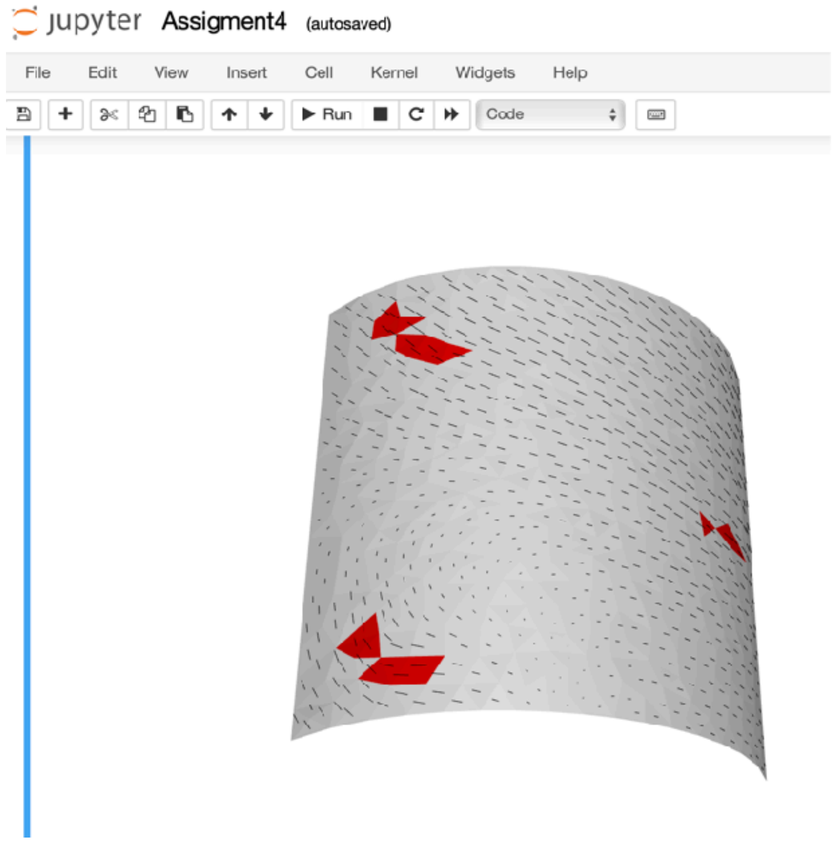
[Ray et al. 2008]

Vector Field Design

- We will code a complete implementation of a simple algorithm to design **vector field**
- We will implement a method based on [Knöppel et al. 2013] and [Diamanti et. al. 2014]
  - The field will be discretized on **faces**
  - The topology will be **free**
  - The objective will be **smoothness (i.e. as-constant-as-possible)**
  - We will support **soft alignment constraints**

Framework

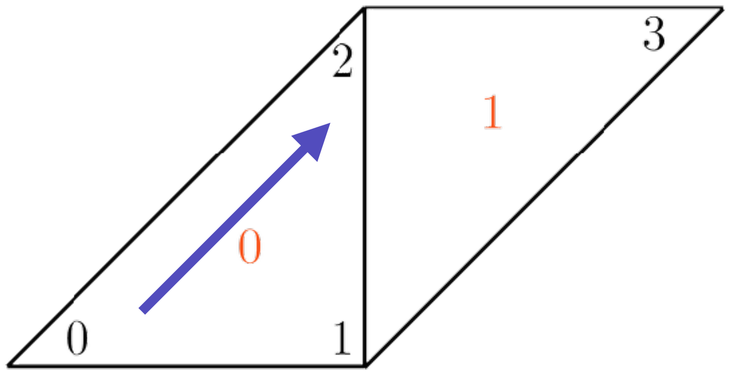
- The reference implementation is provided as part of Assignment 4
- Self-contained
- Constraints are specified programmatically, you might want to write a simple UI to experiment more freely, using https://ipywidgets.readthedocs.io/en/latest/examples/Using%20Interact.html



Input/Output

- Mesh V, F

$$V = \begin{matrix} x & y & z \end{matrix} \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \\ 2 & 1 & 0 \end{pmatrix} \qquad F = \begin{pmatrix} 0 & 1 & 2 \\ 1 & 3 & 2 \end{pmatrix}$$



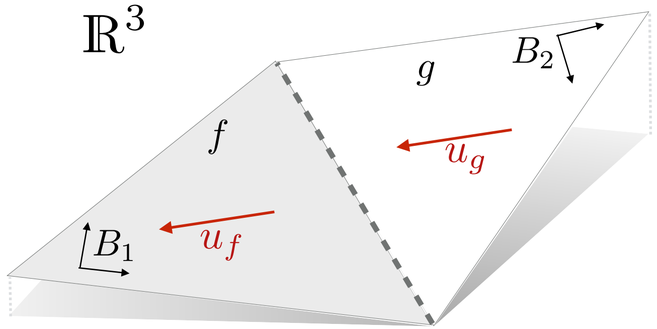
- Adjacency TT

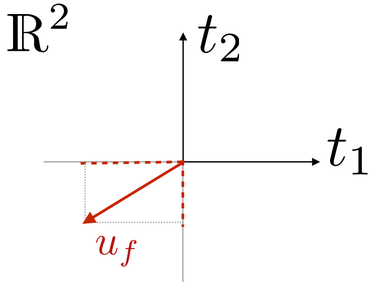
$$TT = \begin{pmatrix} -1 & 1 & -1 \\ -1 & -1 & 0 \end{pmatrix}$$

soft_id = 0, soft_value = (1 1 0)

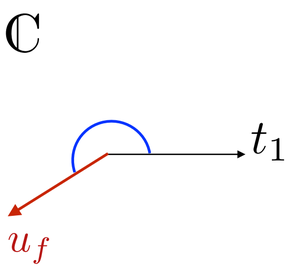
- Constrained face ids *soft_id* and directions *soft_value*

Discretization and Representation



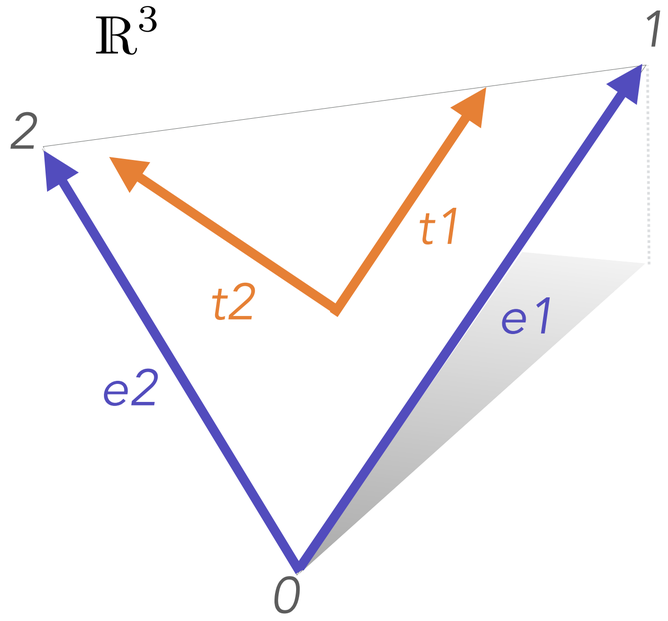


$$u_f = (u_f^x, u_f^y) \in \mathbb{R}^2$$



$$u_f = u_f^x + iu_f^y \in \mathbb{C}$$

Constructing Local Bases

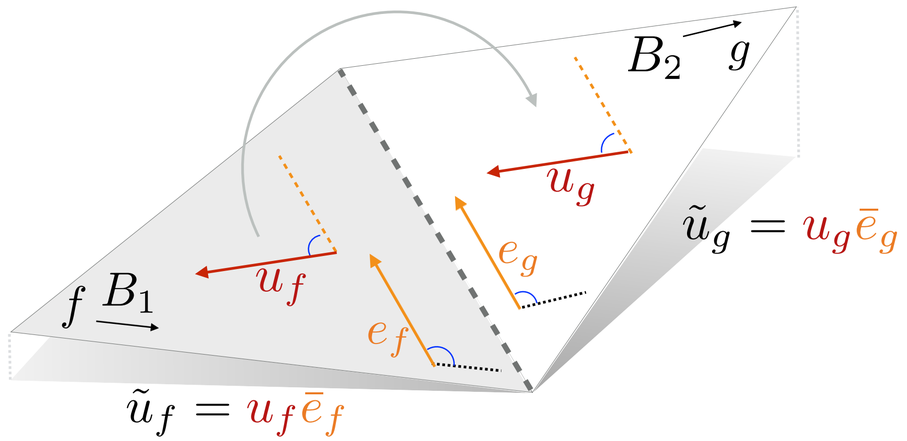



```
# Edges
e1 = V[F[:, 1], :] - V[F[:, 0], :]
e2 = V[F[:, 2], :] - V[F[:, 0], :]

# Compute the local reference systems for each face, T1, T2
T1 = e1 / np.linalg.norm(e1, axis=1)[:,None]

T2 =  np.cross(T1, np.cross(T1, e2))
T2 /= np.linalg.norm(T2, axis=1)[:,None]
```

Discrete Transport



Constant field (over an edge)

$$u_f \bar{e}_f = u_g \bar{e}_g$$

```
# Compute the complex representation of the common edge
e  = V[F[f, (ei+1)%3], :] - V[F[f, ei], :]

vef = np.array([np.dot(e, T1[f, :]), np.dot(e, T2[f, :])])
vef /= np.linalg.norm(vef)
ef = (vef[0] + vef[1]*1j).conjugate()

veg = np.array([np.dot(e, T1[g, :]), np.dot(e, T2[g, :])])
veg /= np.linalg.norm(veg)
eg = (veg[0] + veg[1]*1j).conjugate()
```

Energy Formulation

- An ideally constant field satisfies (for each edge):

$$(u_f \bar{e}_f - u_g \bar{e}_g)$$

- We want to find the field that is as-constant-as-possible:

$$f(\mathbf{u}) = \underbrace{\sum_{f,g \in \mathcal{E}} \|u_f \bar{e}_f - u_g \bar{e}_g\|^2}_{\text{As-constant-as-possible}} + \lambda \underbrace{\sum_{f \in \mathcal{C}} \|u_f - c_f\|2}_{\text{Soft-constraints}}$$

- We can rewrite this expression in matrix form:

$$f(\mathbf{u}) = \underbrace{\|\mathbf{L}\mathbf{u}\|^2}_{\text{As-constant-as-possible}} + \lambda \underbrace{\|\mathbf{C}\mathbf{u} - \mathbf{d}\|^2}_{\text{Soft-constraints}}$$

Energy Formulation

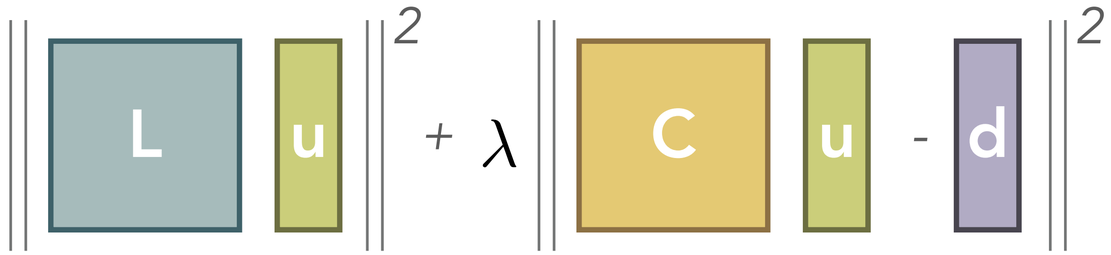
$$f(\mathbf{u}) = \|\mathbf{L}\mathbf{u}\|^2 + \lambda\|\mathbf{C}\mathbf{u} - \mathbf{d}\|^2$$

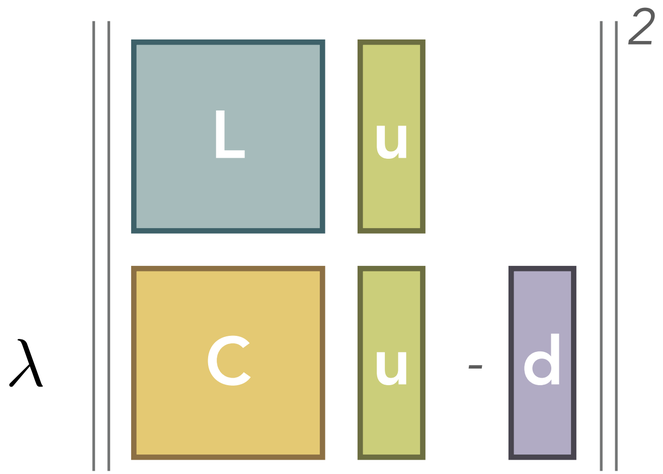
- We can rearrange the expression in a single norm

- which can be minimized solving a *complex* linear system

Energy Formulation

$$f(\mathbf{u}) = \|\mathbf{L}\mathbf{u}\|^2 + \lambda\|\mathbf{C}\mathbf{u} - \mathbf{d}\|^2$$





- We can rearrange the expression in a single norm
- which can be minimized solving a *complex* linear system

Energy Formulation

$$f(\mathbf{u}) = \|\mathbf{L}\mathbf{u}\|^2 + \lambda\|\mathbf{C}\mathbf{u} - \mathbf{d}\|^2$$

- We can rearrange the expression in a single norm

$$f(\mathbf{u}) = \left\| \begin{matrix} \mathbf{L}\mathbf{u} \\ \sqrt{\lambda}\mathbf{C}\mathbf{u} - \sqrt{\lambda}\mathbf{d} \end{matrix} \right\|^2 = \|\mathbf{A}\mathbf{u} - \mathbf{b}\|^2$$

$$\mathbf{A} = \begin{pmatrix} \mathbf{L} \\ \sqrt{\lambda}\mathbf{C} \end{pmatrix}$$

$$\mathbf{b} = \begin{pmatrix} \mathbf{0} \\ \sqrt{\lambda}\mathbf{d} \end{pmatrix}$$

- which can be minimized solving a complex linear system

$$\nabla f(\mathbf{u}) = \mathbf{A}^*\mathbf{A}\mathbf{u} - \mathbf{A}^*\mathbf{b} = 0$$

$\mathbf{u}$ *Optimized Field*

System Assembly - Smoothness

- For each edge we want one row in **A** $\|u_f \bar{e}_f - u_g \bar{e}_g\|^2$ $\|\mathbf{Lu}\|^2$

```
index = 0
for f in range(F.shape[0]):
    for ei in range(3): # Loop over the edges

        # Look up the opposite face
        g = TT[f, ei]

        # If it is a boundary edge, it does not contribute to the energy
        # or avoid to count every edge twice
        if g == -1 or f > g:
            continue

        # Compute the complex representation of the common edge
        e  = V[F[f, (ei+1)%3], :] - V[F[f, ei], :]

        vef = np.array([np.dot(e, T1[f, :]), np.dot(e, T2[f, :])])
        vef /= np.linalg.norm(vef)
        ef = (vef[0] + vef[1]*1j).conjugate()

        veg = np.array([np.dot(e, T1[g, :]), np.dot(e, T2[g, :])])
        veg /= np.linalg.norm(veg)
        eg = (veg[0] + veg[1]*1j).conjugate()


        # Add the term conj(f)^n*ui - conj(g)^n*uj to the energy matrix
        data.append(ef);  ii.append(index); jj.append(f)
        data.append(-eg); ii.append(index); jj.append(g)

        index += 1
```

System Assembly - Soft Constraints

- Similarly, we add a row to **A** for each constrained face

- Note that the corresponding entry of **b** is not zero

$$\lambda \sum_{f \in \mathcal{C}} \|u_f - c_f\|^2 \qquad \|\sqrt{\lambda}\mathbf{C}\mathbf{u} - \sqrt{\lambda}\mathbf{d}\|^2$$

```
b = np.zeros(index + soft_id.shape[0], dtype=complex)

for ci in range(soft_id.shape[0]):
    f = soft_id[ci]
    v = soft_value[ci, :]

    # Project on the local frame
    c = np.dot(v, T1[f, :]) + np.dot(v, T2[f, :])*1j

    data.append(sqrtl); ii.append(index); jj.append(f)
    b[index] = c * sqrtl

    index += 1
```

Solving the Linear System

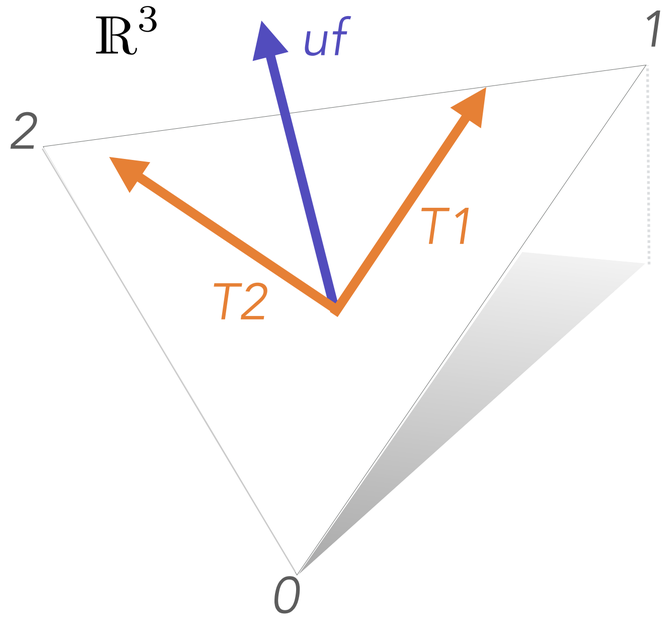
- The linear system is solved using a sparse direct solver.

$$\|\mathbf{A}\mathbf{u} - \mathbf{b}\|^2$$

$$\nabla f(\mathbf{u}) = \mathbf{A}^*\mathbf{A}\mathbf{u} - \mathbf{A}^*\mathbf{b} = 0$$

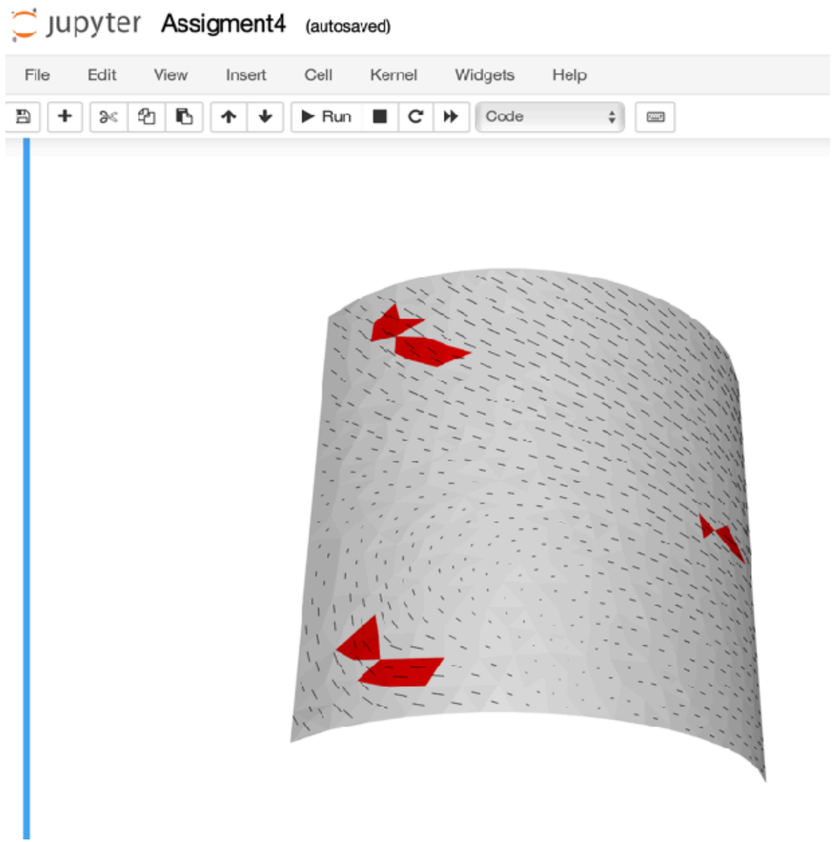
```
# Solve the linear system
A = sp.coo_matrix((data, (ii, jj)), shape=(index, F.shape[0])).asformat("csr")
u = sp.linalg.spsolve(A.H @ A, A.H @ b)
```

Extraction of the Interpolated Field




```
R = T1 * u.real[:,None] + T2 * u.imag[:,None]
```

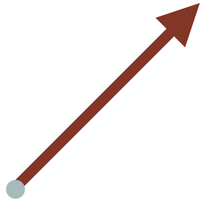
Demo


jupyter Assigment4 (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Run
Code

Taxonomy for general directional fields

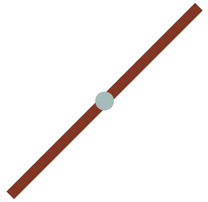
Taxonomy

1-vector



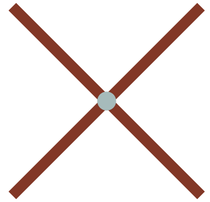
"classical vector"

2-direction



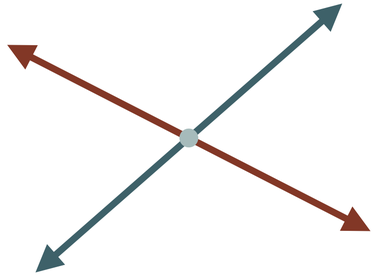
"line", "2-RoSy"

4-direction



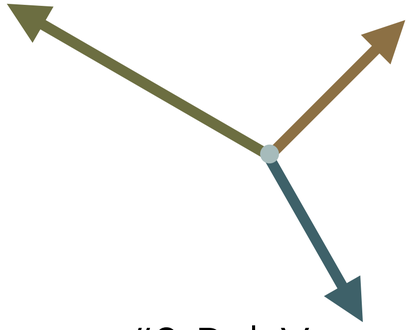
"unit cross", "4-RoSy"

2^2-vector



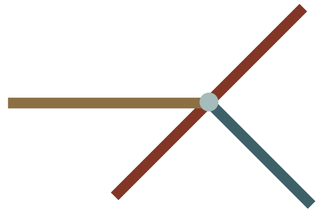
"Frame field"

1^3-vector



"3-PolyVector"

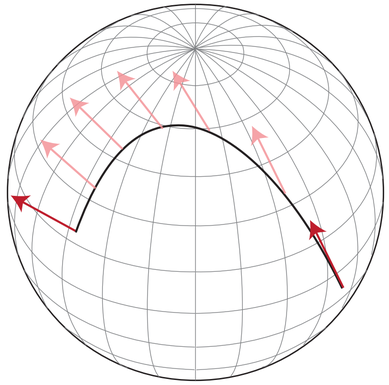
$(1^2, 2)$-direction



"4-PolyDirection
including a line"

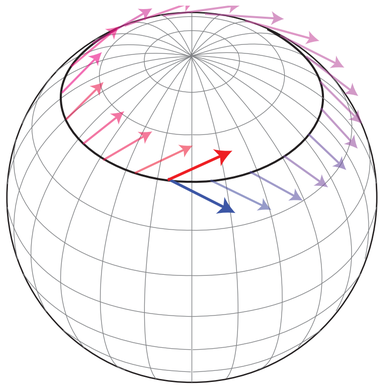
Discretization

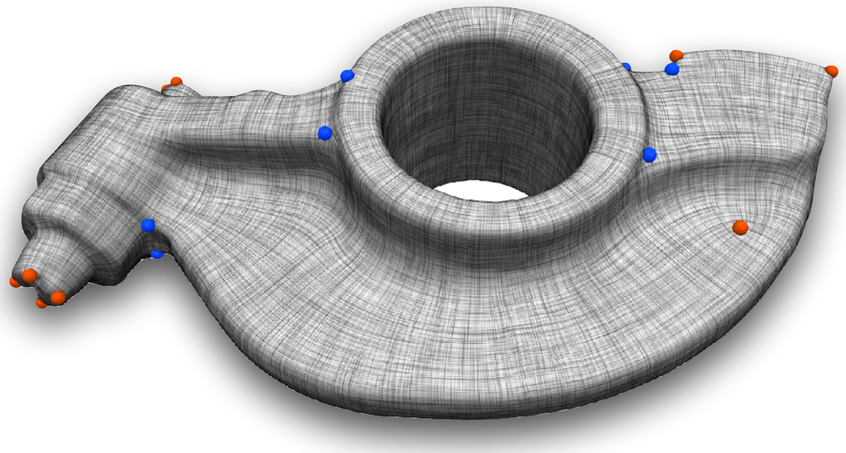
Well Studied in the Continuum



Parallel Transport

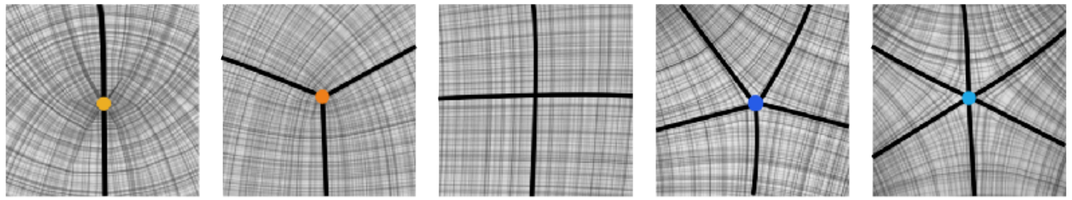
Holonomy





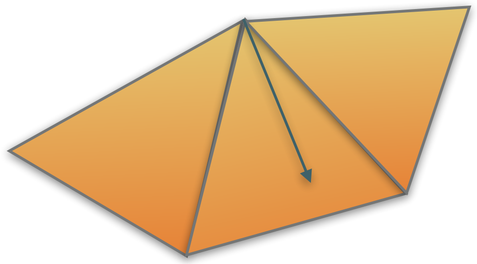
Poincare-Hopf Theorem

Singularities



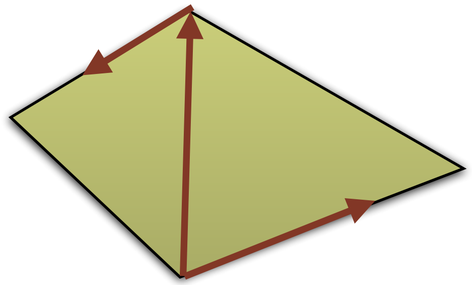
Discretizations

Vertex based



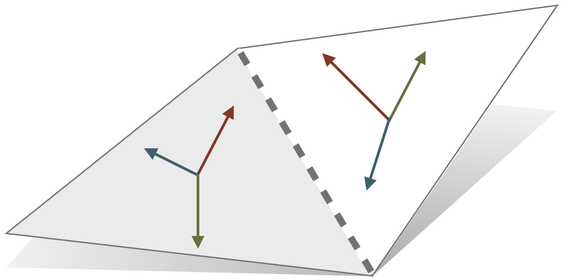
[Polthier and Schmies 98]
[Zhang *et al.* 2006]
[Knöppel *et al.* 2013]

Edge based



[Desbrun *et al.* 2005]
[Fisher *et al.* 2007]
[Ben-Chen *et al.* 2010]

Face based

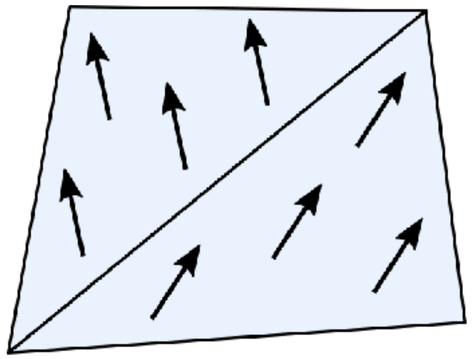


[Bommes *et al.* 2009]
[Crane *et al.* 2010]
[Diamanti *et al.* 2014]

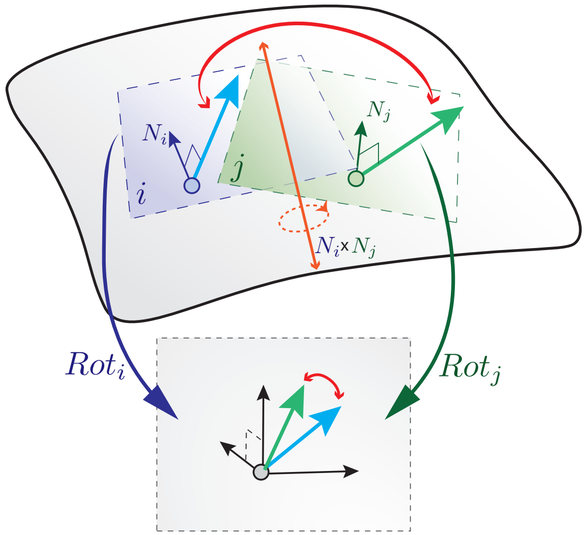
Choice of tangent space and differential implications: see [deGoes *et al.* 2016].

Challenges in the Discrete Setting

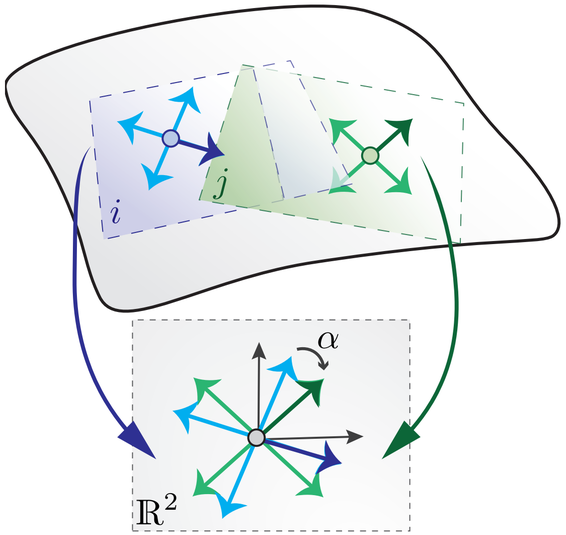
Discontinuity



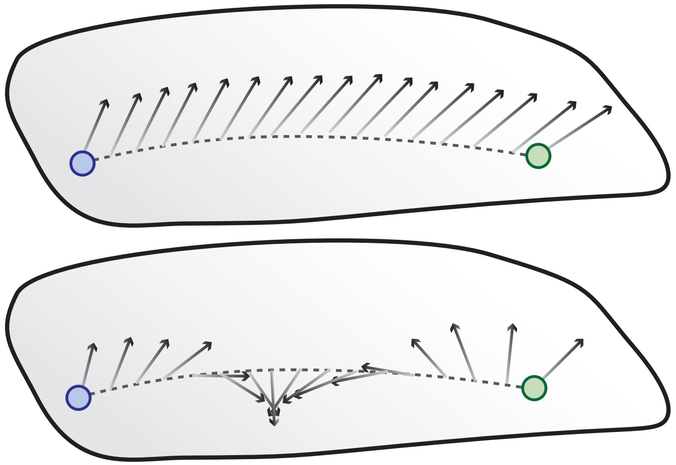
Connection



Matching



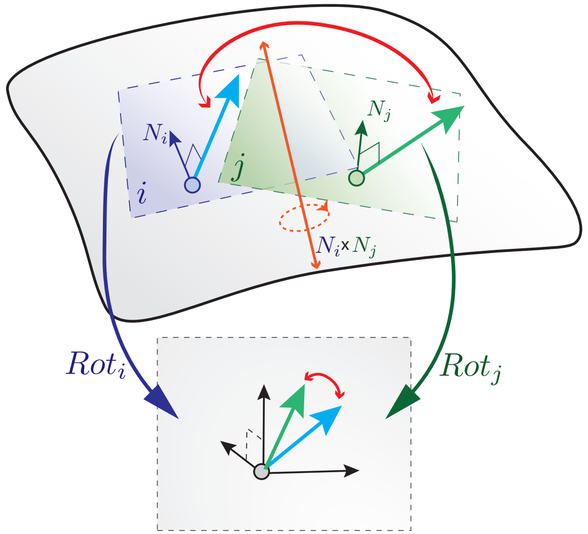
Interpolation



Discrete Connection

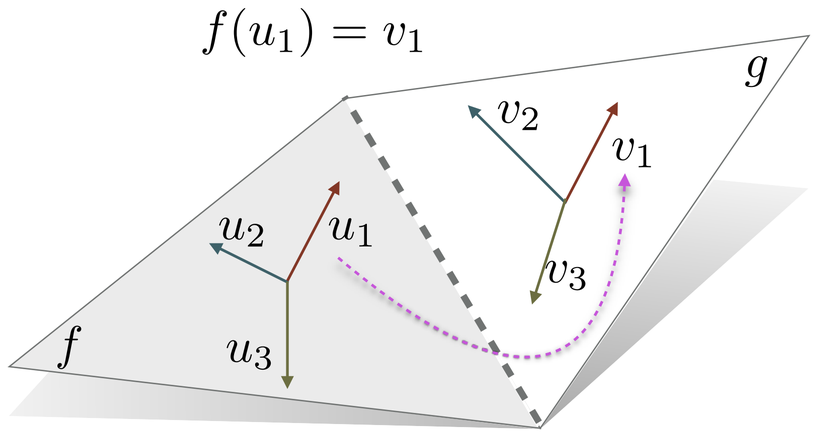
- Bijective linear map between adjacent tangent spaces.
- Popular choice: flattening

[Ray *et al.* 2008]
[Crane *et al.* 2010]
[Knöppel and Pinkall 2015]



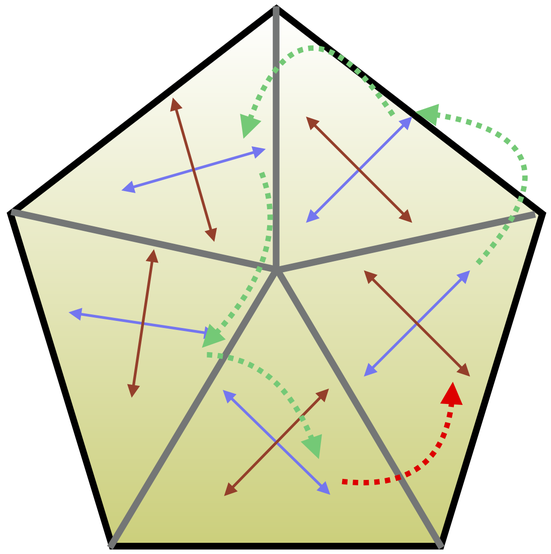
Discrete Matching

- Which direction to which other?
- *N*-directional: *N* possible choices.
  - How best to choose?



Singularities

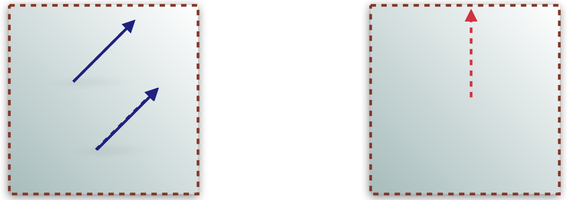
- Around a matching cycle, directional returns to itself.
    - Up to a different matching!



Index of singularity: $\frac{1}{k}$

Discrete Interpolation

- What happens in between?
  - Valid rotation choices:



$$\delta_{ij} = \frac{\pi}{4} + 2\pi k, \; k \in \mathbb{Z}$$

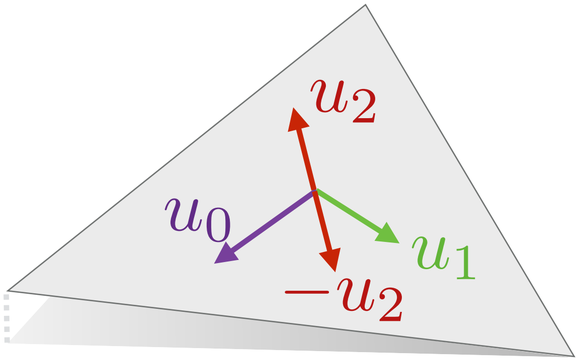
Principal Rotation *Period Jump*

[Li *et al.* 2006]

- Implicit: can only assume principal.
- Explicit: period given.

Polyvector Representation

N-PolyVectors



$$P(z) = (z - u_0)(z - u_1)(z^2 - {u_2}^2)$$

$$P(z) = z^4 \overbrace{- (u_0 + u_1)}^{x_3} z^3 + \overbrace{(u_0 u_1 - {u_2}^2)}^{x_2} z^2 + \overbrace{{u_2}^2 (u_0 + u_1)}^{x_1} z - \overbrace{u_0 u_1 {u_2}^2}^{x_0}$$

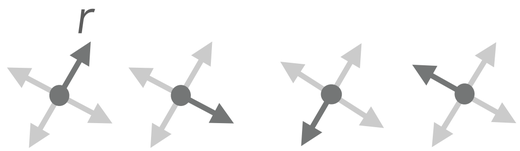
order: 1 order: 2 order: 3 order: 4

[Knöppel et al. 2013] [Diamanti et al. 2014]

N-vector fields

- Instead of representing the field with a 2D vector (complex number) we use a complex polynomial to represent a n-vector field.

$$p(z) = z^n - r^n$$



- Our new variable to interpolate is

$$u = r^n$$

- The transport slightly changes in

$$(u_f(\overline{e}_f)^n - u_g(\overline{e}_g)^n)^2$$

- The constraints need to be converted with

$$u = r^n$$

- To extract the field, we need to find the roots of *p*

[Knöppel et al. 2013] [Diamanti et al. 2014]

Roots of a complex polynomial - companion matrix

- The roots of a polynomial in the form

$$p(t) = c_0 + c_1 t + \cdots + c_{n-1} t^{n-1} + t^n$$

- are the eigenvalues of the corresponding companion matrix

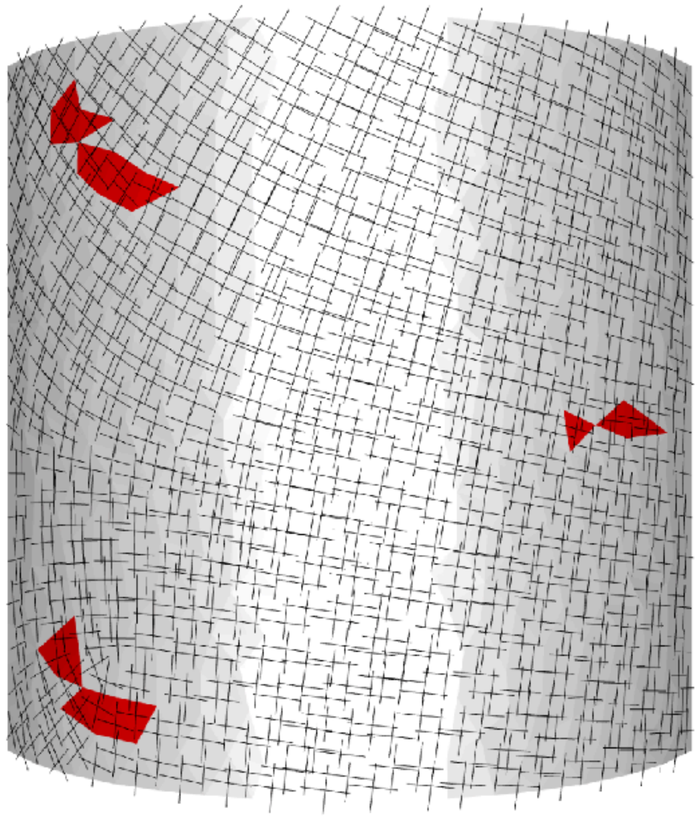
$$C(p) = \begin{bmatrix} 0 & 0 & \dots & 0 & -c_0 \\ 1 & 0 & \dots & 0 & -c_1 \\ 0 & 1 & \dots & 0 & -c_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & -c_{n-1} \end{bmatrix}.$$

```
def find_root(c0,n):
    M = np.zeros((n,n),dtype=complex)
    for i in range(1,n):
        M[i,i-1] = 1.0
    M[0,n-1] = c0
    w, v = np.linalg.eig(M)
    return w[0]
```

- For more information: https://en.wikipedia.org/wiki/Companion_matrix

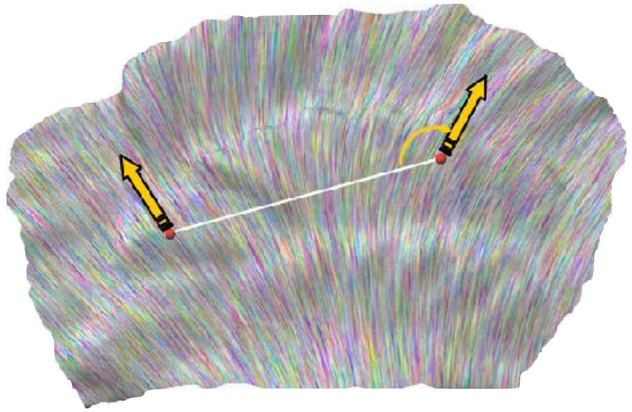
Polyvector Demo

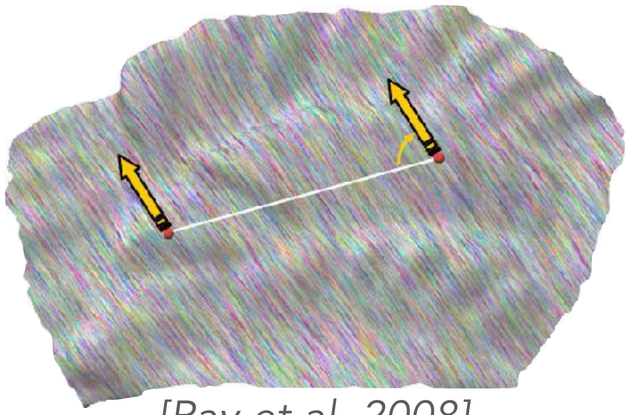
```
In [6]: R = align_field(v, f, tt, cf, c, 1e6, 4)
        plot_mesh_field(v, f, R, cf, 4)
```



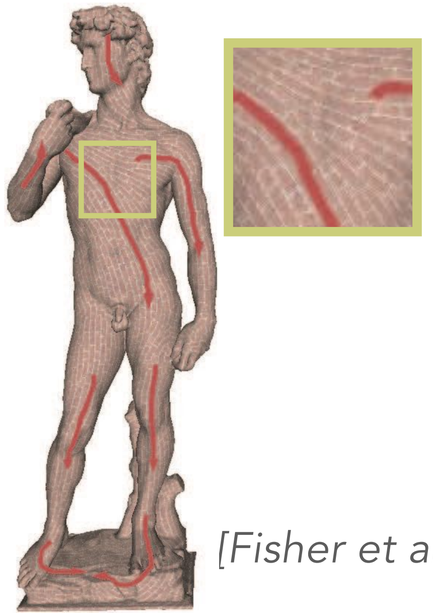
Constraints

Constraints - Alignment





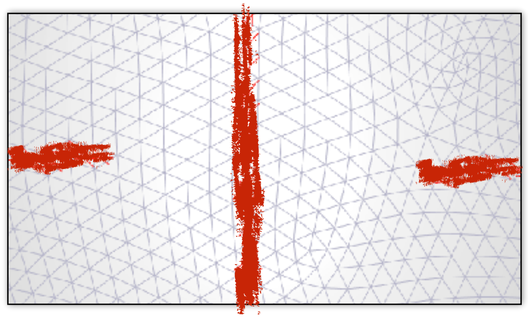
[Ray et al. 2008]

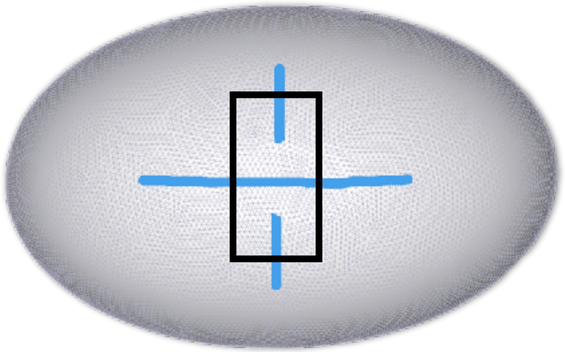


[Fisher et al. 2007]

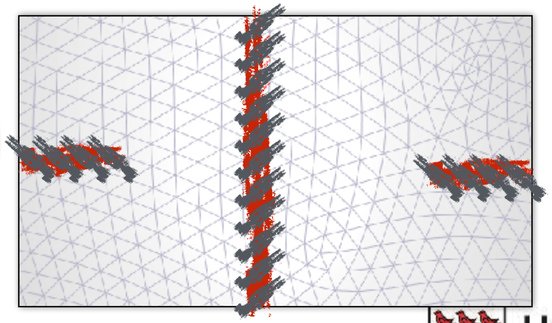
- Complete or Partial

partial constraints

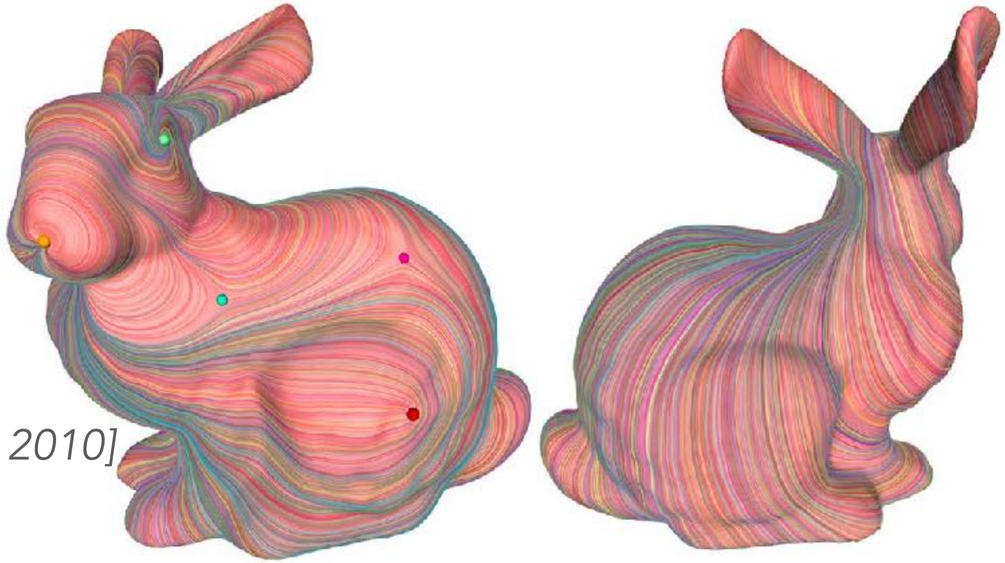




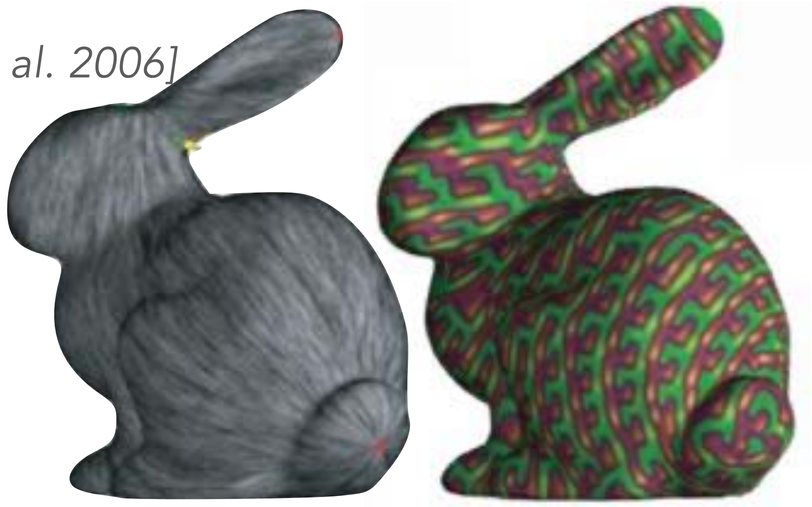
full constraints



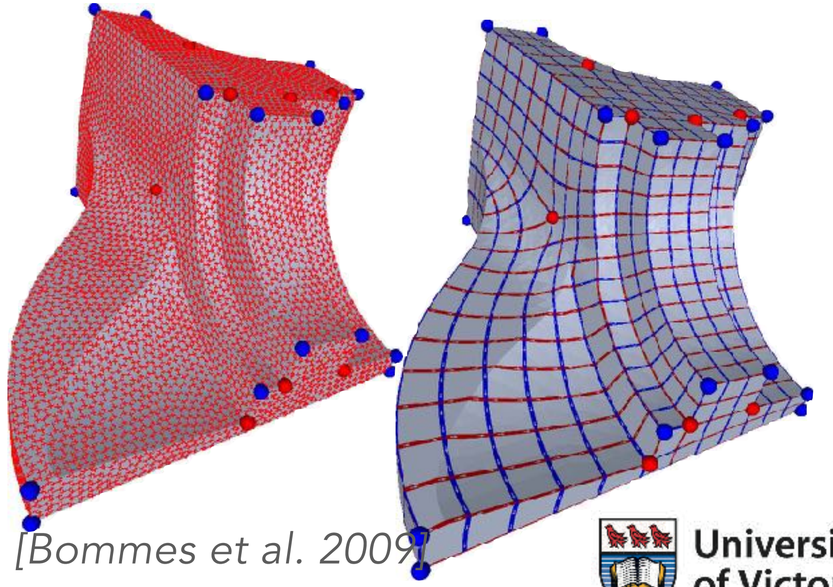
Constraints - Topology



[Crane et al. 2010]

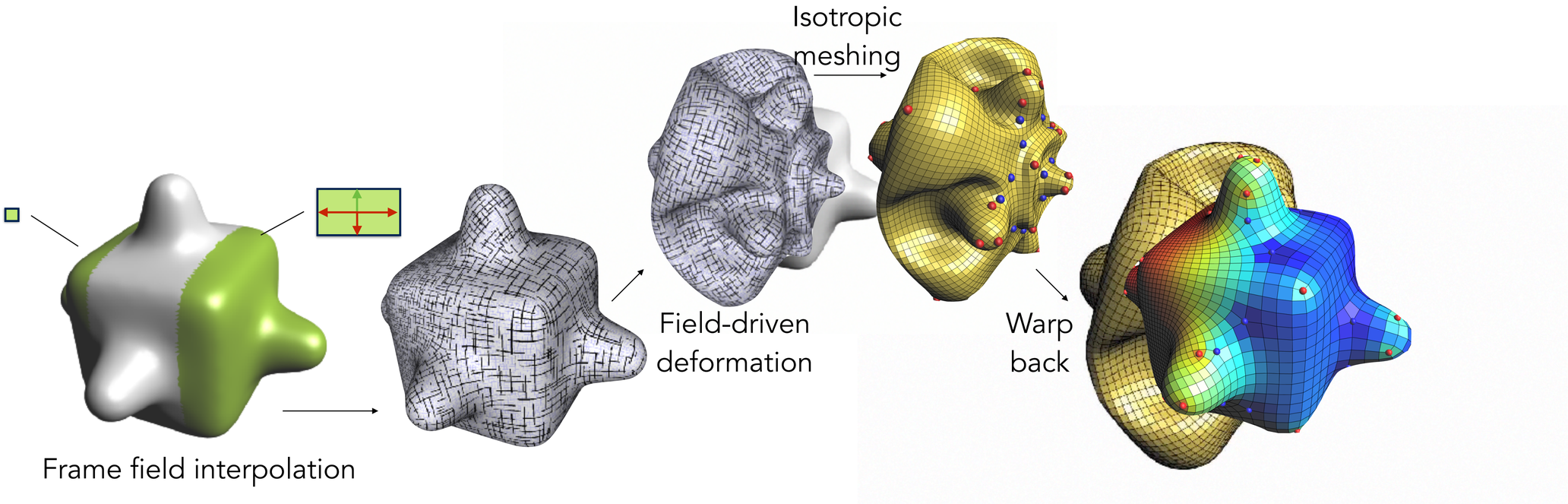


[Zhang et al. 2006]



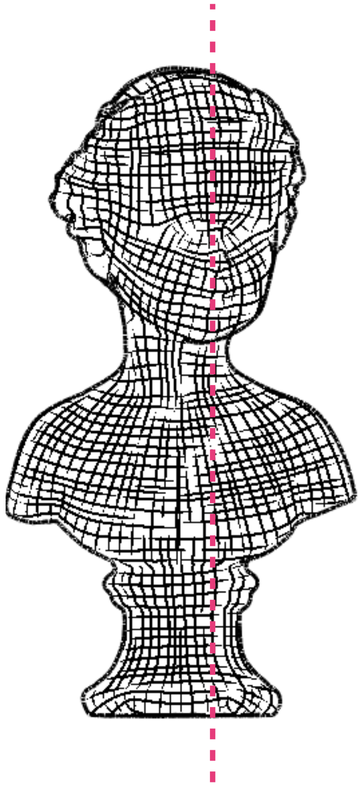
[Bommes et al. 2009]

Constraints - Scale

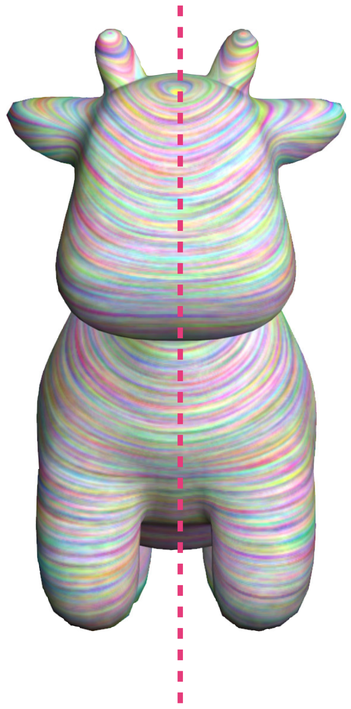


[Panozzo et al. 2014]

Constraints - SYMMETRY

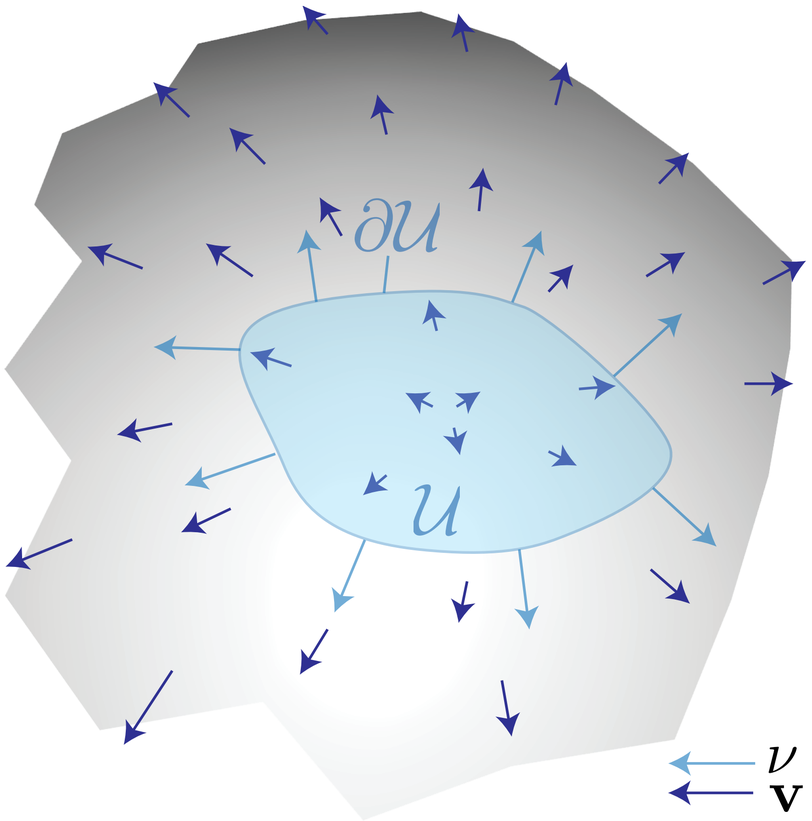


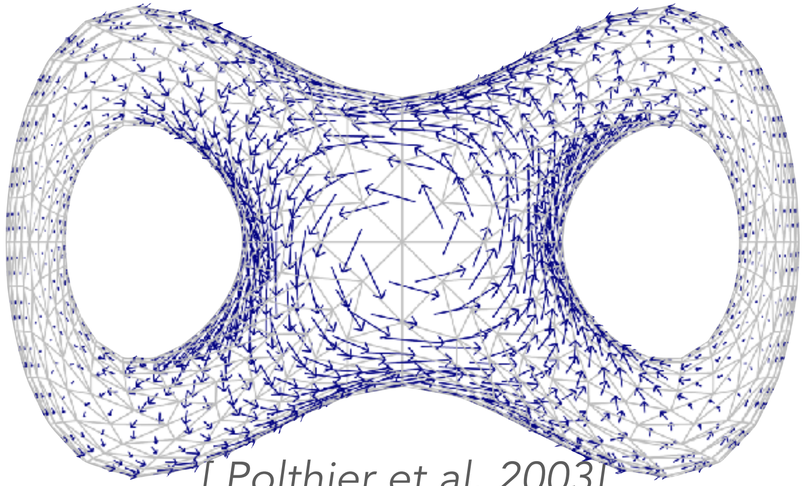
[Panozzo et al. 2012]



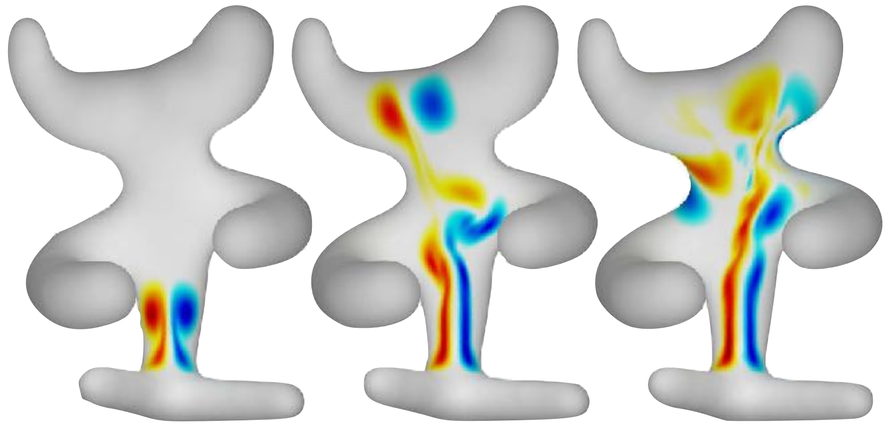
[Azencot et al. 2013]

Constraints - Differential - Divergence



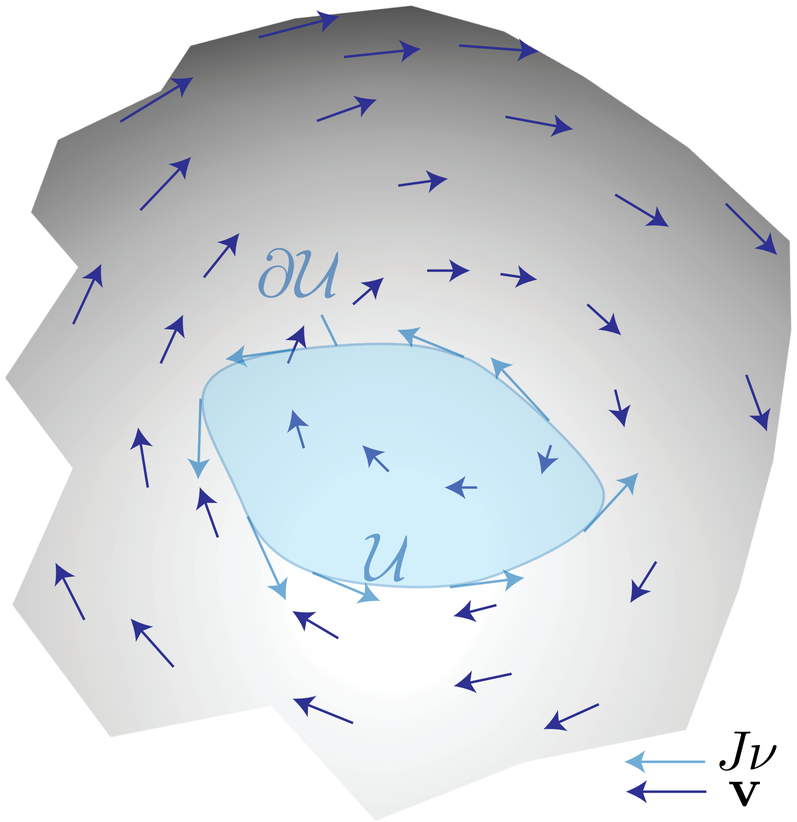


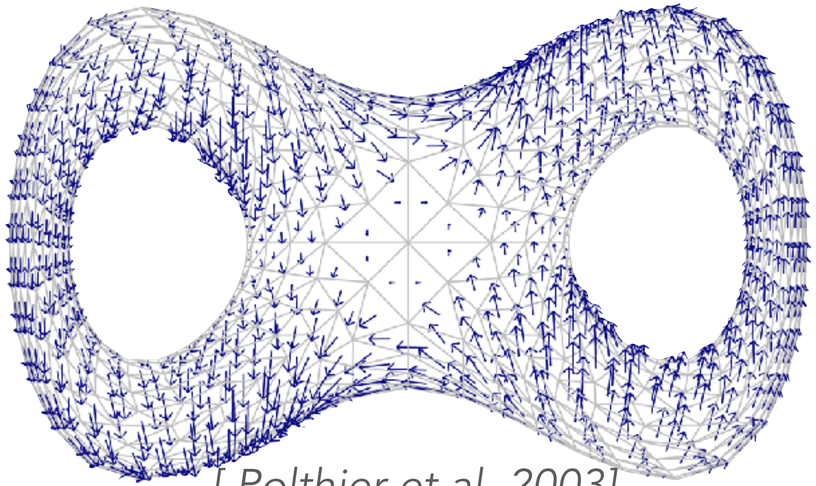
[Polthier et al. 2003]



[Azencot et al. 2014]

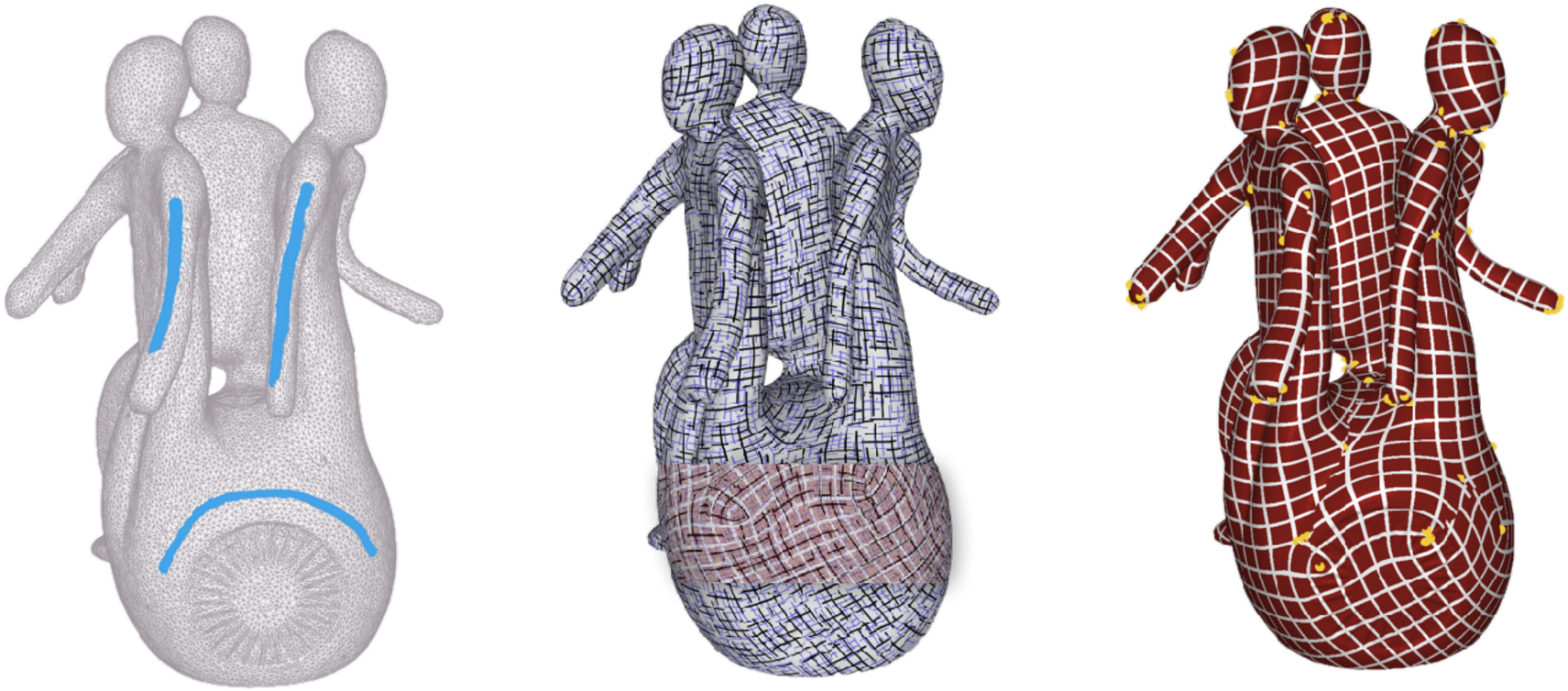
Constraints - Differential - curl





[Polthier et al. 2003]

Constraints - Differential - curl



[Diamanti et al. 2015]

Remarks

- We implemented a simple but powerful algorithm to design vector and polyvector fields
- You will need this algorithm for the next assignment. You can copy this code but you will have to implement hard constraints instead of soft ones
- A lot more information on this topic is available at:

  https://github.com/avaxman/DirectionalFieldSynthesis

- Takeaway: You now know how to process scalar and directional fields on surfaces!

References

- github.com/avaxman/DirectionalFieldSynthesis