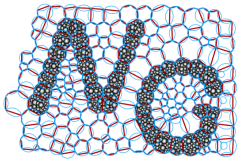
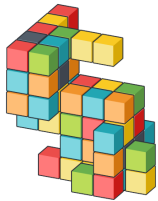


ccas.ru/gridgen



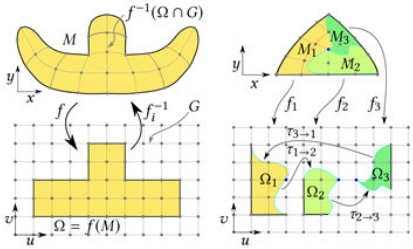
pixel.inria.fr

How to compute locally invertible maps

Vladimir Garanzha, Igor Kaporin, Liudmila Kudryavtseva,
François Protais, Nicolas Ray, Dmitry Sokolov

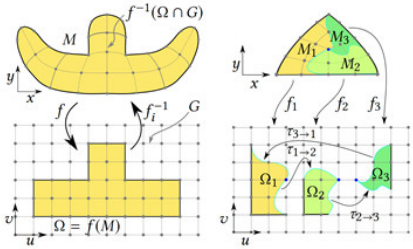
MIT Vision and Graphics Seminar, March 2, 2021

Maps are everywhere

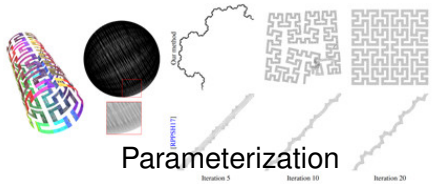


Hexahedral meshing

Maps are everywhere

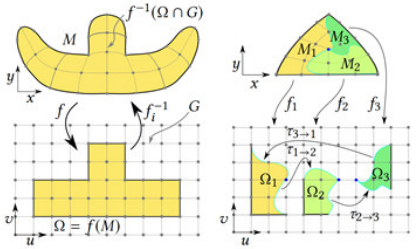


Hexahedral meshing

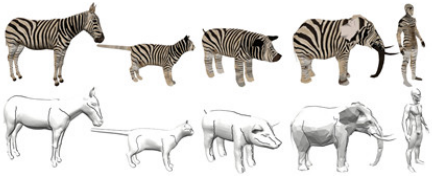


Parameterization

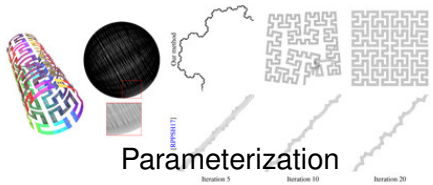
Maps are everywhere



Hexahedral meshing

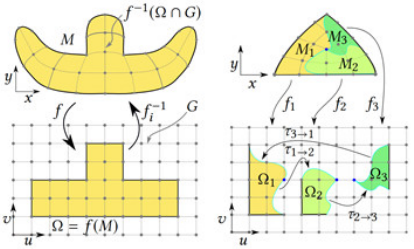


Functional maps

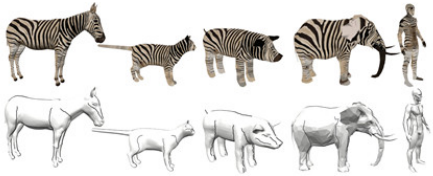


Parameterization

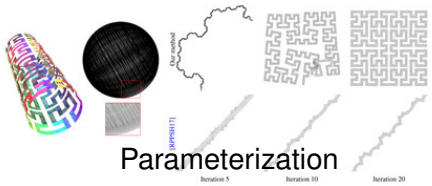
Maps are everywhere



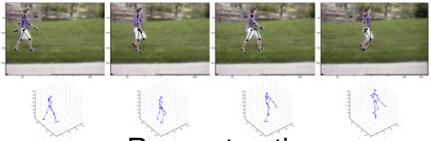
Hexahedral meshing



Functional maps

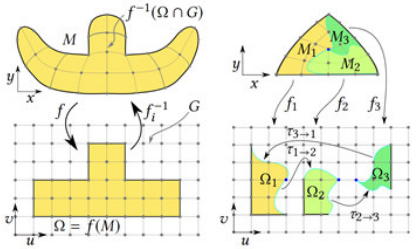


Parameterization

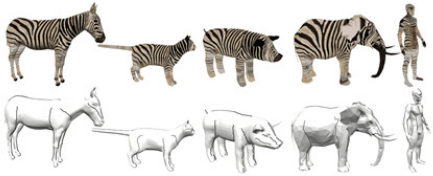


Reconstruction

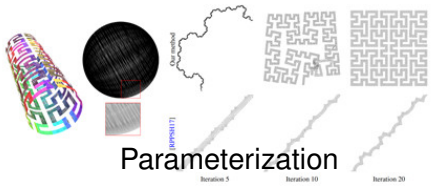
Maps are everywhere



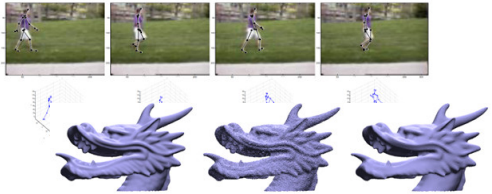
Hexahedral meshing



Functional maps

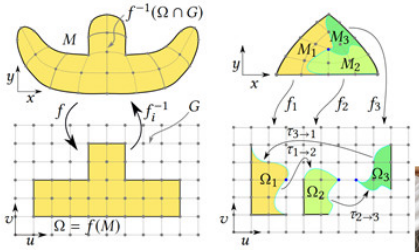


Parameterization

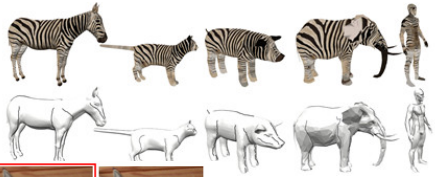


Mesh smoothing

Maps are everywhere

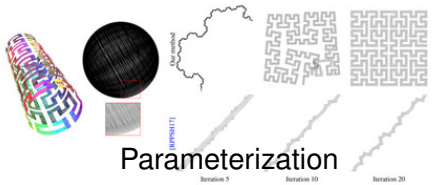


Hexahedral meshing

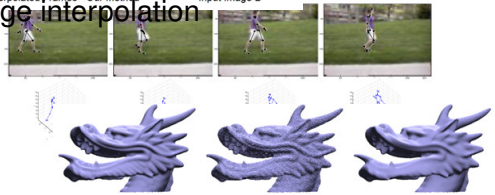


ial maps

Image interpolation

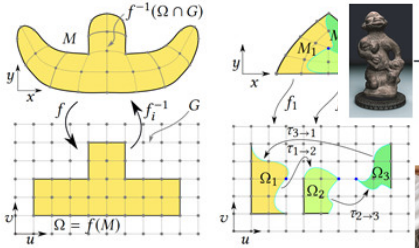


Parameterization



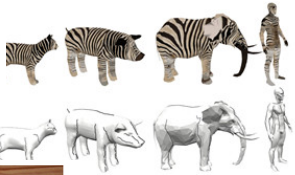
Mesh smoothing

Maps are everywhere



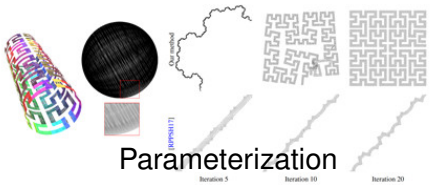
Hexahedral meshing

Texture transfer

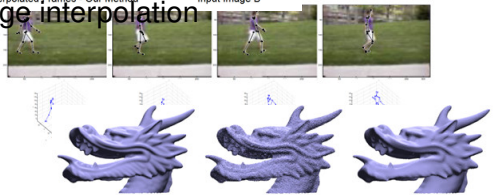


ial maps

Image interpolation

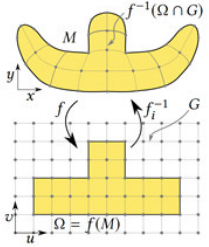


Parameterization

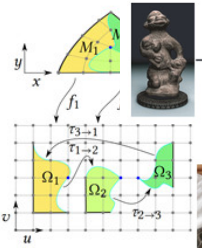


Mesh smoothing

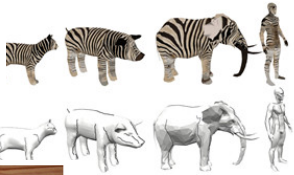
Maps are everywhere



Hexahedral meshing

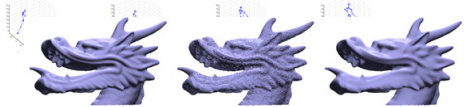


Texture transfer



ial maps

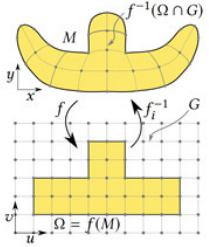
Image interpolation



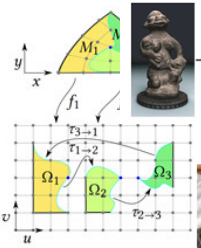
Mesh smoothing

Style transfer

Maps are everywhere



Hexahedral meshing



Texture transfer

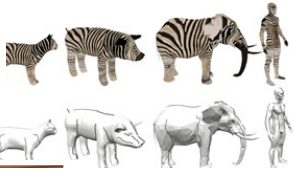
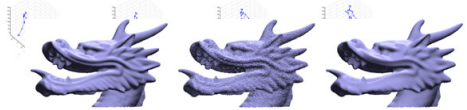


Image int

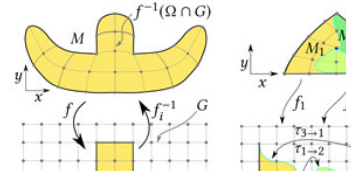
Pose reconstruction

Style transfer

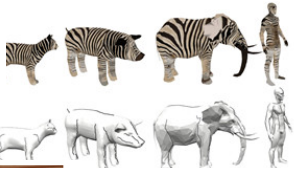


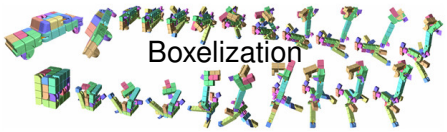
Mesh smoothing

Maps are everywhere



Texture transfer



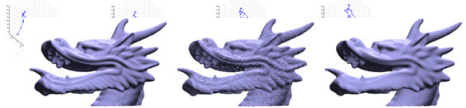


Boxelization

Image int

Pose reconstruction

Style transfer



Mesh smoothing

Maps are everywhere

Liu, H., Zhang, P., Chien, E., Solomon, J., & Bommes, D. (2018). Singularity-constrained octahedral fields for hexahedral meshing. ACM Trans. Graph., 37(4), 93-1.

Claici, S., Bessmeltsev, M., Schaefer, S., & Solomon, J. (2017, August). Isometry-aware preconditioning for mesh parameterization. In Computer Graphics Forum (Vol. 36, No. 5, pp. 37-47)

Gehre, A., Bronstein, M., Kobbelt, L., & Solomon, J. (2018, August). Interactive curve constrained functional maps. In Computer Graphics Forum (Vol. 37, No. 5, pp. 1-12).

Howe, N., Leventon, M. E., & Freeman, W. T. (2000). Bayesian reconstruction of 3d human motion from single-camera video. Advances in neural information processing systems, 12, 820.

Jones, T. R., Durand, F., & Desbrun, M. (2003). Non-iterative, feature-preserving mesh smoothing. In ACM SIGGRAPH 2003 Papers (pp. 943-949).

Mahajan, D., Huang, F. C., Matusik, W., Ramamoorthi, R., & Belhumeur, P. (2009). Moving gradients: a path-based method for plausible image interpolation. ACM Transactions on Graphics (TOG), 28(3), 1-11.

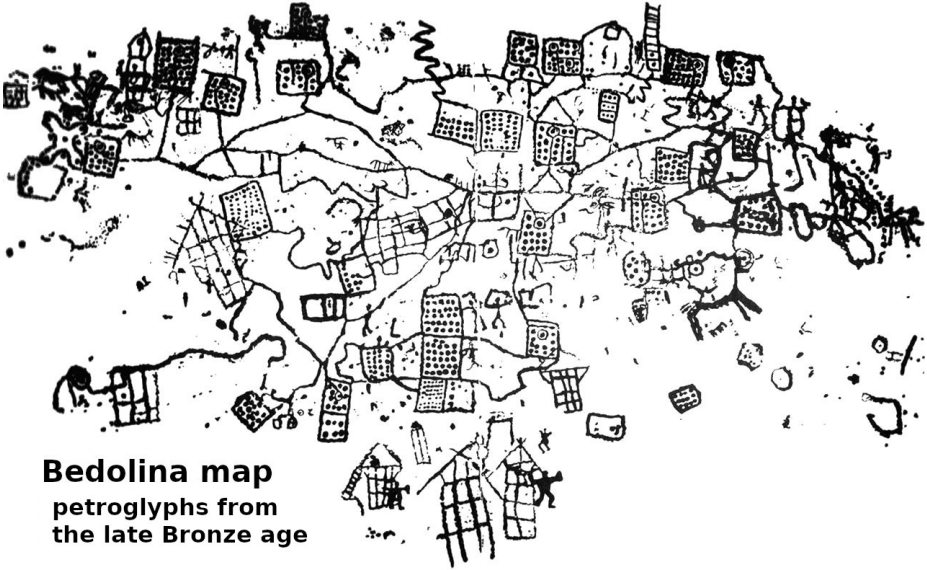
Mertens, T., Kautz, J., Chen, J., Bekaert, P., & Durand, F. (2006). Texture Transfer Using Geometry Correlation. Rendering Techniques, 273(10.2312), 273-284.

YiChang Shih, Sylvain Paris, Connelly Barnes, William T. Freeman, & Fredo Durand. 2014. Style transfer for headshot portraits. ACM Trans. Graph. 33, 4, Article 148 (July 2014),

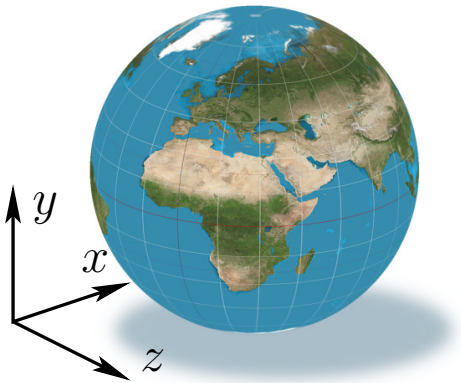
Zhao, M., Tian, Y., Zhao, H., Alsheikh, M. A., Li, T., Hristov, R., . . . & Torralba, A. (2018, August). RF-based 3D skeletons. In Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication (pp. 267-281).

Zhou, Y., Sueda, S., Matusik, W., & Shamir, A. (2014). Boxelization: Folding 3D objects into boxes. ACM Transactions on Graphics (TOG), 33(4), 1-8.

The motivation



And what if the Earth was not flat?



$$\vec{u}(\vec{x})$$

$$\vec{x}(\vec{u})$$

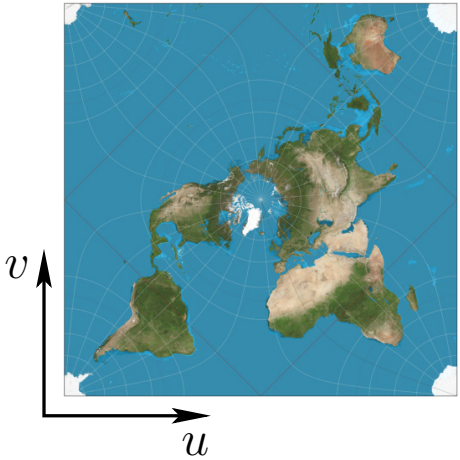
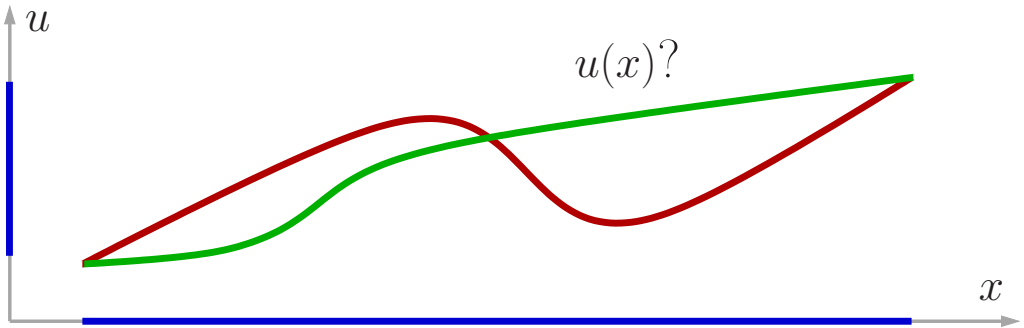


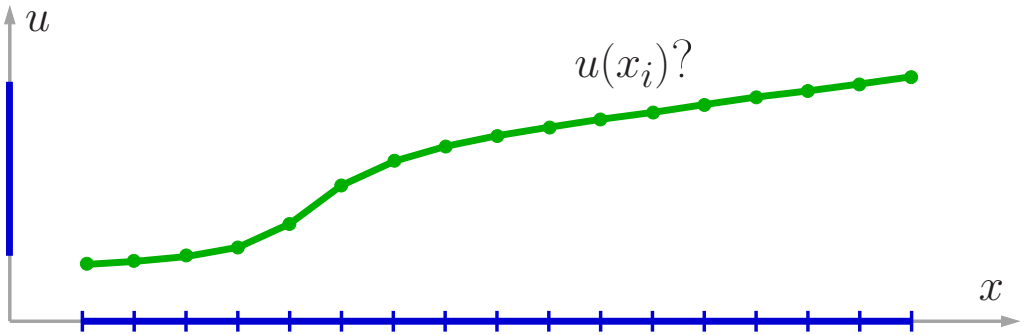
Table of Contents

How to compute a map between two segments?



N.B. We want the map to be invertible!

How to compute a map between two segments?

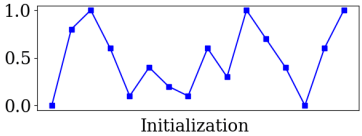


N.B. We want the map to be invertible!

represent $u(x)$ as a piecewise affine function.

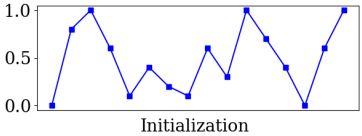
Compute the first map

```
1  x = [0, .8, 1, .6, .1, .4, .2, .1, .6, .3, 1, .7, .4, 0, .6, 1]
2
3  for _ in range(128):
4      for i in range(1, len(x)-1):
5          x[i] = ( x[i-1] + x[i+1] )/2.
```

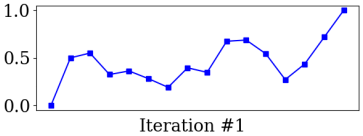


Compute the first map

```
x = [0, .8, 1, .6, .1, .4, .2, .1, .6, .3, 1, .7, .4, 0, .6, 1]

for _ in range(128):
    for i in range(1, len(x)-1):
        x[i] = ( x[i-1] + x[i+1] )/2.
```



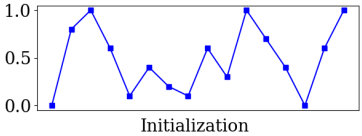
Initialization



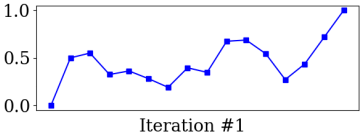
Iteration #1

Compute the first map

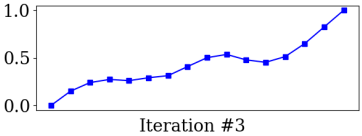
```
x = [0, .8, 1, .6, .1, .4, .2, .1, .6, .3, 1, .7, .4, 0, .6, 1]

for _ in range(128):
    for i in range(1, len(x)-1):
        x[i] = ( x[i-1] + x[i+1] )/2.
```



Initialization



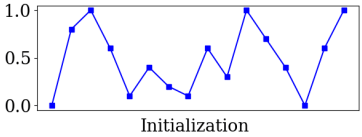
Iteration #1



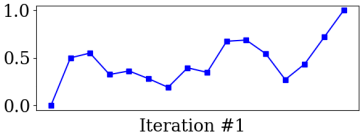
Iteration #3

Compute the first map

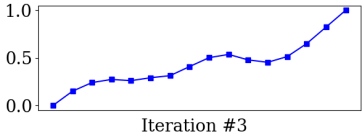
```
x = [0, .8, 1, .6, .1, .4, .2, .1, .6, .3, 1, .7, .4, 0, .6, 1]

for _ in range(128):
    for i in range(1, len(x)-1):
        x[i] = ( x[i-1] + x[i+1] )/2.
```



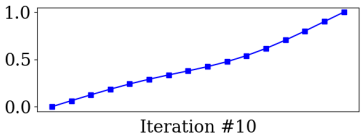
Initialization



Iteration #1



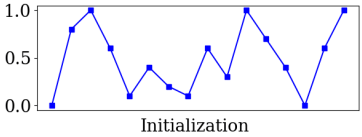
Iteration #3



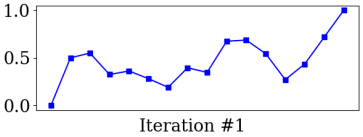
Iteration #10

Compute the first map

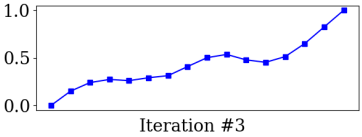
```
x = [0, .8, 1, .6, .1, .4, .2, .1, .6, .3, 1, .7, .4, 0, .6, 1]

for _ in range(128):
    for i in range(1, len(x)-1):
        x[i] = ( x[i-1] + x[i+1] )/2.
```



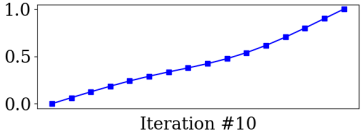
Initialization



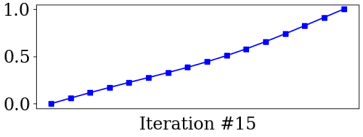
Iteration #1



Iteration #3



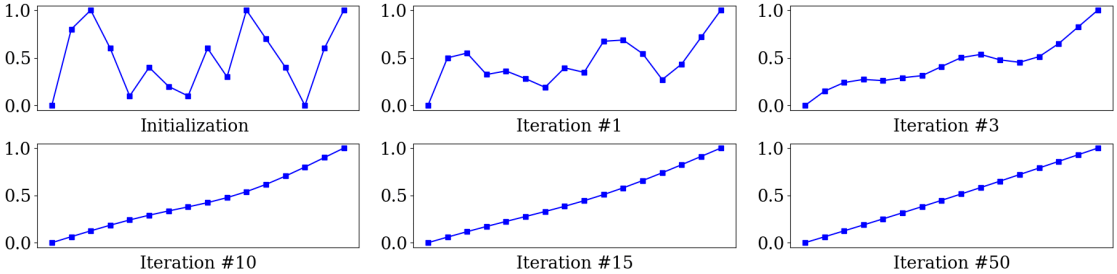
Iteration #10



Iteration #15

Compute the first map

```
x = [0, .8, 1, .6, .1, .4, .2, .1, .6, .3, 1, .7, .4, 0, .6, 1]

for _ in range(128):
    for i in range(1, len(x)-1):
        x[i] = ( x[i-1] + x[i+1] )/2.
```



The Gauß–Seidel iterative method

Given an ordinary system of linear equations:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\ \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n \end{cases}$$

The Gauß–Seidel iterative method

Given an ordinary system of linear equations:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\ \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n \end{cases}$$

Let us rewrite it as follows:

$$x_1 = \frac{1}{a_{11}}(b_1 - a_{12}x_2 - a_{13}x_3 - \cdots - a_{1n}x_n)$$

$$x_2 = \frac{1}{a_{22}}(b_2 - a_{21}x_1 - a_{23}x_3 - \cdots - a_{2n}x_n)$$

$$\vdots$$

$$x_n = \frac{1}{a_{nn}}(b_n - a_{n1}x_1 - a_{n2}x_2 - \cdots - a_{n,n-1}x_{n-1})$$

The Gauß–Seidel iterative method

Let us start with an arbitrary vector $\vec{x}^{(0)} = \left(x_1^{(0)}, x_2^{(0)}, \ldots, x_n^{(0)}\right)$,

The Gauß–Seidel iterative method

Let us start with an arbitrary vector $\vec{x}^{(0)} = \left(x_1^{(0)}, x_2^{(0)}, \ldots, x_n^{(0)}\right)$,

we can define $\vec{x}^{(1)}$ as follows:

$$
\begin{aligned}
x_1^{(1)} &= \frac{1}{a_{11}}(b_1 - a_{12}x_2^{(0)} - a_{13}x_3^{(0)} - \cdots - a_{1n}x_n^{(0)}) \\
x_2^{(1)} &= \frac{1}{a_{22}}(b_2 - a_{21}x_1^{(1)} - a_{23}x_3^{(0)} - \cdots - a_{2n}x_n^{(0)}) \\
&\vdots \\
x_n^{(1)} &= \frac{1}{a_{nn}}(b_n - a_{n1}x_1^{(1)} - a_{n2}x_2^{(1)} - \cdots - a_{n,n-1}x_{n-1}^{(1)})
\end{aligned}
$$

The Gauß–Seidel iterative method

Let us start with an arbitrary vector $\vec{x}^{(0)} = \left(x_1^{(0)}, x_2^{(0)}, \ldots, x_n^{(0)}\right)$,
we can define $\vec{x}^{(1)}$ as follows:

$$
\begin{aligned}
x_1^{(1)} &= \frac{1}{a_{11}}(b_1 - a_{12}x_2^{(0)} - a_{13}x_3^{(0)} - \cdots - a_{1n}x_n^{(0)}) \\
x_2^{(1)} &= \frac{1}{a_{22}}(b_2 - a_{21}x_1^{(1)} - a_{23}x_3^{(0)} - \cdots - a_{2n}x_n^{(0)}) \\
&\vdots \\
x_n^{(1)} &= \frac{1}{a_{nn}}(b_n - a_{n1}x_1^{(1)} - a_{n2}x_2^{(1)} - \cdots - a_{n,n-1}x_{n-1}^{(1)})
\end{aligned}
$$

Repeating the process k times, the solution can be approximated by the vector
$\vec{x}^{(k)} = \left(x_1^{(k)}, x_2^{(k)}, \ldots, x_n^{(k)}\right)$.

Back to the 1D mapping

```
x = [0, .8, 1, .6, .1, .4, .2, .1, .6, .3, 1, .7, .4, 0, .6, 1]

for _ in range(128):
    for i in range(1, len(x)-1):
        x[i] = ( x[i-1] + x[i+1] )/2.
```

Back to the 1D mapping

```
x = [0, .8, 1, .6, .1, .4, .2, .1, .6, .3, 1, .7, .4, 0, .6, 1]

for _ in range(128):
    for i in range(1, len(x)-1):
        x[i] = ( x[i-1] + x[i+1] )/2.
```

$$
\begin{cases}
x_0 & & & & & & & = 0 \\
-x_0 & +2x_1 & -x_2 & & & & & = 0 \\
 & -x_1 & +2x_2 & -x_3 & & & & = 0 \\
 & & \ddots & \ddots & \ddots & & & \vdots \\
 & & & -x_{12} & +2x_{13} & -x_{14} & & = 0 \\
 & & & & -x_{13} & +2x_{14} & -x_{15} & = 0 \\
 & & & & & & x_{15} & = 1
\end{cases}
$$

Back to the 1D mapping

```
x = [0, .8, 1, .6, .1, .4, .2, .1, .6, .3, 1, .7, .4, 0, .6, 1]

for _ in range(128):
    for i in range(1, len(x)-1):
        x[i] = ( x[i-1] + x[i+1] )/2.
```

$$
\begin{cases}
x_0 & & & & & & & = 0 \\
-x_0 & +2x_1 & -x_2 & & & & & = 0 \\
 & -x_1 & +2x_2 & -x_3 & & & & = 0 \\
 & & \ddots & \ddots & \ddots & & & \vdots \\
 & & & -x_{12} & +2x_{13} & -x_{14} & & = 0 \\
 & & & & -x_{13} & +2x_{14} & -x_{15} & = 0 \\
 & & & & & & x_{15} & = 1
\end{cases}
\qquad
\begin{cases}
\frac{d^2u}{dx^2} & = 0 \\
u(0) & = 0 \\
u(15) & = 1
\end{cases}
$$

The takeaway message

- A map can be represented by simple (e.g. affine) functions defined over a mesh of segments.
- These linear functions can be obtained by solving a system of equations.

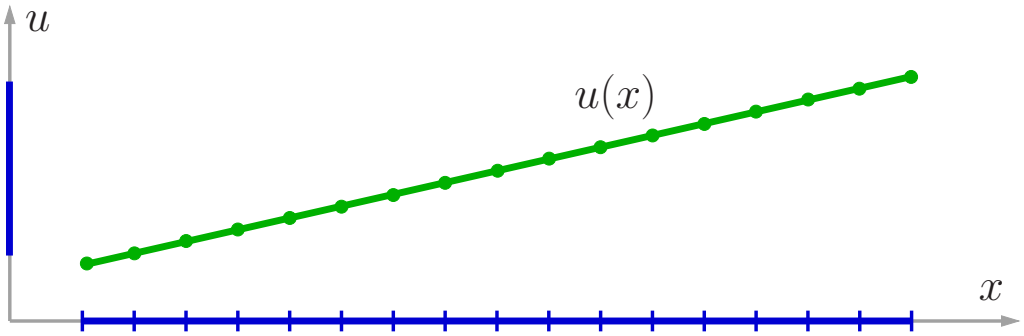
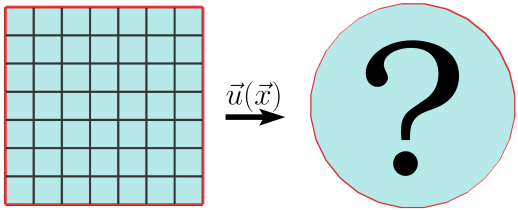
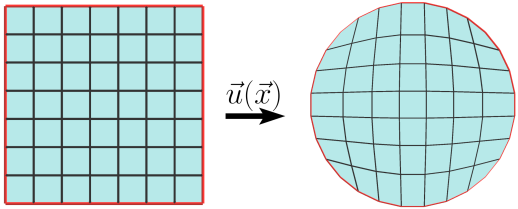


Table of Contents

Tutte embedding

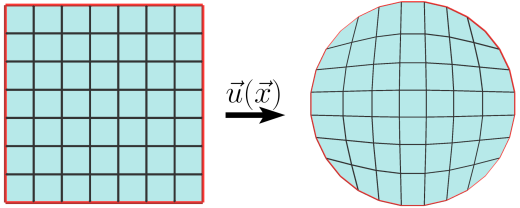


Tutte embedding




```
from mesh import Mesh
mesh,n = Mesh(),Mesh().size # a quad mesh with regular grid connectivity
u,v = mesh.x[:n*n], mesh.x[n*n:] # the grid is made of n*n verts

for _ in range(128): # Gauss-Seidel iterations solving for zero Laplacian
    for j in range(1, n-1):      # the boundary is fixed, so we iterate
        for i in range(1, n-1): # through interior vertices only
            idx = i+j*n
            u[idx] = (u[idx-1] + u[idx+1] + u[idx-n] + u[idx+n])/4.
            v[idx] = (v[idx-1] + v[idx+1] + v[idx-n] + v[idx+n])/4.
```

Tutte embedding



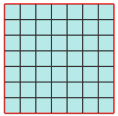
$\Delta\vec{u}(\vec{x}) = \vec{0}$,
Dirichlet boundary condition

Valid map for convex boundary [Tutte, 1963].

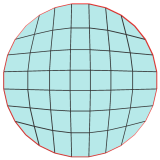
```
from mesh import Mesh
mesh,n = Mesh(),Mesh().size # a quad mesh with regular grid connectivity
u,v = mesh.x[:n*n], mesh.x[n*n:] # the grid is made of n*n verts

for _ in range(128): # Gauss-Seidel iterations solving for zero Laplacian
    for j in range(1, n-1):      # the boundary is fixed, so we iterate
        for i in range(1, n-1): # through interior vertices only
            idx = i+j*n
            u[idx] = (u[idx-1] + u[idx+1] + u[idx-n] + u[idx+n])/4.
            v[idx] = (v[idx-1] + v[idx+1] + v[idx-n] + v[idx+n])/4.
```

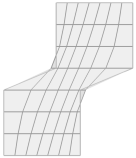
Testing time!

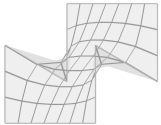
robustness ↓ difficulty ⟶



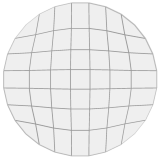
$\Delta \vec{u}(\vec{x}) = \vec{0}$

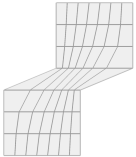


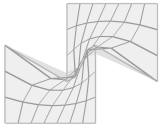




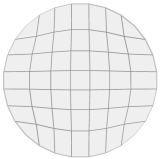
$\Delta \vec{x}(\vec{u}) = \vec{0}$

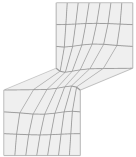


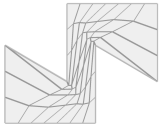




$\arg\min \int_{\Omega} f(J)\, dx$

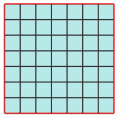




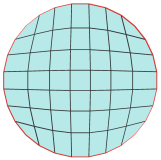


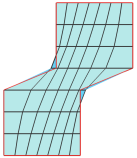
Testing time!

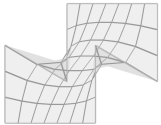
robustness ↓ difficulty ⟶



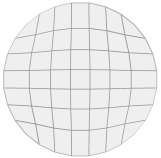
$\Delta \vec{u}(\vec{x}) = \vec{0}$

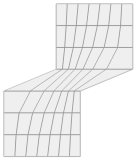


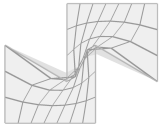




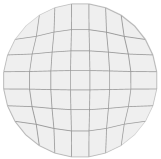
$\Delta \vec{x}(\vec{u}) = \vec{0}$

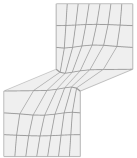


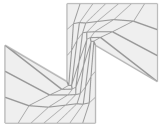




$\arg\min \int_{\Omega} f(J)\, dx$

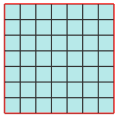




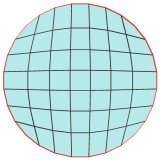


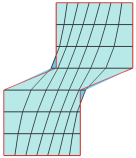
Testing time!

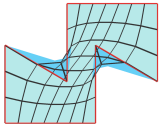
robustness ↓ difficulty ⟶



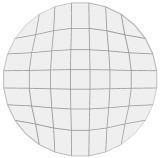
$\Delta \vec{u}(\vec{x}) = \vec{0}$

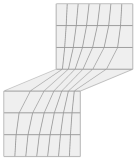


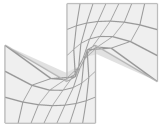




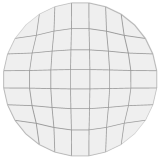
$\Delta \vec{x}(\vec{u}) = \vec{0}$

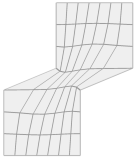


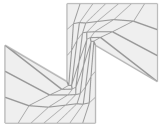




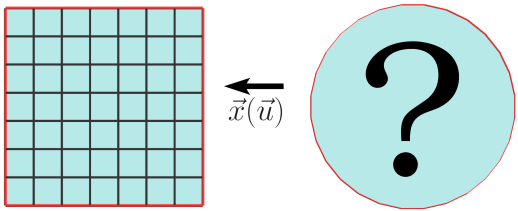
$\arg\min \int_{\Omega} f(J)\, dx$







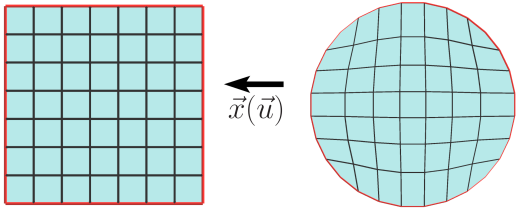
Winslow smoothing



[Crowley, 1962, Winslow, 1966]
Compute an **inverse** harmonic map:

$$\Delta \vec{x} = \vec{0}$$

Winslow smoothing



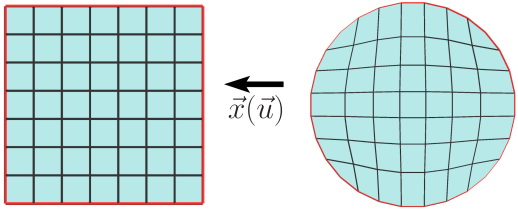
$$\begin{cases} g_{22}\frac{\partial^2 u}{\partial x^2} - 2g_{12}\frac{\partial^2 u}{\partial x \partial y} + g_{11}\frac{\partial^2 u}{\partial y^2} = 0 \\ g_{22}\frac{\partial^2 v}{\partial x^2} - 2g_{12}\frac{\partial^2 v}{\partial x \partial y} + g_{11}\frac{\partial^2 v}{\partial y^2} = 0, \end{cases}$$

[Crowley, 1962, Winslow, 1966]
Compute an **inverse** harmonic map:

$$\Delta \vec{x} = \vec{0}$$

$$\overbrace{\begin{pmatrix} g_{11} & g_{12} \\ g_{12} & g_{22} \end{pmatrix}}^{\text{metric tensor}} := \begin{pmatrix} \frac{\partial u}{\partial x} & \frac{\partial v}{\partial x} \\ \frac{\partial u}{\partial y} & \frac{\partial v}{\partial y} \end{pmatrix} \begin{pmatrix} \frac{\partial u}{\partial x} & \frac{\partial u}{\partial y} \\ \frac{\partial v}{\partial x} & \frac{\partial v}{\partial y} \end{pmatrix}$$

Winslow smoothing



[Crowley, 1962, Winslow, 1966]
Compute an **inverse** harmonic map:

$$\Delta \vec{x} = \vec{0}$$

$$\begin{cases} g_{22}\frac{\partial^2 u}{\partial x^2} - 2g_{12}\frac{\partial^2 u}{\partial x \partial y} + g_{11}\frac{\partial^2 u}{\partial y^2} = 0 \\ g_{22}\frac{\partial^2 v}{\partial x^2} - 2g_{12}\frac{\partial^2 v}{\partial x \partial y} + g_{11}\frac{\partial^2 v}{\partial y^2} = 0, \end{cases}$$

$$\overbrace{\begin{pmatrix} g_{11} & g_{12} \\ g_{12} & g_{22} \end{pmatrix}}^{\text{metric tensor}} := \begin{pmatrix} \frac{\partial u}{\partial x} & \frac{\partial v}{\partial x} \\ \frac{\partial u}{\partial y} & \frac{\partial v}{\partial y} \end{pmatrix} \begin{pmatrix} \frac{\partial u}{\partial x} & \frac{\partial u}{\partial y} \\ \frac{\partial v}{\partial x} & \frac{\partial v}{\partial y} \end{pmatrix}$$

Use finite differences, e.g.:

$$\left(\frac{\partial^2 u}{\partial x^2}\right)_{i,j} = \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2}$$

Quasi-linear system to solve:

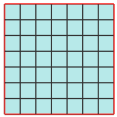
$$a_{i,j}u_{i-1,j} + b_{i,j}u_{i,j} + c_{i,j}u_{i+1,j} + d_{i,j} = 0$$
$$a_{i,j}v_{i-1,j} + b_{i,j}v_{i,j} + c_{i,j}v_{i+1,j} + e_{i,j} = 0$$

Numerical solution of Winslow equations

```
from mesh import Mesh
mesh,n = Mesh(),Mesh().size # a quad mesh with regular grid connectivity
u,v = mesh.x[:n*n], mesh.x[n*n:] # the grid is made of n*n verts
def g11(i,j): # metric tensor estimation via finite differences
    return (u[i+1+j*n]-u[i-1+j*n])**2/4. + (v[i+1+j*n]-v[i-1+j*n])**2/4.
def g22(i,j):
    return (u[i+j*n+n]-u[i+j*n-n])**2/4. + (v[i+j*n+n]-v[i+j*n-n])**2/4.
def g12(i,j):
    return (u[i+1+j*n]-u[i-1+j*n])*(u[i+j*n+n]-u[i+j*n-n])/4. + \
           (v[i+1+j*n]-v[i-1+j*n])*(v[i+j*n+n]-v[i+j*n-n])/4.
for _ in range(128): # Gauss-Seidel iterations, zero Laplacian of the inverse map
     for j in range(1, n-1):      # the boundary is fixed, so we iterate
         for i in range(1, n-1): # through interior vertices only
             a,b,c = g22(i,j), 2*g22(i,j)+2*g11(i,j), g22(i,j)
             d = g11(i,j)*(u[i+j*n+n] + u[i+j*n-n]) - 2*g12(i,j)* \
                 (u[i+1+j*n+n] + u[i-1+j*n-n] - u[i-1+j*n+n] - u[i+1+j*n-n])/4.
             e = g11(i,j)*(v[i+j*n+n] + v[i+j*n-n]) - 2*g12(i,j)* \
                 (v[i+1+j*n+n] + v[i-1+j*n-n] - v[i-1+j*n+n] - v[i+1+j*n-n])/4.
             u[i+j*n] = (d + a*u[i-1+j*n] + c*u[i+1+j*n])/b # actual Gauss-Seidel
             v[i+j*n] = (e + a*v[i-1+j*n] + c*v[i+1+j*n])/b # linear system update
```

Testing time!

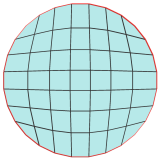
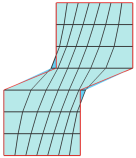
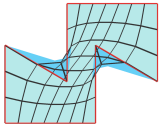
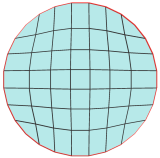
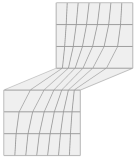
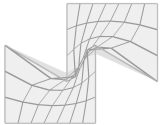
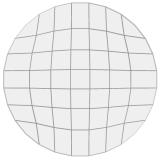
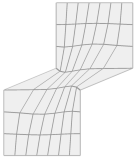
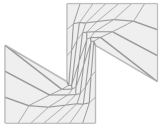
robustness ↓ difficulty ⟶



$\Delta \vec{u}(\vec{x}) = \vec{0}$

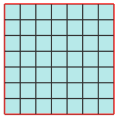
$\Delta \vec{x}(\vec{u}) = \vec{0}$

$\arg\min \int_{\Omega} f(J)\, dx$

Testing time!

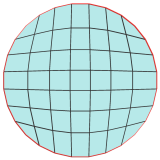
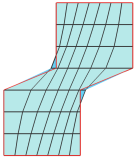
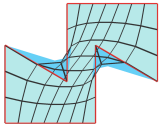
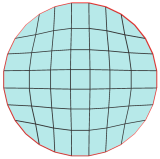
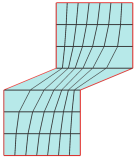
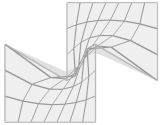
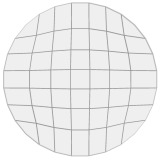
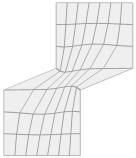
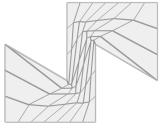
robustness ↓ difficulty ⟶



$\Delta \vec{u}(\vec{x}) = \vec{0}$

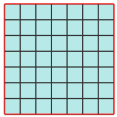
$\Delta \vec{x}(\vec{u}) = \vec{0}$

$\arg\min \int_{\Omega} f(J)\, dx$

Testing time!

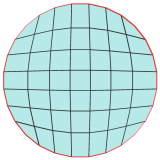
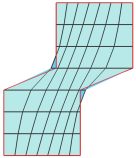
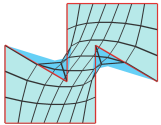
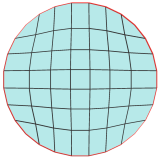
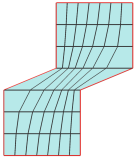
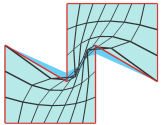
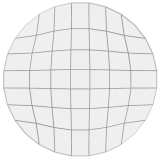
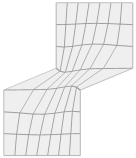
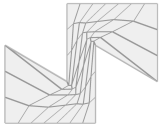
robustness ↓ difficulty ⟶



$\Delta \vec{u}(\vec{x}) = \vec{0}$

$\Delta \vec{x}(\vec{u}) = \vec{0}$

$\arg\min \int_{\Omega} f(J)\, dx$

Variational formulation: problem statement

Solving Laplace's equation $\Delta \vec{u}(\vec{x}) = \vec{0}$ is equivalent* to Dirichlet energy minimization $\arg\min_{\vec{u}} \int_{\Omega} \operatorname{tr} J^{\top} J \, dx$, where J is the Jacobian matrix of the mapping $\vec{u}(\vec{x})$.

Variational formulation: problem statement

Solving Laplace's equation $\Delta \vec{u}(\vec{x}) = \vec{0}$ is equivalent* to Dirichlet energy minimization $\arg\min_{\vec{u}} \int_\Omega \operatorname{tr} J^\top J \, dx$, where J is the Jacobian matrix of the mapping $\vec{u}(\vec{x})$.

[Brackbill and Saltzman, 1982] **Let us try the same with the inverse harmonic map!**

$\Delta \vec{x}(\vec{u}) = \vec{0}$ is equivalent to $\arg\min_{\vec{u}} \int_\Omega \frac{\operatorname{tr} J^\top J}{\det J} \, dx$ provided that $\vec{u}(\vec{x})$ is a diffemorphism.

Variational formulation: problem statement

Solving Laplace's equation $\Delta \vec{u}(\vec{x}) = \vec{0}$ is equivalent* to Dirichlet energy minimization $\arg\min_{\vec{u}} \int_\Omega \operatorname{tr} J^\top J \, dx$, where J is the Jacobian matrix of the mapping $\vec{u}(\vec{x})$.

[Brackbill and Saltzman, 1982] **Let us try the same with the inverse harmonic map!**

$\Delta \vec{x}(\vec{u}) = \vec{0}$ is equivalent to $\arg\min_{\vec{u}} \int_\Omega \frac{\operatorname{tr} J^\top J}{\det J} \, dx$ provided that $\vec{u}(\vec{x})$ is a diffemorphism.

[Ivanenko, 1988] **Reformulate as a barrier problem for FEM approximation:**

$$\arg\min_{\vec{u}} \int_\Omega f(J) \, dx, \qquad f(J) := \begin{cases} \frac{\operatorname{tr} J^\top J}{\det J}, & \det J > 0 \\ +\infty, & \det J \le 0 \end{cases}$$

Variational formulation: problem statement

Solving Laplace's equation $\Delta\vec{u}(\vec{x}) = \vec{0}$ is equivalent* to Dirichlet energy minimization $\arg\min_{\vec{u}} \int_\Omega \operatorname{tr} J^\top J \, dx$, where J is the Jacobian matrix of the mapping $\vec{u}(\vec{x})$.

[Brackbill and Saltzman, 1982] **Let us try the same with the inverse harmonic map!**

$\Delta\vec{x}(\vec{u}) = \vec{0}$ is equivalent to $\arg\min_{\vec{u}} \int_\Omega \frac{\operatorname{tr} J^\top J}{\det J} \, dx$ provided that $\vec{u}(\vec{x})$ is a diffemorphism.

[Ivanenko, 1988] **Reformulate as a barrier problem for FEM approximation:**

$$\arg\min_{\vec{u}} \int_\Omega f(J)\, dx, \qquad f(J) := \begin{cases} \frac{\operatorname{tr} J^\top J}{\det J}, & \det J > 0 \\ +\infty, & \det J \le 0 \end{cases}$$

[Ball, 1981] **Polyconvex hyperelastic energy to minimize, very solid theoretical results on invertibility in 2D and 3D!**

* fine print statement asking for the first-born child

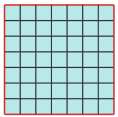
Testing time!

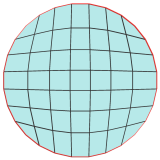
robustness ↓ difficulty ⟶

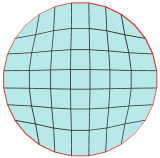
$\Delta \vec{u}(\vec{x}) = \vec{0}$

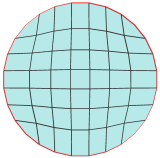
$\Delta \vec{x}(\vec{u}) = \vec{0}$

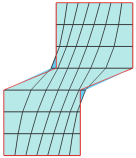
$\arg\min \int_{\Omega} f(J)\, dx$

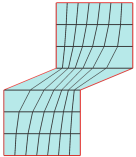


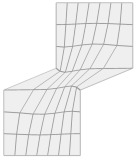


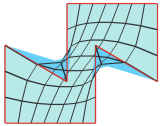


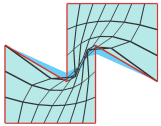


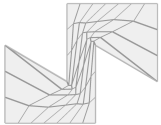












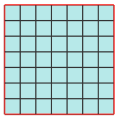
Testing time!

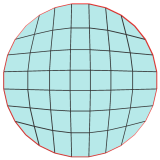
robustness ↓ difficulty ⟶

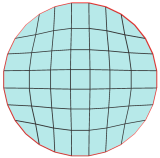
$\Delta \vec{u}(\vec{x}) = \vec{0}$

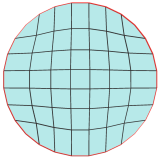
$\Delta \vec{x}(\vec{u}) = \vec{0}$

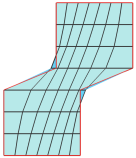
$\arg\min \int_{\Omega} f(J)\, dx$

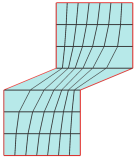


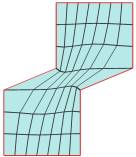


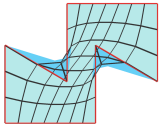


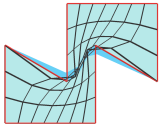


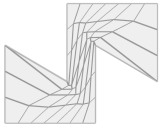












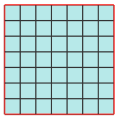
Testing time!

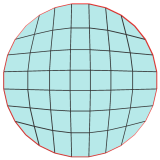
robustness ↓ difficulty ⟶

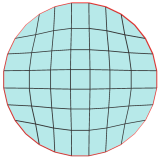
$\Delta \vec{u}(\vec{x}) = \vec{0}$

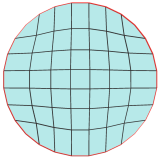
$\Delta \vec{x}(\vec{u}) = \vec{0}$

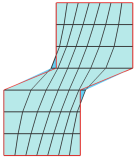
$\arg\min \int_{\Omega} f(J)\, dx$

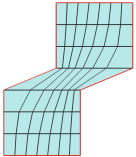


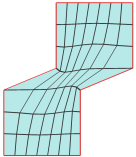


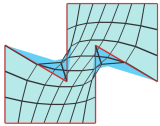


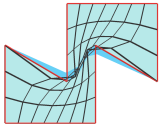


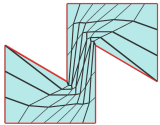












Regularization of the barrier: the main idea

"Winslow functional":

$$\arg\min_{\vec{u}} \int_{\Omega} f(J)\, dx, \qquad f(J) := \begin{cases} \frac{\operatorname{tr} J^{\top} J}{\det J}, & \det J > 0 \\ +\infty, & \det J \leq 0 \end{cases}$$

Regularization of the barrier: the main idea

"Winslow functional":

$$\arg\min_{\vec{u}} \int_{\Omega} f(J)\, dx, \qquad f(J) := \begin{cases} \frac{\operatorname{tr} J^\top J}{\det J}, & \det J > 0 \\ +\infty, & \det J \le 0 \end{cases} = \frac{\operatorname{tr} J^\top J}{\max\{0, \det J\}}$$

Regularization of the barrier: the main idea

"Winslow functional":

$$\arg\min_{\vec{u}} \int_{\Omega} f(J)\, dx, \qquad f(J) := \begin{cases} \frac{\operatorname{tr} J^\top J}{\det J}, & \det J > 0 \\ +\infty, & \det J \leq 0 \end{cases} = \frac{\operatorname{tr} J^\top J}{\max\{0, \det J\}}$$

Theoretical guarantees for FEM
(no guarantees for continuous case),
however no practical way to find the minimum.

Regularization of the barrier: the main idea

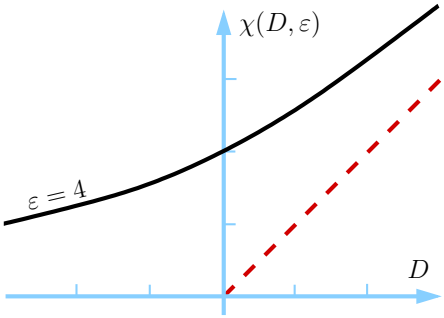
"Winslow functional":

$$\arg\min_{\vec{u}} \int_{\Omega} f(J)\,dx, \qquad f(J) := \begin{cases} \frac{\operatorname{tr} J^\top J}{\det J}, & \det J > 0 \\ +\infty, & \det J \le 0 \end{cases} = \frac{\operatorname{tr} J^\top J}{\max\{0, \det J\}}$$

Theoretical guarantees for FEM
(no guarantees for continuous case),
however no practical way to find the minimum.

Define a regularization function:

$$\chi(D, \varepsilon) := \frac{D + \sqrt{\varepsilon^2 + D^2}}{2}$$



Regularization of the barrier: the main idea

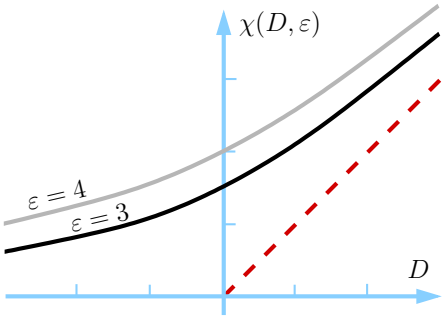
"Winslow functional":

$$\arg\min_{\vec{u}} \int_{\Omega} f(J)\,dx, \qquad f(J) := \begin{cases} \frac{\operatorname{tr} J^\top J}{\det J}, & \det J > 0 \\ +\infty, & \det J \le 0 \end{cases} = \frac{\operatorname{tr} J^\top J}{\max\{0, \det J\}}$$

Theoretical guarantees for FEM
(no guarantees for continuous case),
however no practical way to find the minimum.

Define a regularization function:

$$\chi(D, \varepsilon) := \frac{D + \sqrt{\varepsilon^2 + D^2}}{2}$$



Regularization of the barrier: the main idea

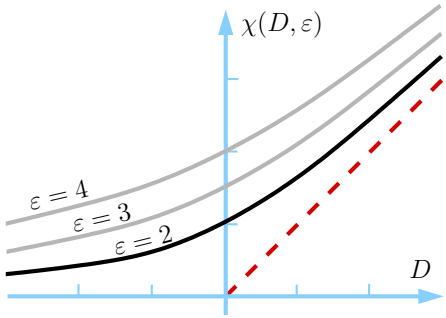
"Winslow functional":

$$\arg\min_{\vec{u}} \int_{\Omega} f(J)\, dx, \qquad f(J) := \begin{cases} \frac{\operatorname{tr} J^\top J}{\det J}, & \det J > 0 \\ +\infty, & \det J \le 0 \end{cases} = \frac{\operatorname{tr} J^\top J}{\max\{0, \det J\}}$$

Theoretical guarantees for FEM
(no guarantees for continuous case),
however no practical way to find the minimum.

Define a regularization function:

$$\chi(D, \varepsilon) := \frac{D + \sqrt{\varepsilon^2 + D^2}}{2}$$



Regularization of the barrier: the main idea

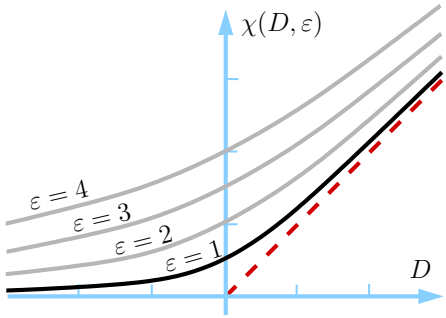
"Winslow functional":

$$\arg\min_{\vec{u}} \int_{\Omega} f(J)\, dx, \qquad f(J) := \begin{cases} \frac{\operatorname{tr} J^\top J}{\det J}, & \det J > 0 \\ +\infty, & \det J \le 0 \end{cases} = \frac{\operatorname{tr} J^\top J}{\max\{0, \det J\}}$$

Theoretical guarantees for FEM
(no guarantees for continuous case),
however no practical way to find the minimum.

Define a regularization function:

$$\chi(D, \varepsilon) := \frac{D + \sqrt{\varepsilon^2 + D^2}}{2}$$



Regularization of the barrier: the main idea

"Winslow functional":

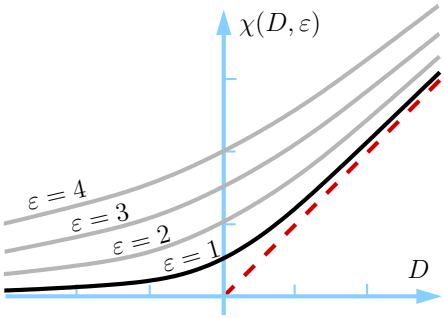
$$\arg\min_{\vec{u}} \int_\Omega f(J)\,dx, \qquad f(J) := \begin{cases} \frac{\operatorname{tr} J^\top J}{\det J}, & \det J > 0 \\ +\infty, & \det J \le 0 \end{cases} = \frac{\operatorname{tr} J^\top J}{\max\{0, \det J\}}$$

Theoretical guarantees for FEM
(no guarantees for continuous case),
however no practical way to find the minimum.

Define a regularization function:

$$\chi(D, \varepsilon) := \frac{D + \sqrt{\varepsilon^2 + D^2}}{2}$$

Solve $\lim_{\varepsilon \to 0^+} \arg\min_{\vec{u}} \int_\Omega \frac{\operatorname{tr} J^\top J}{\chi(\det J, \varepsilon)}\,dx$



A way to recover from tangled initializations!

Regularization of the barrier: resolution scheme

In practice, the map $\vec{u}(\vec{x})$ can be approximated with P1/Q1 elements, so our variables are the coordinates of the vertices $U := \left(\vec{u}_1^\top \dots \vec{u}_{\#V}^\top\right)^\top$.

Regularization of the barrier: resolution scheme

In practice, the map $\vec{u}(\vec{x})$ can be approximated with P1/Q1 elements, so our variables are the coordinates of the vertices $U := \left(\vec{u}_1^\top \dots \vec{u}_{\#V}^\top\right)^\top$.

Discretize the problem: $$\lim_{\varepsilon \to 0^+} \arg\min_U \sum_{t=1}^{\#T} \frac{\operatorname{tr} J_t^\top J_t}{\chi(\det J_t, \varepsilon)} \operatorname{vol}(T_t)$$

Regularization of the barrier: resolution scheme

In practice, the map $\vec{u}(\vec{x})$ can be approximated with P1/Q1 elements, so our variables are the coordinates of the vertices $U := \left(\vec{u}_1^\top \dots \vec{u}_{\#V}^\top\right)^\top$.

Discretize the problem: $$\lim_{\varepsilon \to 0^+} \arg\min_U \sum_{t=1}^{\#T} \frac{\operatorname{tr} J_t^\top J_t}{\chi(\det J_t, \varepsilon)} \operatorname{vol}(T_t)$$

✓ Start from initial guess U^0, build a sequence $U^{k+1} := U^k + \delta^k$.

Regularization of the barrier: resolution scheme

In practice, the map $\vec{u}(\vec{x})$ can be approximated with P1/Q1 elements, so our variables are the coordinates of the vertices $U := \left(\vec{u}_1^\top \dots \vec{u}_{\#V}^\top\right)^\top$.

Discretize the problem: $$\lim_{\varepsilon \to 0^+} \arg\min_U \sum_{t=1}^{\#T} \frac{\operatorname{tr} J_t^\top J_t}{\chi(\det J_t, \varepsilon)} \operatorname{vol}(T_t)$$

- ✓ Start from initial guess U^0, build a sequence $U^{k+1} := U^k + \delta^k$.
- ✓ For each iteration carefully choose ε^k as a function of U^k, U^{k-1} and ε^{k-1}.

Regularization of the barrier: resolution scheme

In practice, the map $\vec{u}(\vec{x})$ can be approximated with P1/Q1 elements, so our variables are the coordinates of the vertices $U := \left(\vec{u}_1^\top \dots \vec{u}_{\#V}^\top\right)^\top$.

Discretize the problem: $$\lim_{\varepsilon \to 0^+} \arg\min_U \sum_{t=1}^{\#T} \frac{\operatorname{tr} J_t^\top J_t}{\chi(\det J_t, \varepsilon)} \operatorname{vol}(T_t)$$

- ✓ Start from initial guess U^0, build a sequence $U^{k+1} := U^k + \delta^k$.
- ✓ For each iteration carefully choose ε^k as a function of U^k, U^{k-1} and ε^{k-1}.
- ✓ For each ε^k solve a minimization problem to obtain δ^k.
  $\rightarrow$ This can be done via Newton or a quasi-Newton method.

Regularization of the barrier: resolution scheme

In practice, the map $\vec{u}(\vec{x})$ can be approximated with P1/Q1 elements, so our variables are the coordinates of the vertices $U := \left(\vec{u}_1^\top \dots \vec{u}_{\#V}^\top\right)^\top$.

Discretize the problem: $$\lim_{\varepsilon \to 0^+} \arg\min_U \sum_{t=1}^{\#T} \frac{\operatorname{tr} J_t^\top J_t}{\chi(\det J_t, \varepsilon)} \operatorname{vol}(T_t)$$

- ✓ Start from initial guess U^0, build a sequence $U^{k+1} := U^k + \delta^k$.
- ✓ For each iteration carefully choose ε^k as a function of U^k, U^{k-1} and ε^{k-1}.
- ✓ For each ε^k solve a minimization problem to obtain δ^k.
  $\rightarrow$ This can be done via Newton or a quasi-Newton method.

Under certain assumptions* on the minimization toolbox chosen, **untangling is guaranteed in a finite number of iterations!**

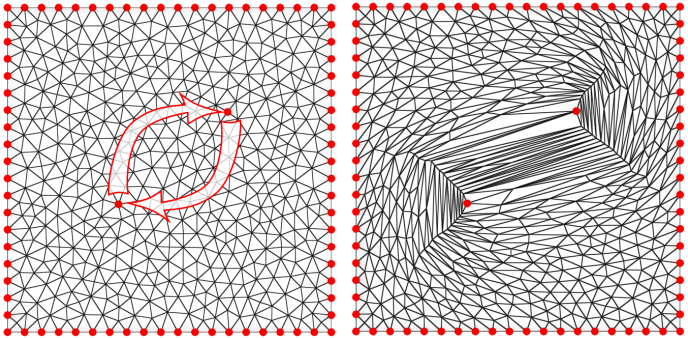
In practice less than 100 Newton iterations with a **positive definite** matrix.

* Nobody reads the fine print.

Numerical solution of the penalty method

```
from mesh import Mesh
import numpy as np
from scipy.optimize import fmin_l_bfgs_b
mesh,n = Mesh(),Mesh().nverts # generate a test quad mesh
Q = [ np.matrix('-1,-1;1,0;0,0;0,1'), np.matrix('-1,0;1,-1;0,1;0,0'),  # quadratures for
      np.matrix('0,0;0,-1;1,1;-1,0'), np.matrix('0,-1;0,0;1,0;-1,1') ] # every quad corner
def jacobian(U, qc, quad): # evaluate the Jacobian matrix at the given quadrature point
    return np.matrix([[U[quad[0]  ], U[quad[1]  ], U[quad[2]  ], U[quad[3]  ]],
                      [U[quad[0]+n], U[quad[1]+n], U[quad[2]+n], U[quad[3]+n]]]) * Q[qc]
for iter in range(10): # outer L-BFGS loop
    mindet = min( [ np.linalg.det( jacobian(mesh.x, qc, quad) ) for quad in mesh.quads for qc in range(4) ] )
    eps = np.sqrt(1e-6**2 + .04*min(mindet, 0)**2) # the regularization parameter e
    def energy(U): # compute the energy and its gradient for the map u
        F,G = 0, np.zeros(2*n)
        for quad in mesh.quads: # sum over all quads
            for qc in range(4): # evaluate the Jacobian matrix for every quad corner
                J = jacobian(U, qc, quad)
                det = np.linalg.det(J)
                chi  = det/2 + np.sqrt(eps**2 + det**2)/2    # the penalty function
                chip = .5 + det/(2*np.sqrt(eps**2 + det**2)) # its derivative
                f = np.trace(np.transpose(J)*J)/chi # quad corner shape quality
                F += f
                dfdj = (2*J - np.matrix([[J[1,1],-J[1,0]],[-J[0,1],J[0,0]]])*f*chip)/chi
                dfdu = Q[qc] * np.transpose(dfdj) # chain rule for the actual variables
                for i,v in enumerate(quad):
                    if (mesh.boundary[v]): continue # the boundary verts are locked
                    G[v  ] += dfdu[i,0]
                    G[v+n] += dfdu[i,1]
        return F,G
    mesh.x = fmin_l_bfgs_b(energy, mesh.x, factr=1e12)[0] # inner L-BFGS loop
```

Invertible mapping sanity check



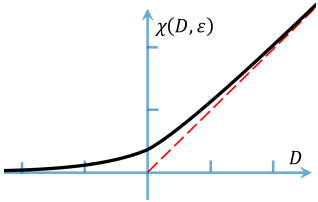
Sanity check passed.

Table of Contents

Penalty method for mesh untangling

Define a regularization function for the denominator:

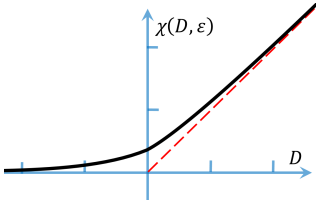
$$\chi(D, \varepsilon) := \frac{D + \sqrt{\varepsilon^2 + D^2}}{2}$$



Penalty method for mesh untangling

Define a regularization function for the denominator:

$$\chi(D, \varepsilon) := \frac{D + \sqrt{\varepsilon^2 + D^2}}{2}$$



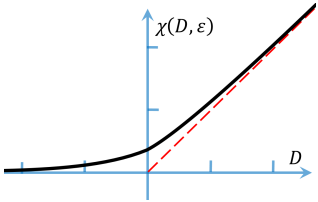
Define distortion measures:

$$f(J) := \frac{\operatorname{tr} J^\top J}{(\det J)^{\frac{2}{d}}} \to f_\varepsilon(J) := \frac{\operatorname{tr} J^\top J}{(\chi(\det J, \varepsilon))^{\frac{2}{d}}} \qquad g(J) := \det J + \frac{1}{\det J} \to g_\varepsilon(J) := \frac{\det^2 J + 1}{\chi(\det J, \varepsilon)}$$

Penalty method for mesh untangling

Define a regularization function for the denominator:

$$\chi(D, \varepsilon) := \frac{D + \sqrt{\varepsilon^2 + D^2}}{2}$$



Define distortion measures:

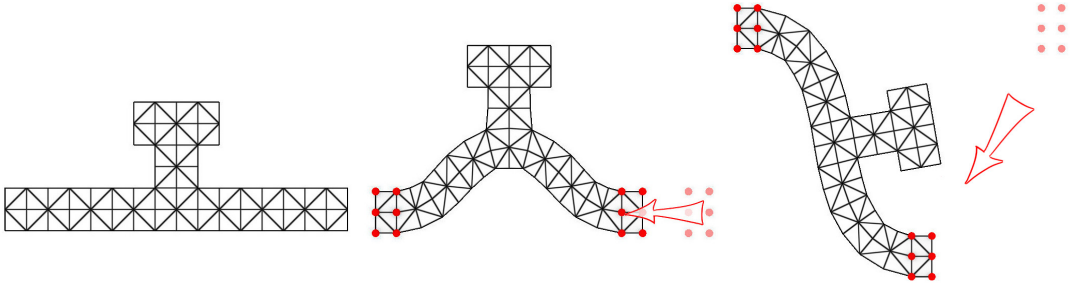
$$f(J) := \frac{\operatorname{tr} J^\top J}{(\det J)^{\frac{2}{d}}} \to f_\varepsilon(J) := \frac{\operatorname{tr} J^\top J}{(\chi(\det J, \varepsilon))^{\frac{2}{d}}} \qquad g(J) := \det J + \frac{1}{\det J} \to g_\varepsilon(J) := \frac{\det^2 J + 1}{\chi(\det J, \varepsilon)}$$

Solve $\lim_{\varepsilon \to 0^+} \arg\min_{\vec{u}} \int_\Omega \left(f_\varepsilon(J) + \lambda g_\varepsilon(J)\right) dx$

Polyconvex energy for $\varepsilon = 0$, polyconvexity is lost for $\varepsilon > 0$.

Amazingly robust, works both in 2D and 3D, not limited to regular grids and simplicial meshes, well-defined in continuous settings.

Testing time! 2D free boundary injective mapping



Testing time! Shape-area tradeoff

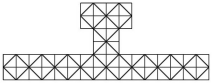
Recall that we solve

$$\lim_{\varepsilon \to 0^+} \arg\min_{\vec{u}} \int_{\Omega} \Big(f_\varepsilon(J) + \lambda\, g_\varepsilon(J) \Big)\, dx,$$

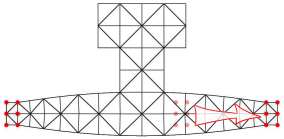
where

$$f_\varepsilon(J) := \frac{\operatorname{tr} J^\top J}{(\chi(\det J, \varepsilon))^{\frac{2}{d}}}$$

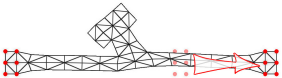
$$g_\varepsilon(J) := \frac{\det^2 J + 1}{\chi(\det J, \varepsilon)}.$$



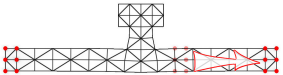
rest shape



angle-preserving mapping ($\lambda = 0$)

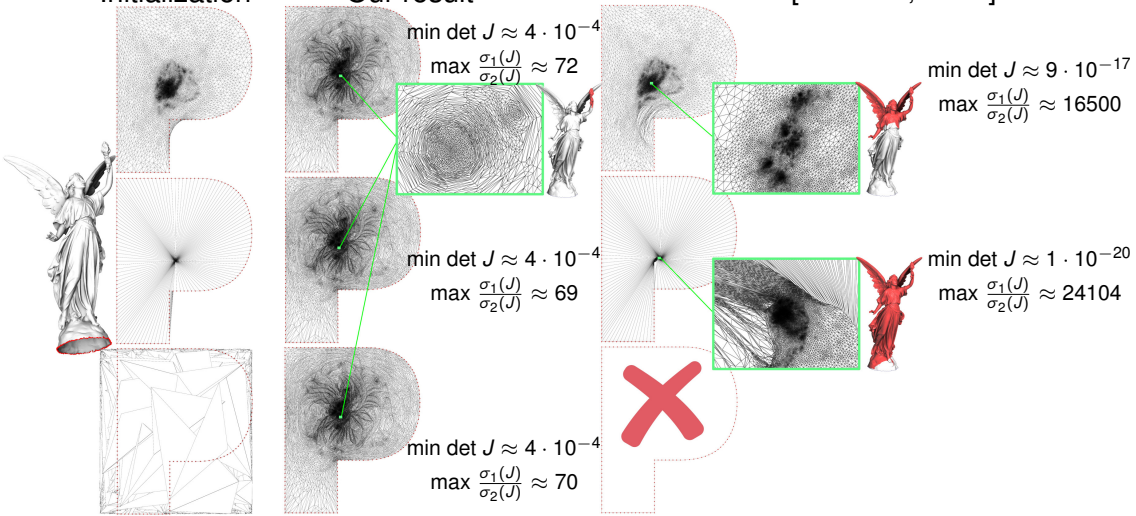


area-preserving mapping ($\lambda = 10^4$)



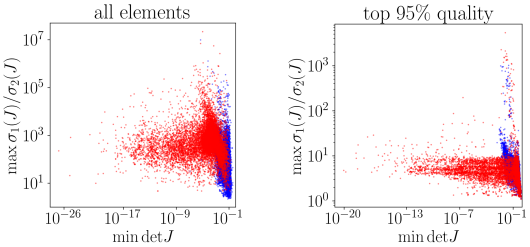
shape-area trade-off ($\lambda = 1$)

Testing time! 2D constrained boundary mapping

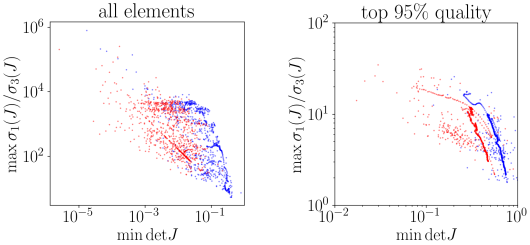


Constrained boundary mapping benchmark

2D dataset
(10743 challenges)

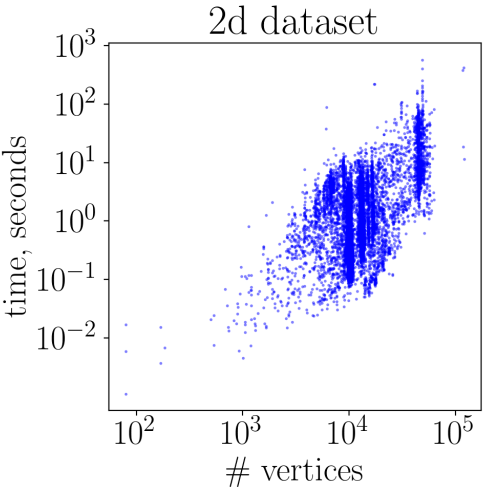


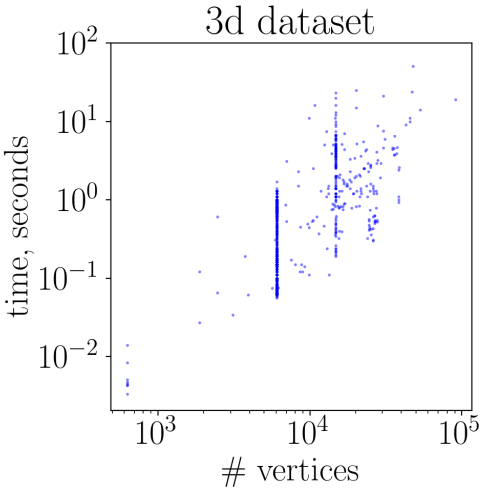
3D dataset
(904 challenges)



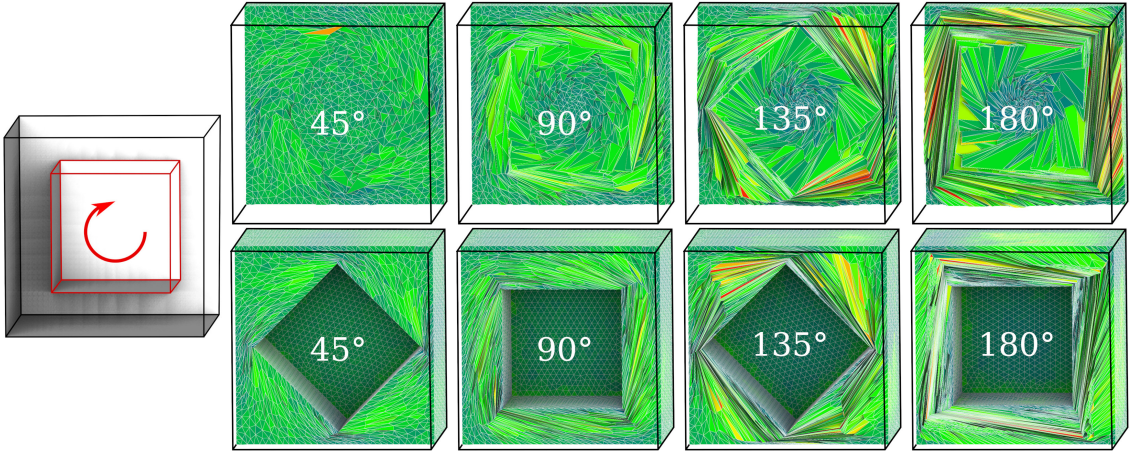
https://github.com/duxingyi-charles/Locally-Injective-Mappings-Benchmark

Running times





Testing time! 3D constrained boundary mapping



Slippery road a.k.a. limitations

Numerical challenge: stiffest problems require a careful implementation

In 2D we have never encountered a practical test case we were unable to treat.

In 3D, however, it can fail due to the numerical challenges in very anisotropic and highly twisted meshes.

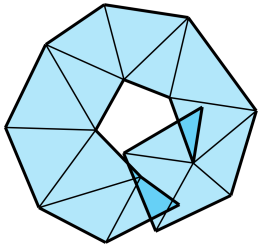
Slippery road a.k.a. limitations

Numerical challenge: stiffest problems require a careful implementation

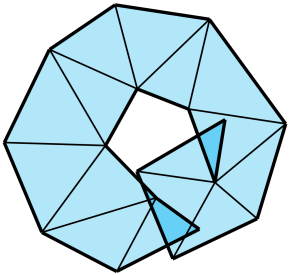
In 2D we have never encountered a practical test case we were unable to treat.

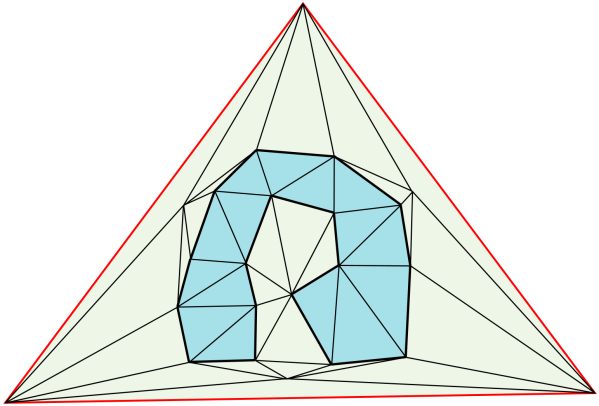
In 3D, however, it can fail due to the numerical challenges in very anisotropic and highly twisted meshes.

Free of inverted elements $\neq$ invertible

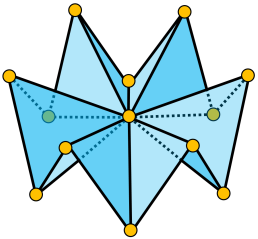


Prevent overlaps altogether: bi-material optimization



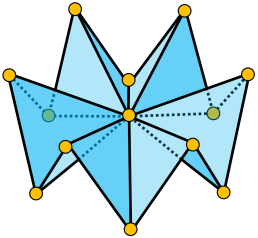


Fun fact: k-coverings

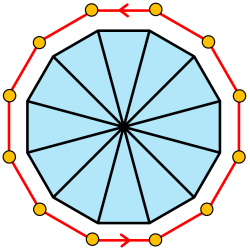


Surface to flatten

Fun fact: k-coverings

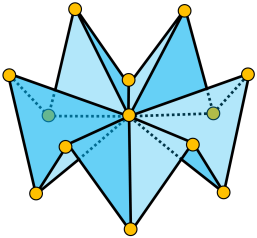


Surface to flatten

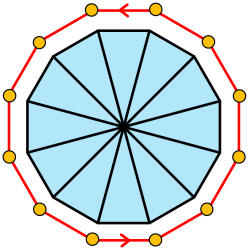


Local minimum, invertible map

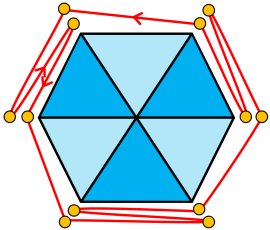
Fun fact: k-coverings



Surface to flatten

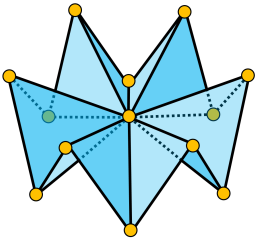


Local minimum, invertible map

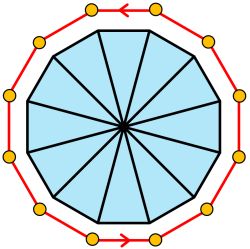


Inverted elements, forbidden

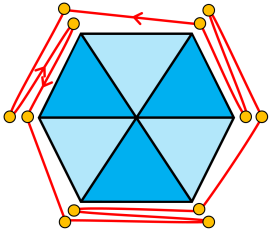
Fun fact: k-coverings



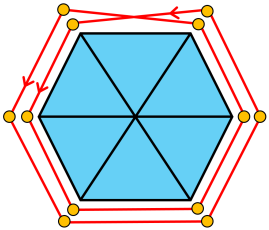
Surface to flatten



Local minimum, invertible map

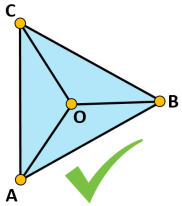


Inverted elements, forbidden

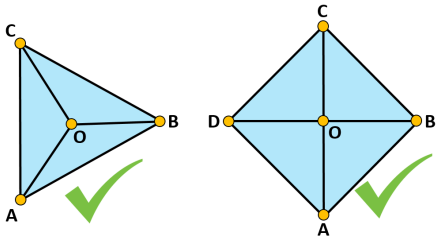


Global minimum, no inverted elements, **not invertible** locally in one point

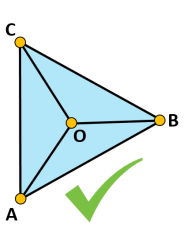
Preventing k-coverings in 2D

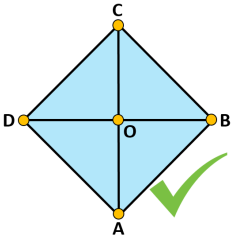


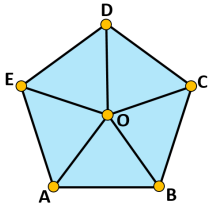
Preventing k-coverings in 2D

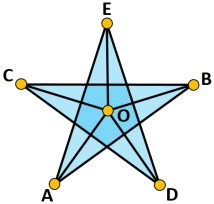


Preventing k-coverings in 2D

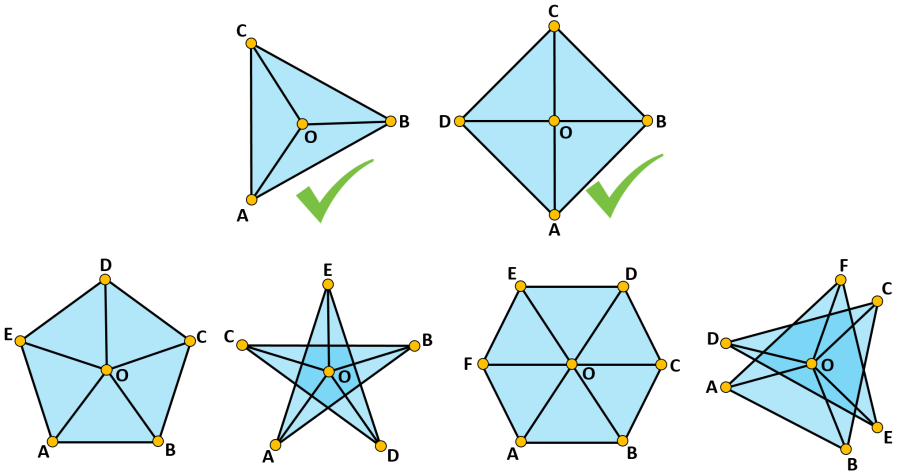




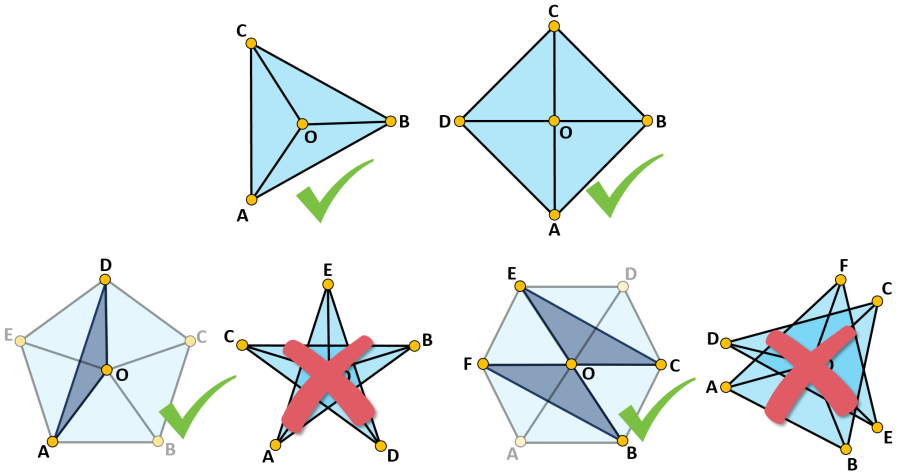




Preventing k-coverings in 2D



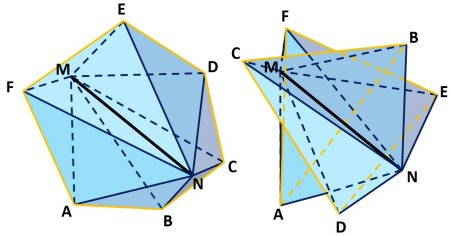
Preventing k-coverings in 2D



Valences 5+ can be reduced to 3 and 4, effectively preventing k-coverings.

No tweaking of the optimization, pre-processing the data!

Preventing k-coverings in 3D



Preventing k-coverings in 3D

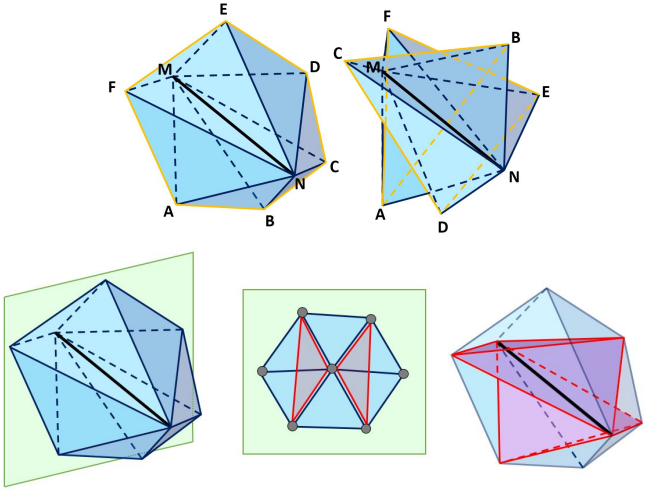
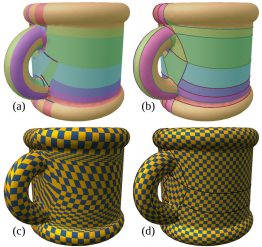


Table of Contents

Conclusion

Up to now, the options we had for parameterization:

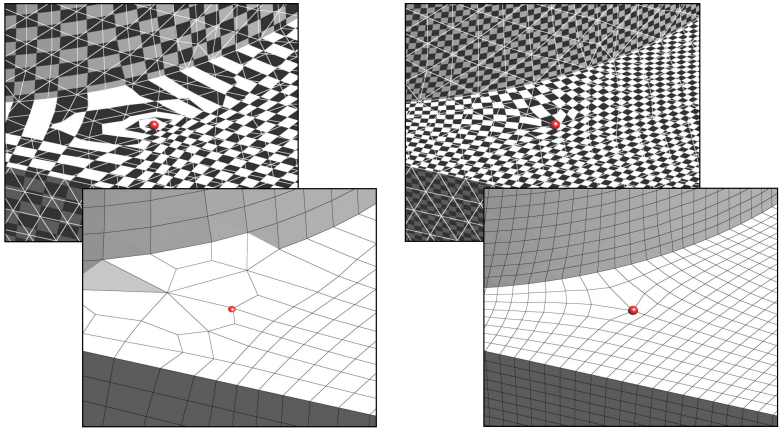
Conformal maps
[Sawhney and Crane 2017]



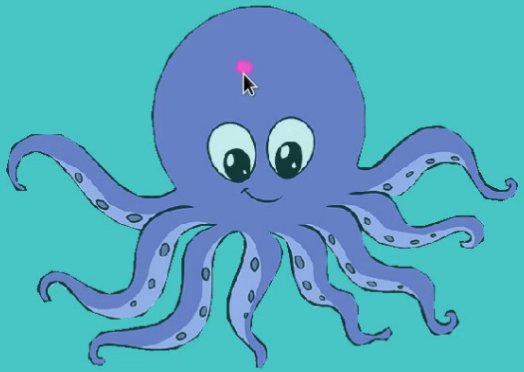
Tutte embedding
[Myles et al 2014]

Conclusion

Now we can make more things!



Inversion-free deformation



with extra triangles to
prevent 2-coverings

It works.

Bibliography

J. M. Ball. Global invertibility of sobolev functions and the interpenetration of matter. *Proceedings of the Royal Society of Edinburgh: Section A Mathematics*, 88(3-4):315–328, 1981. doi: 10.1017/S030821050002014X.

J.U Brackbill and J.S Saltzman. Adaptive zoning for singular problems in two dimensions. *Journal of Computational Physics*, 46(3):342 – 368, 1982. ISSN 0021-9991. doi: https://doi.org/10.1016/0021-9991(82)90020-1. URL http://www.sciencedirect.com/science/article/pii/0021999182900201.

WP Crowley. An equipotential zoner on a quadrilateral mesh. *Memo, Lawrence Livermore National Lab*, 5, 1962.

Xingyi Du, Noam Aigerman, Qingnan Zhou, Shahar Z. Kovalsky, Yajie Yan, Danny M. Kaufman, and Tao Ju. Lifting simplices to find injectivity. *ACM Trans. Graph.*, 39(4), July 2020. ISSN 0730-0301. doi: 10.1145/3386569.3392484. URL https://doi.org/10.1145/3386569.3392484.

SA Ivanenko. Construction of nondegenerate grids. *Zh. Vychisl. Mat. Mat. Fiz*, 28(10):1498, 1988.

W. T. Tutte. How to Draw a Graph. *Proceedings of the London Mathematical Society*, s3-13(1):743–767, 01 1963. ISSN 0024-6115. doi: 10.1112/plms/s3-13.1.743. URL https://doi.org/10.1112/plms/s3-13.1.743.

Alan M Winslow. Numerical solution of the quasilinear poisson equation in a nonuniform triangle mesh. *Journal of computational physics*, 1(2):149–172, 1966.

How to compute locally invertible maps

Vladimir Garanzha, Igor Kaporin, Liudmila Kudryavtseva, François Protais, Nicolas Ray, Dmitry Sokolov