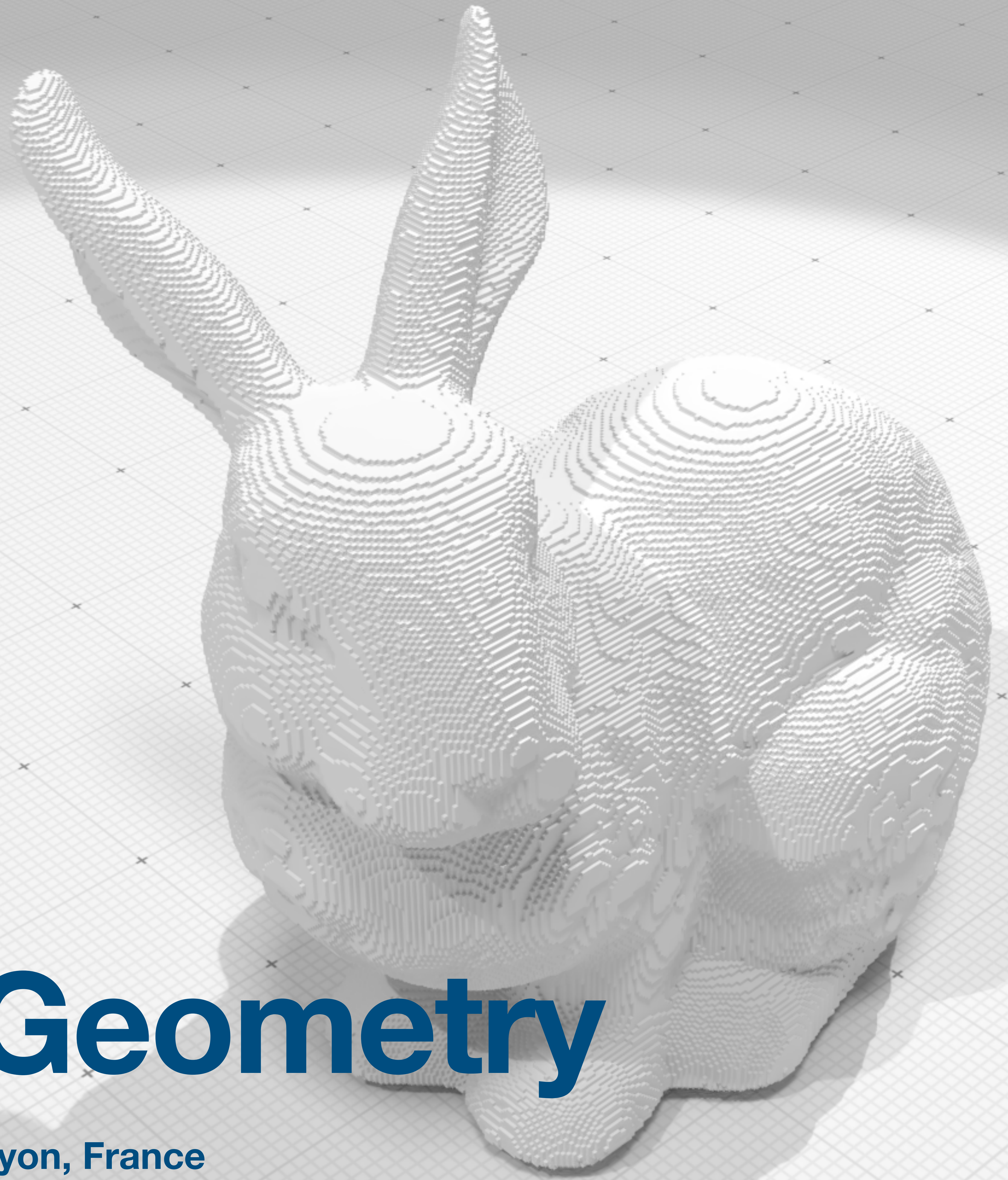
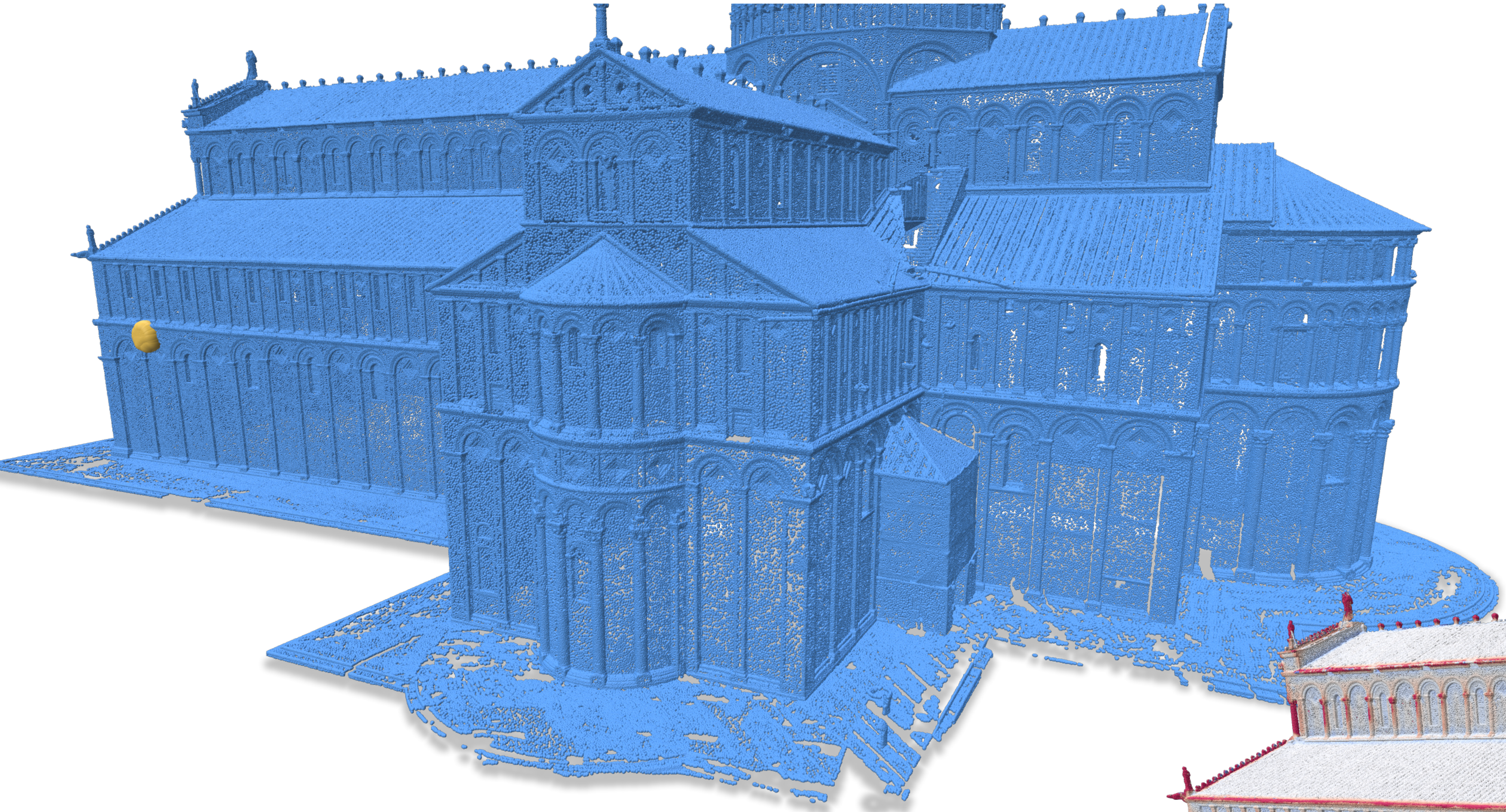
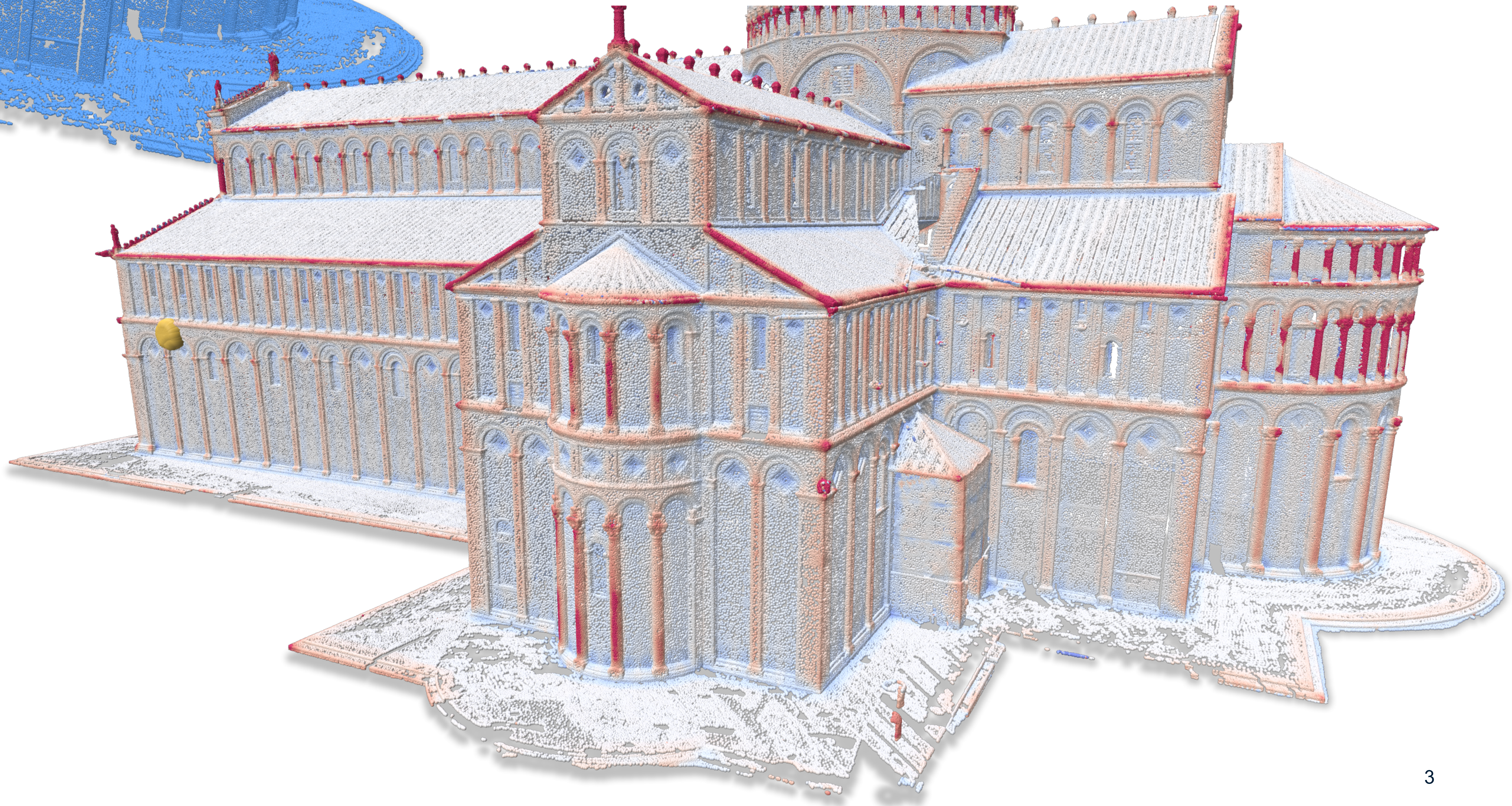


Digital Geometry

David Coeurjolly, CNRS, Lyon, France





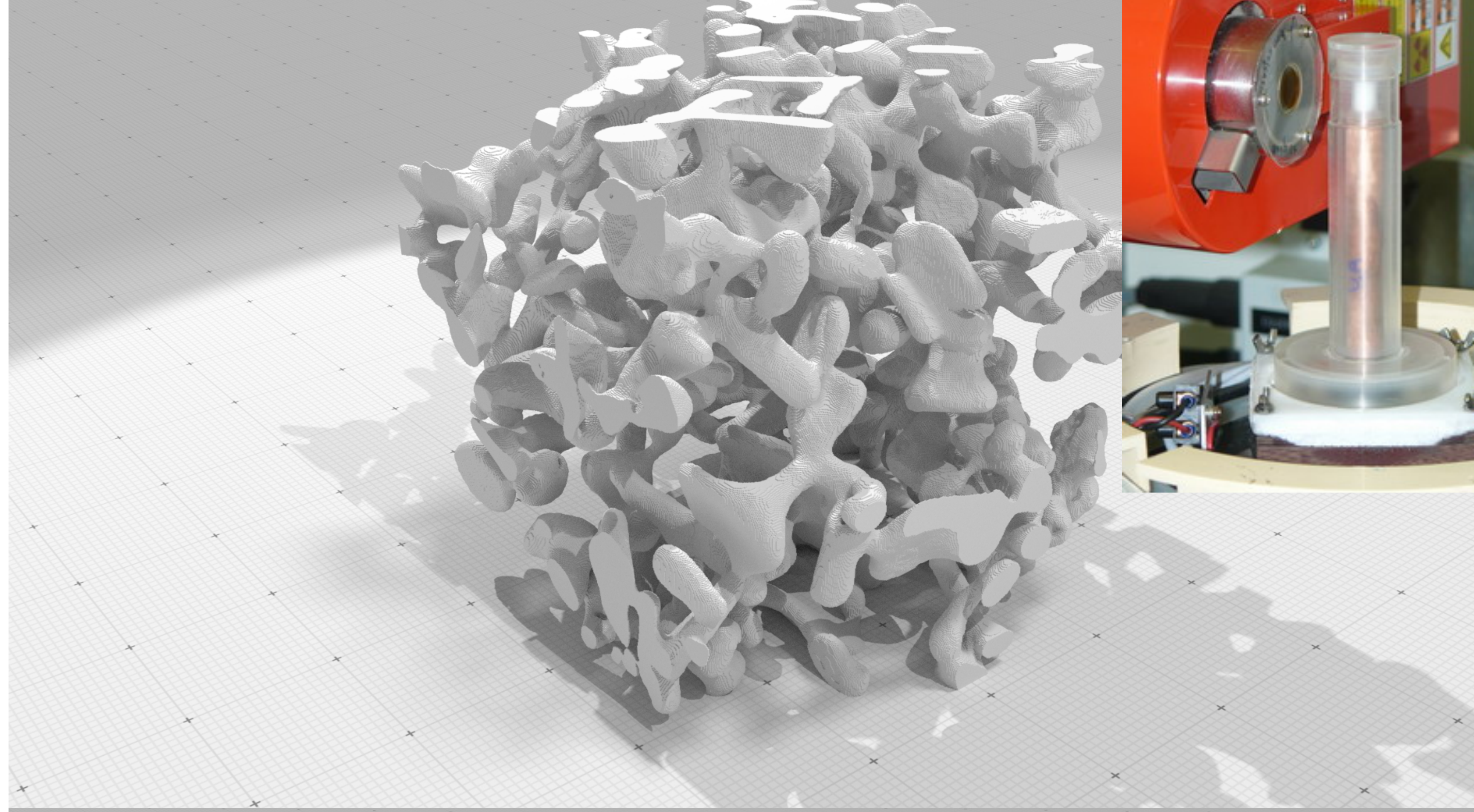
Outline

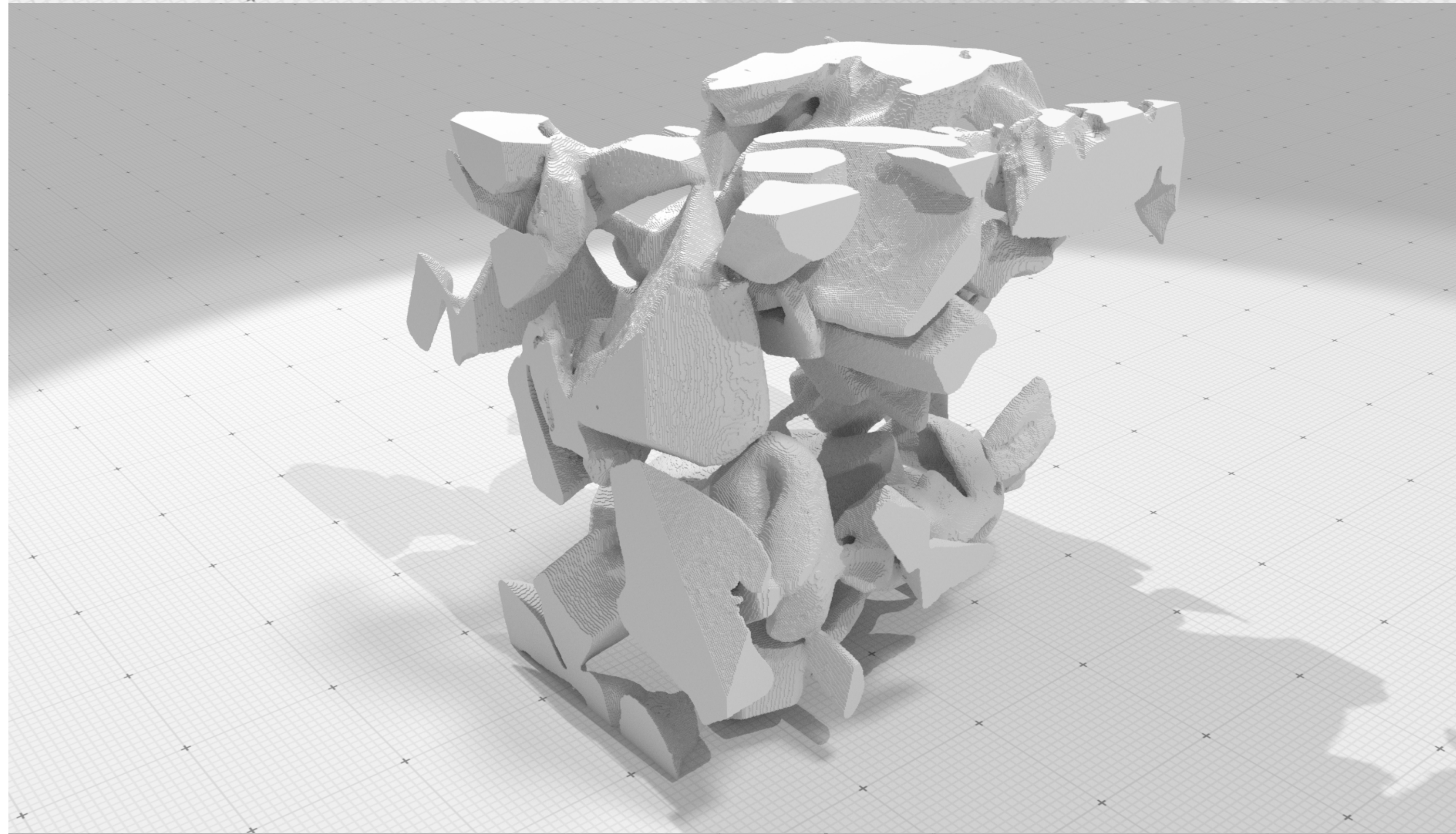
- context
- dgtal.org
- $\mathbb{Z}$ -- geometry with integers
- $\mathbb{Z}^d$ -- geometry processing on grids
- digital surface processing
- conclusion

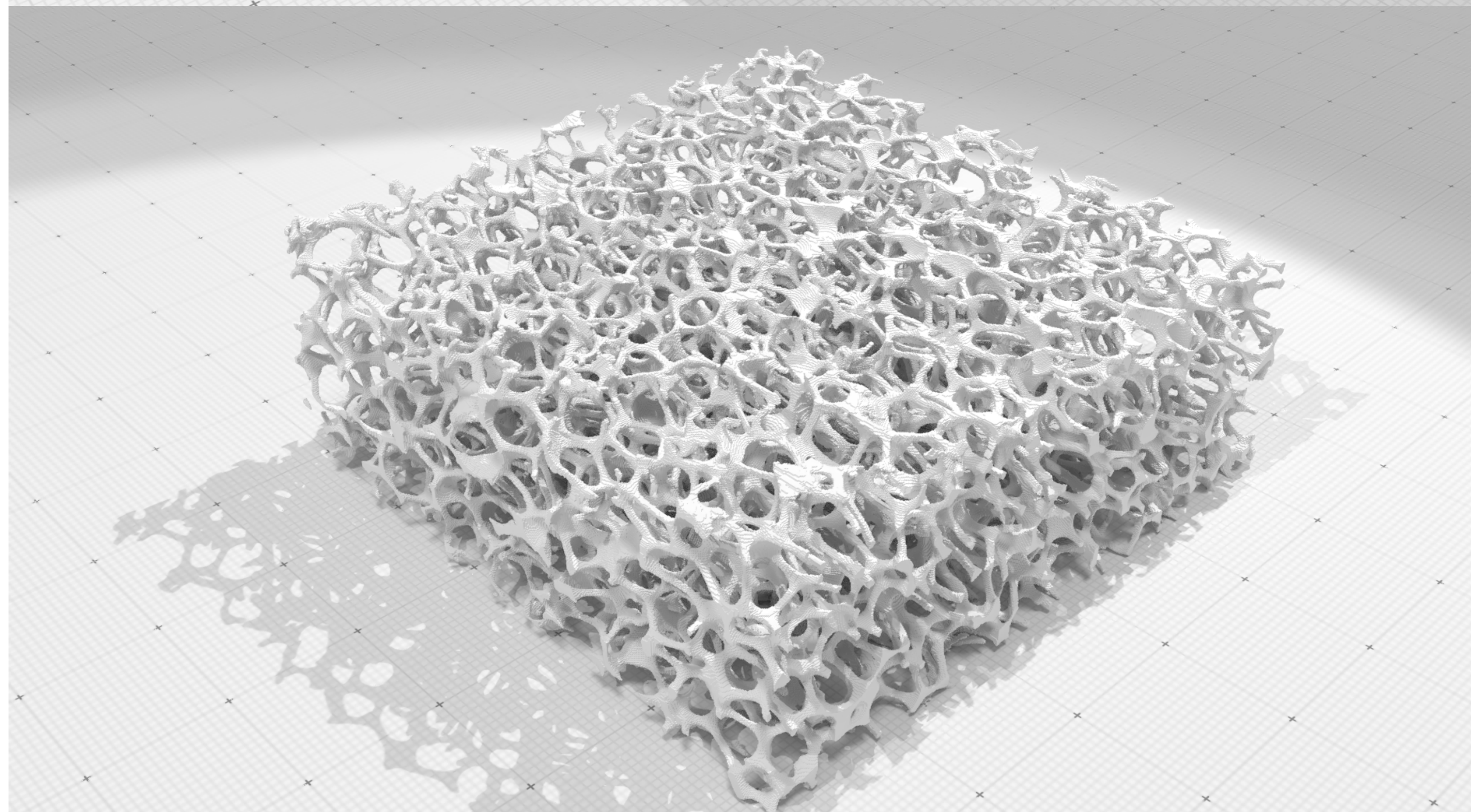
Motivations (1): devices

- Micro-tomographic images
  - material sciences
  - medical images

- Process geometry/topology of images partitions





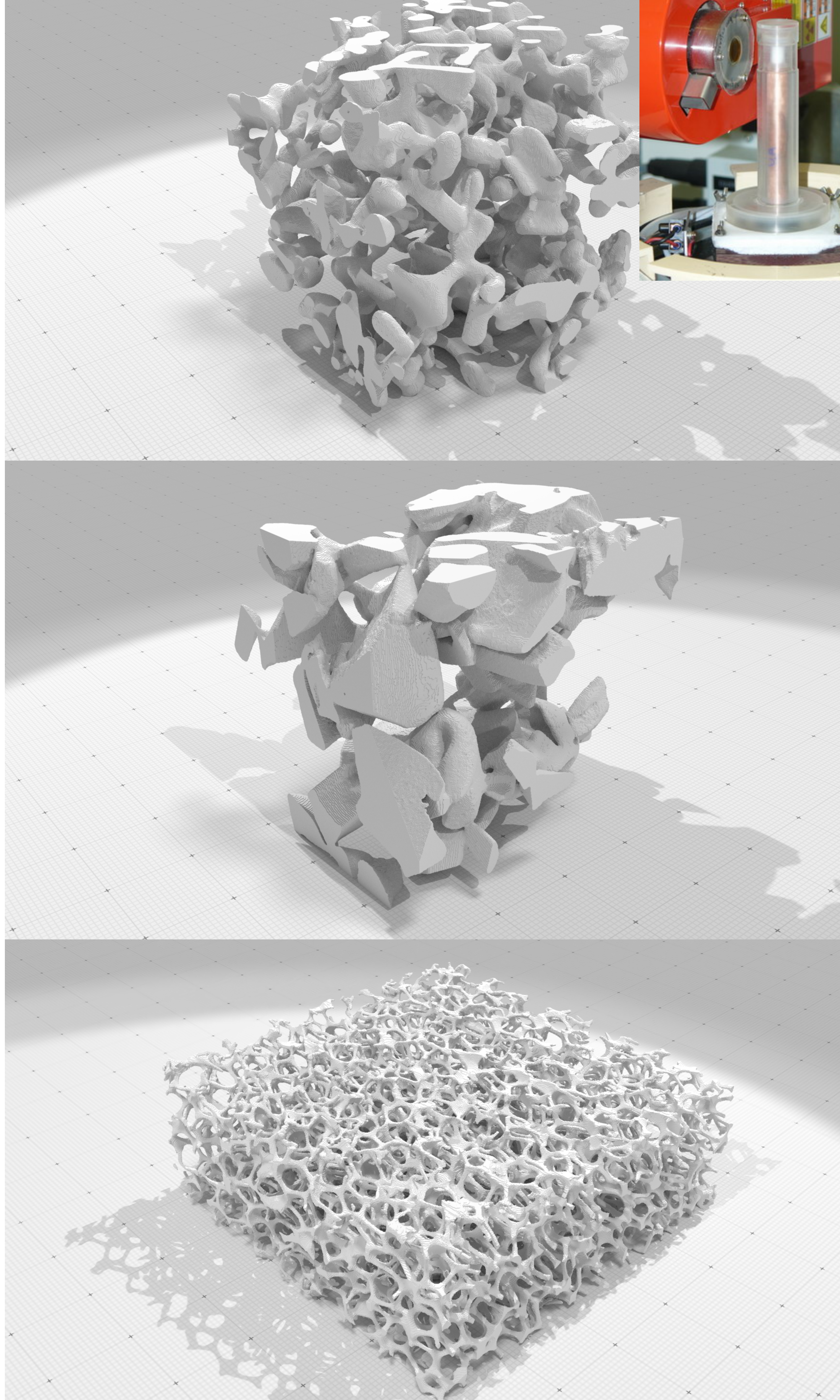


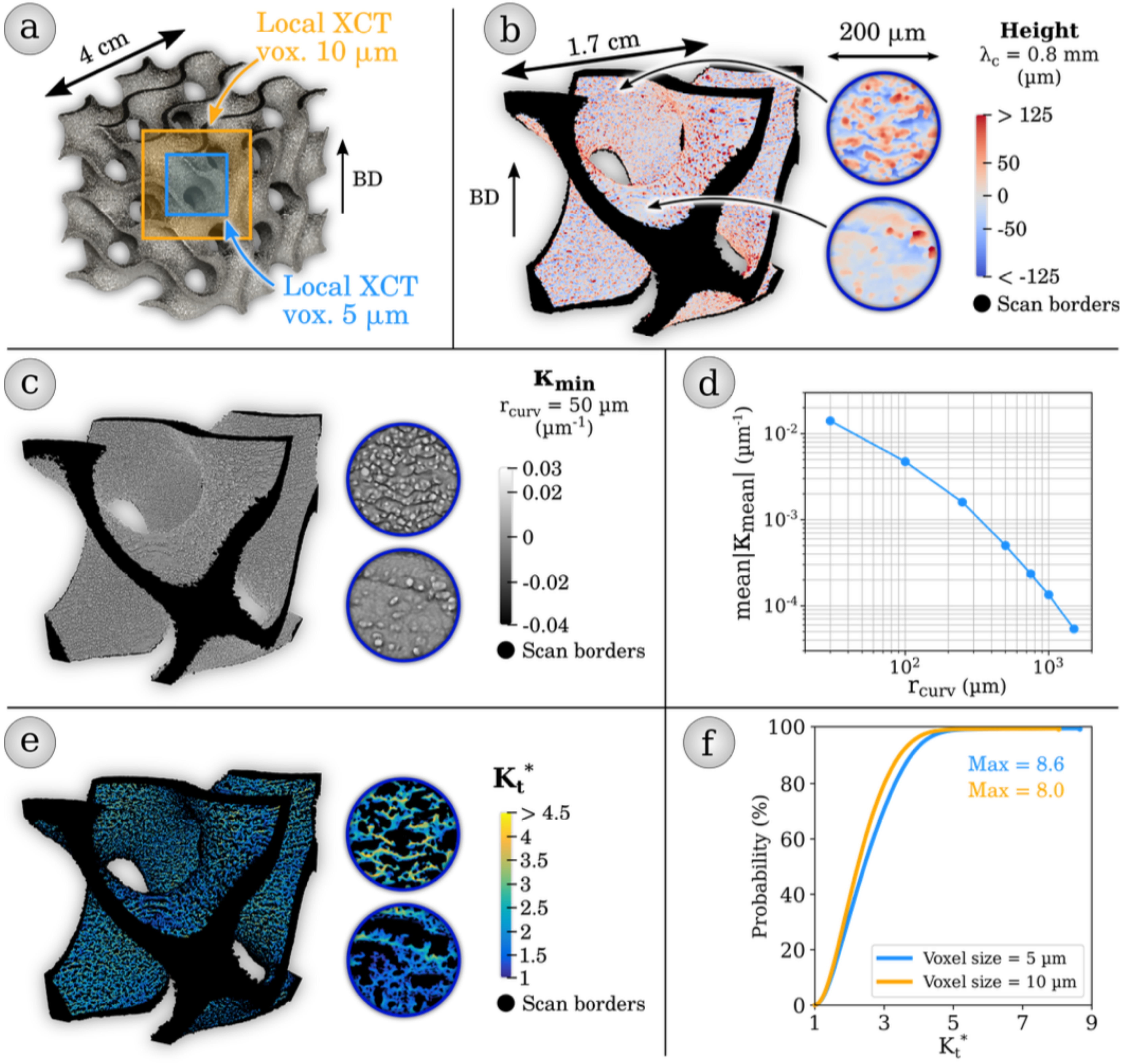
Motivations (1): devices

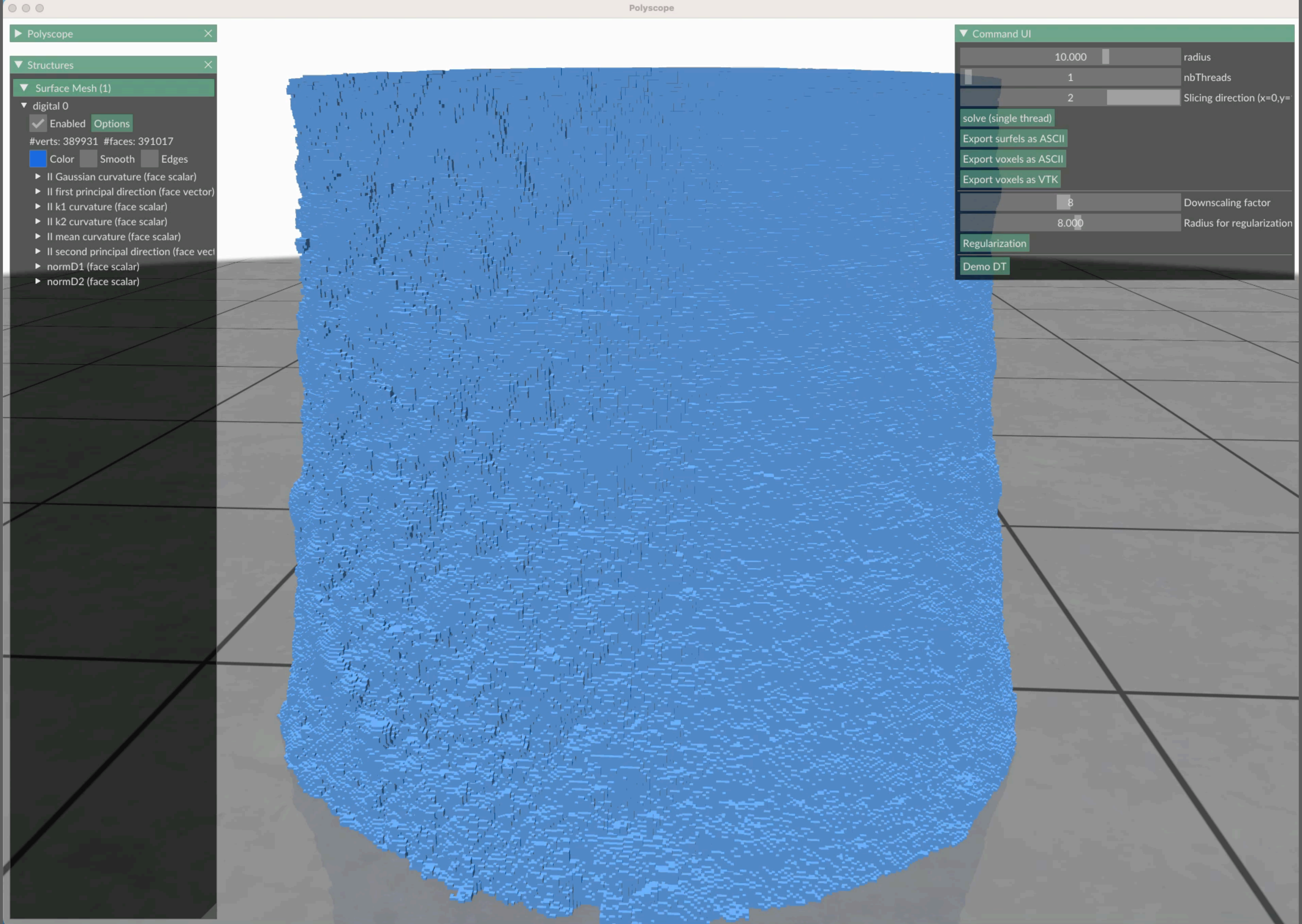
- Micro-tomographic images
  - material sciences
  - medical images

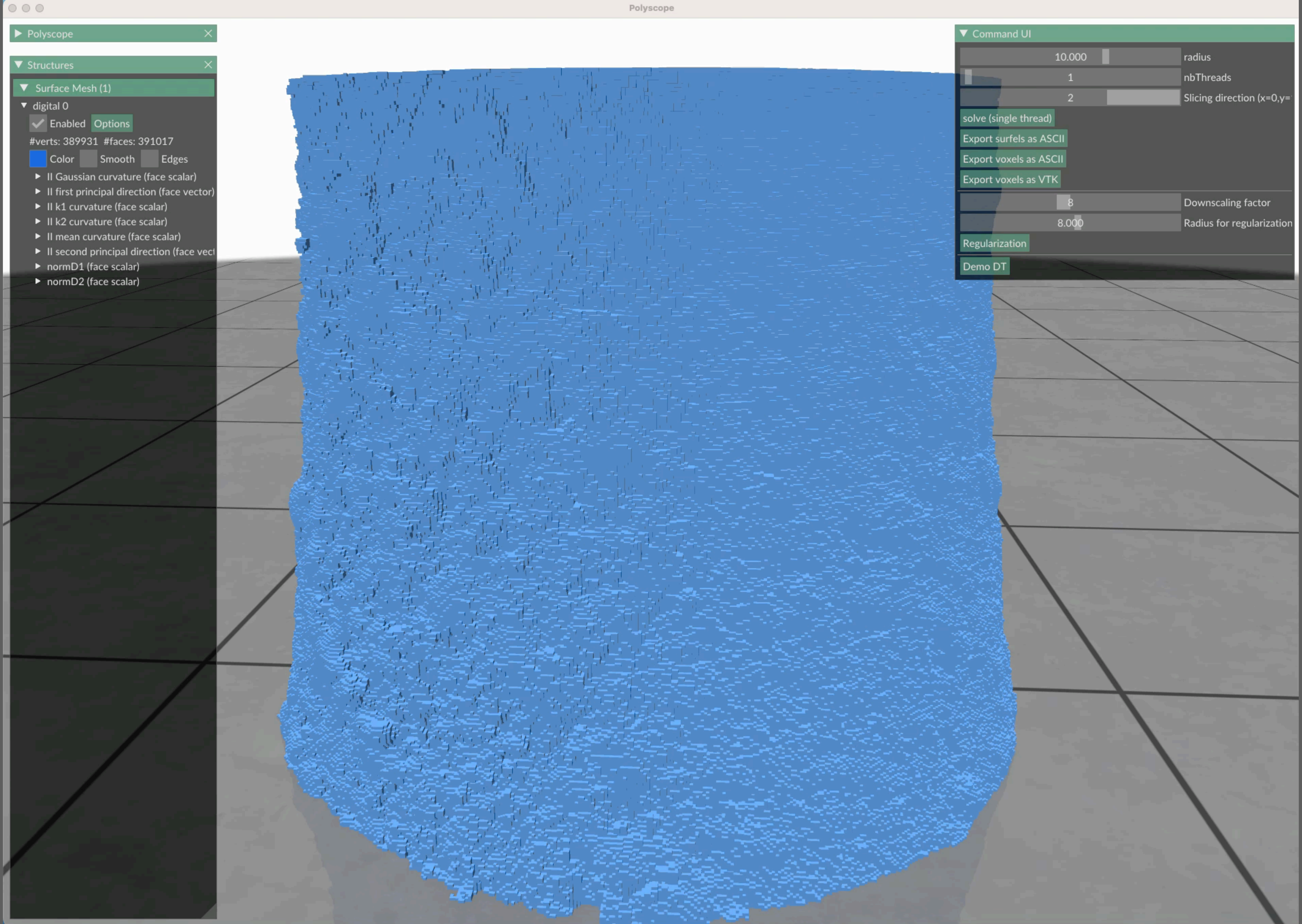
- Process geometry/topology of images partitions

$$\Rightarrow X \subset \mathbb{Z}^3$$




a
4 cm
Local XCT
vox. 10 µm
Local XCT
vox. 5 µm
BD
b
1.7 cm
200 µm
Height
λc = 0.8 mm
(µm)
> 125
50
0
-50
< -125
Scan borders
BD
c
Kmin
rcurv = 50 µm
(µm-1)
0.03
0.02
0
-0.02
-0.04
Scan borders
d
mean|Kmean| (µm-1)
10-2
10-3
10-4
102
103
rcurv (µm)
e
Kt*
> 4.5
4
3.5
3
2.5
2
1.5
1
Scan borders
f
Probability (%)
100
80
60
40
20
0
Max = 8.6
Max = 8.0
Voxel size = 5 µm
Voxel size = 10 µm
1
3
5
7
9
Kt*


Polyscope
Polyscope
Structures
Surface Mesh (1)
digital 0
Enabled
Options
#verts: 389931 #faces: 391017
Color
Smooth
Edges
II Gaussian curvature (face scalar)
II first principal direction (face vector)
II k1 curvature (face scalar)
II k2 curvature (face scalar)
II mean curvature (face scalar)
II second principal direction (face vect
normD1 (face scalar)
normD2 (face scalar)
Command UI
10.000
radius
1
nbThreads
2
Slicing direction (x=0,y=
solve (single thread)
Export surfels as ASCII
Export voxels as ASCII
Export voxels as VTK
8
Downscaling factor
8.000
Radius for regularization
Regularization
Demo DT

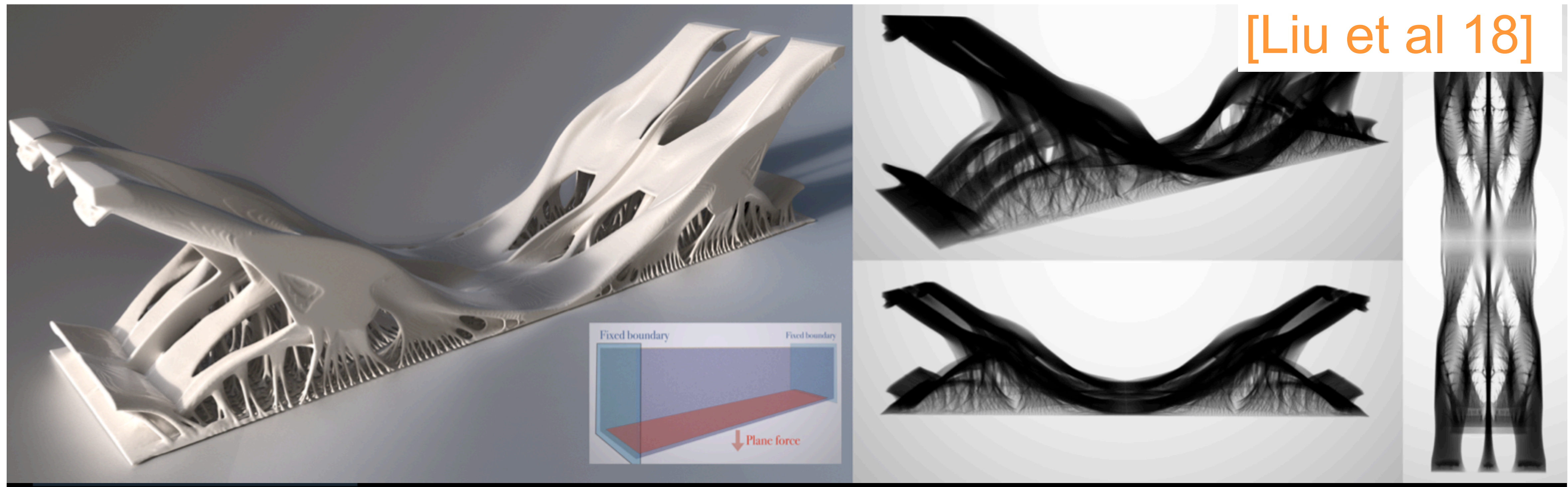

Polyscope
Polyscope
Structures
Surface Mesh (1)
digital 0
Enabled Options
#verts: 389931 #faces: 391017
Color Smooth Edges
II Gaussian curvature (face scalar)
II first principal direction (face vector)
II k1 curvature (face scalar)
II k2 curvature (face scalar)
II mean curvature (face scalar)
II second principal direction (face vect
normD1 (face scalar)
normD2 (face scalar)
Command UI
10.000 radius
1 nbThreads
2 Slicing direction (x=0,y=
solve (single thread)
Export surfels as ASCII
Export voxels as ASCII
Export voxels as VTK
8 Downscaling factor
8.000 Radius for regularization
Regularization
Demo DT

Motivations (2): $\mathbb{Z}^d$ as an efficient modelling space

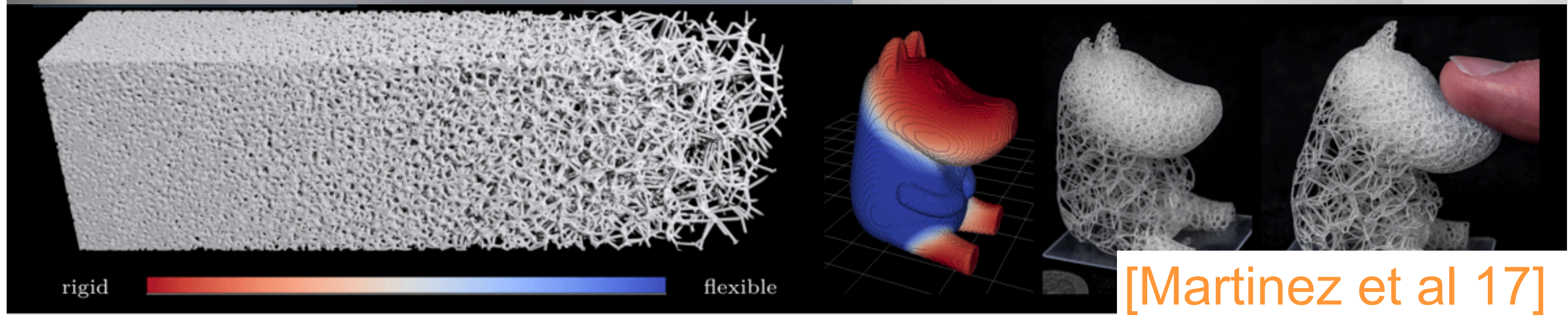
- Shape optimization / fabrication

- As a proxy or an intermediate representation

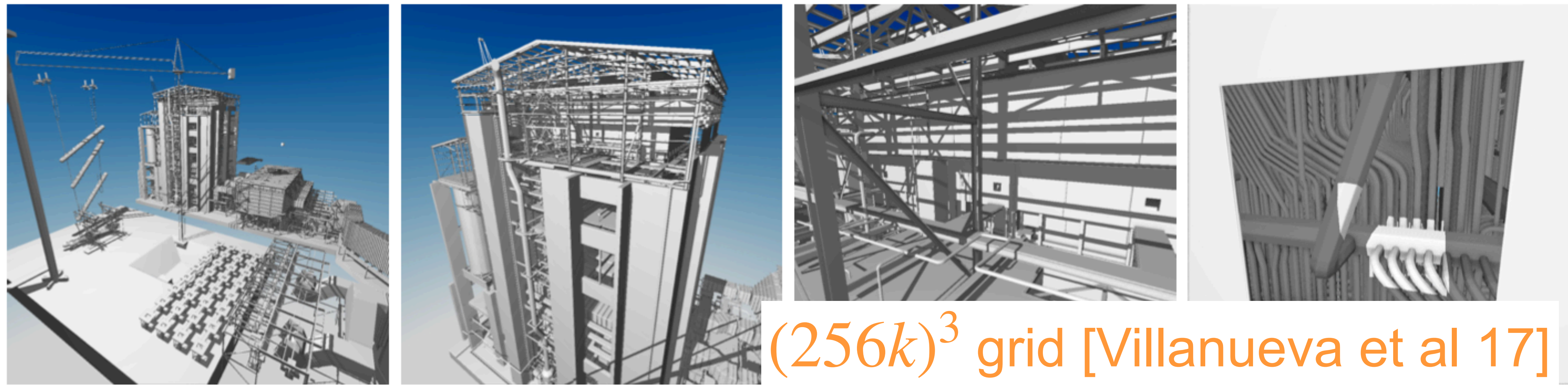
  light transport simulation, booleans, medial axis, distance fields, multiple interfaces/objects tracking in a simulation loop…



[Liu et al 18]



[Martinez et al 17]



$(256k)^3$ grid [Villanueva et al 17]

***Focus**: characteristic functions / labelled images / level sets / …*

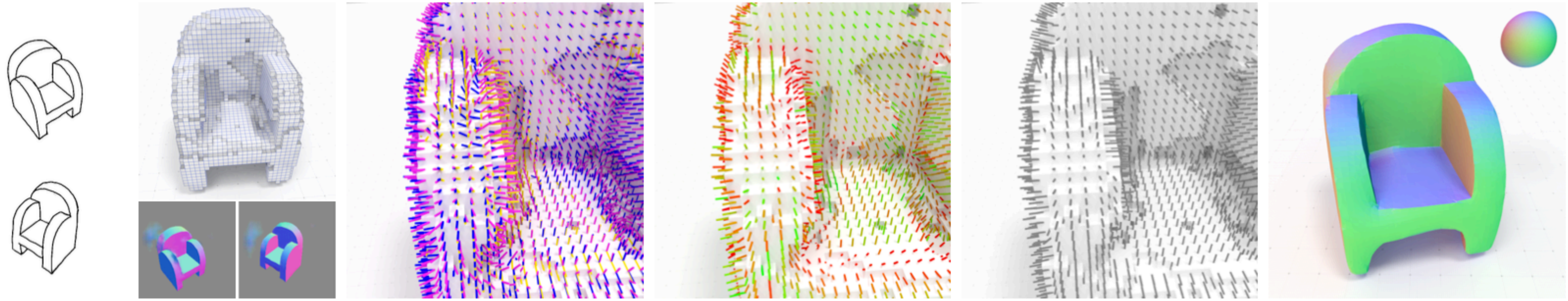

File
Options
ColorSelector
dragon
indirect
HDR
Bloom
Motion Blur
SMAA
SSAO
DoF
Global illumination
Second Bounce
40 fps


File
Options
ColorSelector
dragon
indirect
HDR
Bloom
Motion Blur
SMAA
SSAO
DoF
Global illumination
Second Bounce
40 fps


TEARDOWN
RELEASE DATE
TRAILER


TEARDOWN
RELEASE DATE
TRAILER

Example



(a) Input (b) CNNs predictions (c) Candidate normals (d) Aggregated normals (e) Piecewise-smooth normals (f) Final surface

[Delanoy et al 19]

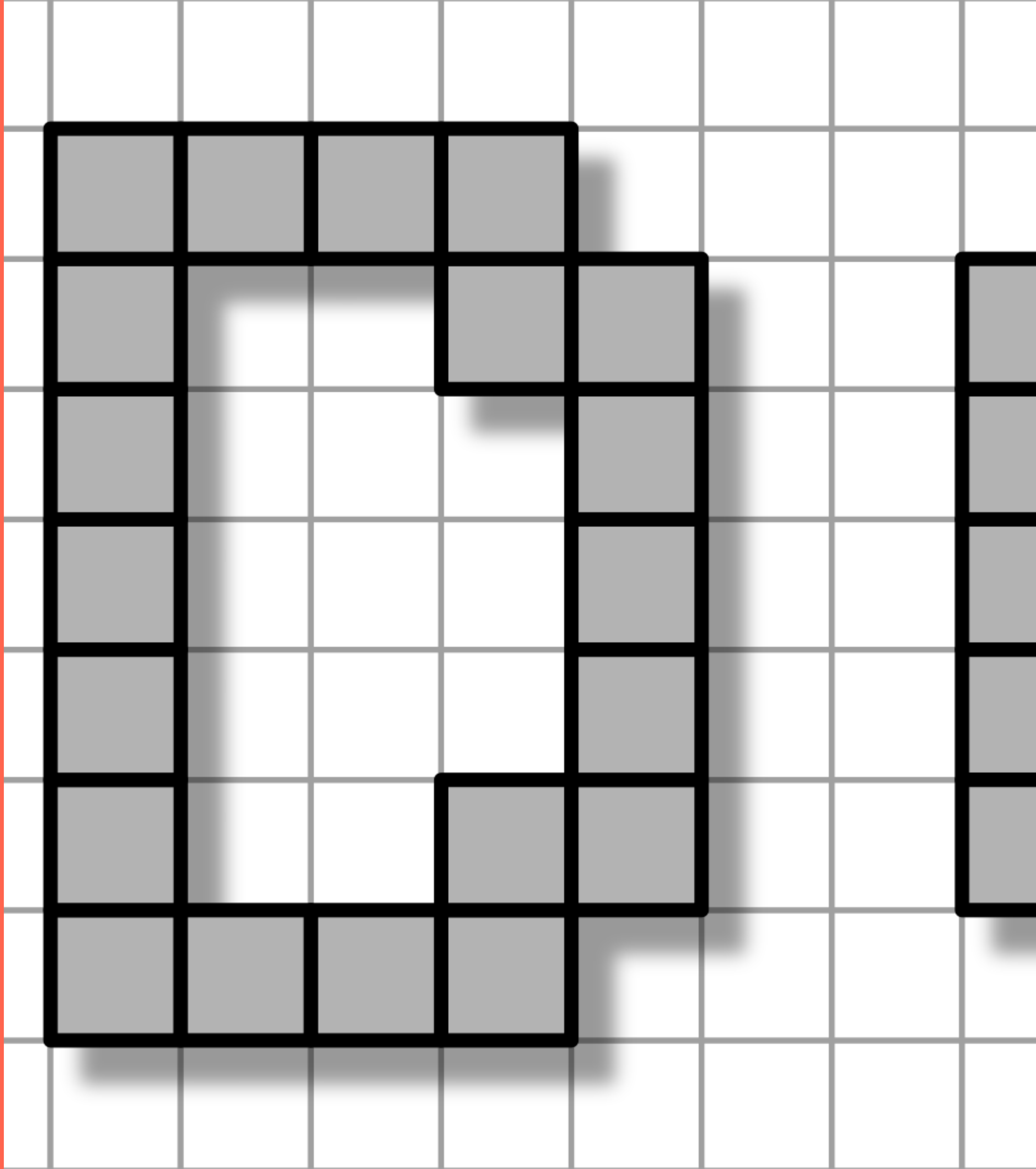
Digital Geometry

Topology and geometry processing on regular data:

- fast algorithms thanks to the regularity of the data
- simple topological structure
- integer based computations
- advanced surface based geometry processing

… in $\mathbb{Z}^d$

dgtal.org



DGtal

Digital Geometry Tools and Algorithms Library

Fork me on GitHub

News

DGtal release 1.2

Posted on June 1, 2021

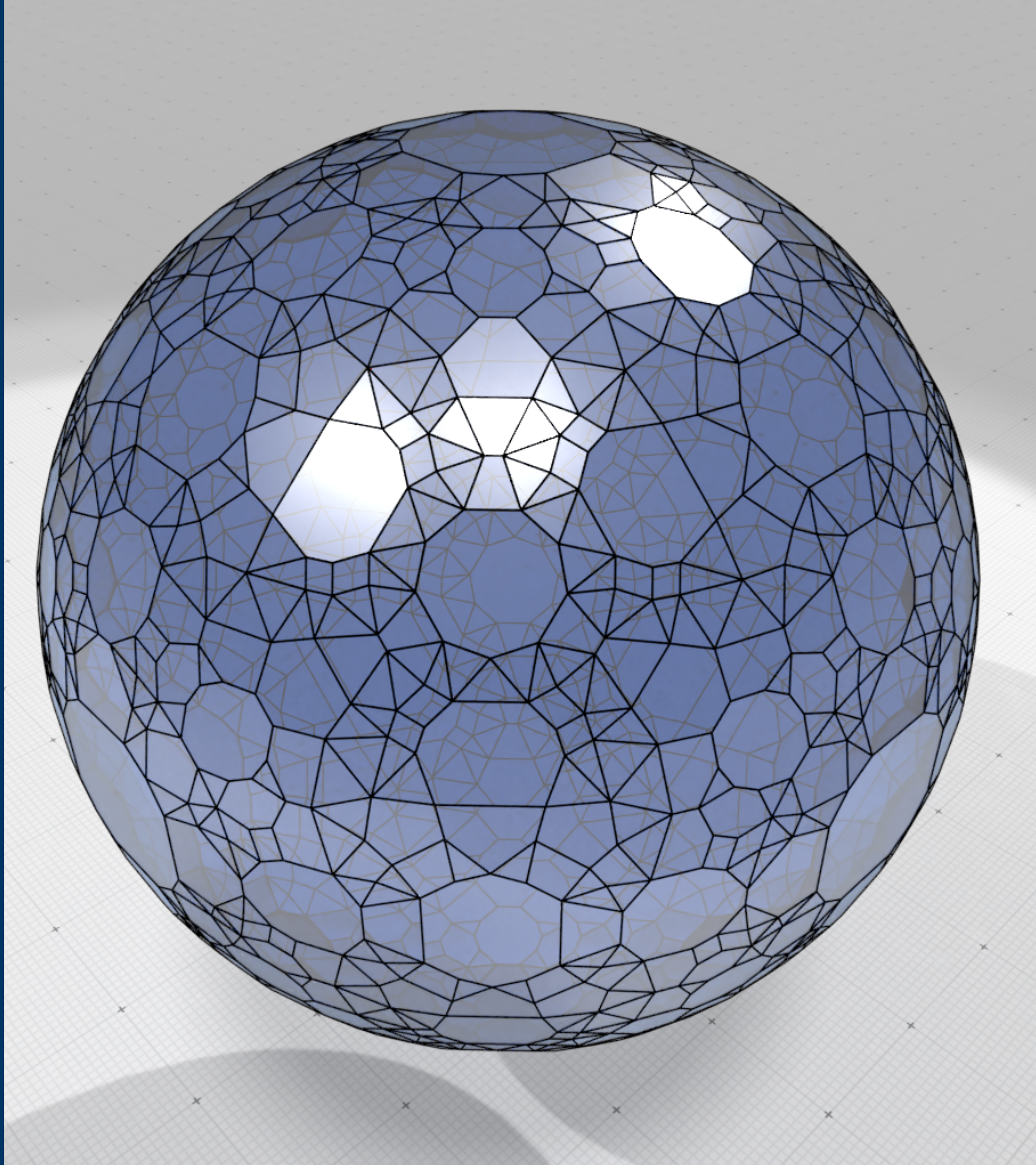
We are really excited to share with you the release 1.2 of DGtal and its tools. As usual, all edits and bugfixes are listed in the Changelog, and we would like to thank all devs involved in this release. In this short review, we would like to focus on only...

[Read More]

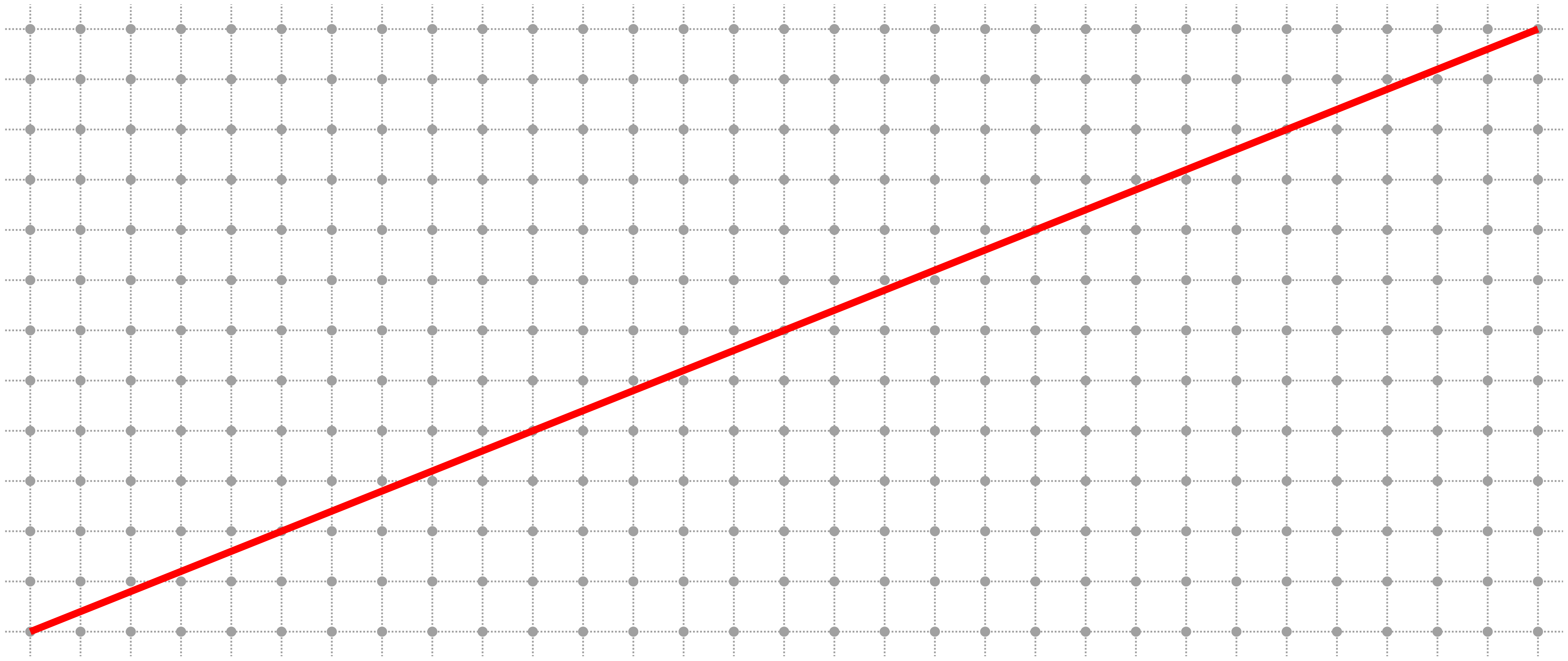
DGtal release 1.1

https://dgtal.org

python



Quick example



$$a_0 + \cfrac{b_1}{a_1 + \cfrac{b_2}{a_2 + \cfrac{b_3}{a_3 + \ddots}}}$$

- Rational slope $\alpha = \dfrac{p}{q}$

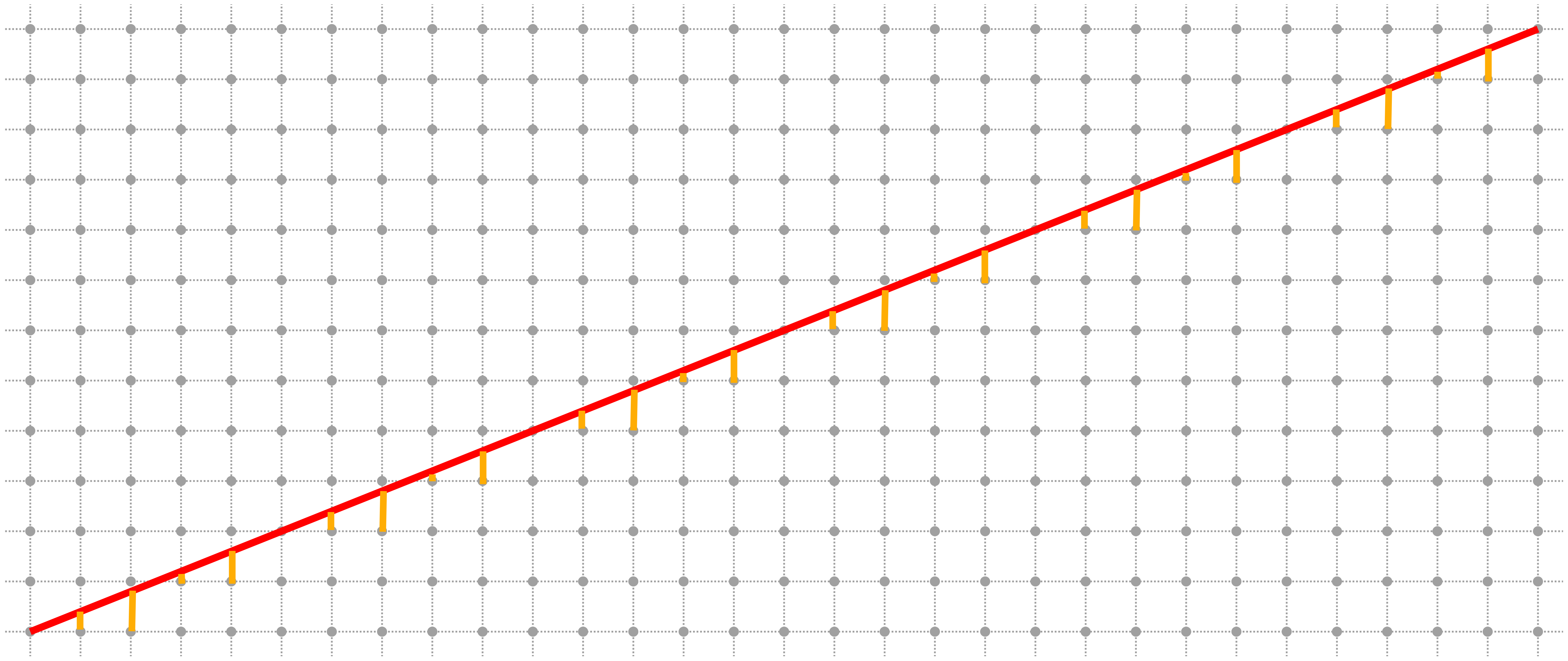
  ⇒ finite set of remainders

  ⇒ periodic structure $q/\gcd(p, q)$

  ⇒ canonical pattern from **continued fraction**

- *arithmetization* to speed-up tracing (e.g. fast ray marching on SVO)

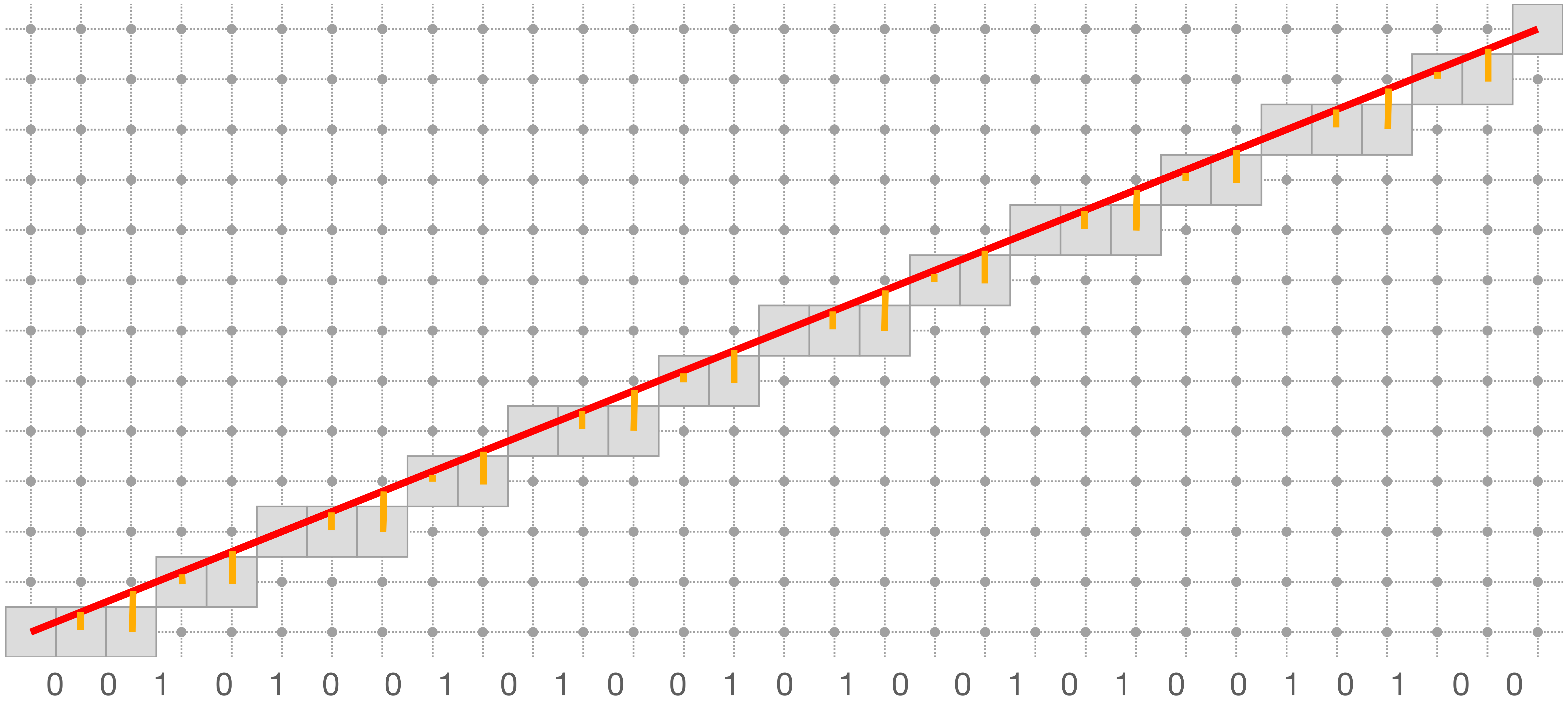
Quick example



$$a_0 + \cfrac{b_1}{a_1 + \cfrac{b_2}{a_2 + \cfrac{b_3}{a_3 + \ddots}}}$$

- Rational slope $\alpha = \frac{p}{q}$
  $\Rightarrow$ finite set of remainders
  $\Rightarrow$ periodic structure $q/\gcd(p,q)$
  $\Rightarrow$ canonical pattern from **continued fraction**
- *arithmetization* to speed-up tracing (e.g. fast ray marching on SVO)

Quick example



$$a_0 + \cfrac{b_1}{a_1 + \cfrac{b_2}{a_2 + \cfrac{b_3}{a_3 + \ddots}}}$$

- Rational slope $\alpha = \dfrac{p}{q}$

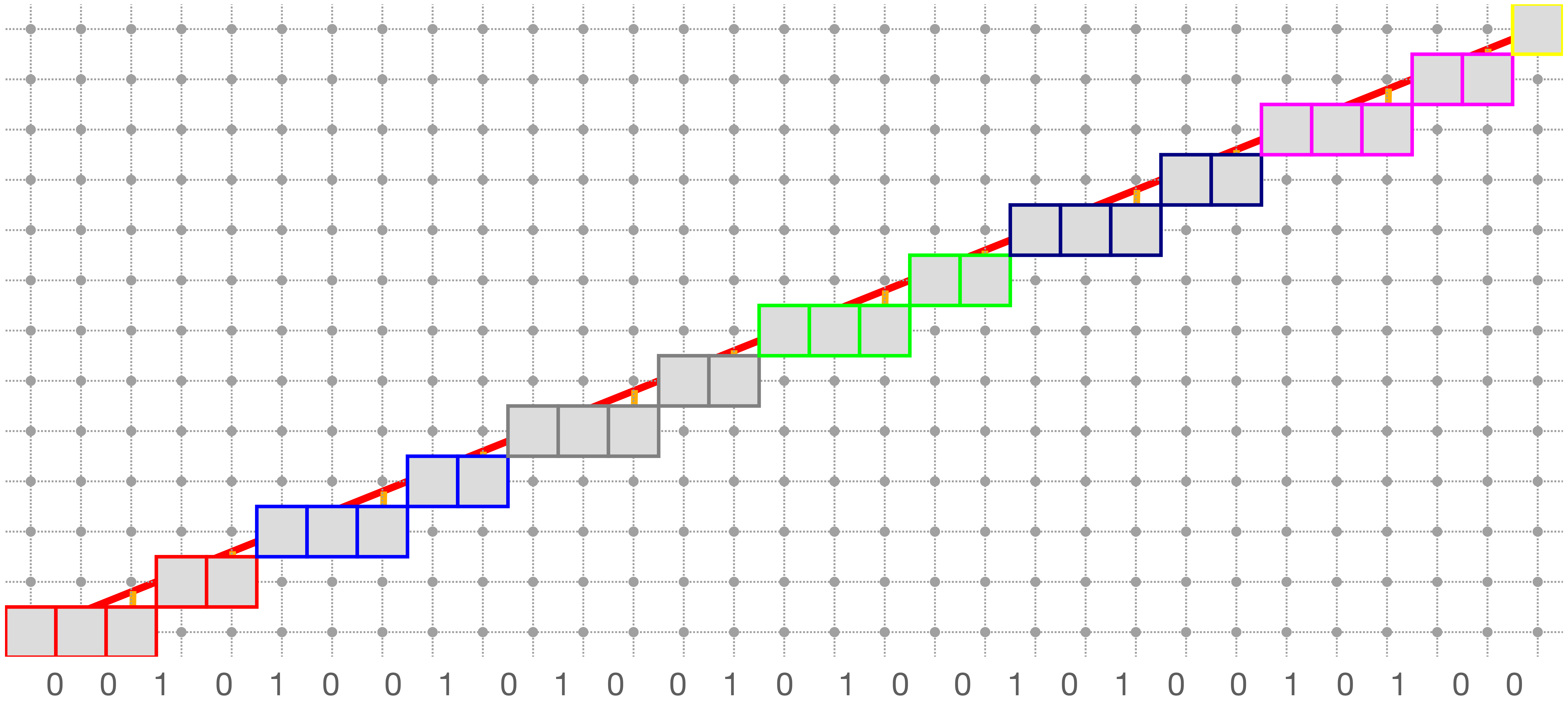
  $\Rightarrow$ finite set of remainders

  $\Rightarrow$ periodic structure $q/\mathrm{gcd}(p, q)$

  $\Rightarrow$ canonical pattern from **continued fraction**

- *arithmetization* to speed-up tracing (e.g. fast ray marching on SVO)

Quick example



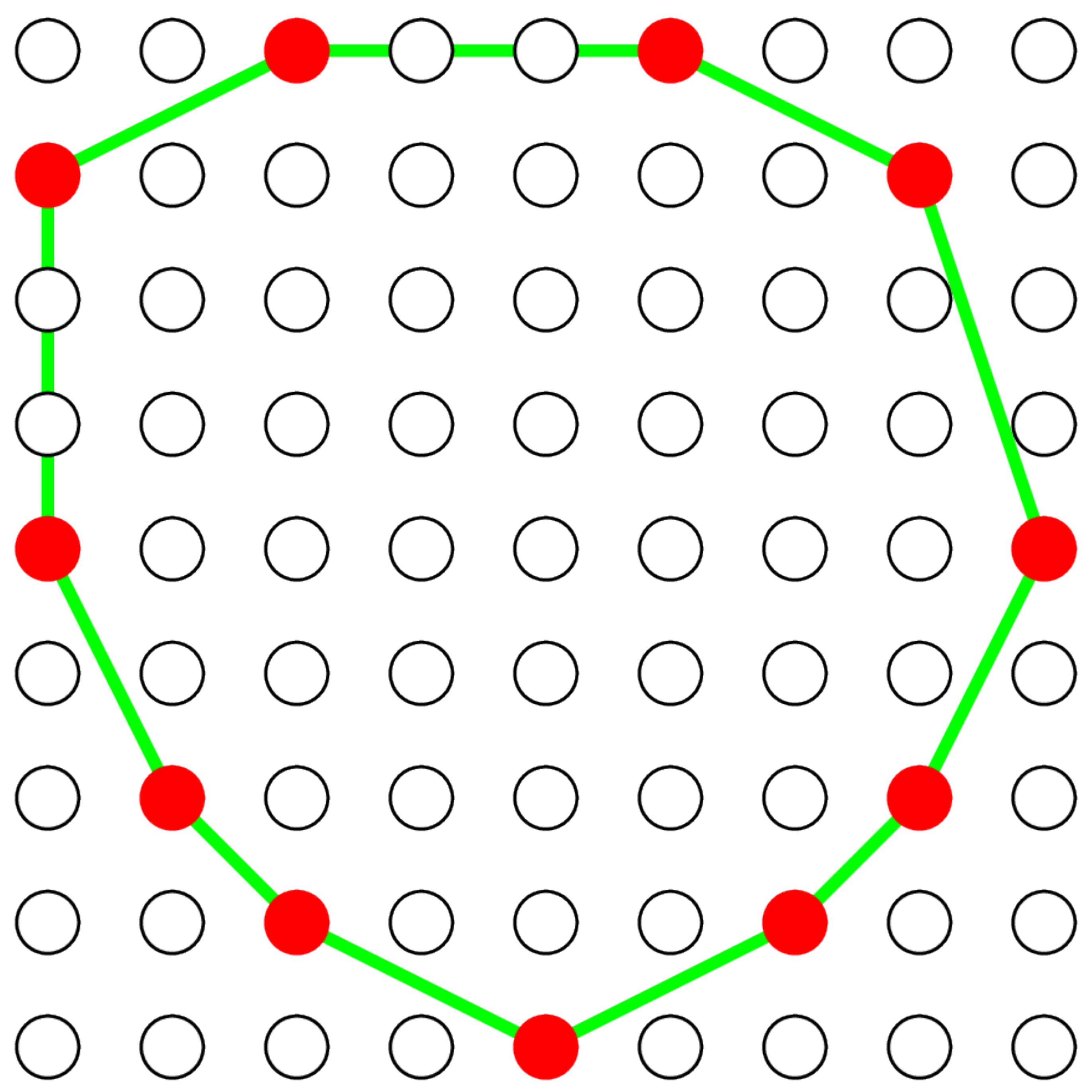
$$a_0 + \cfrac{b_1}{a_1 + \cfrac{b_2}{a_2 + \cfrac{b_3}{a_3 + \ddots}}}$$

- Rational slope $\alpha = \dfrac{p}{q}$
  $\Rightarrow$ finite set of remainders
  $\Rightarrow$ periodic structure $q/\gcd(p,q)$
  $\Rightarrow$ canonical pattern from **continued fraction**
- *arithmetization* to speed-up tracing (e.g. fast ray marching on SVO)

Convex hull in 2d

For n points in $\mathbb{R}^d$, #CVXVertices is in $O(n)$

Total size of the CVX $\Theta\left(n^{\lfloor d/2 \rfloor}\right)$



Largest convex polygon in $[1..N]^2$ as at most

$$\frac{12}{(4\pi^2)^{1/3}} N^{2/3} + O(N^{1/3}\log(N))$$

vertices/edges

Further elements

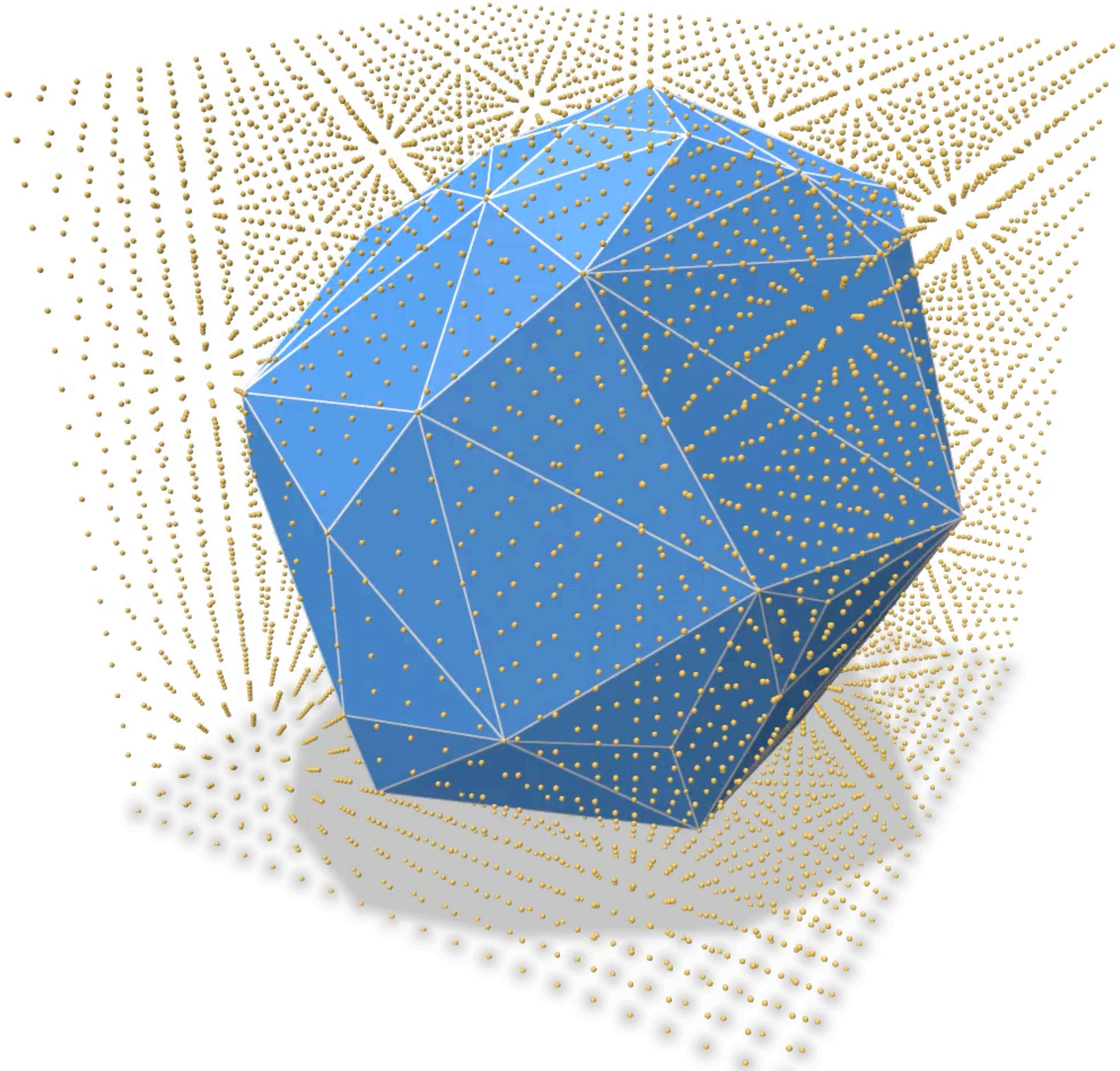
Let $P \subset \mathbb{Z}^d$ *a lattice polytope with non-empty interior, then:* $f_k \ll c_d (Vol\, P)^{\frac{d-1}{d+1}}$

Convex on the lattice $[1,n]^2$ *grid has* $O(n^{2/3})$ *edges*

Let $P \subset [1,U]^2$ *(with* $U \leq 2^m$*) and* $n := |P|$*, the expected time for Voronoi diagram / Delaunay triangulation is:*

$$O\left(\min\{n \log n, n\sqrt{U}\}\right)$$

⋮



Further elements

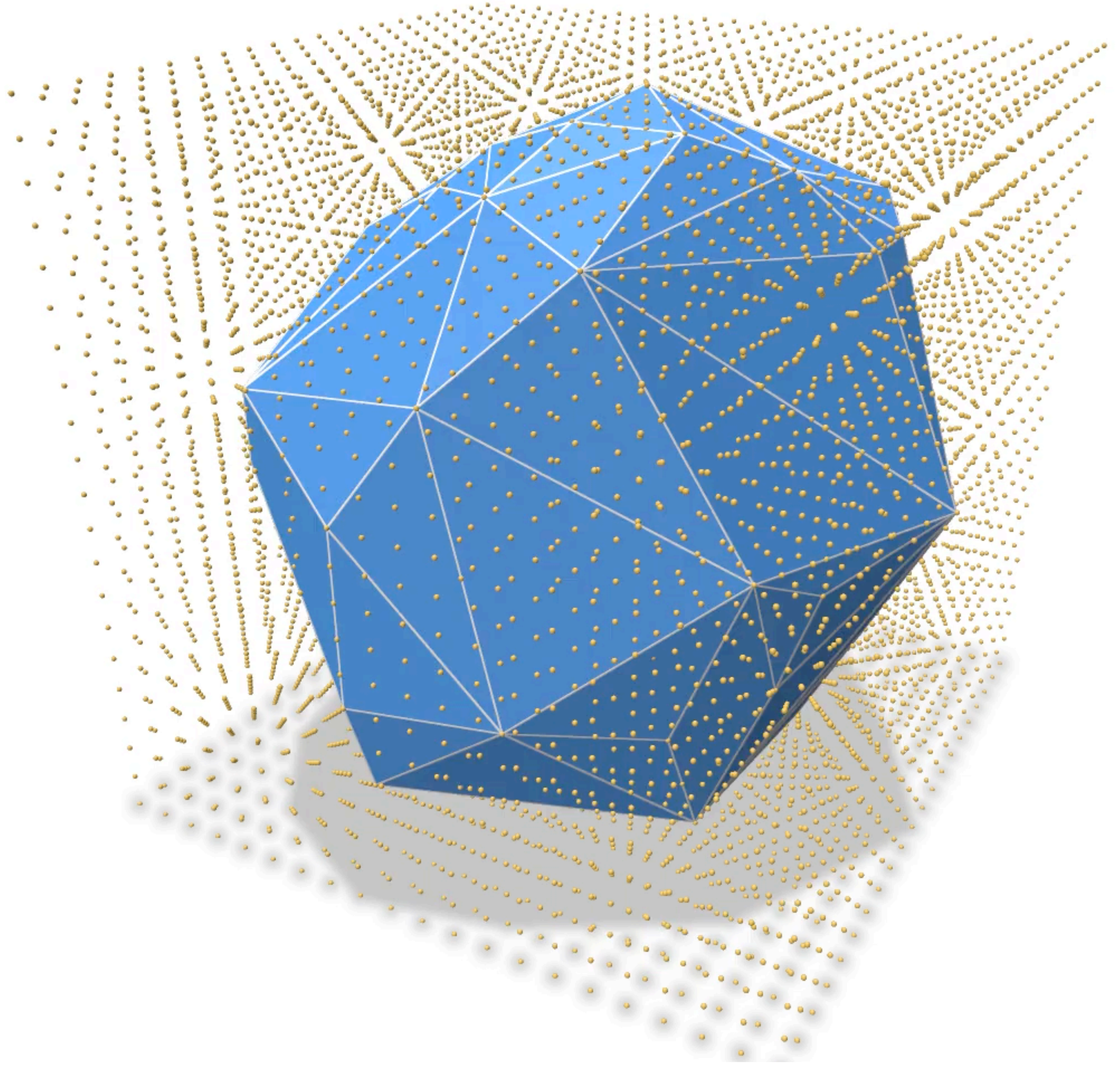
Let $P \subset \mathbb{Z}^d$ *a lattice polytope with non-empty interior, then:* $f_k \ll c_d (Vol\, P)^{\frac{d-1}{d+1}}$

Convex on the lattice $[1,n]^2$ *grid has* $O(n^{2/3})$ *edges*

Let $P \subset [1,U]^2$ *(with* $U \leq 2^m$*) and* $n := |P|$*, the expected time for Voronoi diagram / Delaunay triangulation is:*

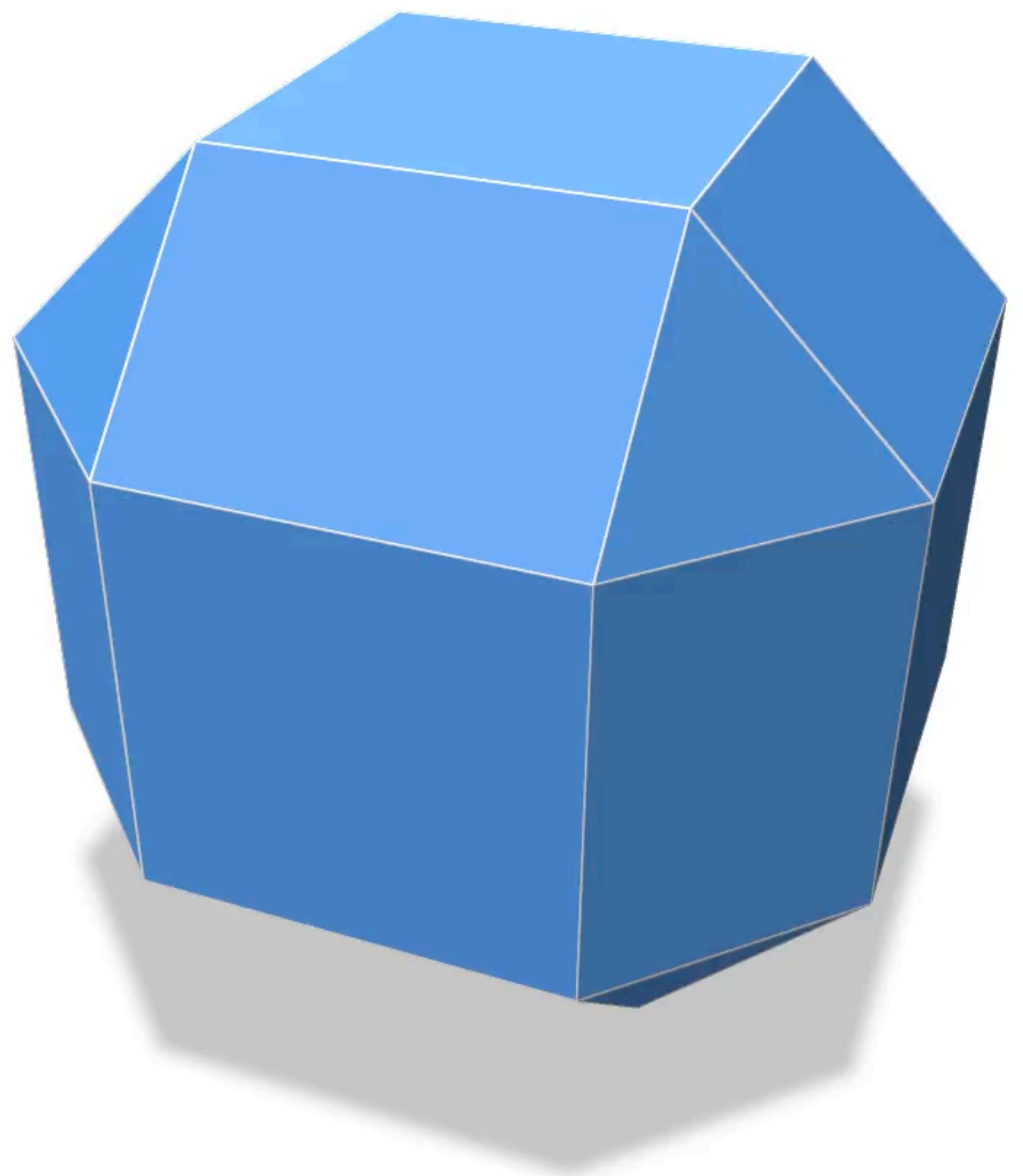
$$O\left(\min\{n \log n, n\sqrt{U}\}\right)$$

⋮

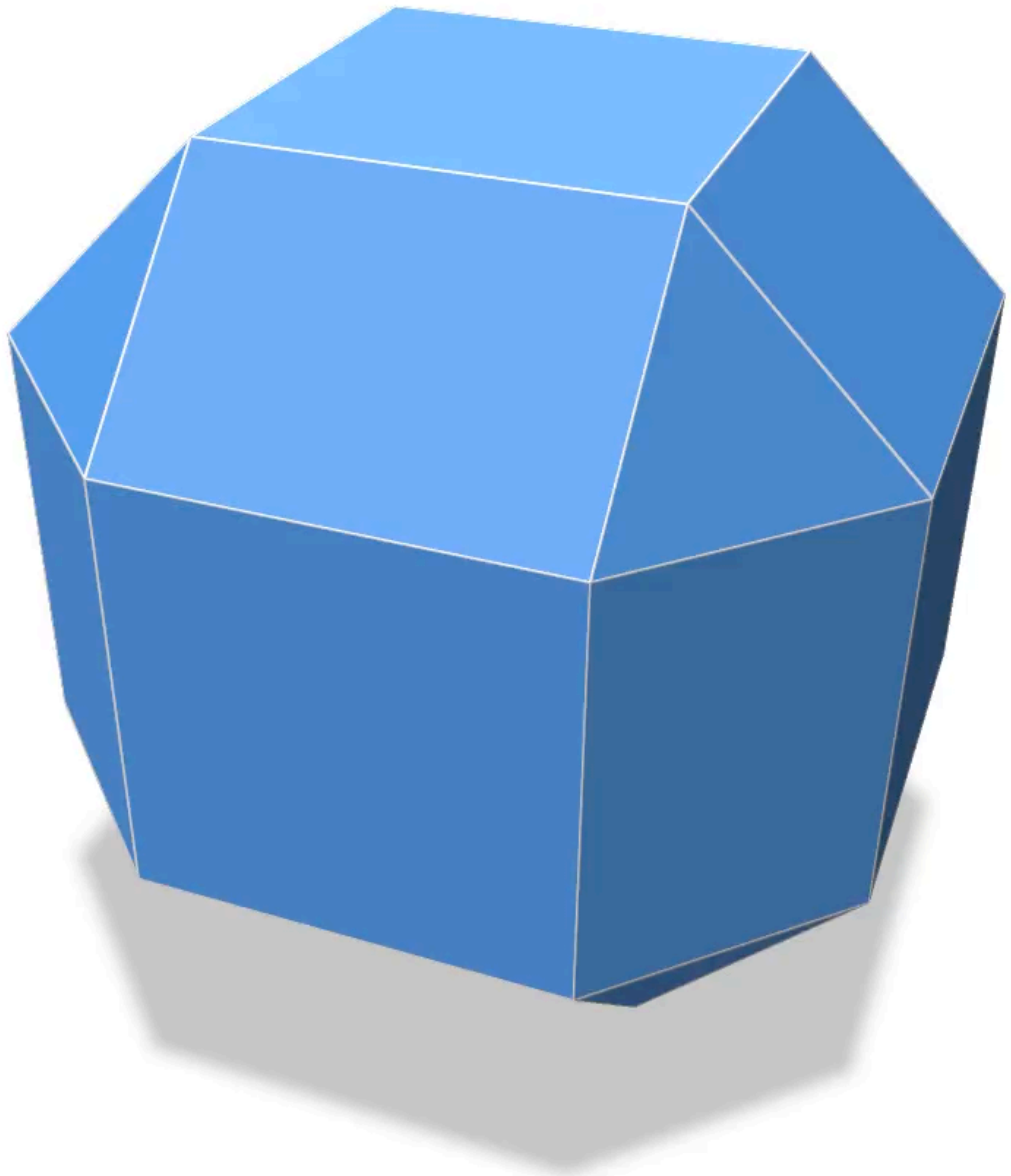


hands on…

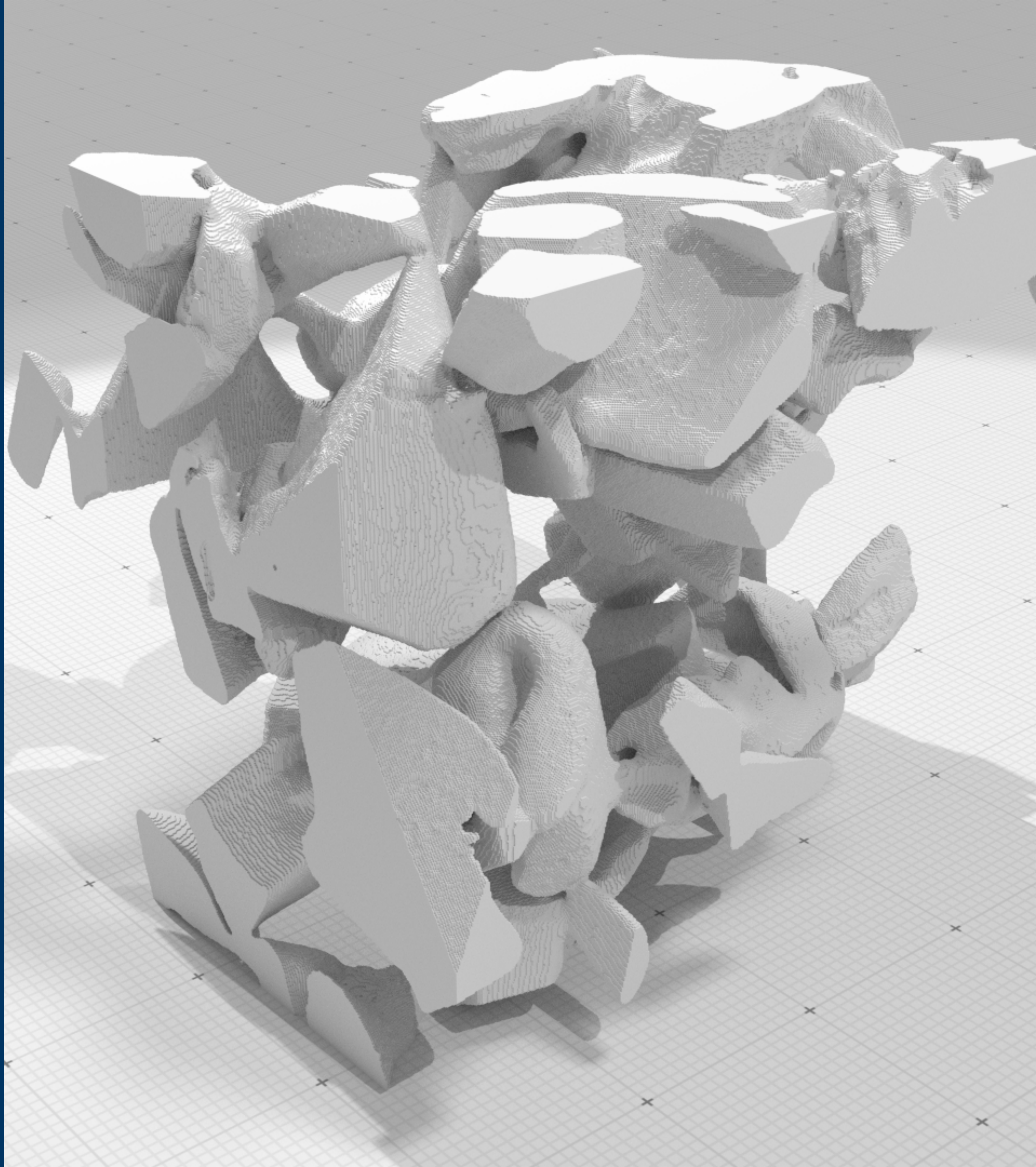
```cpp
void oneStep(double myh)
{
  auto params = SH3::defaultParameters();
  params( "polynomial", "sphere1" )( "gridstep", myh )
        ( "minAABB", -1.25 )( "maxAABB", 1.25 );
  auto implicit_shape  = SH3::makeImplicitShape3D  ( params );
  auto digitized_shape = SH3::makeDigitizedImplicitShape3D( implicit_shape, params );

  std::vector<Point> points;
  std::cout << "Digitzing shape" << std::endl;
  auto domain = digitized_shape->getDomain();
  for(auto &p: domain)
    if (digitized_shape->operator()(p))
      points.push_back(p);

  std::cout << "Computing convex hull" << std::endl;
  QuickHull3D hull;
  hull.setInput( points );
  hull.computeConvexHull();
  std::cout << "#points="    << hull.nbPoints()
            << " #vertices=" << hull.nbVertices()
            << " #facets="   << hull.nbFacets() << std::endl;
  std::vector< RealPoint > vertices;
  hull.getVertexPositions( vertices );
  std::vector< std::vector< std::size_t > > facets;
  hull.getFacetVertices( facets );

  polyscope::registerSurfaceMesh("Convex hull", vertices, facets)->rescaleToUnit();
}
```



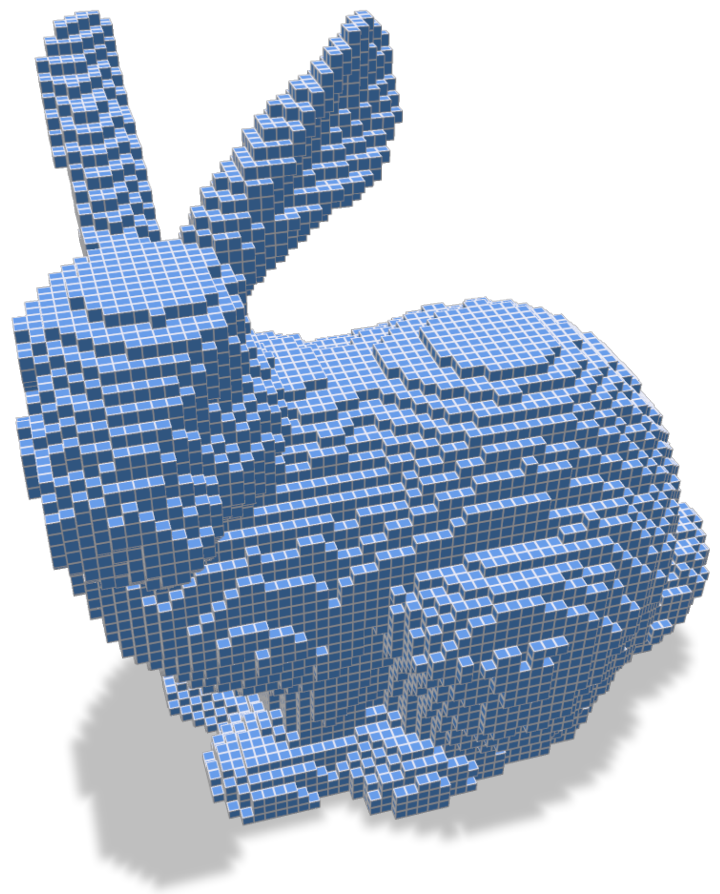
```
void oneStep(double myh)
{
  auto params = SH3::defaultParameters();
  params( "polynomial", "sphere1" )( "gridstep", myh )
        ( "minAABB", -1.25 )( "maxAABB", 1.25 );
  auto implicit_shape  = SH3::makeImplicitShape3D  ( params );
  auto digitized_shape = SH3::makeDigitizedImplicitShape3D( implicit_shape, params );

  std::vector<Point> points;
  std::cout << "Digitzing shape" << std::endl;
  auto domain = digitized_shape->getDomain();
  for(auto &p: domain)
    if (digitized_shape->operator()(p))
      points.push_back(p);

  std::cout << "Computing convex hull" << std::endl;
  QuickHull3D hull;
  hull.setInput( points );
  hull.computeConvexHull();
  std::cout << "#points="    << hull.nbPoints()
            << " #vertices=" << hull.nbVertices()
            << " #facets="   << hull.nbFacets() << std::endl;
  std::vector< RealPoint > vertices;
  hull.getVertexPositions( vertices );
  std::vector< std::vector< std::size_t > > facets;
  hull.getFacetVertices( facets );

  polyscope::registerSurfaceMesh("Convex hull", vertices, facets)->rescaleToUnit();
}
```



$\mathbb{Z}^d$

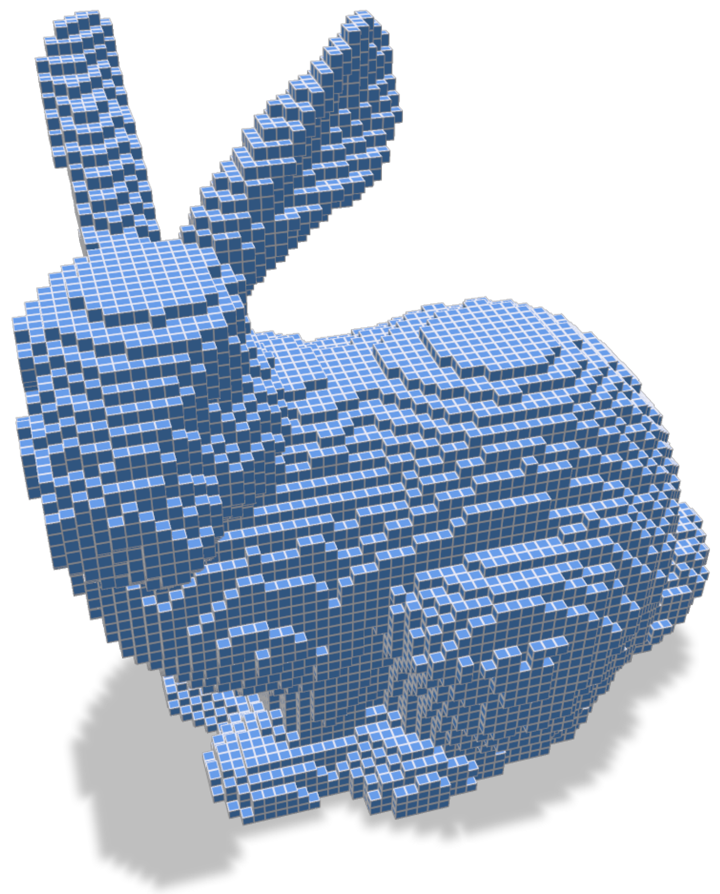


Volumetric analysis



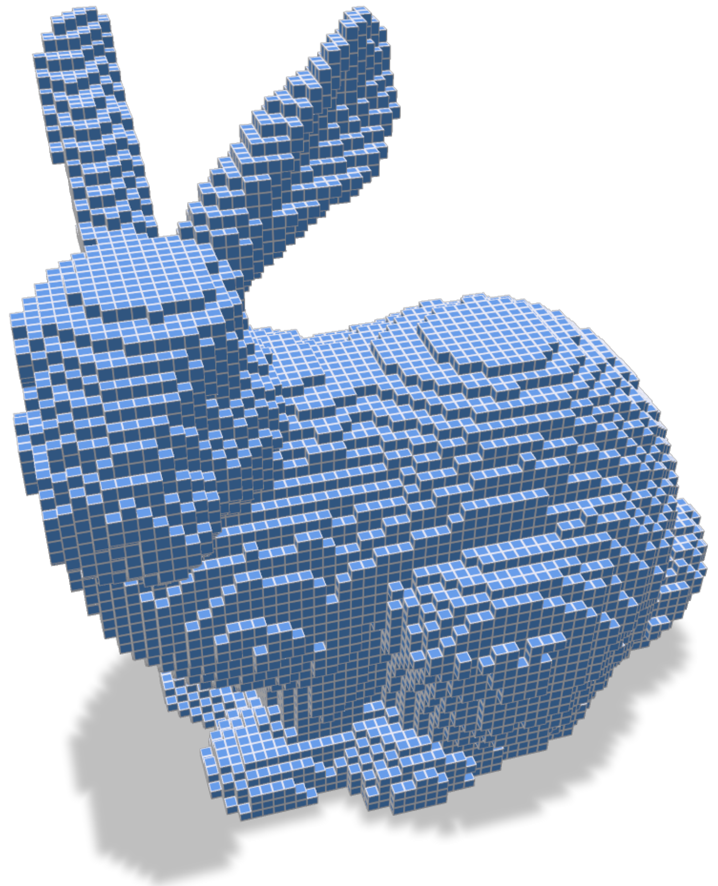
Given $X \subset \mathbb{Z}^d$ and a domain $[0,n]^d$, compute:

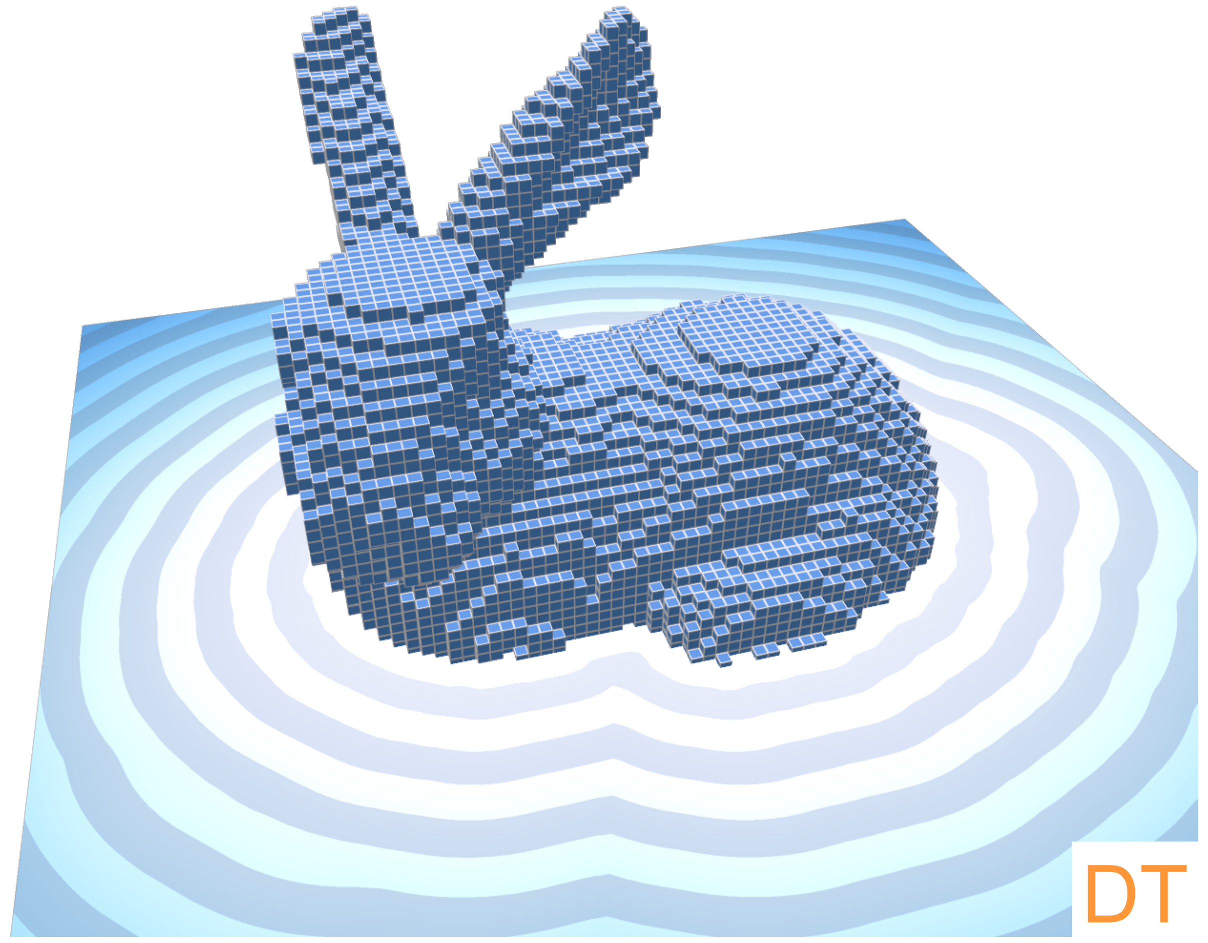
Volumetric analysis



Given $X \subset \mathbb{Z}^d$ and a domain $[0,n]^d$, compute:

Volumetric analysis



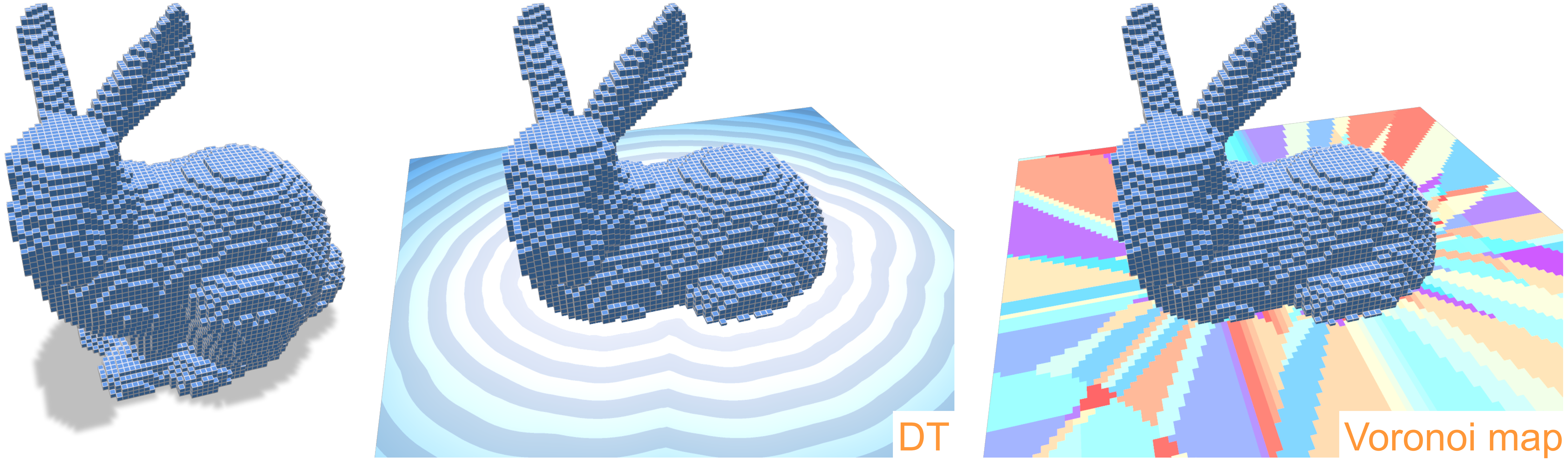


Given $X \subset \mathbb{Z}^d$ ***and a domain*** $[0,n]^d$***, compute:***

$$DT(x) = min_{y \in D \setminus X}\ d(x,y)$$

(aka distance map)

Volumetric analysis

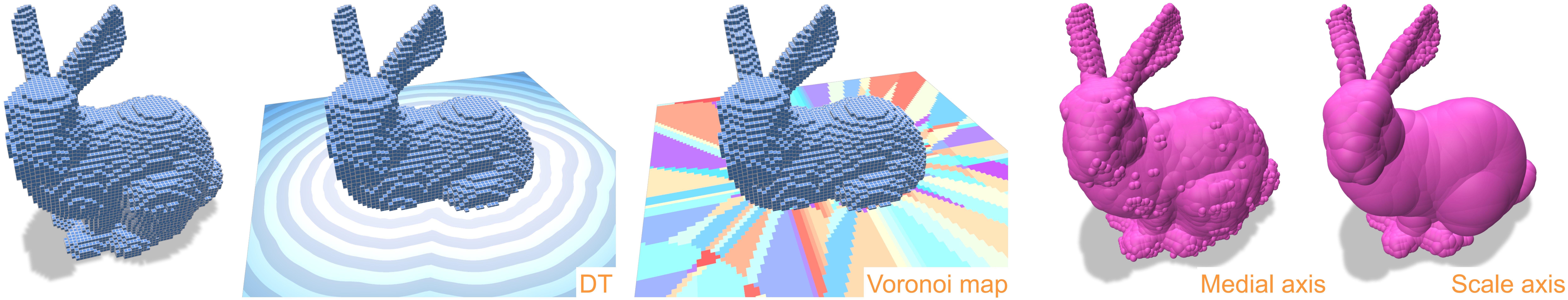


Given $X \subset \mathbb{Z}^d$ and a domain $[0,n]^d$, compute:

$$DT(x) = min_{y \in D \setminus X} \; d(x,y)$$ *(aka distance map)*

$$\sigma(x) = \text{argmin}_{y \in D \setminus X} \; d(x,y)$$ *(aka Voronoi map $\mathcal{V}(X) \cap \mathbb{Z}^d$)*

Volumetric analysis



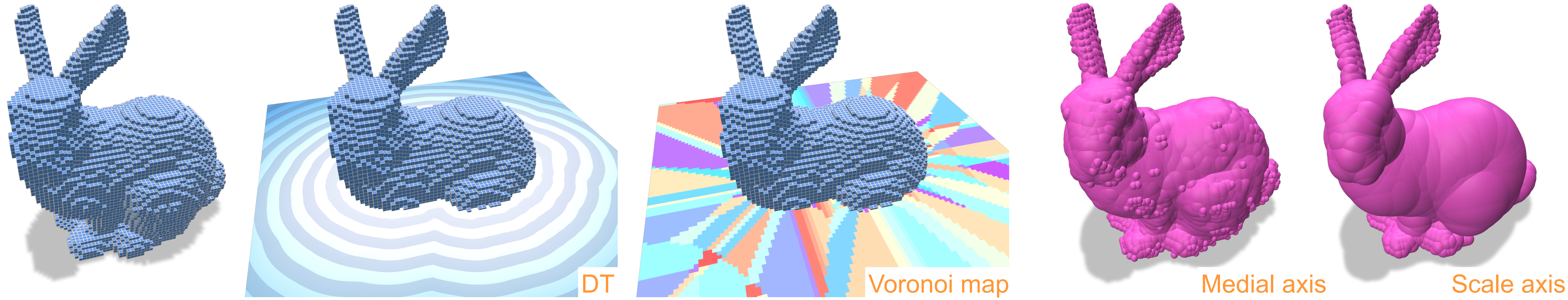
Given $X \subset \mathbb{Z}^d$ ***and a domain*** $[0,n]^d$***, compute:***

$$DT(x) = min_{y \in D \setminus X} \ d(x,y)$$ *(aka distance map)*

$$\sigma(x) = \text{argmin}_{y \in D \setminus X} \ d(x,y)$$ *(aka Voronoi map* $\mathcal{V}(X) \cap \mathbb{Z}^d$*)*

$$M = \{(x,r) \in \mathbb{Z}^{d+1} \mid \mathcal{B}(x,r) \cap \mathbb{Z}^d \subset X, \text{there is no } (x',r') \text{ s.t. } \mathcal{B}(x,r) \subset \mathcal{B}(x',r')\}$$ *(aka discrete medial axis)*

Volumetric analysis



Given $X \subset \mathbb{Z}^d$ ***and a domain*** $[0,n]^d$***, compute:***

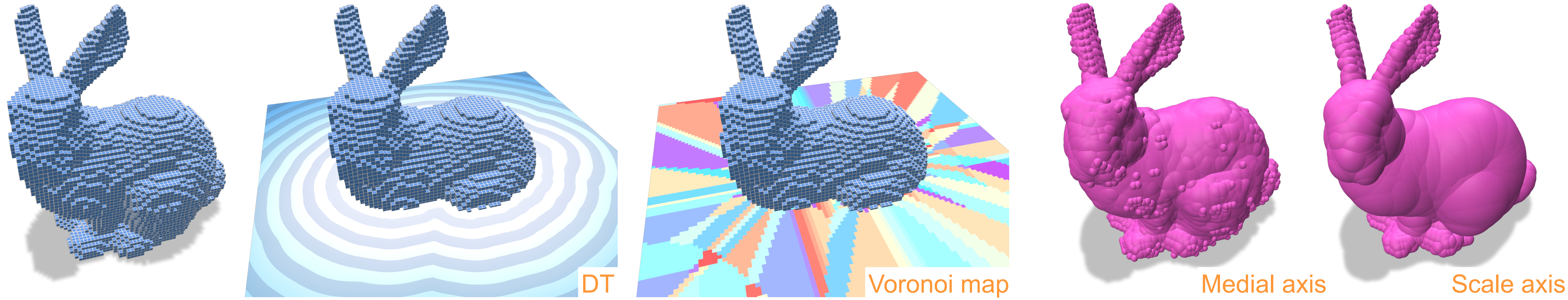
$$DT(x) = min_{y \in D \setminus X} \; d(x,y)$$ *(aka distance map)*

$$\sigma(x) = \text{argmin}_{y \in D \setminus X} \; d(x,y)$$ *(aka Voronoi map* $\mathcal{V}(X) \cap \mathbb{Z}^d$*)*

$$M = \{(x,r) \in \mathbb{Z}^{d+1} \mid \mathcal{B}(x,r) \cap \mathbb{Z}^d \subset X, \text{there is no } (x',r') \text{ s.t. } \mathcal{B}(x,r) \subset \mathcal{B}(x',r')\}$$ *(aka discrete medial axis)*

$$\pi(x) = \text{argmin}_{(y,r) \in M} \; \|x-y\|_2^2 - r^2$$ *(aka* l_2 *Power map* $\mathcal{P}(M) \cap \mathbb{Z}^d$*)*

Volumetric analysis



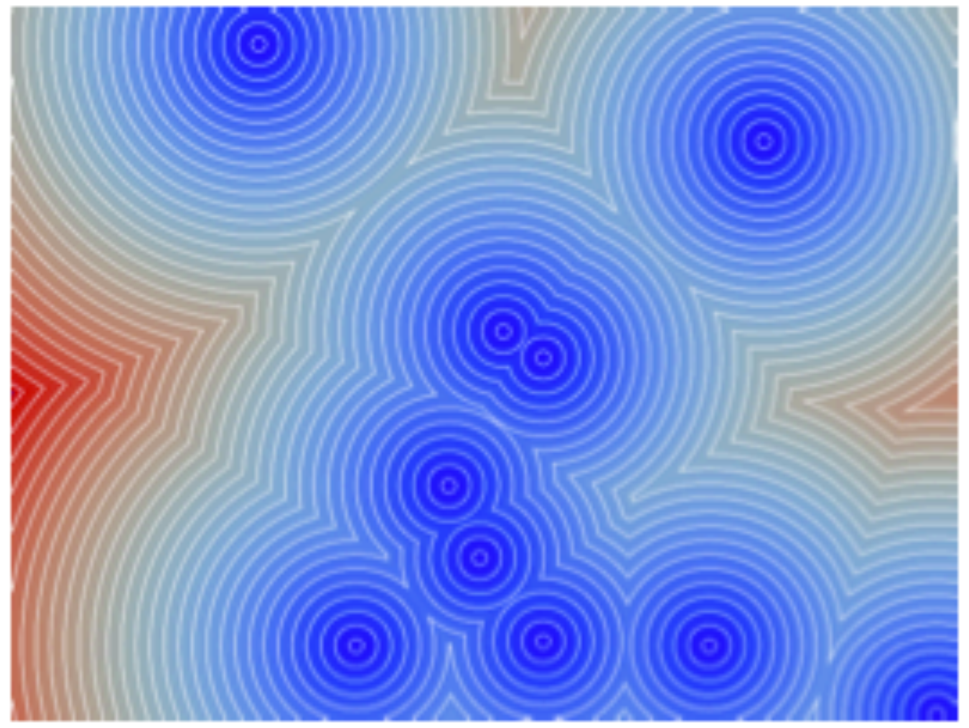
Given $X \subset \mathbb{Z}^d$ ***and a domain*** $[0,n]^d$***, compute:***

- $DT(x) = min_{y \in D \setminus X} \; d(x, y)$ *(aka distance map)*
- $\sigma(x) = \text{argmin}_{y \in D \setminus X} \; d(x, y)$ *(aka Voronoi map* $\mathcal{V}(X) \cap \mathbb{Z}^d$*)*

$M = \{(x, r) \in \mathbb{Z}^{d+1} \mid \mathcal{B}(x, r) \cap \mathbb{Z}^d \subset X$, *there is no* (x', r') *s.t.* $\mathcal{B}(x, r) \subset \mathcal{B}(x', r')\}$ *(aka discrete medial axis)*

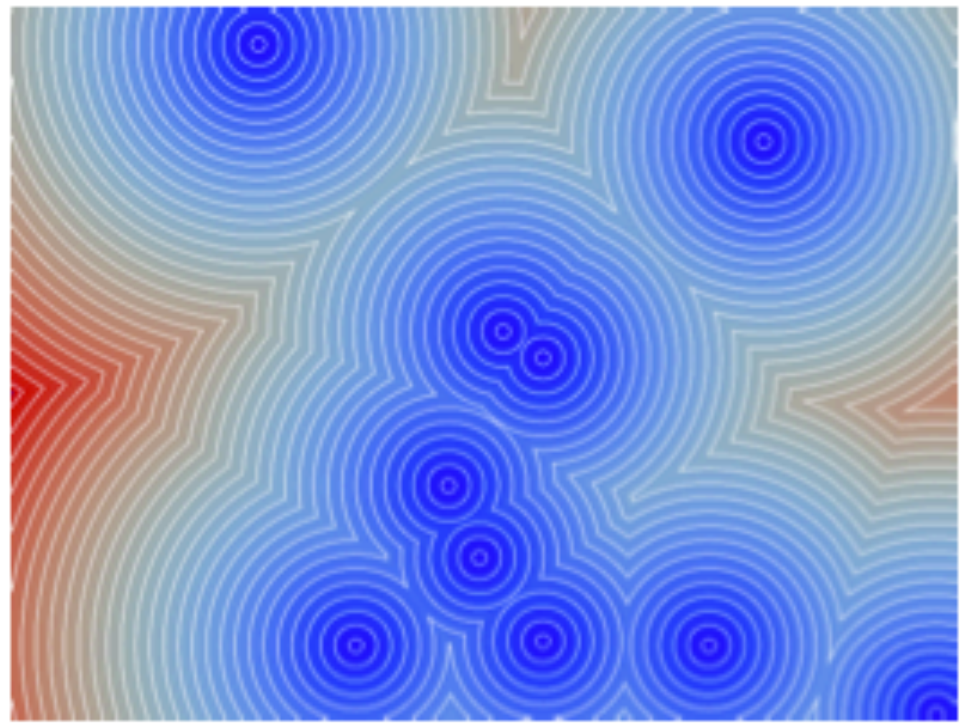
$\pi(x) = \text{argmin}_{(y,r) \in M} \; \|x - y\|_2^2 - r^2$ *(aka* l_2 *Power map* $\mathcal{P}(M) \cap \mathbb{Z}^d$*)*

Separable distance field



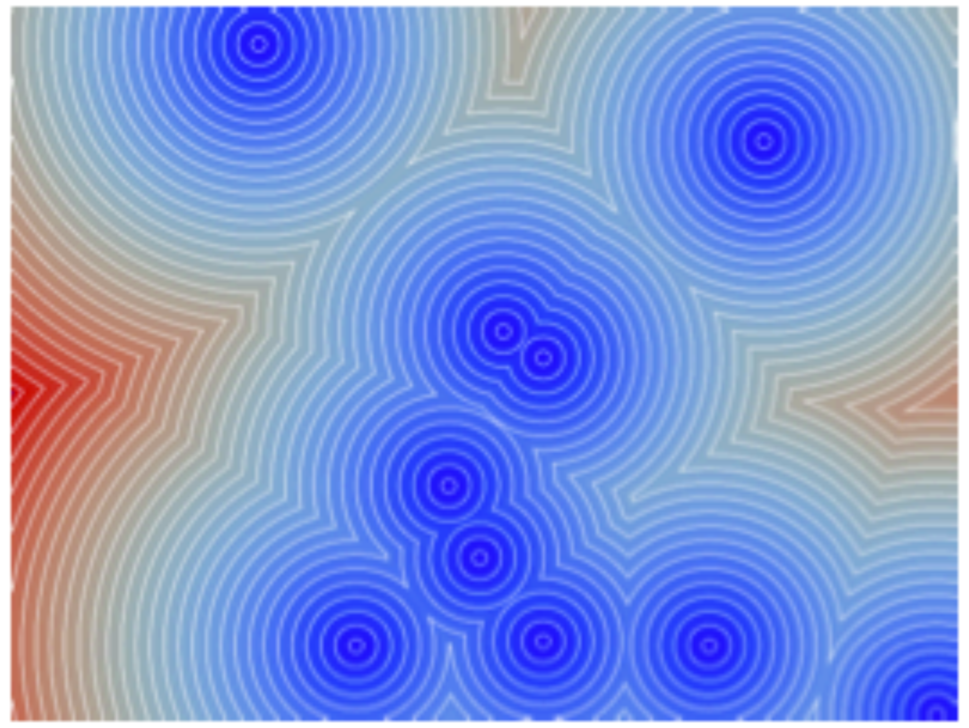
$$DT(x) = \min_{y \in D \setminus X} ||x - y||_2$$

Separable distance field



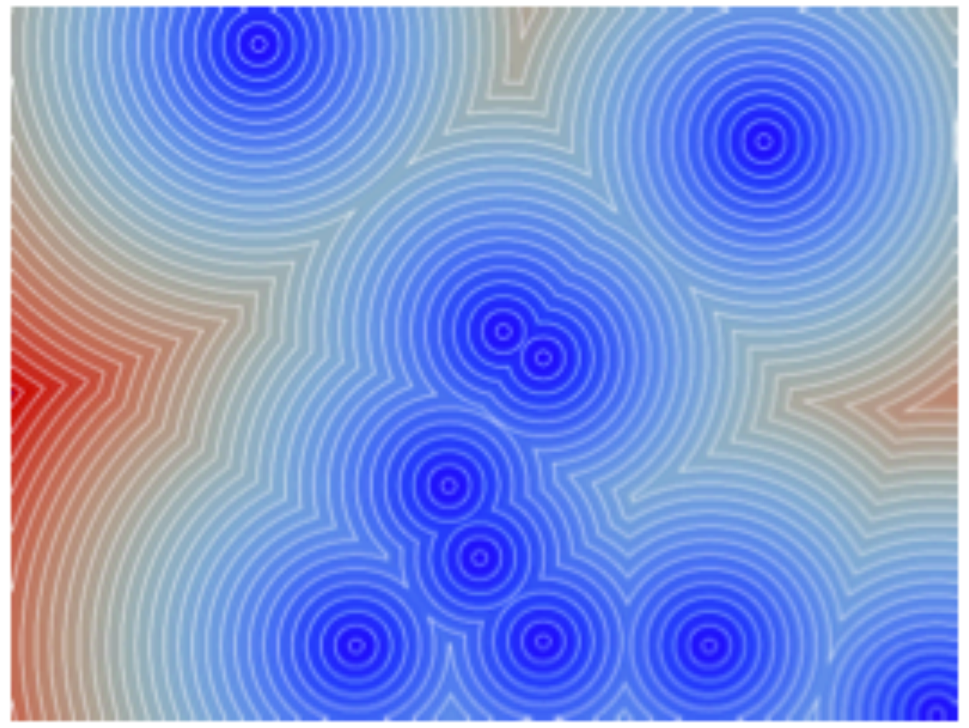
$$
\begin{aligned}
DT(x) &= \min_{y \in D \setminus X} ||x - y||_2 \\
&= \min_{(u,v) \not\subset X} (i-u)^2 + (j-v)^2
\end{aligned}
$$

Separable distance field



$$
\begin{aligned}
DT(x) &= \min_{y \in D \setminus X} ||x - y||_2 \\
&= \min_{(u,v) \not\subset X} (i-u)^2 + (j-v)^2 \\
&= \min_{v} \left((\min_{u} (i-u)^2) + (j-v)^2 \right)
\end{aligned}
$$

Separable distance field



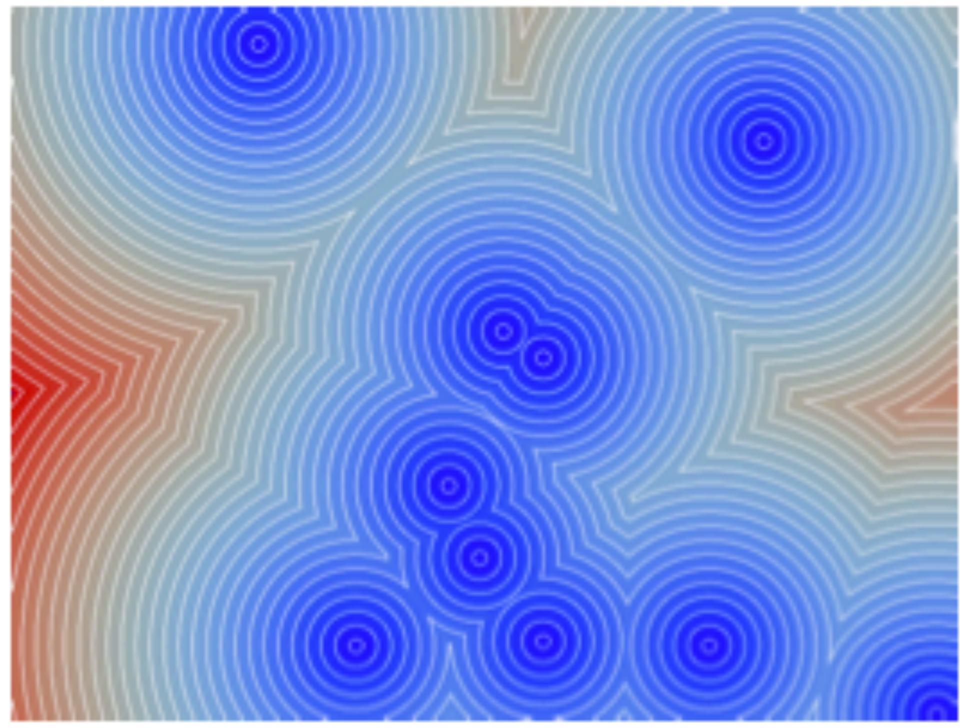
$$DT(x) = \min_{y \in D \setminus X} ||x - y||_2$$

$$= \min_{(u,v) \not\subset X} (i-u)^2 + (j-v)^2$$

$$= \min_{v} \left(\left(\min_{u} (i-u)^2 \right) + (j-v)^2 \right)$$

per line double-scan = $O(n)$

Separable distance field



$$DT(x) = \min_{y \in D \setminus X} ||x - y||_2$$

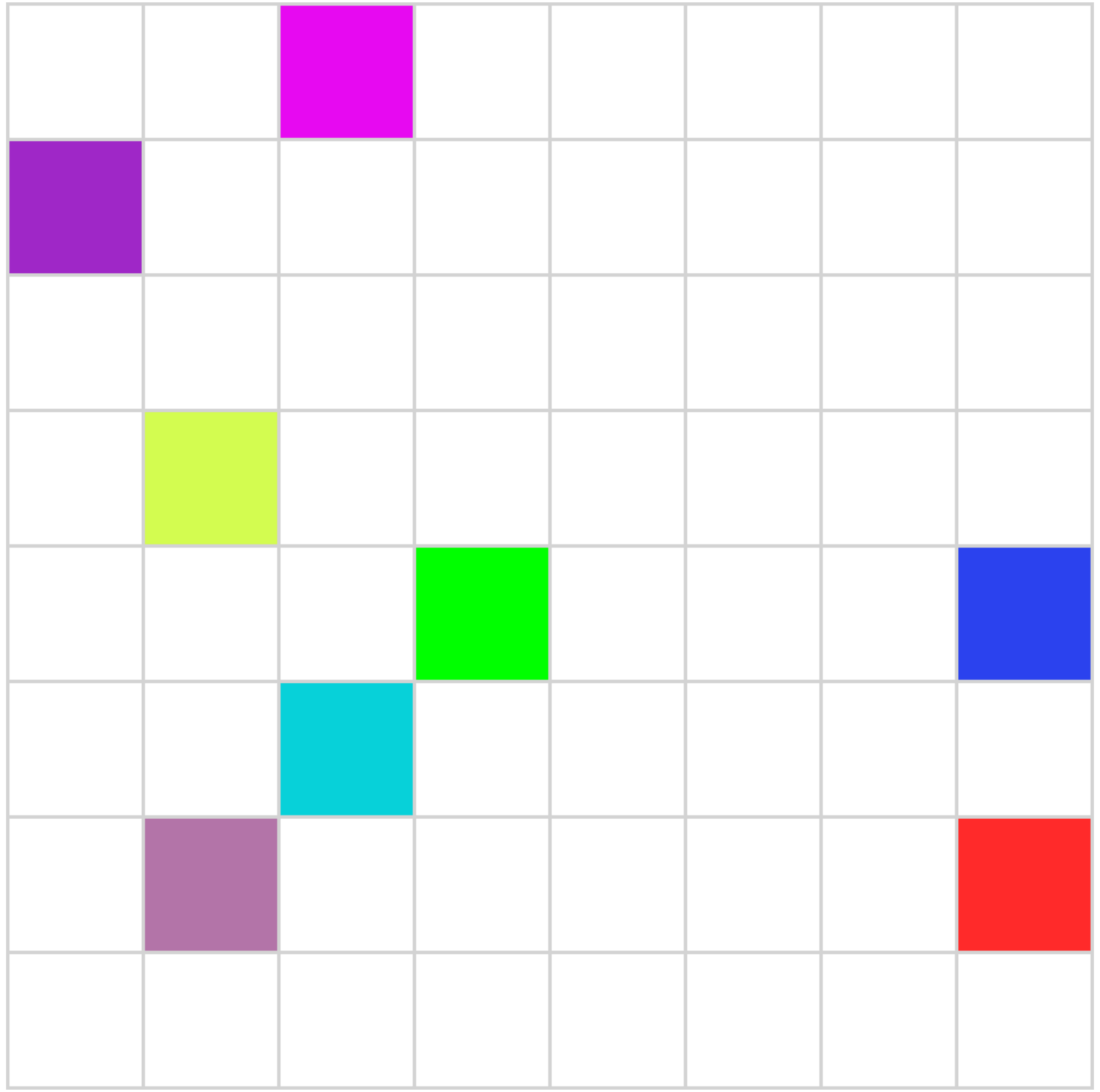
$$= \min_{(u,v) \not\subset X} (i-u)^2 + (j-v)^2$$

$$= \min_v \left(\left(\min_u (i-u)^2\right) + (j-v)^2 \right)$$

per line double-scan = $O(n)$

1D lower enveloppe computation of a set of parabolas = $O(n)$

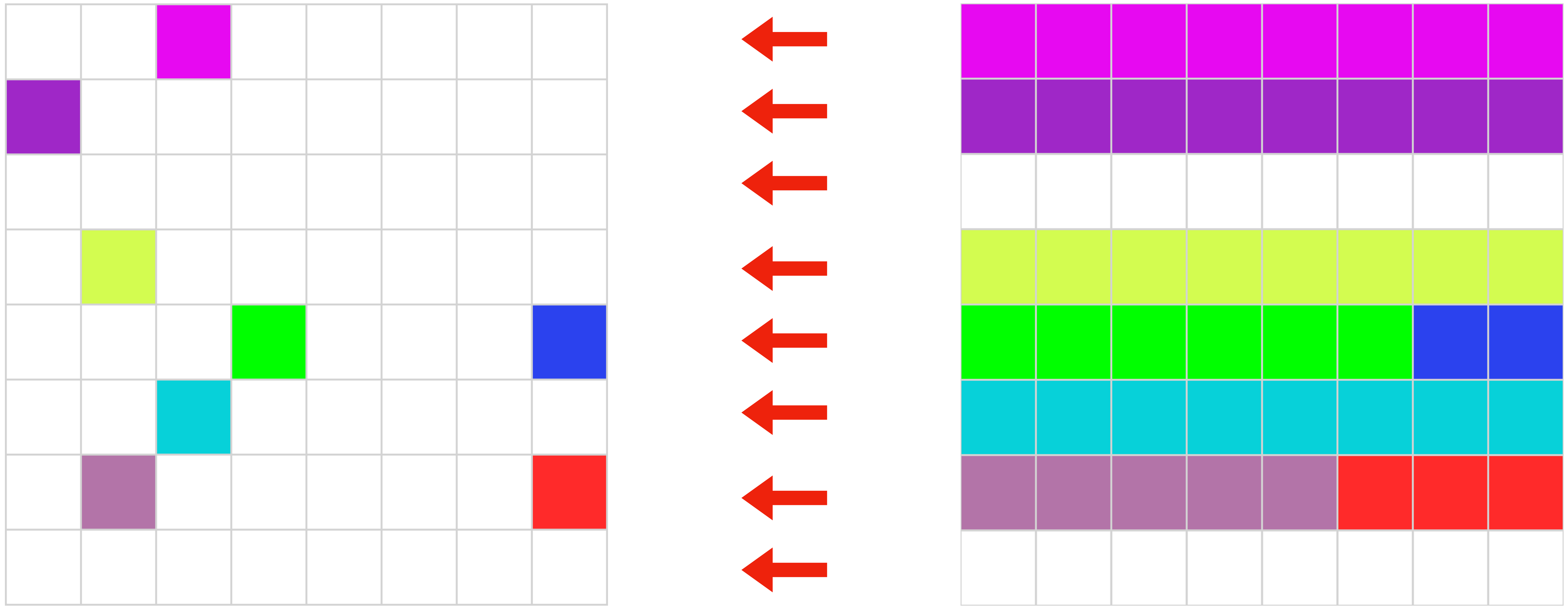
Separable Voronoi map: step 1



Separable Voronoi map: step 1

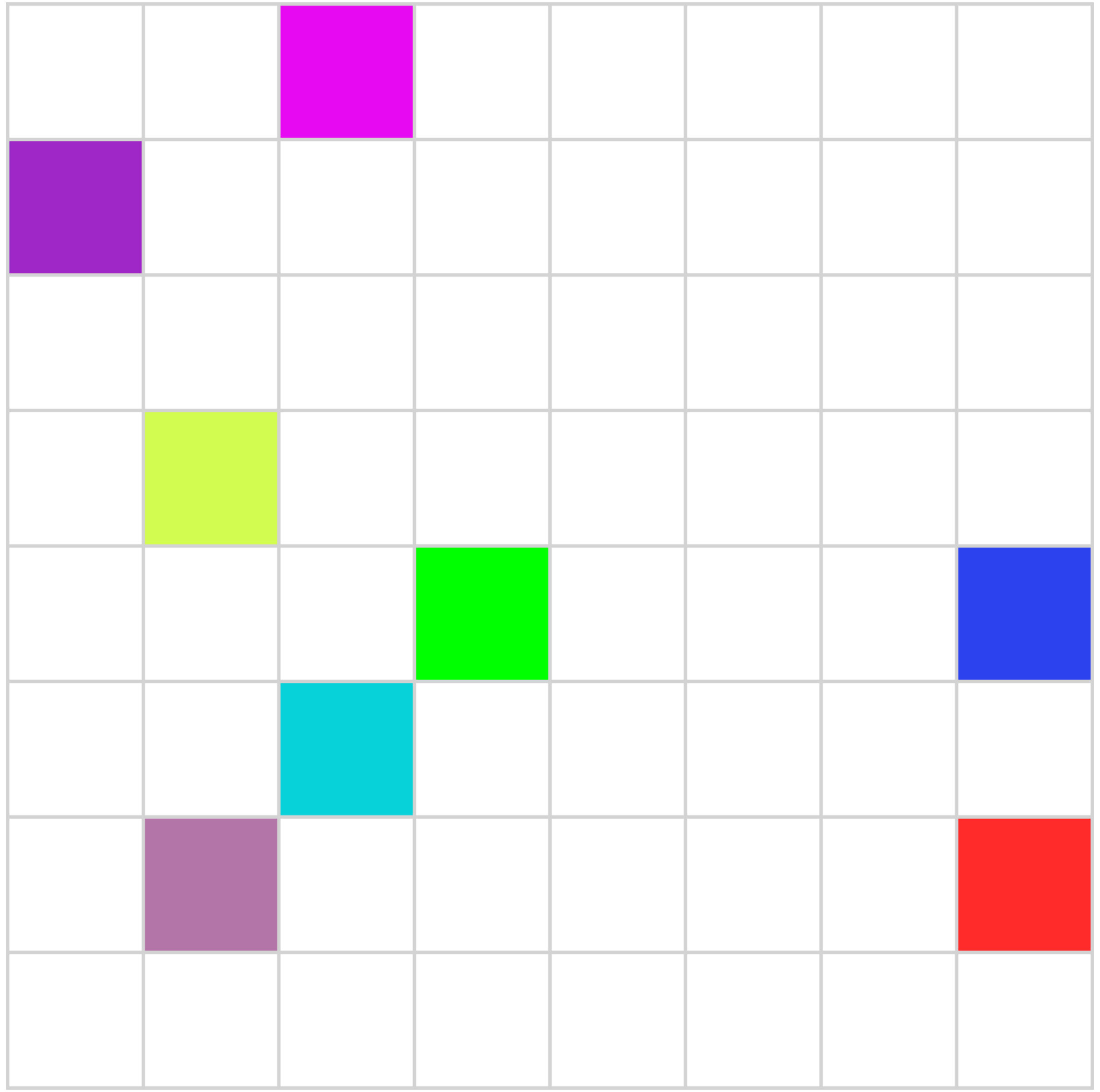
Separable Voronoi map: step 1

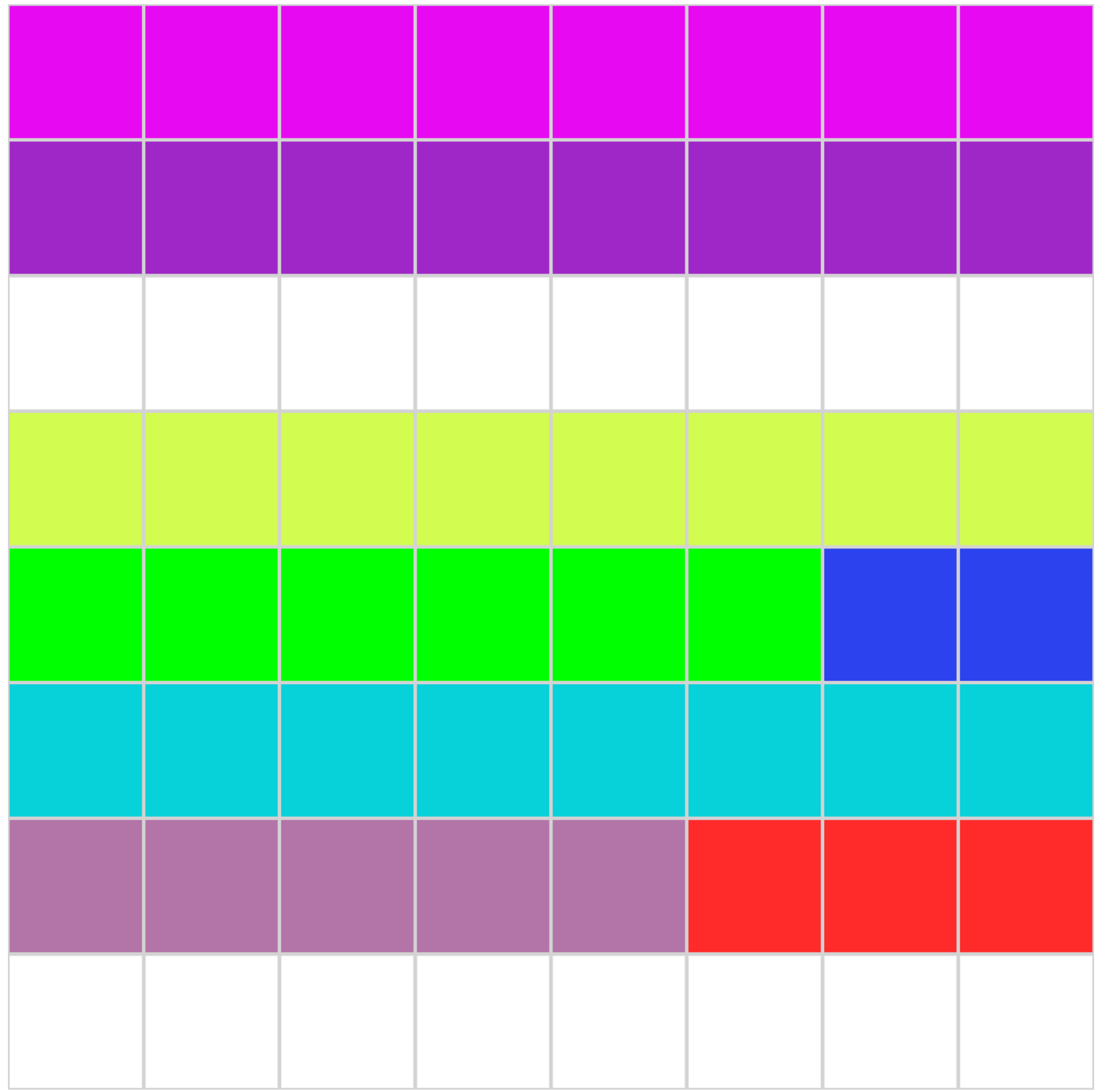
Separable Voronoi map: step 1



$\Rightarrow O(n)$ per row

Separable Voronoi map: step 1

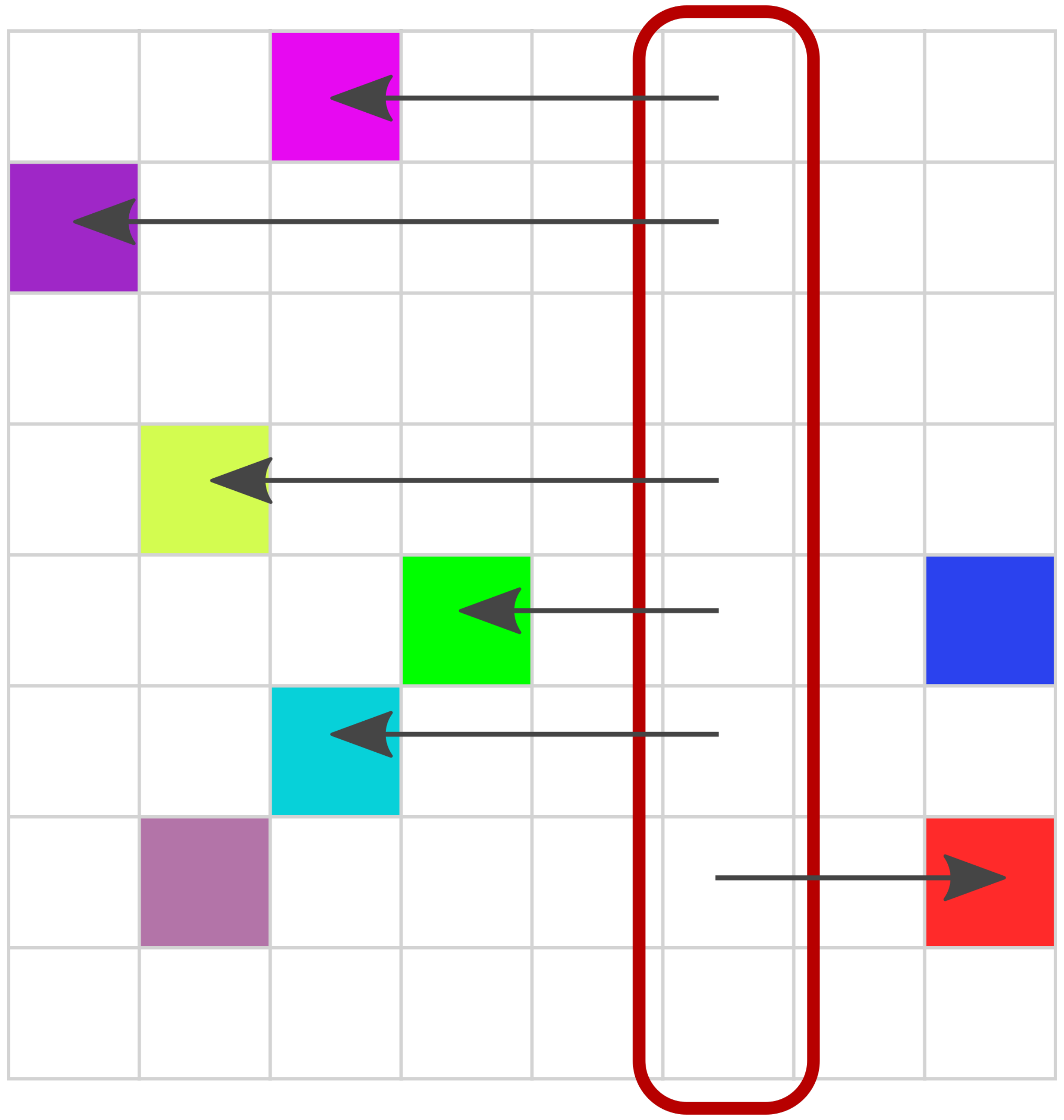




$\Rightarrow O(n)$ per row

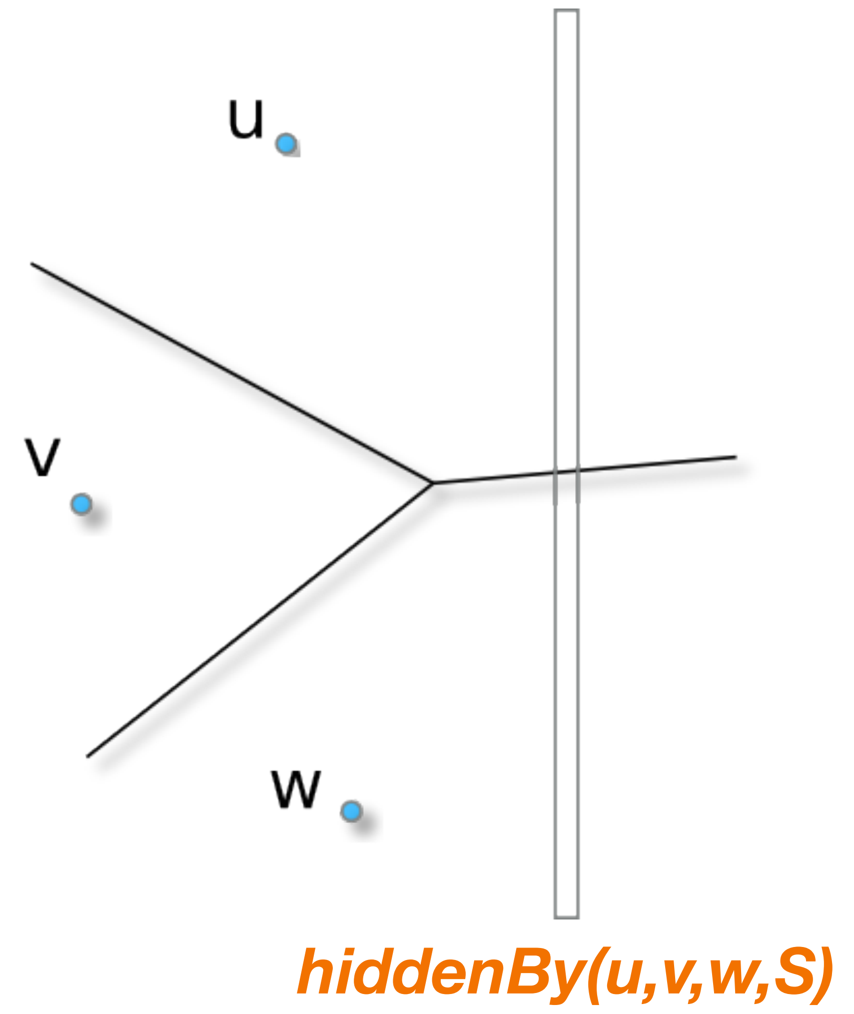
Separable Voronoi map: step 1

Separable Voronoi map: step 1

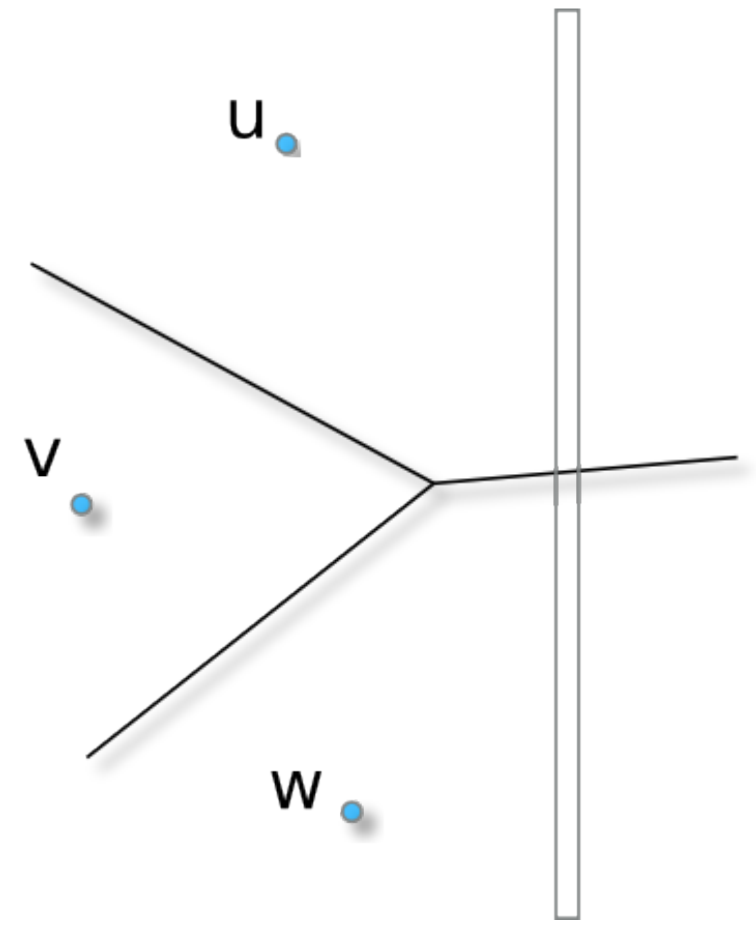


Separable Voronoi map: step 2

Stack based algorithm using a 3-ary *hiddenBy* predicate, *à la* sweep line ⇒ $O(n)$ per column

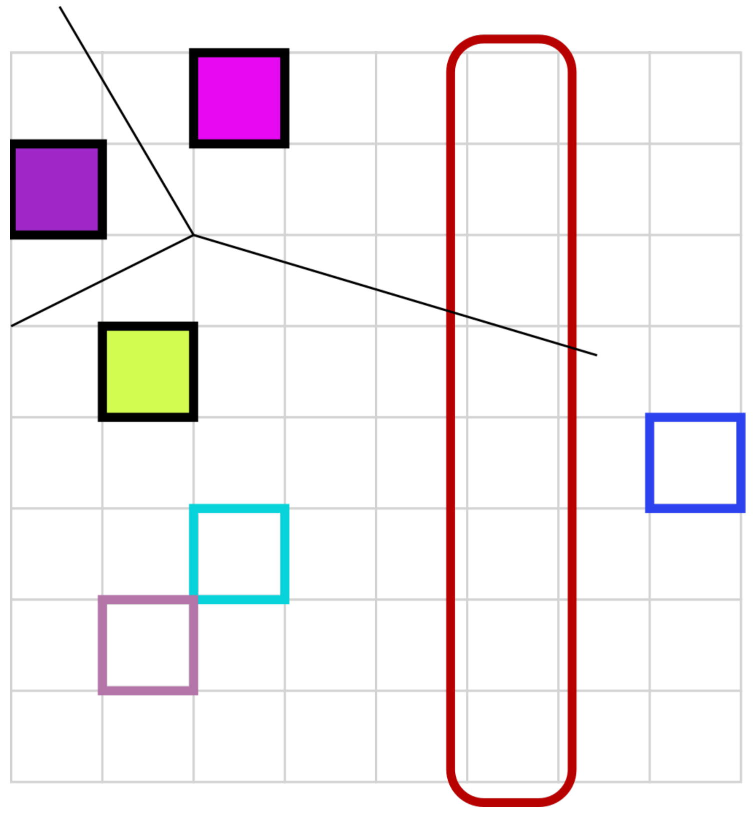


Separable Voronoi map: step 2

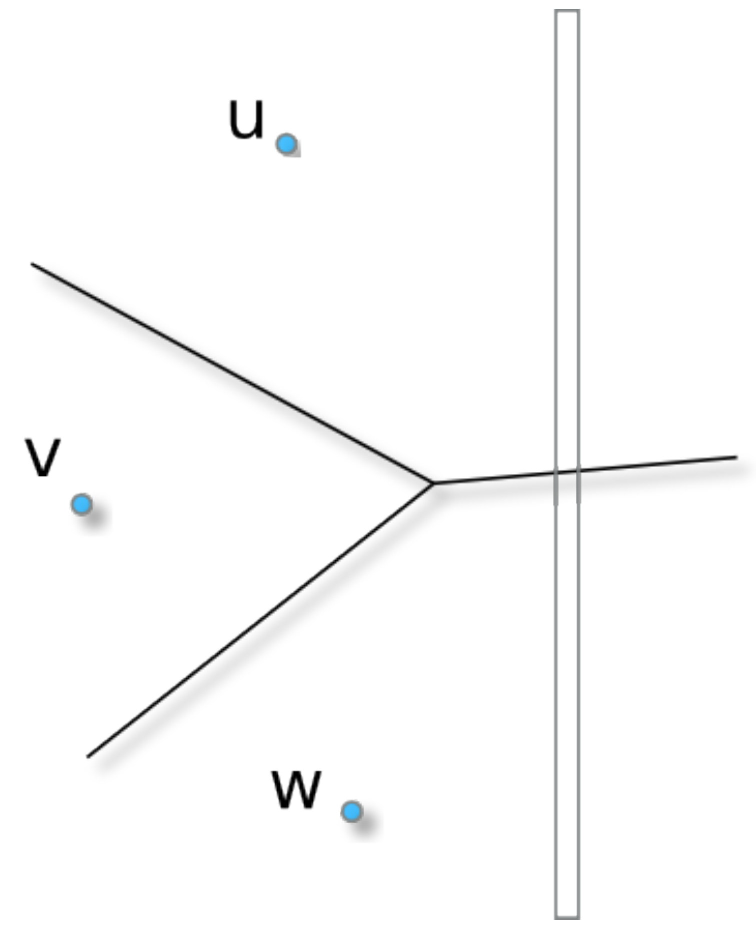


hiddenBy(u,v,w,S)

Stack based algorithm using a 3-ary *hiddenBy* predicate, *à la* sweep line ⇒ $O(n)$ per column

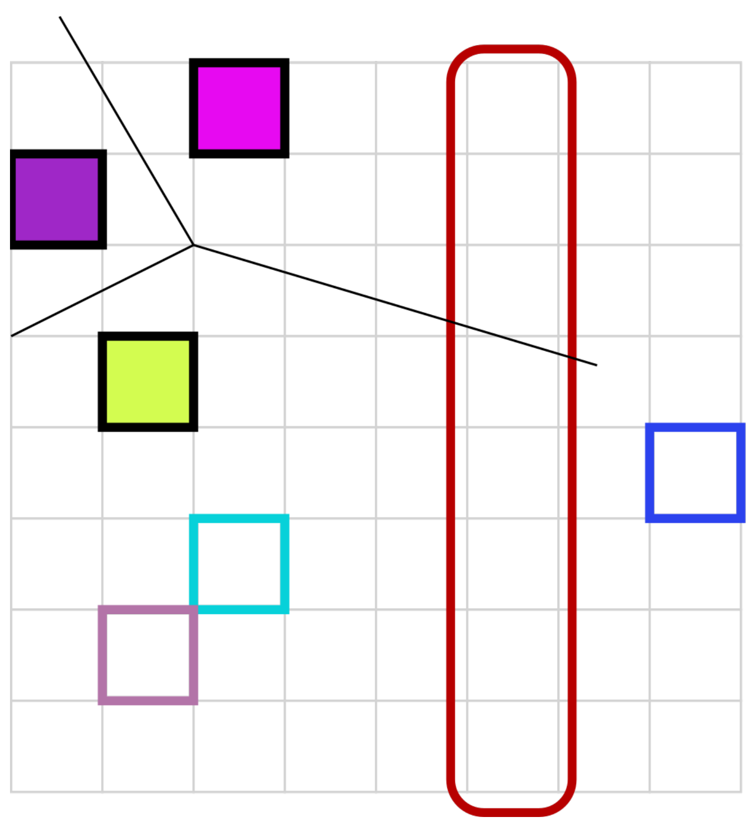


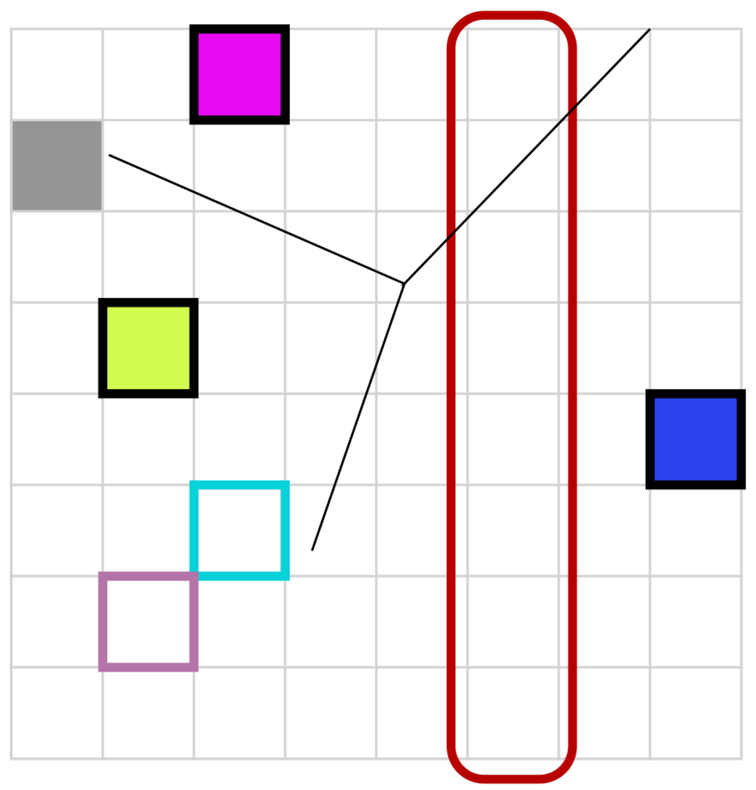
Separable Voronoi map: step 2



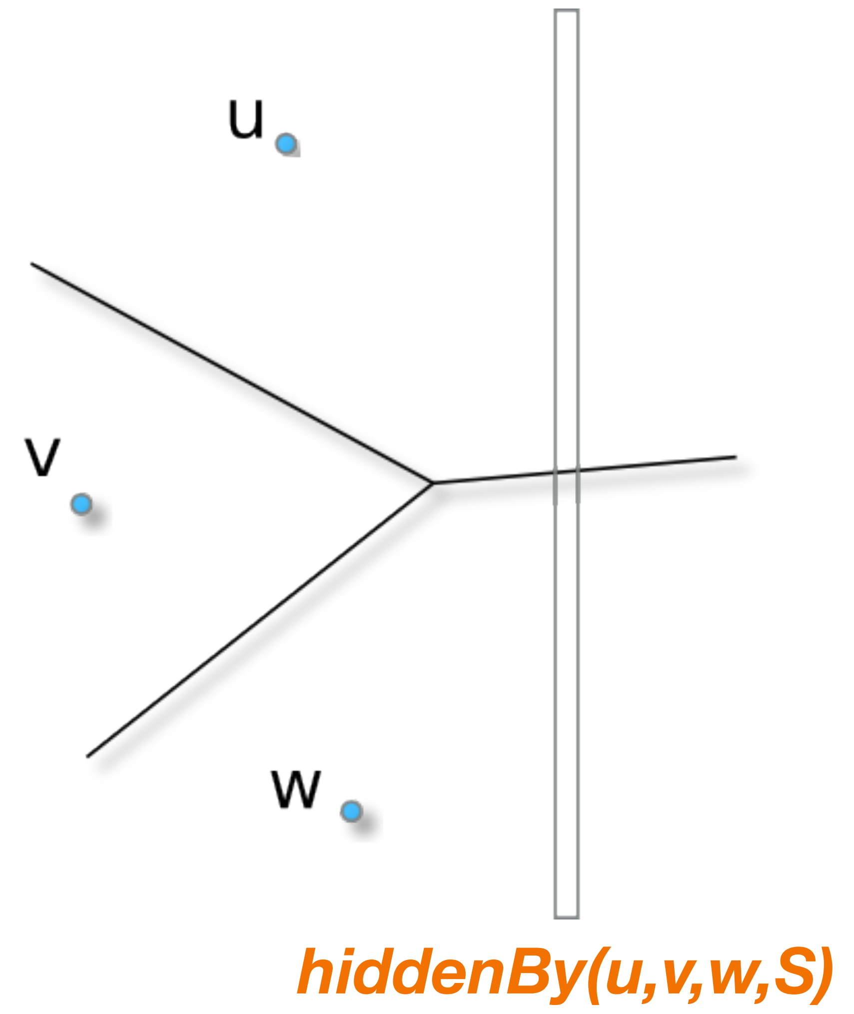
hiddenBy(u,v,w,S)

Stack based algorithm using a 3-ary *hiddenBy* predicate, *à la* sweep line ⇒ $O(n)$ per column

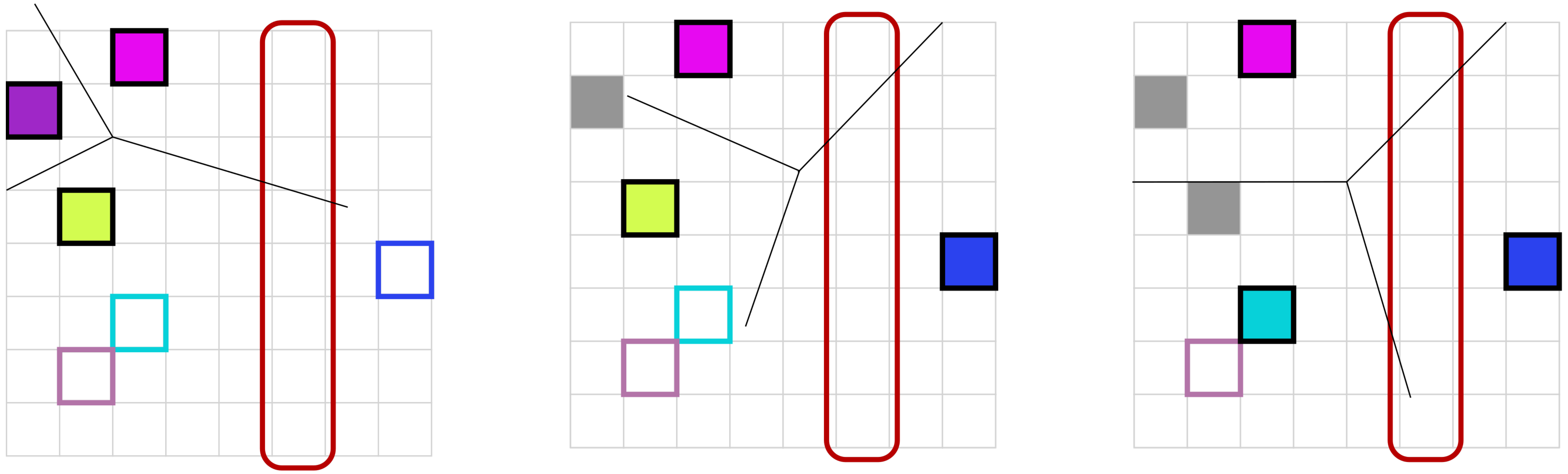




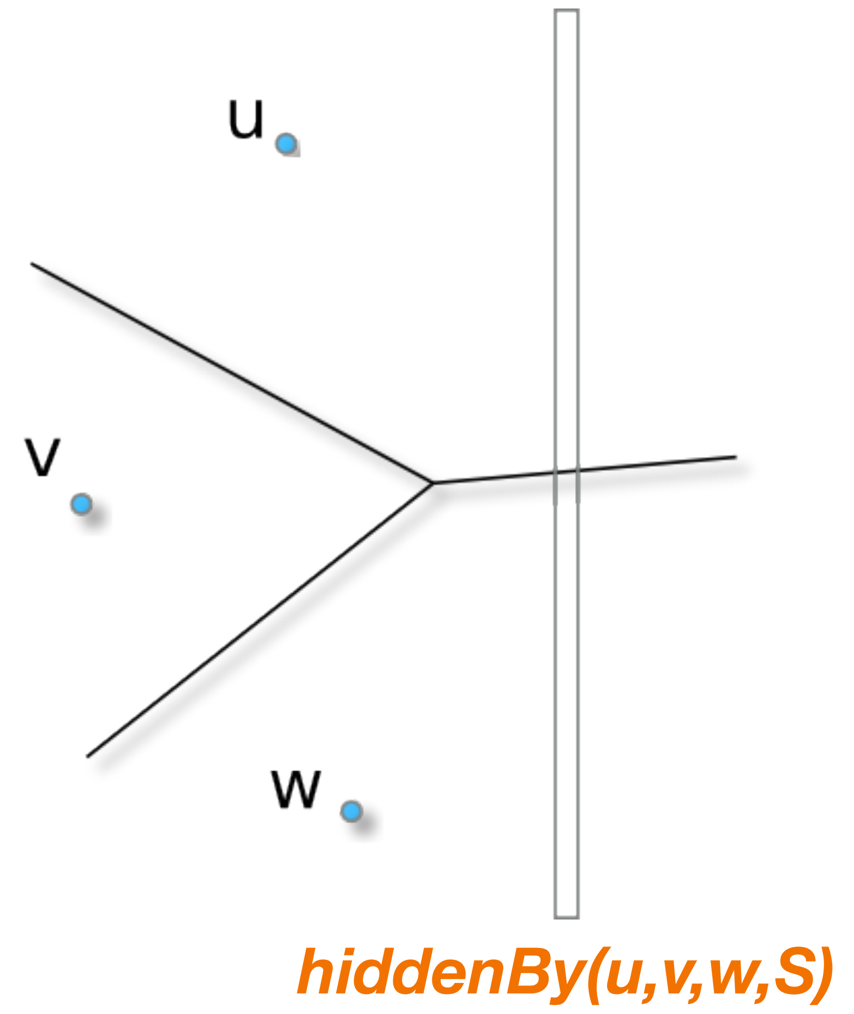
Separable Voronoi map: step 2



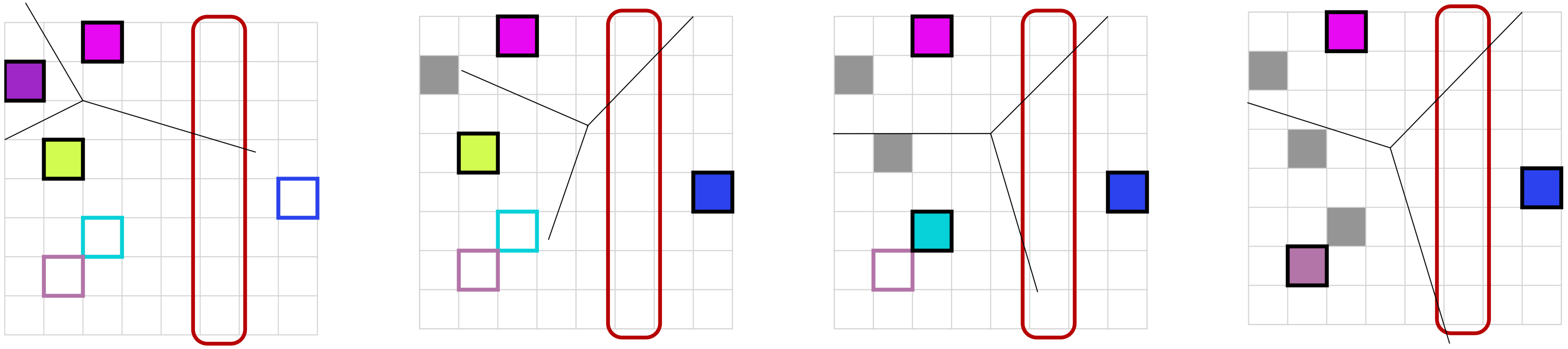
Stack based algorithm using a 3-ary *hiddenBy* predicate, *à la* sweep line $\Rightarrow$ $O(n)$ per column



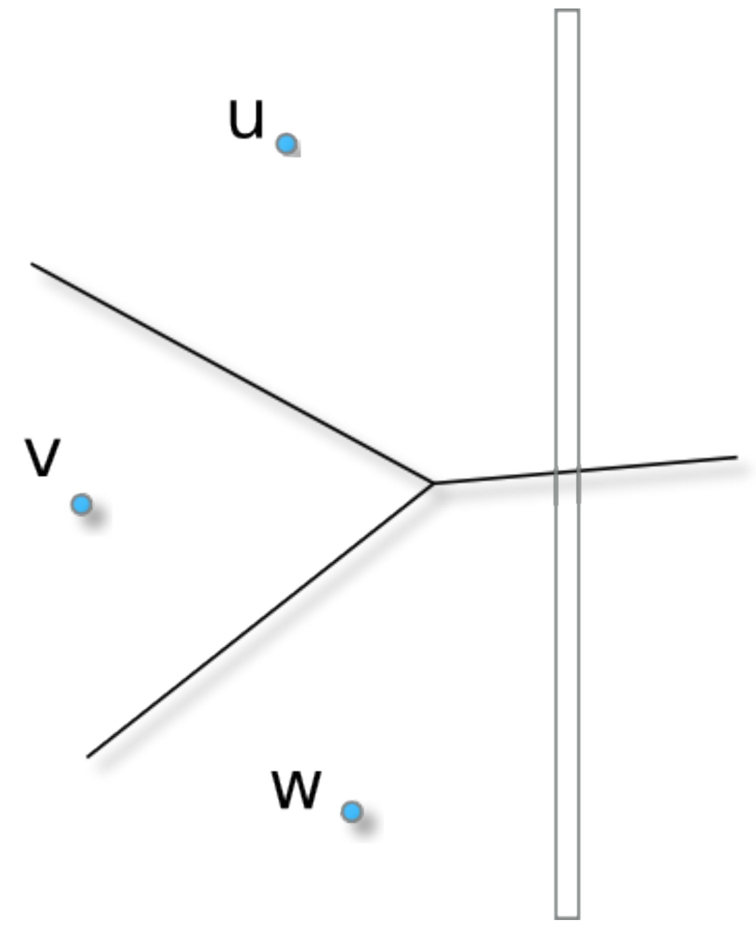
Separable Voronoi map: step 2



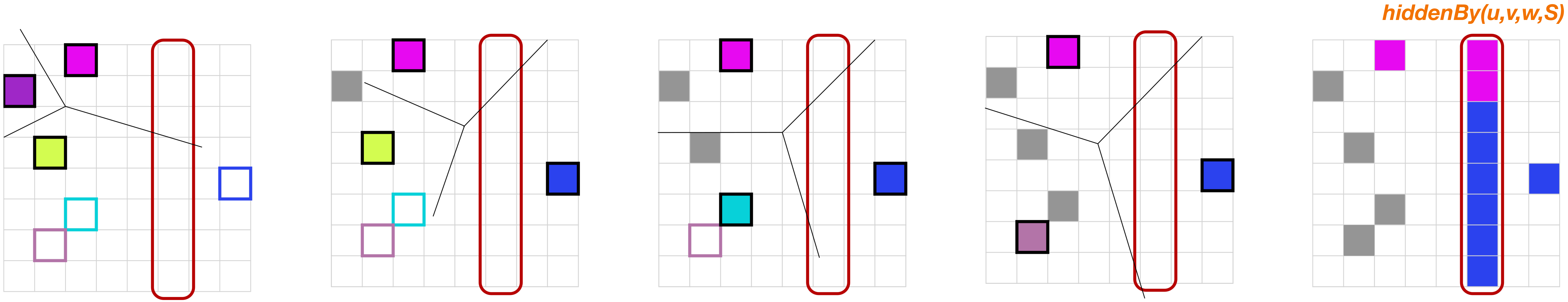
Stack based algorithm using a 3-ary *hiddenBy* predicate, *à la* sweep line ⇒ $O(n)$ per column



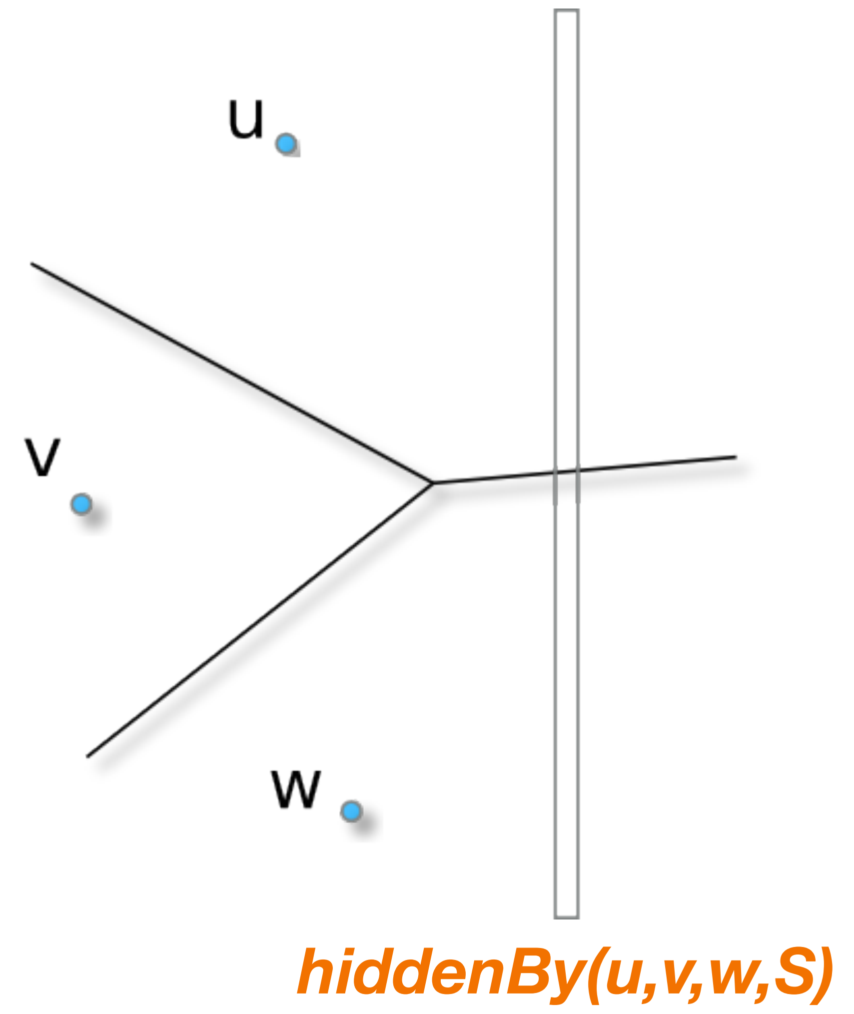
Separable Voronoi map: step 2



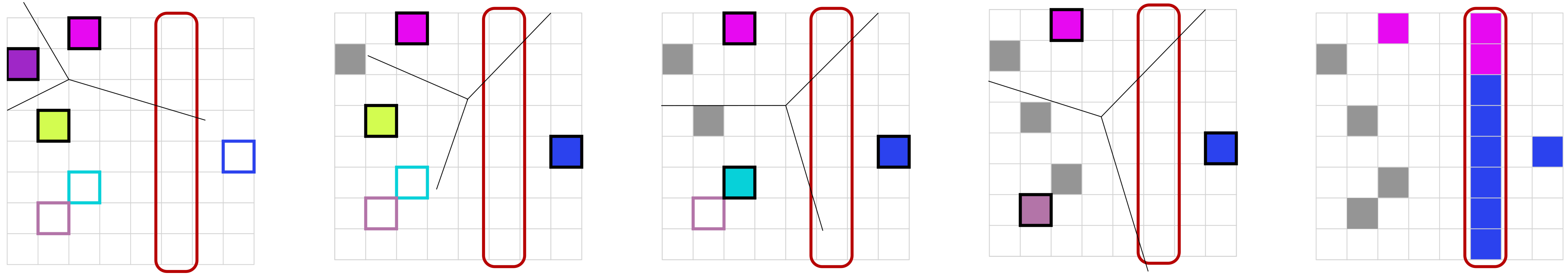
Stack based algorithm using a 3-ary *hiddenBy* predicate, *à la* sweep line ⇒ $O(n)$ per column

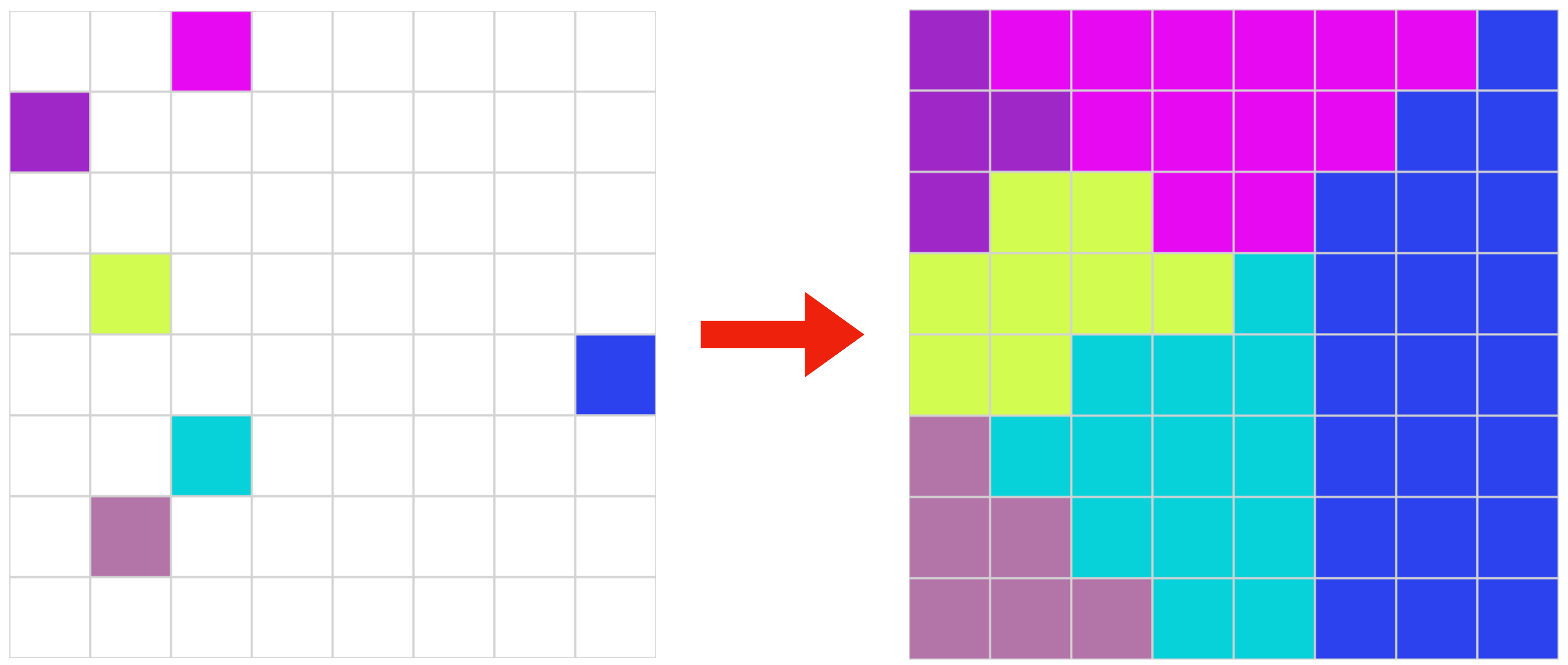


Separable Voronoi map: step 2

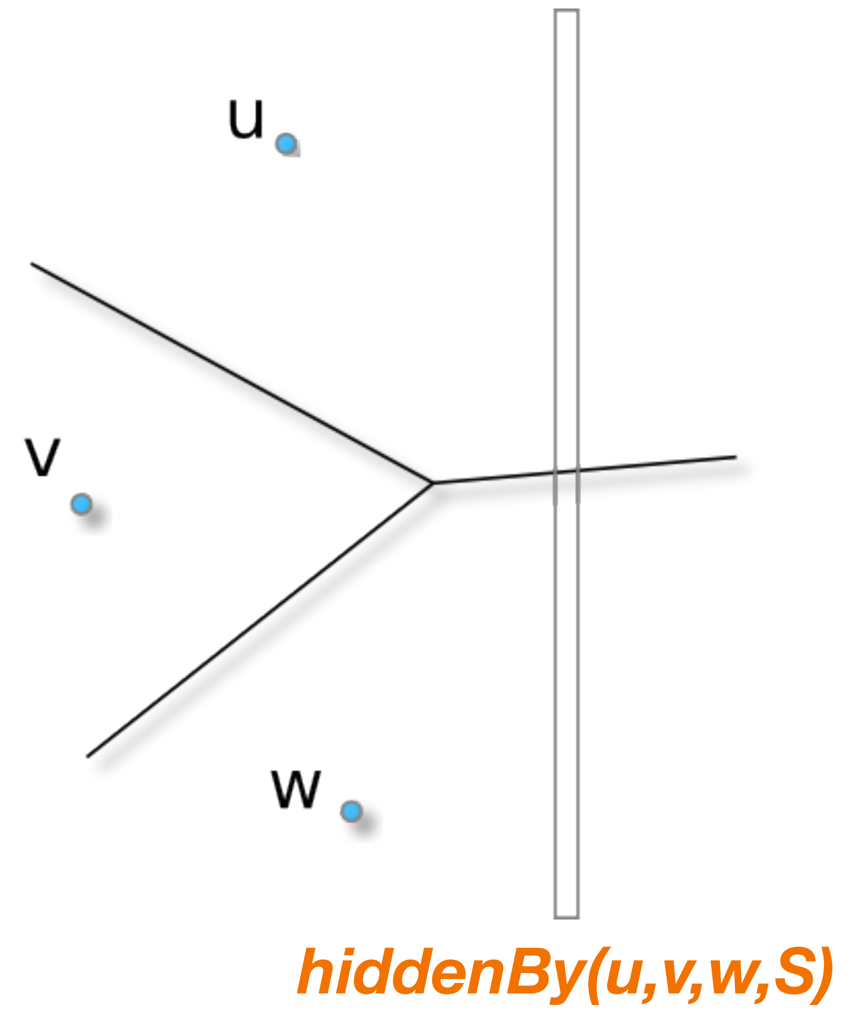


Stack based algorithm using a 3-ary *hiddenBy* predicate, *à la* sweep line ⇒ $O(n)$ per column

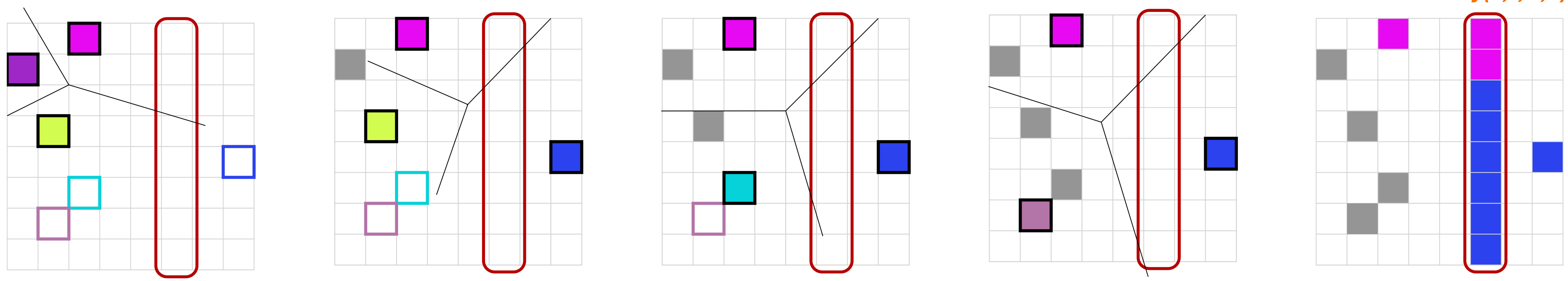


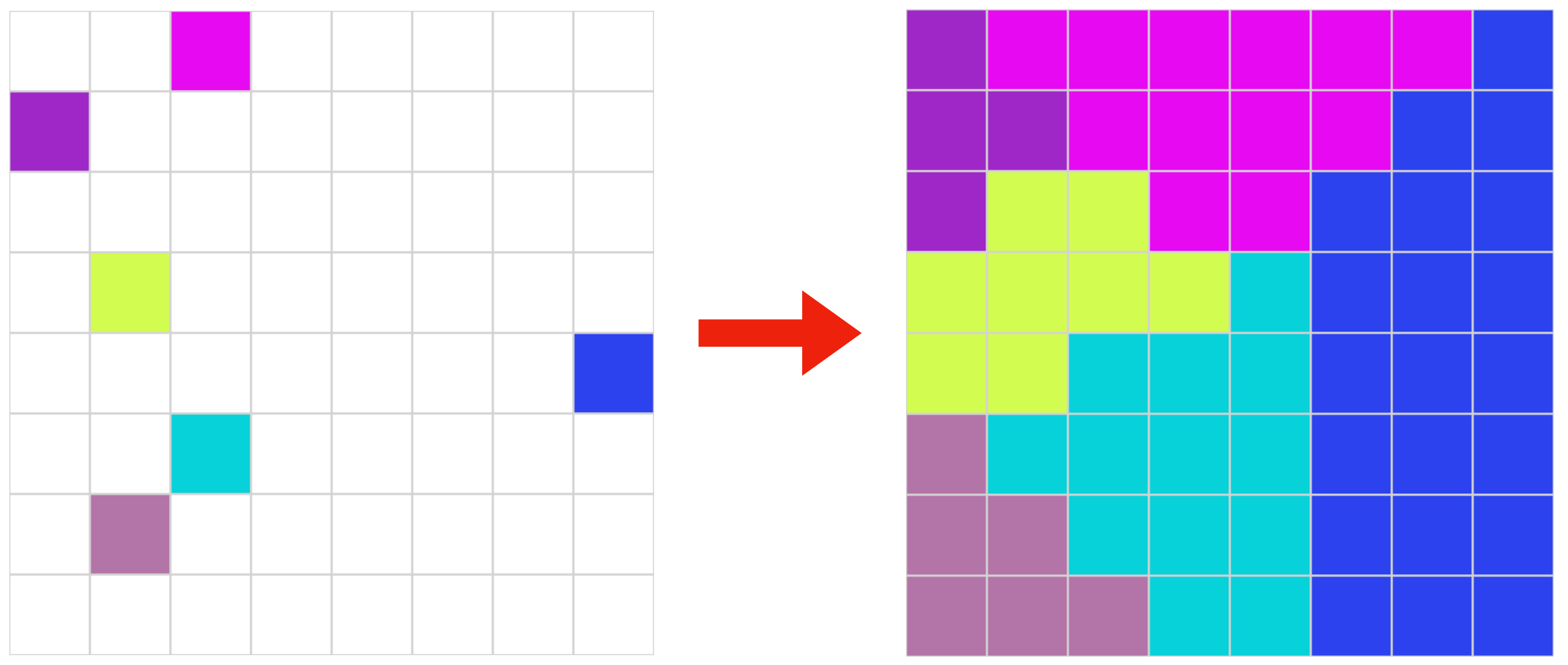


Separable Voronoi map: step 2



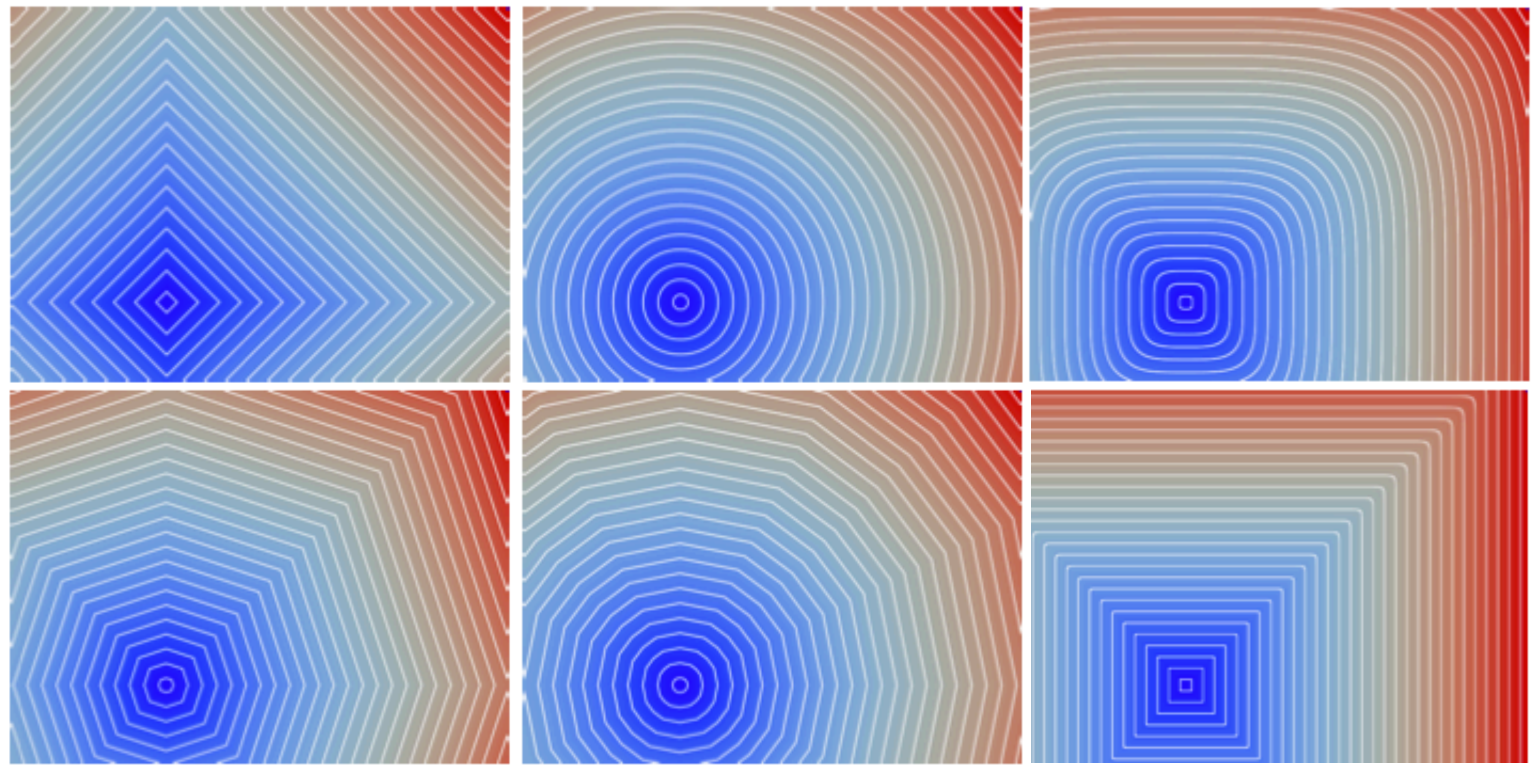
Stack based algorithm using a 3-ary *hiddenBy* predicate, *à la* sweep line $\Rightarrow O(n)$ per column

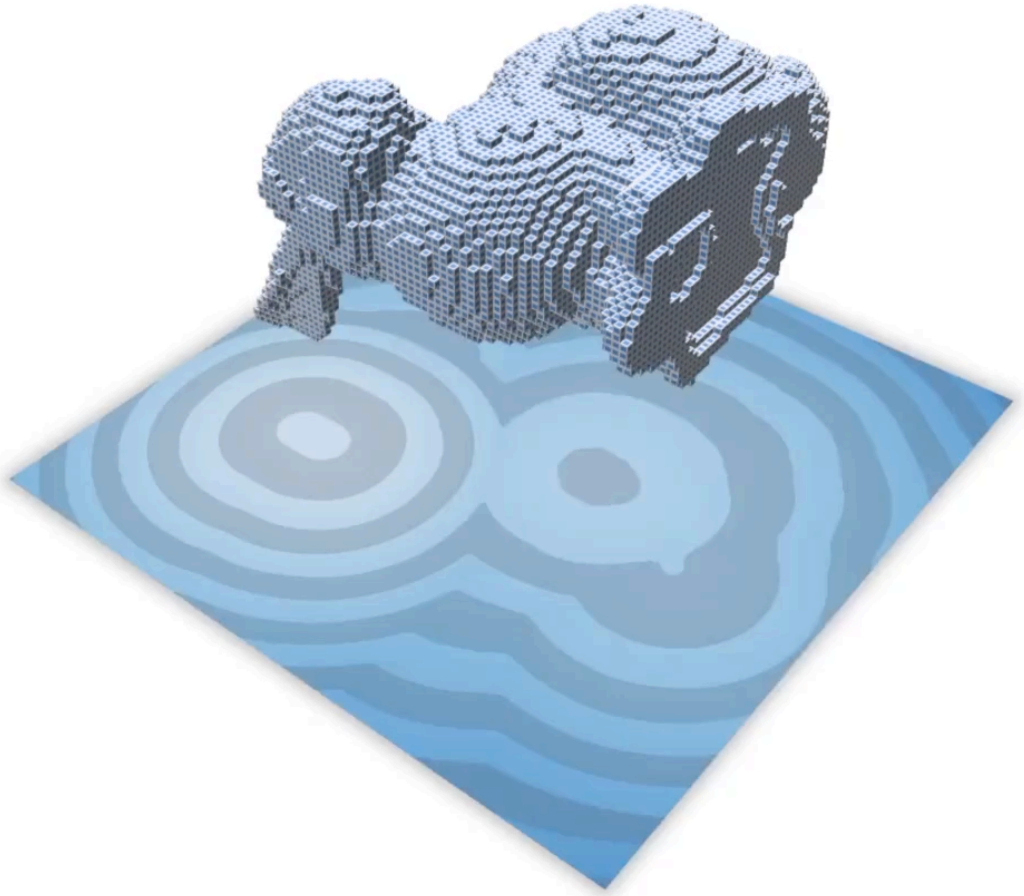


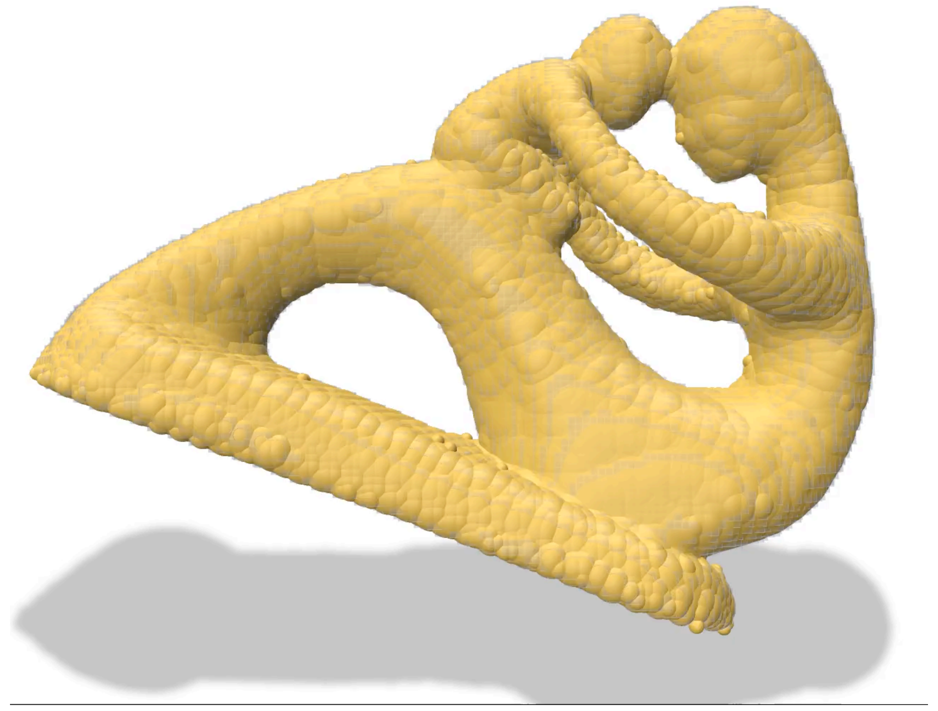


$\Rightarrow O(n^2)$ in total in 2D

Separable approaches

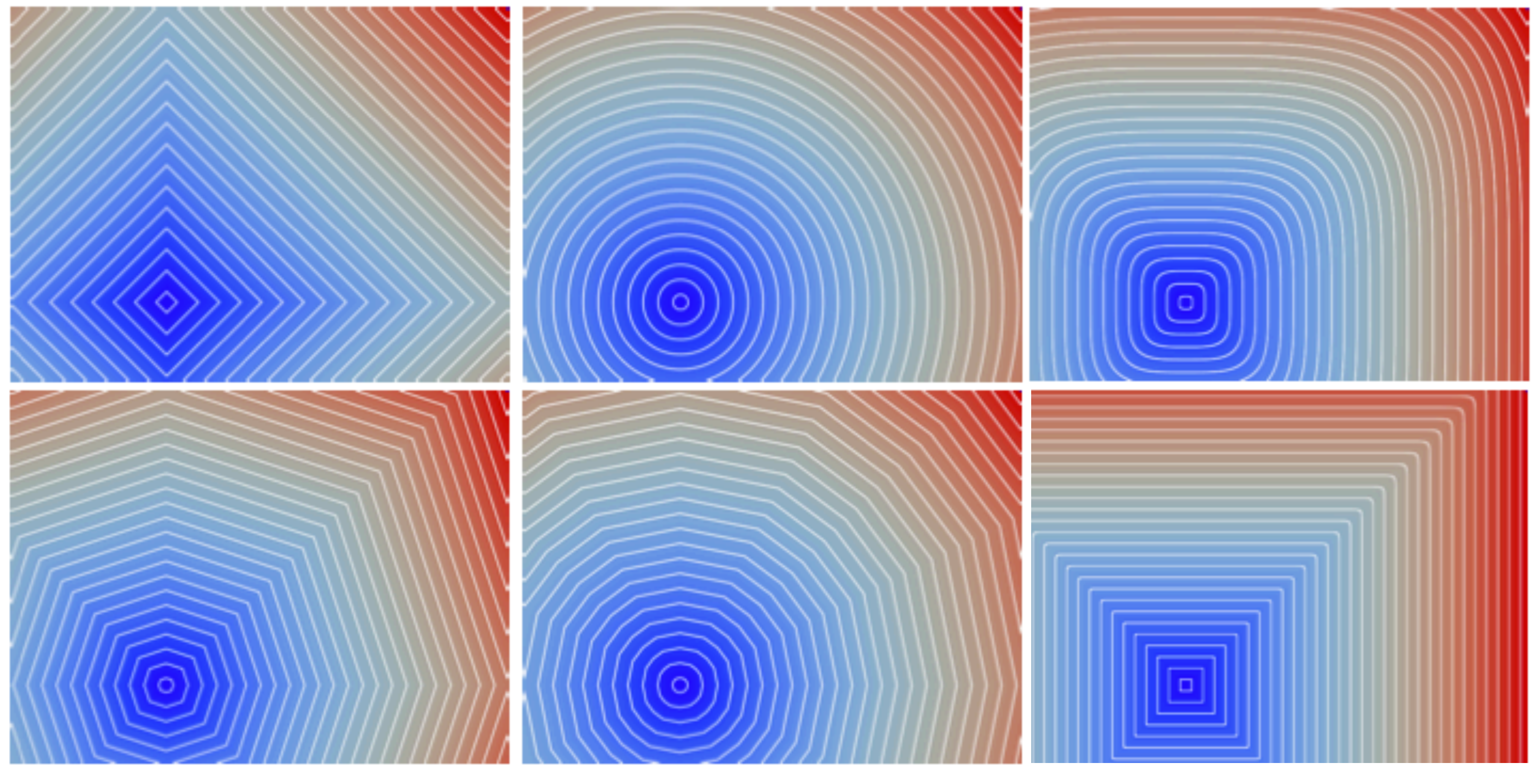


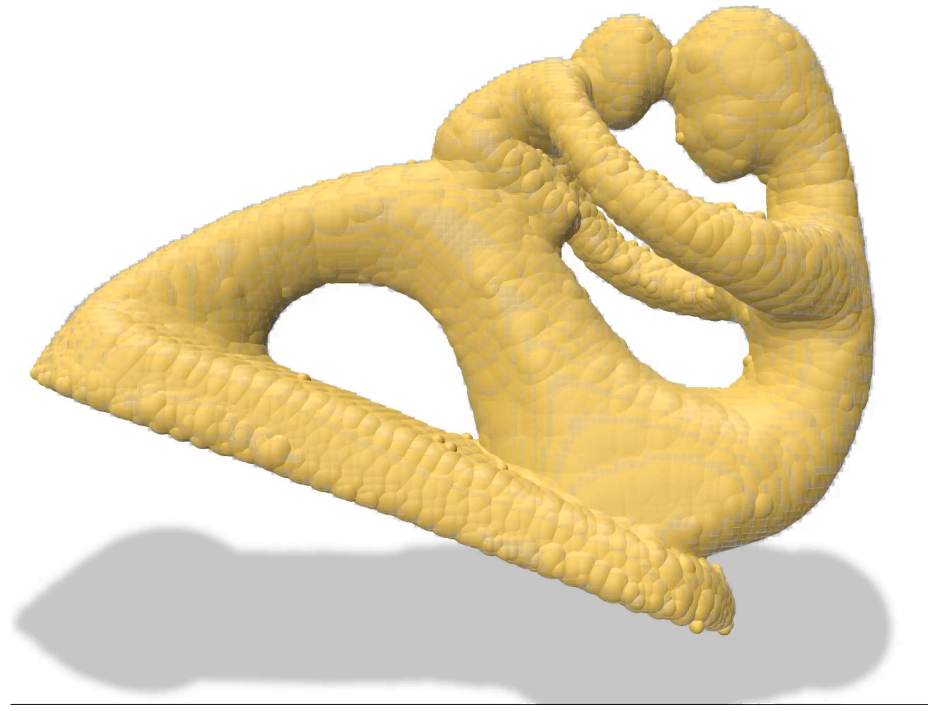




Separable approaches

The algorithm is correct:

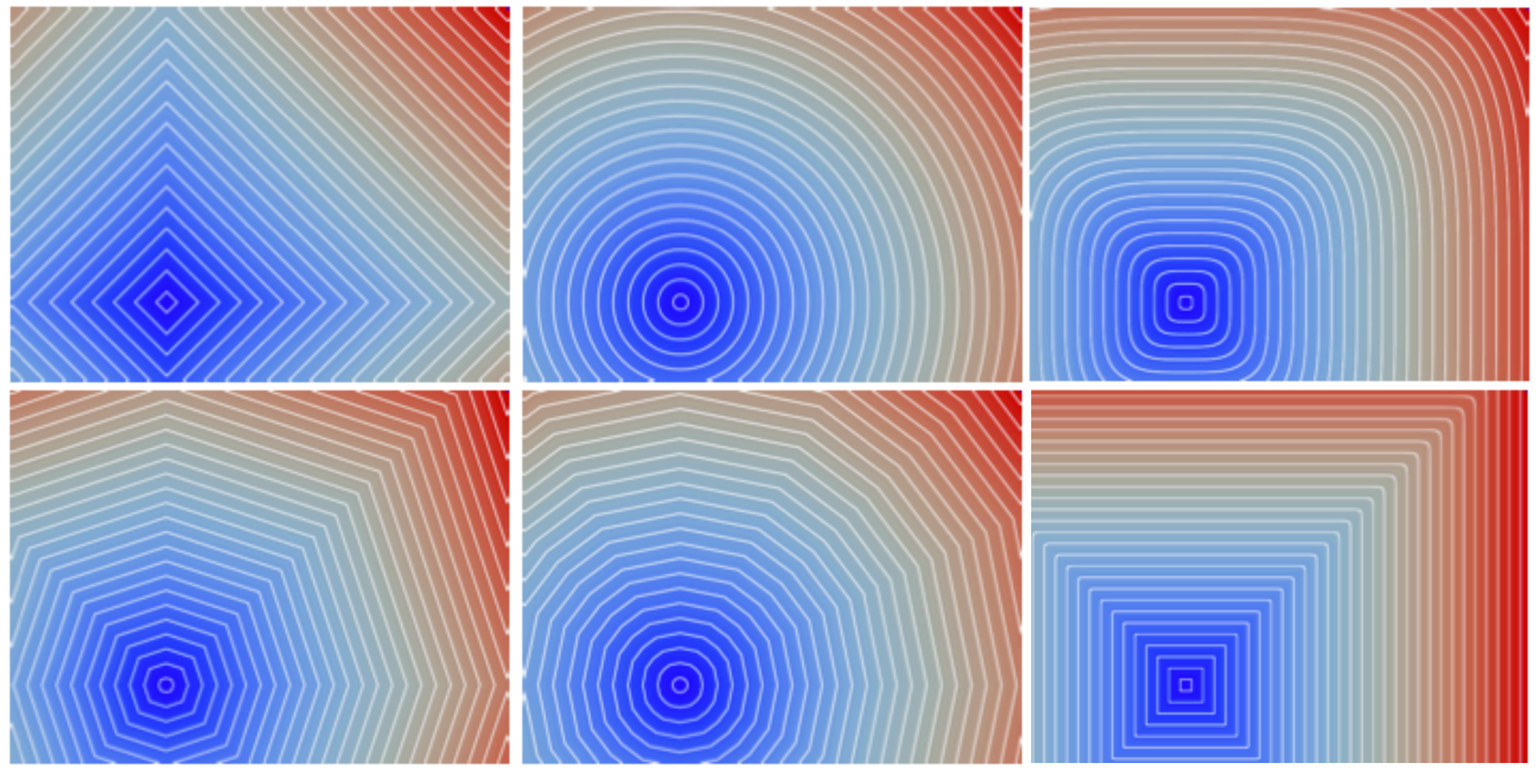


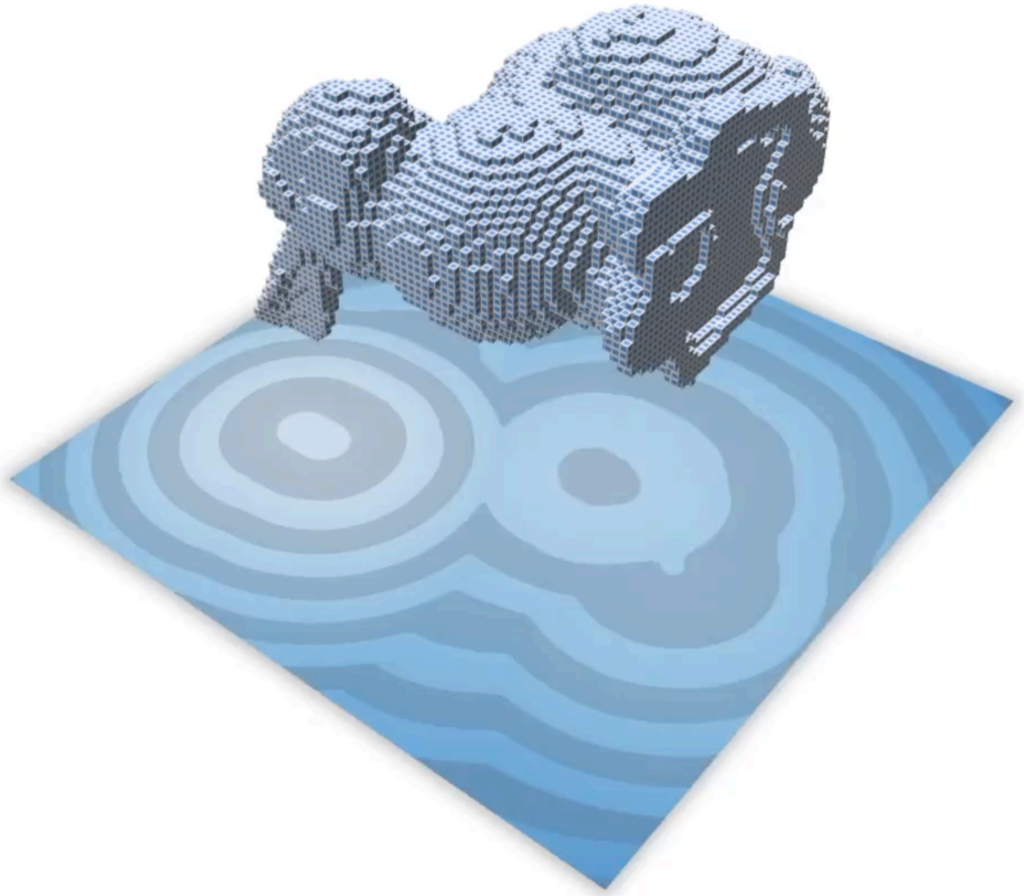


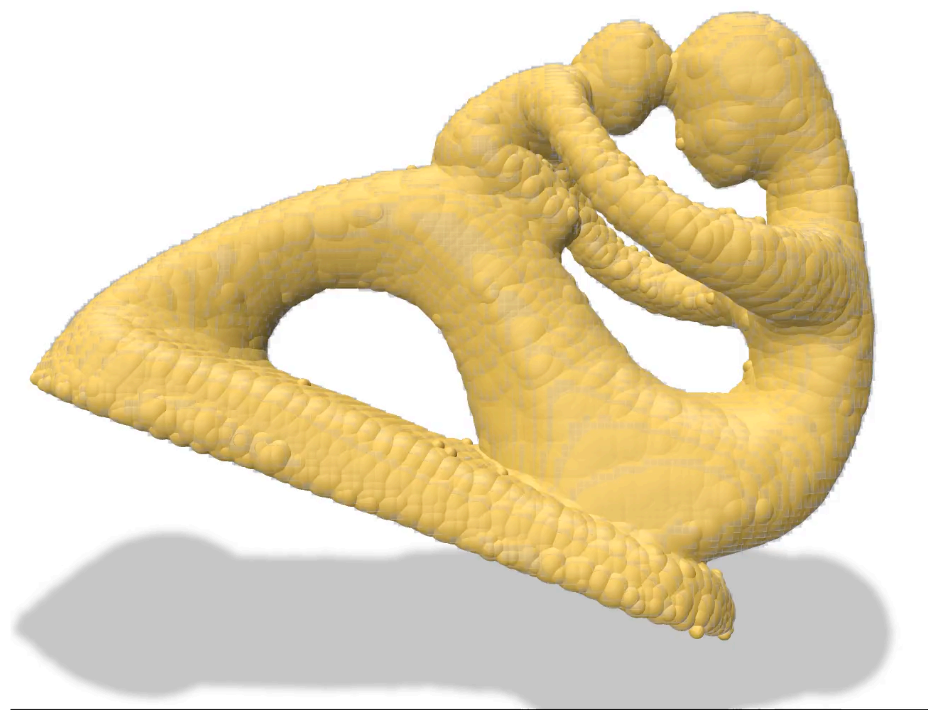
Separable approaches

The algorithm is correct:

- for any dimension



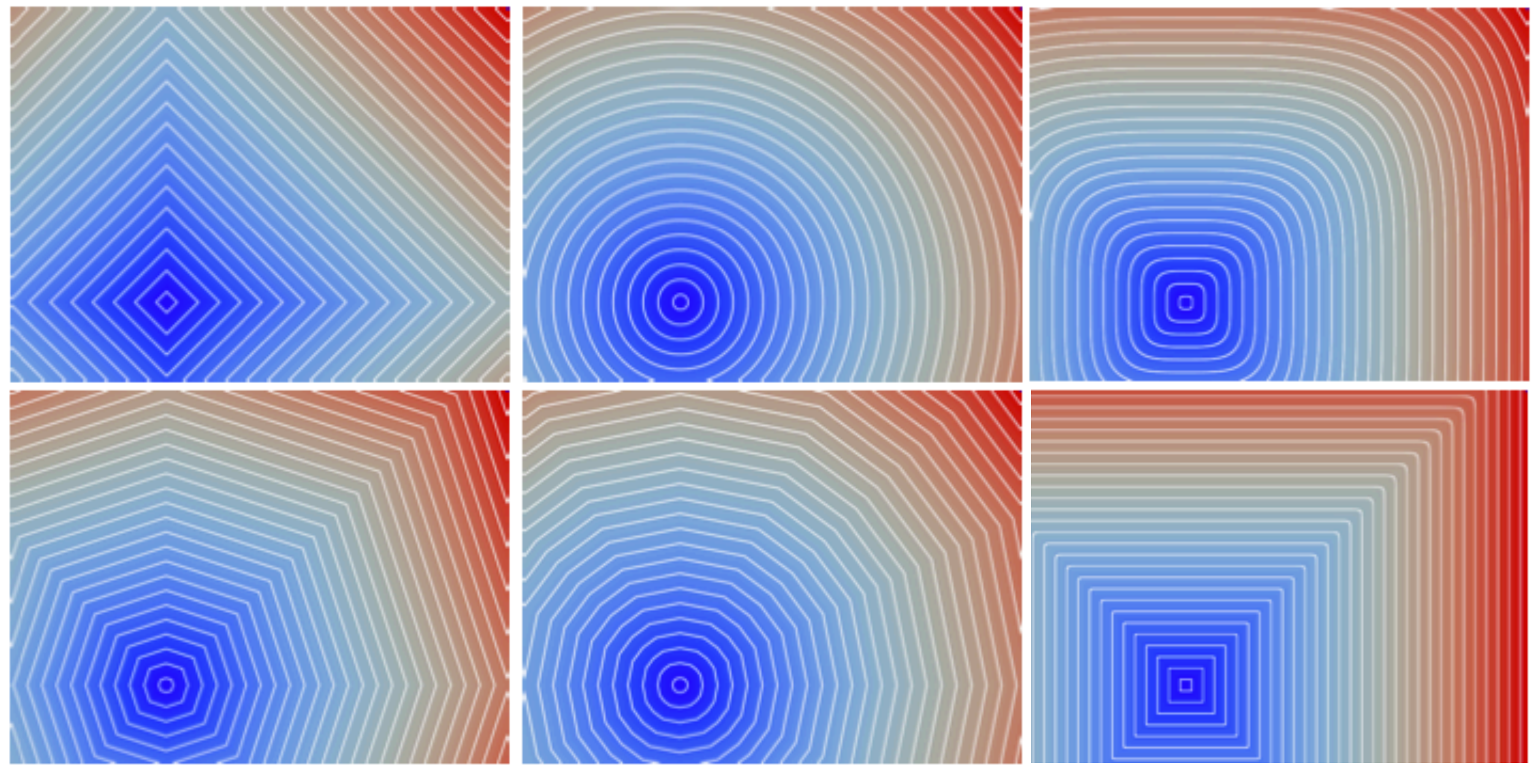


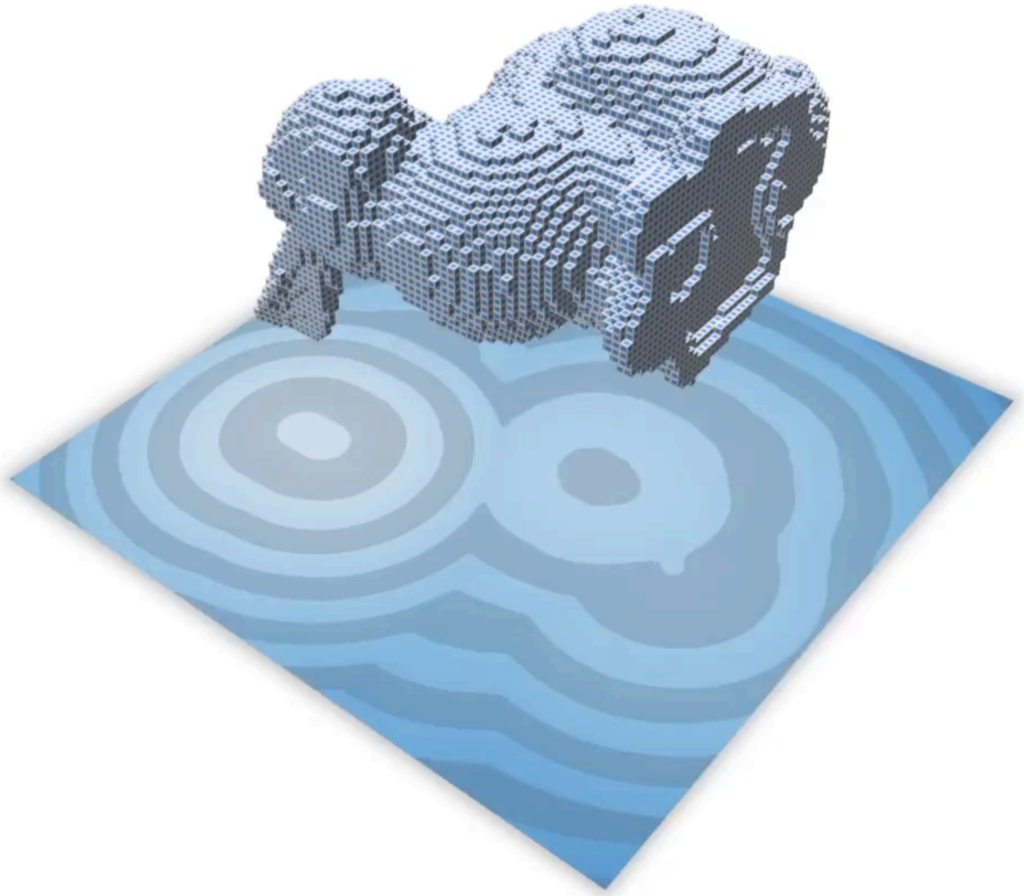


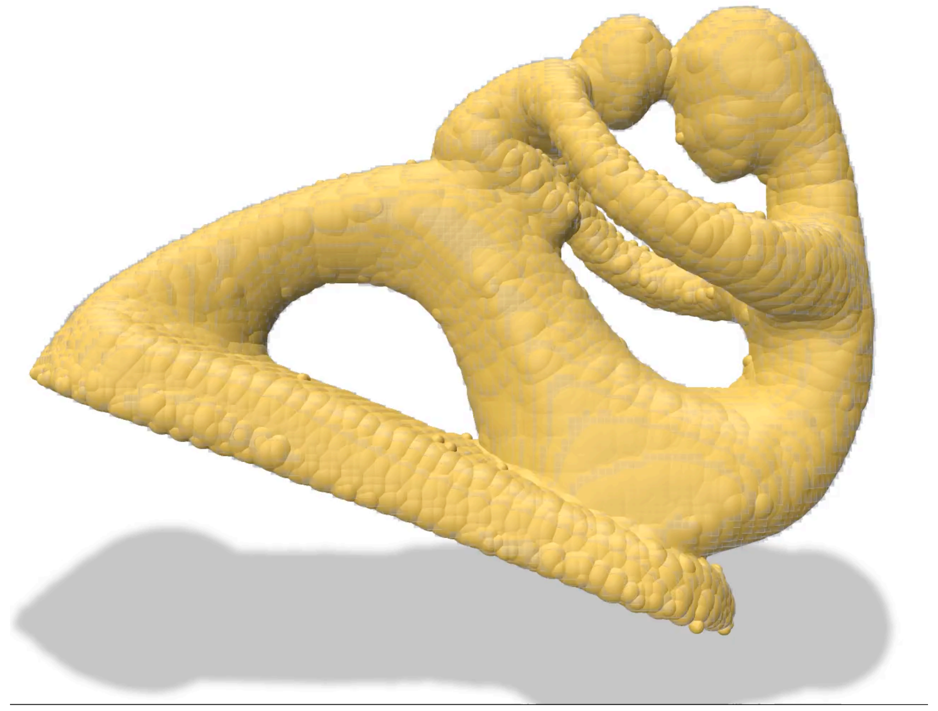
Separable approaches

The algorithm is correct:

- for any dimension
- for any metric with axis symmetric unit ball (e.g. any l_p)



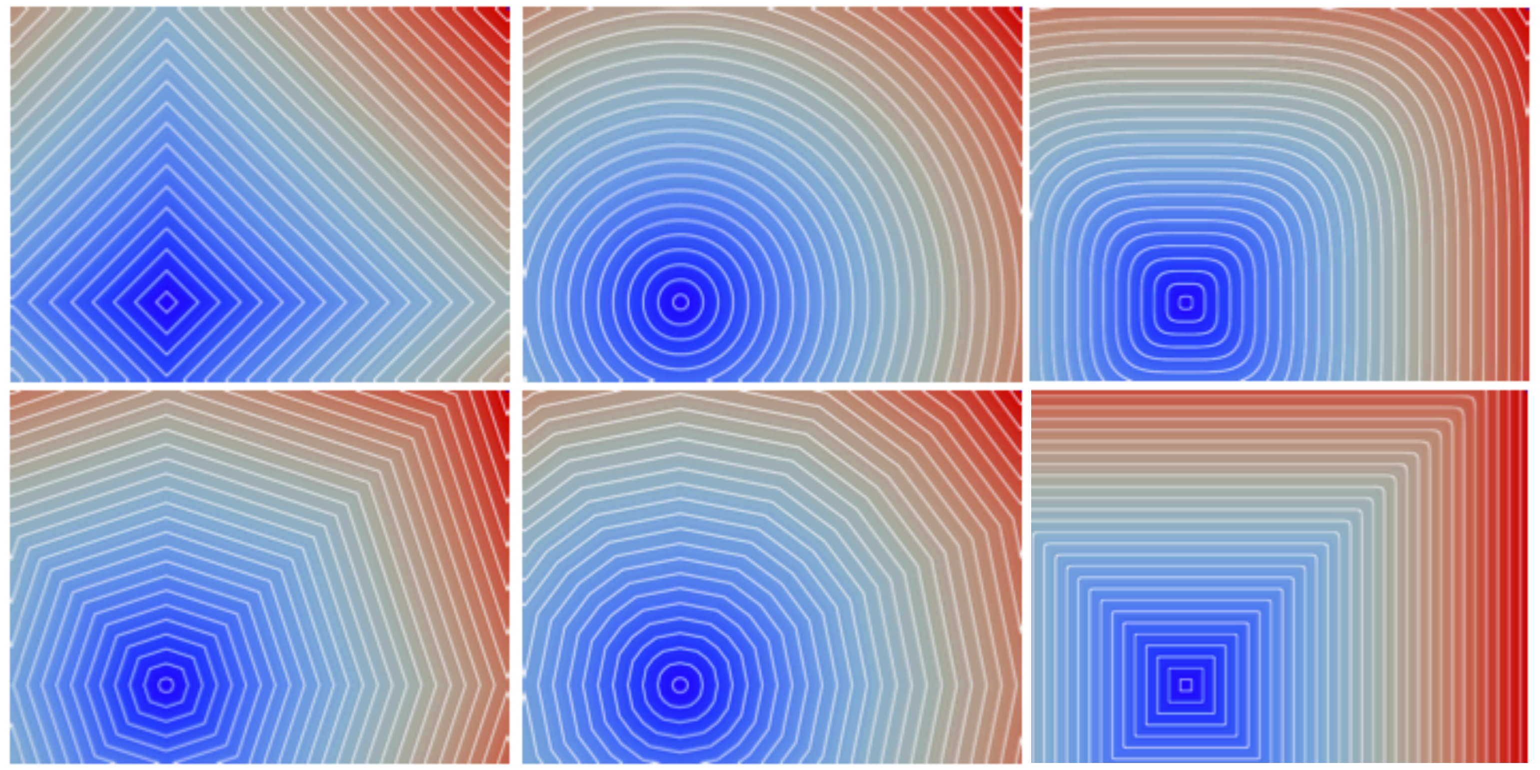


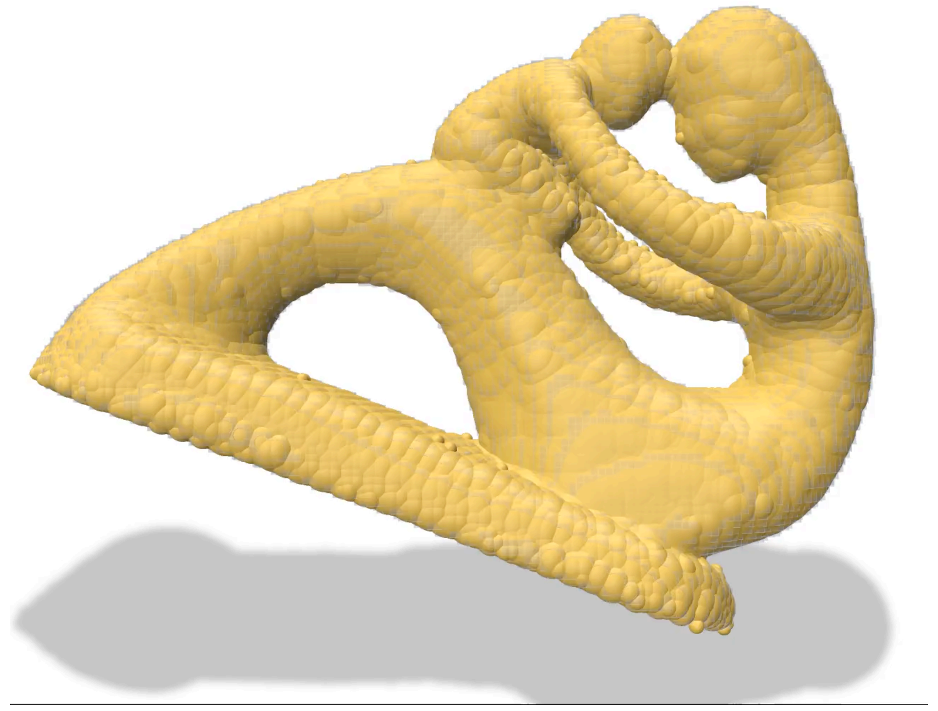


Separable approaches

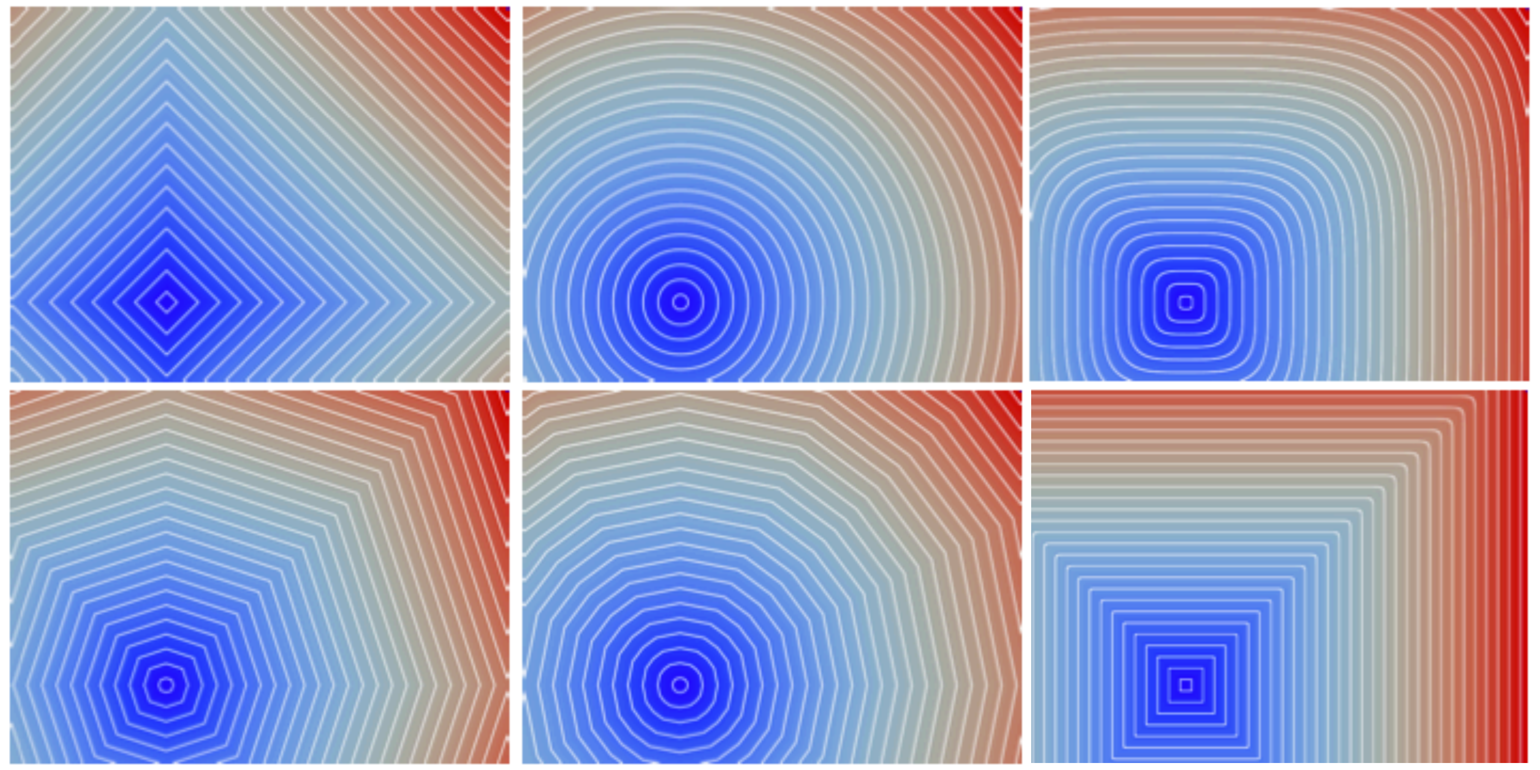
The algorithm is correct:

- for any dimension
- for any metric with axis symmetric unit ball (e.g. any l_p)
- on any toroidal nD domains





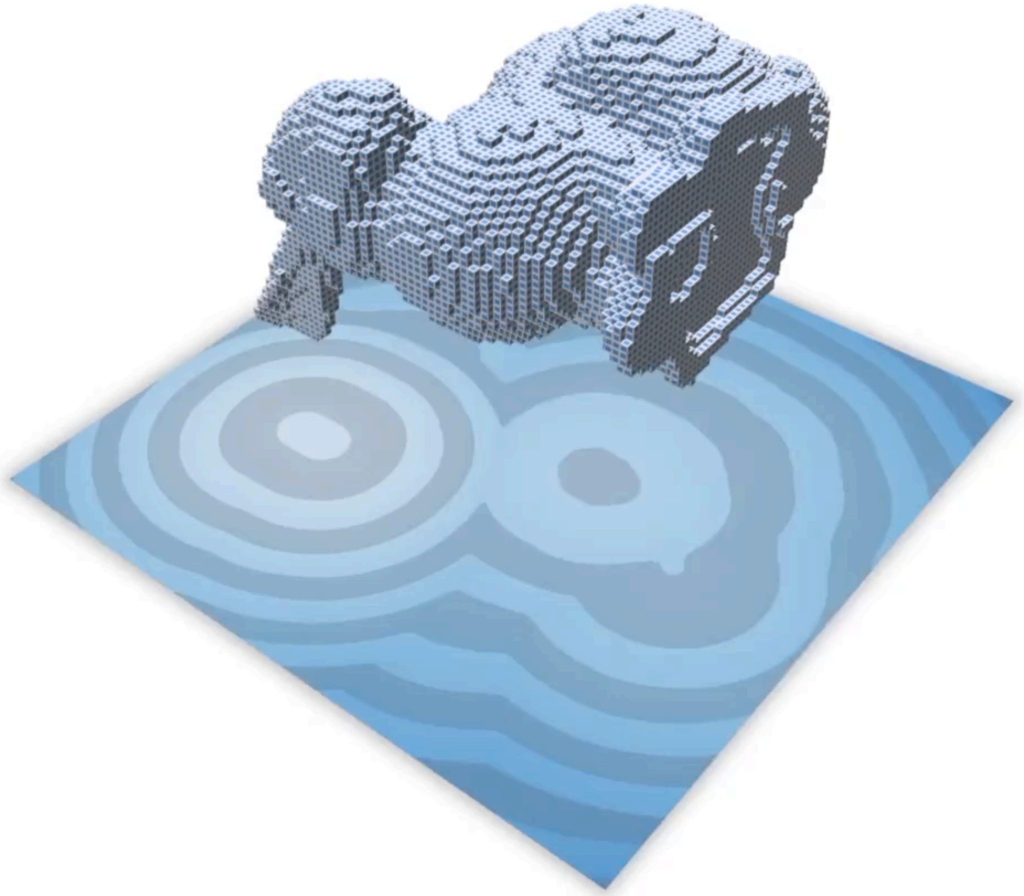
Separable approaches

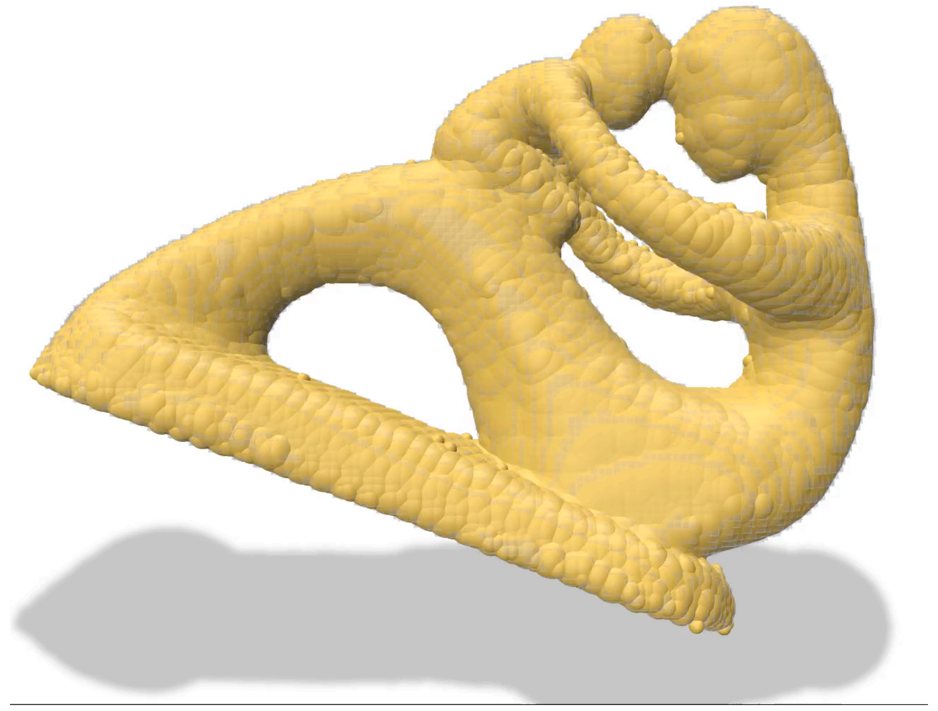


The algorithm is correct:

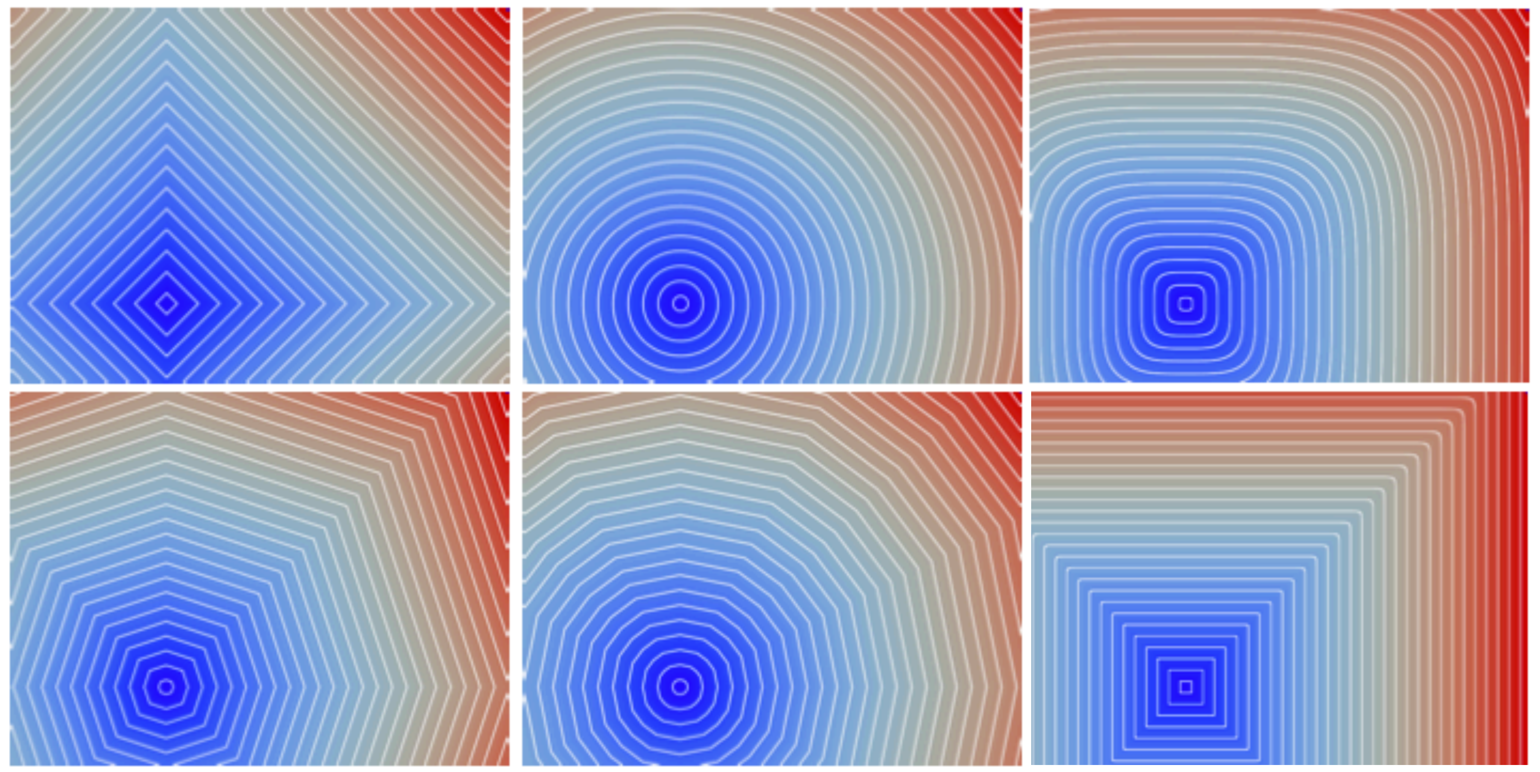
- for any dimension
- for any metric with axis symmetric unit ball (e.g. any l_p)
- on any toroidal nD domains

Exact and linear in time w.r.t. the number of grid points $O(d \cdot n^d)$ for l_2





Separable approaches

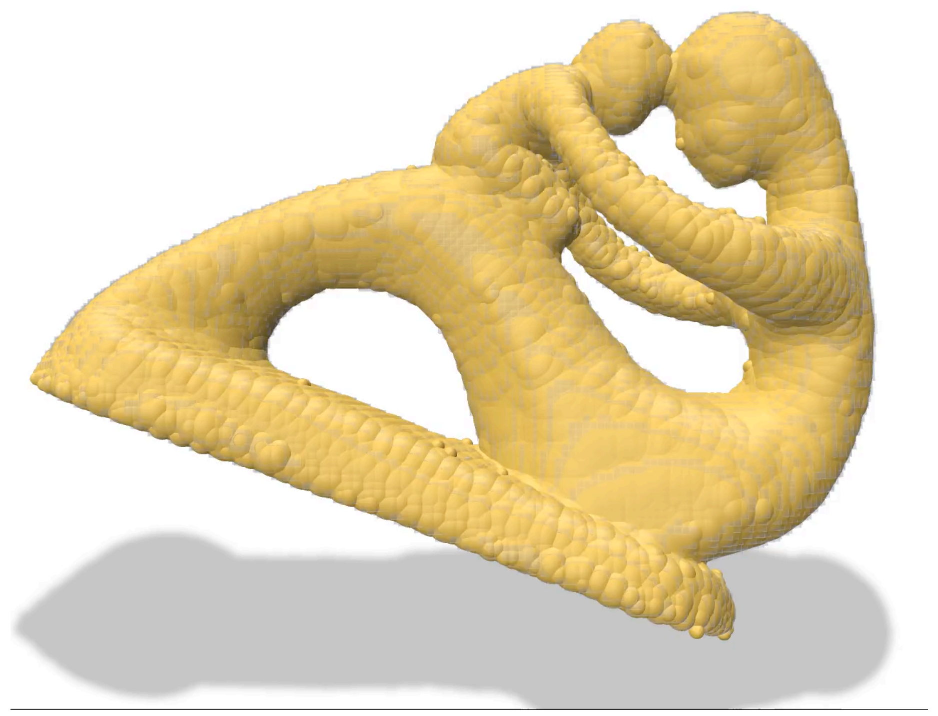


The algorithm is correct:

- for any dimension
- for any metric with axis symmetric unit ball (e.g. any l_p)
- on any toroidal nD domains

Exact and linear in time w.r.t. the number of grid points $O(d \cdot n^d)$ for l_2

$O(d^2 \cdot \log(p) \cdot \log(n) \cdot n^d)$ for exact l_p ($p \in \mathbb{Z}^+$), $O(d \cdot n^d)$ approx.



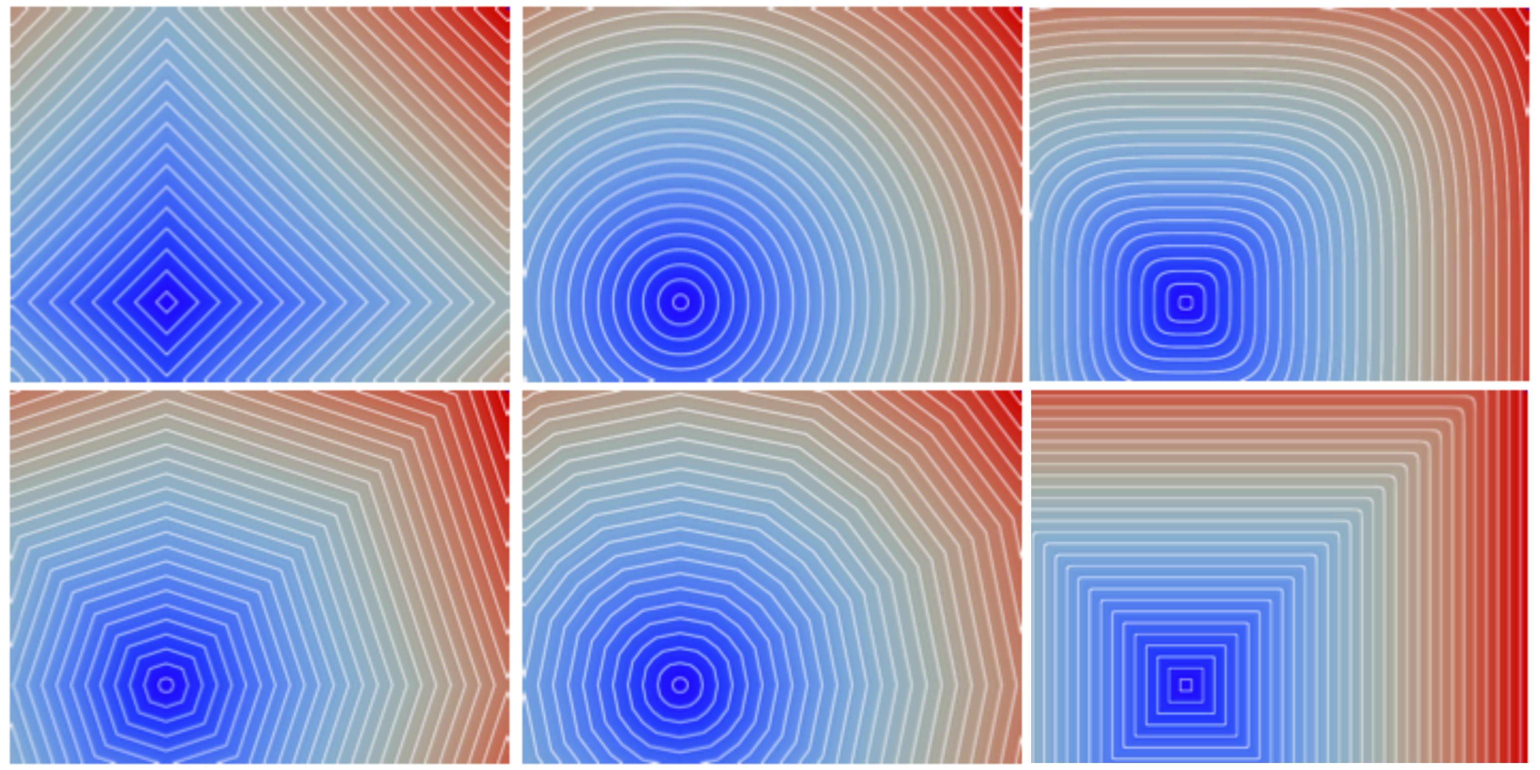
Separable approaches

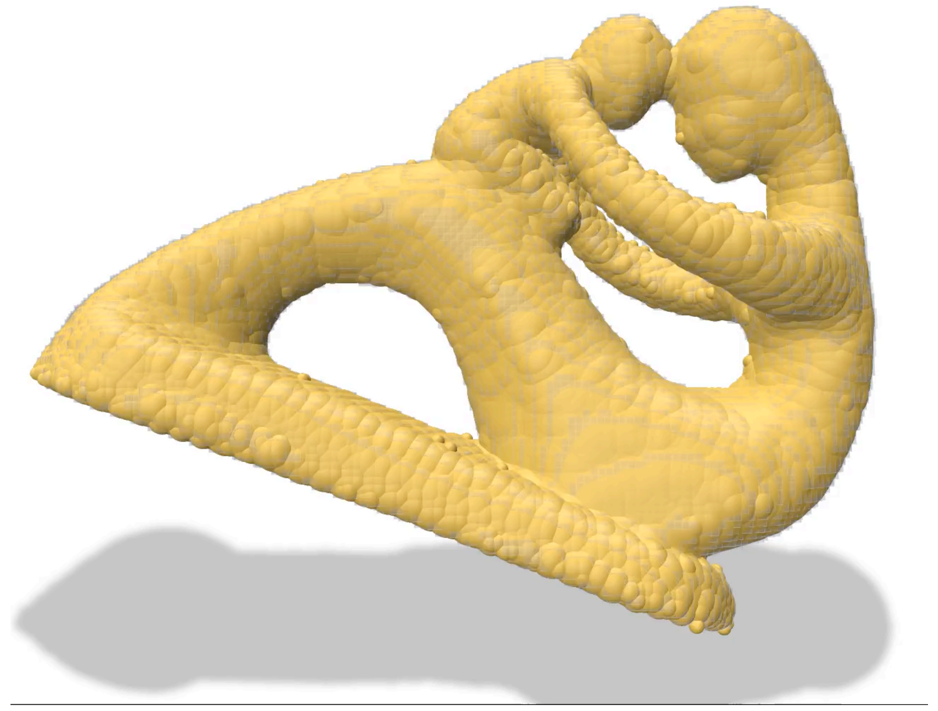
The algorithm is correct:

- for any dimension
- for any metric with axis symmetric unit ball (e.g. any l_p)
- on any toroidal nD domains

Exact and linear in time w.r.t. the number of grid points $O(d \cdot n^d)$ for l_2

$O(d^2 \cdot \log(p) \cdot \log(n) \cdot n^d)$ for exact l_p $(p \in \mathbb{Z}^+)$, $O(d \cdot n^d)$ approx.





Separable approaches

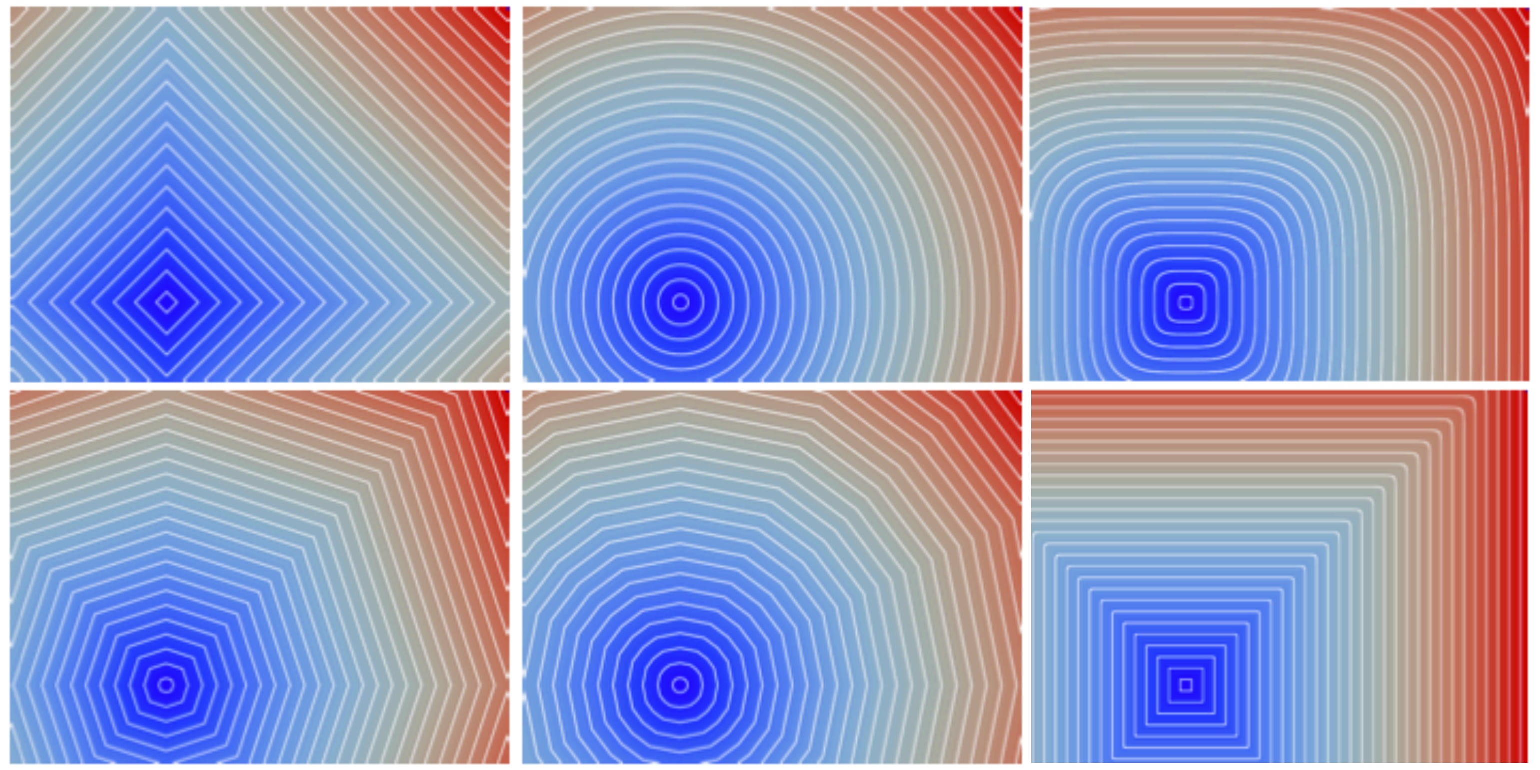
The algorithm is correct:

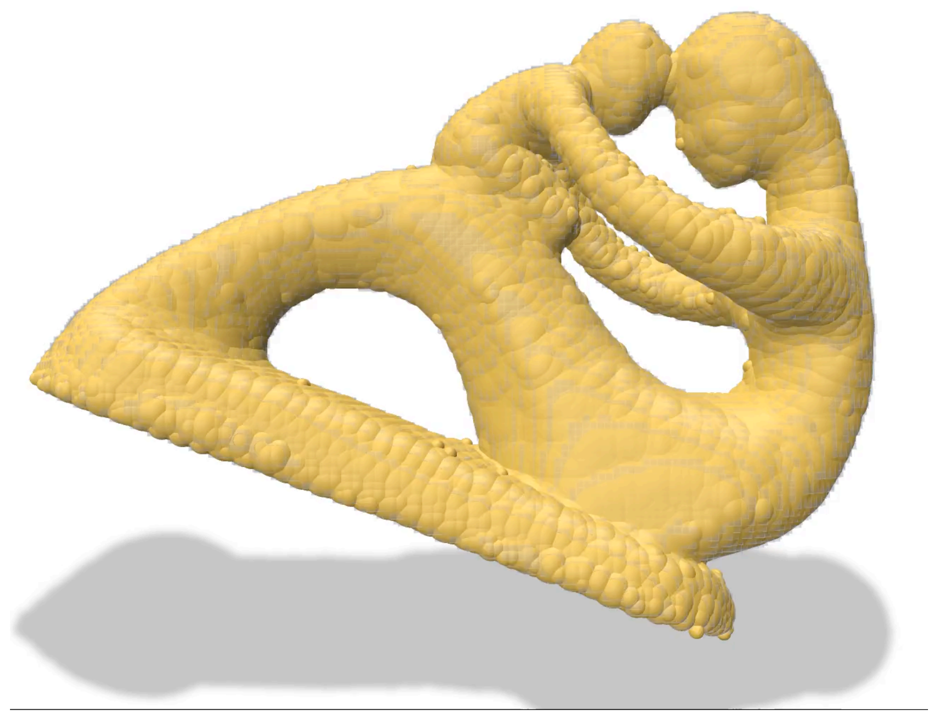
- for any dimension
- for any metric with axis symmetric unit ball (e.g. any l_p)
- on any toroidal nD domains

Exact and linear in time w.r.t. the number of grid points $O(d \cdot n^d)$ for l_2

$O(d^2 \cdot \log(p) \cdot \log(n) \cdot n^d)$ for exact l_p ($p \in \mathbb{Z}^+$), $O(d \cdot n^d)$ approx.

Trivial multithread / GPU / out-of-core implementations





Separable approaches

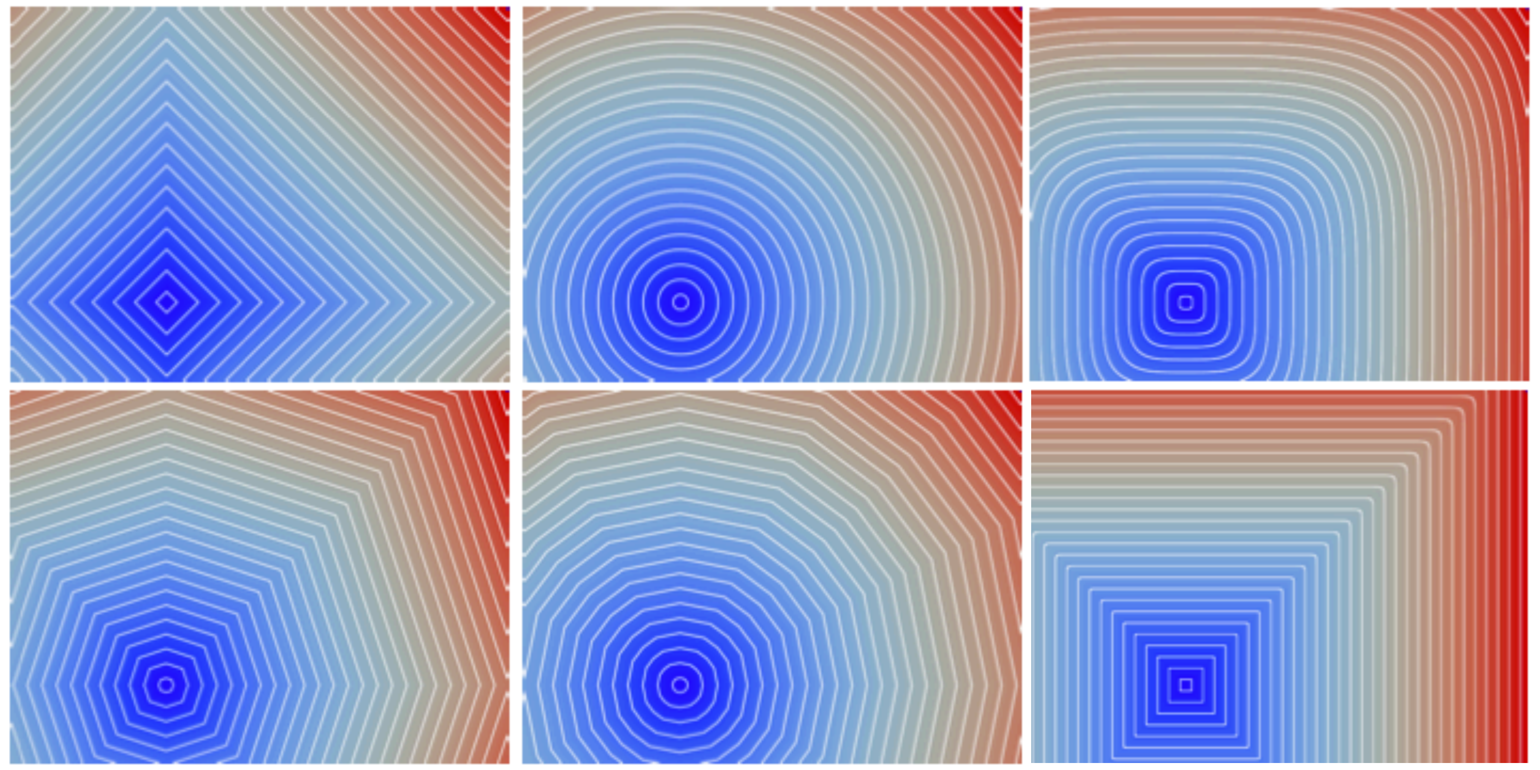
The algorithm is correct:

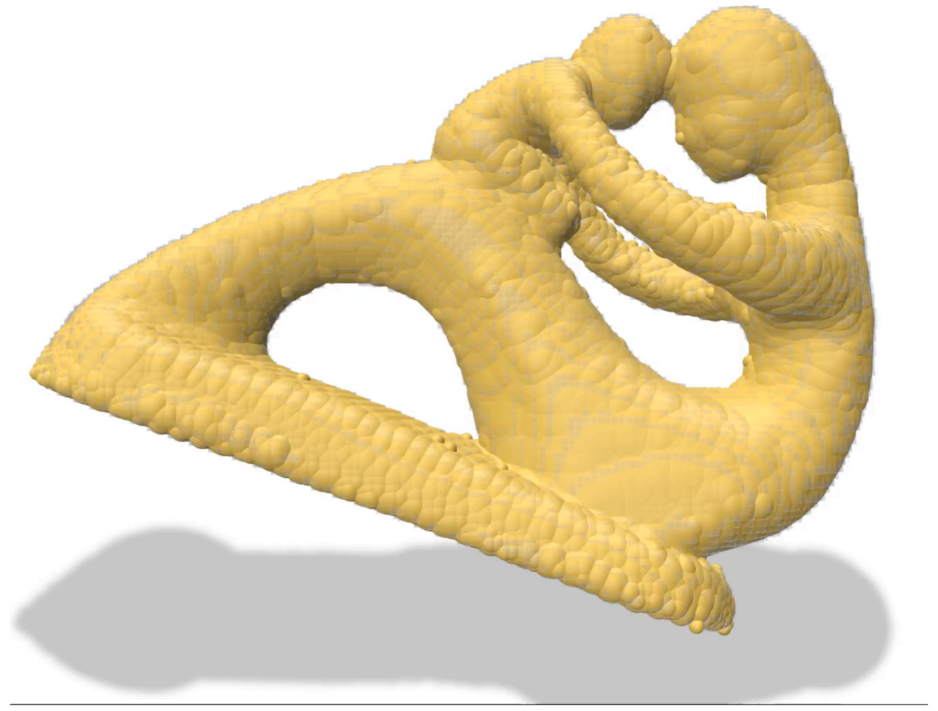
- for any dimension
- for any metric with axis symmetric unit ball (e.g. any l_p)
- on any toroidal nD domains

Exact and linear in time w.r.t. the number of grid points $O(d \cdot n^d)$ for l_2

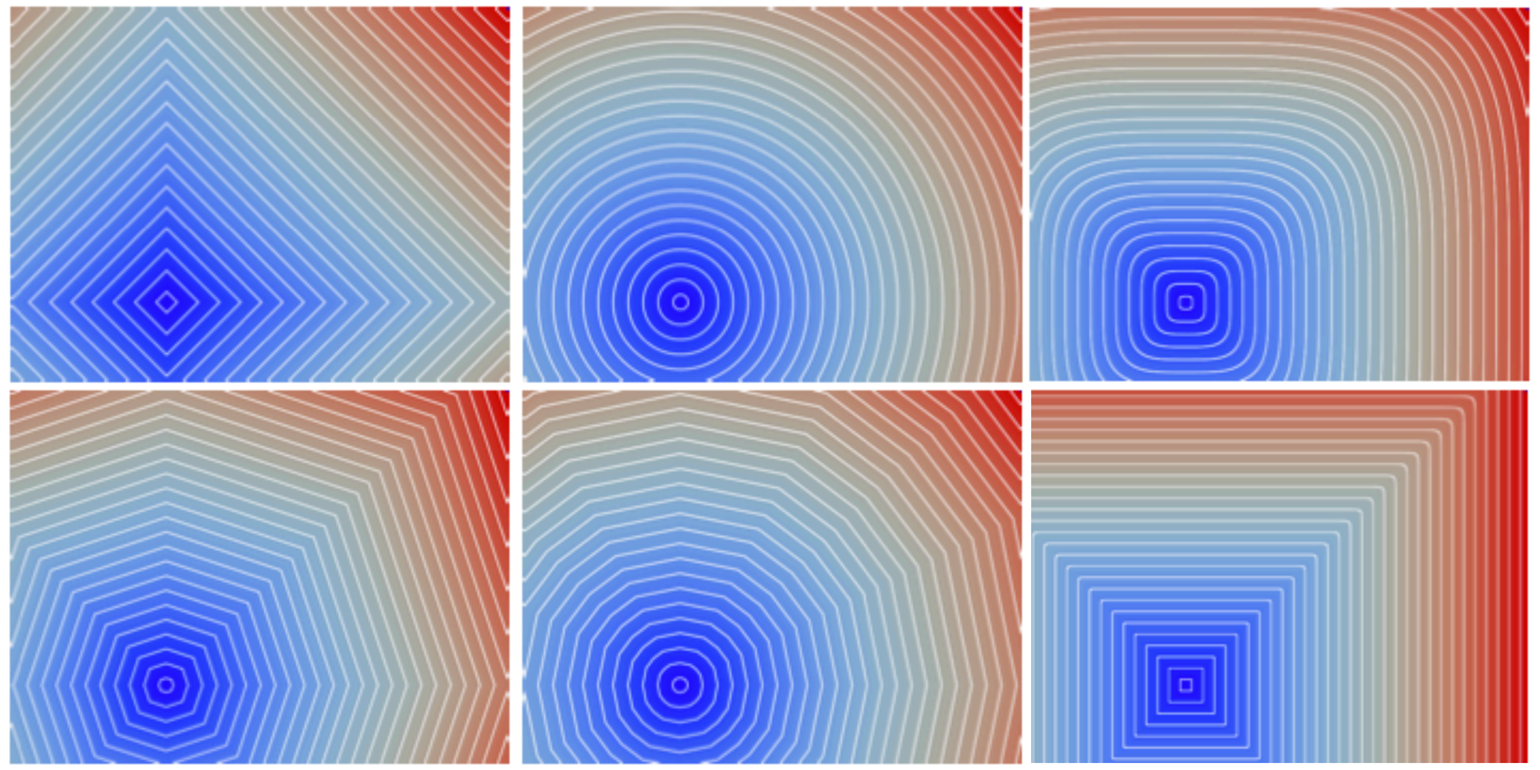
$O(d^2 \cdot \log(p) \cdot \log(n) \cdot n^d)$ for exact l_p ($p \in \mathbb{Z}^+$), $O(d \cdot n^d)$ approx.

Trivial multithread / GPU / out-of-core implementations





Separable approaches



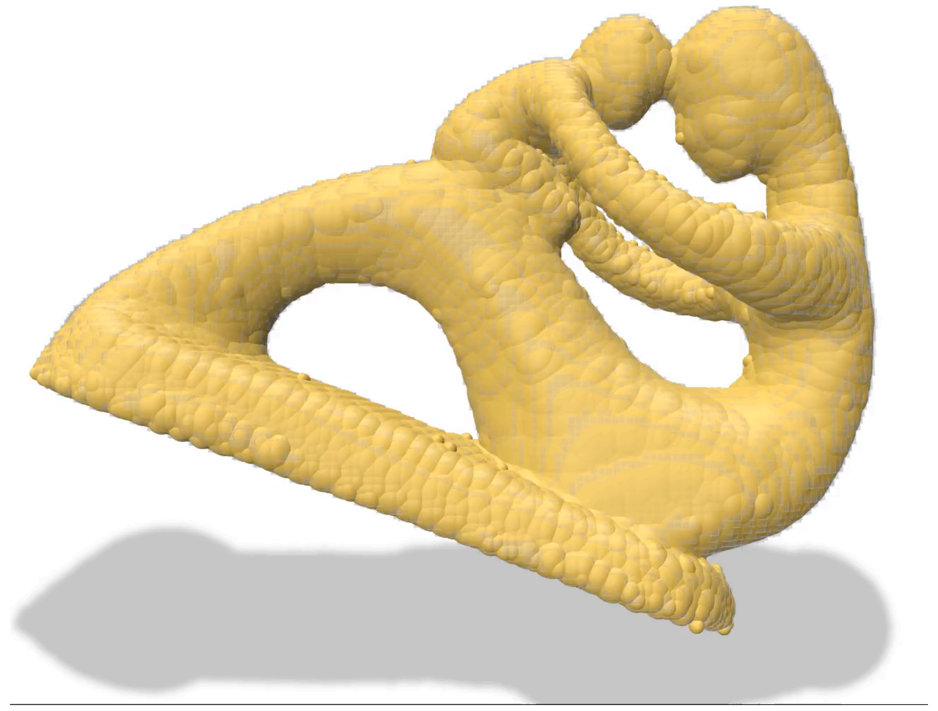
The algorithm is correct:

- for any dimension
- for any metric with axis symmetric unit ball (e.g. any l_p)
- on any toroidal nD domains

Exact and linear in time w.r.t. the number of grid points $O(d \cdot n^d)$ for l_2

$O(d^2 \cdot \log(p) \cdot \log(n) \cdot n^d)$ for exact l_p ($p \in \mathbb{Z}^+$), $O(d \cdot n^d)$ approx.

Trivial multithread / GPU / out-of-core implementations



Separable approaches

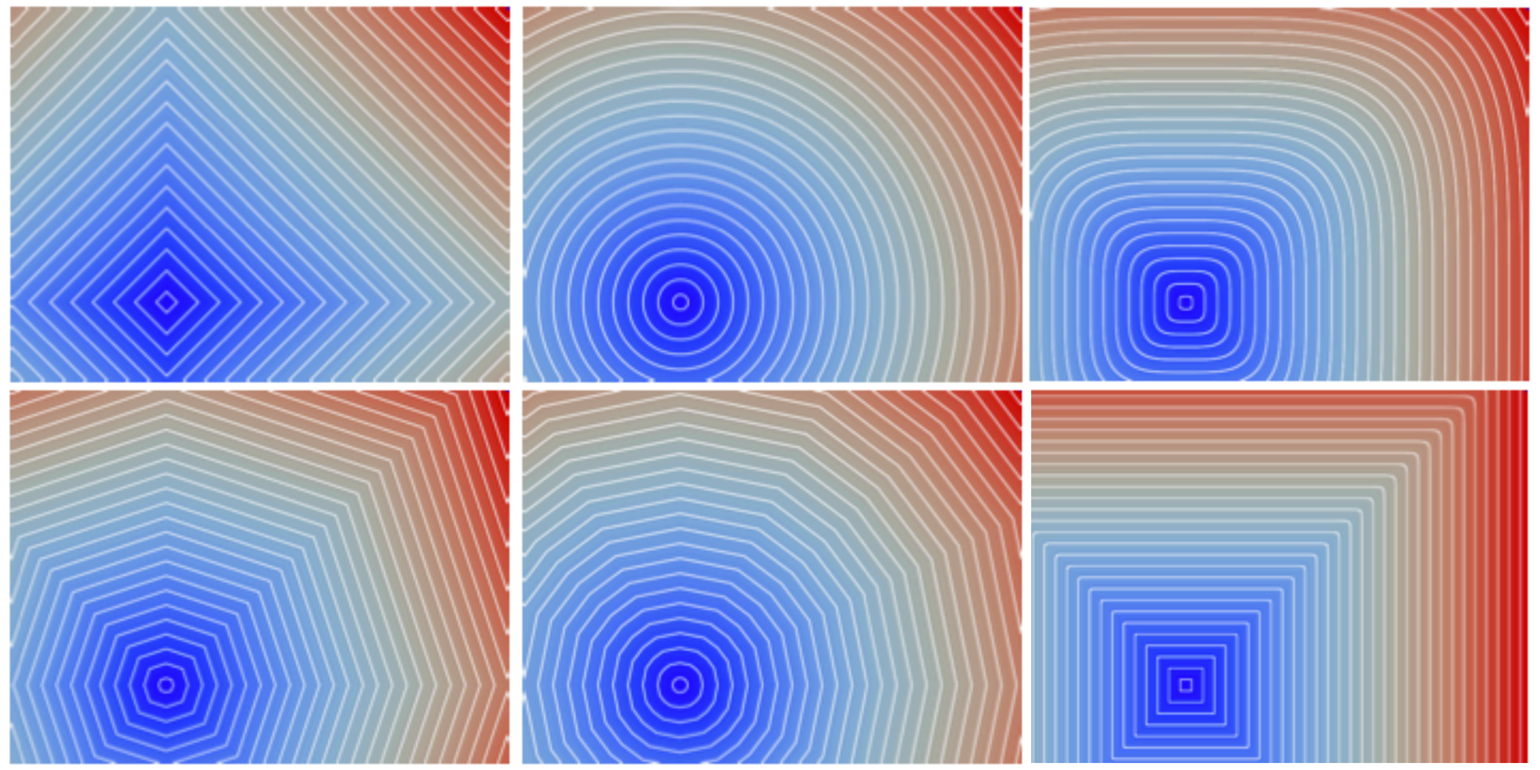
The algorithm is correct:

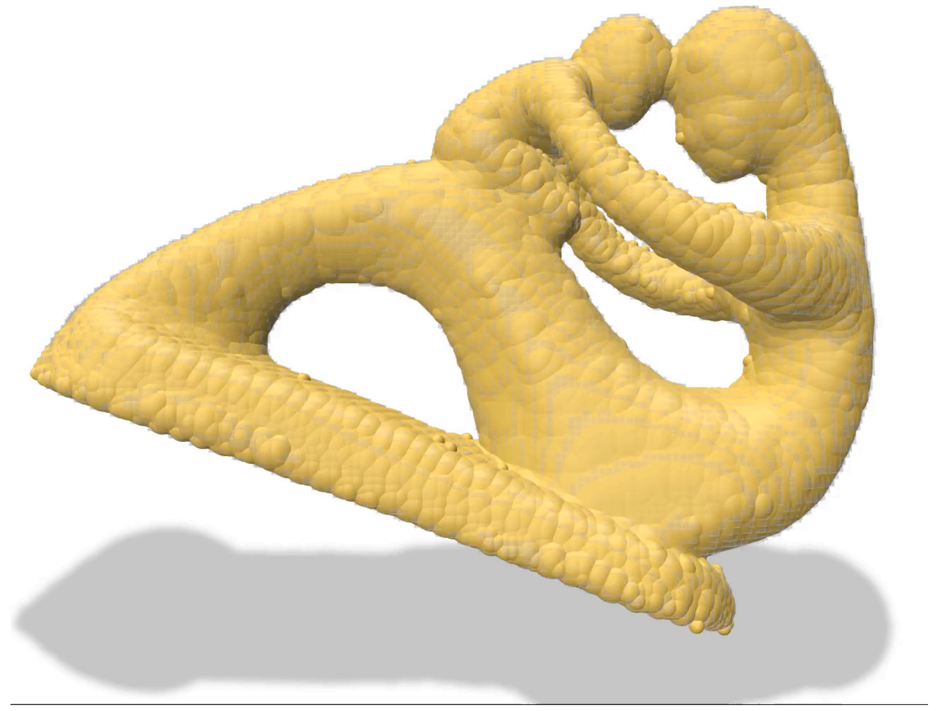
- for any dimension
- for any metric with axis symmetric unit ball (e.g. any l_p)
- on any toroidal nD domains

Exact and linear in time w.r.t. the number of grid points $O(d \cdot n^d)$ for l_2

$O(d^2 \cdot \log(p) \cdot \log(n) \cdot n^d)$ for exact l_p $(p \in \mathbb{Z}^+)$, $O(d \cdot n^d)$ approx.

Trivial multithread / GPU / out-of-core implementations





Separable approaches

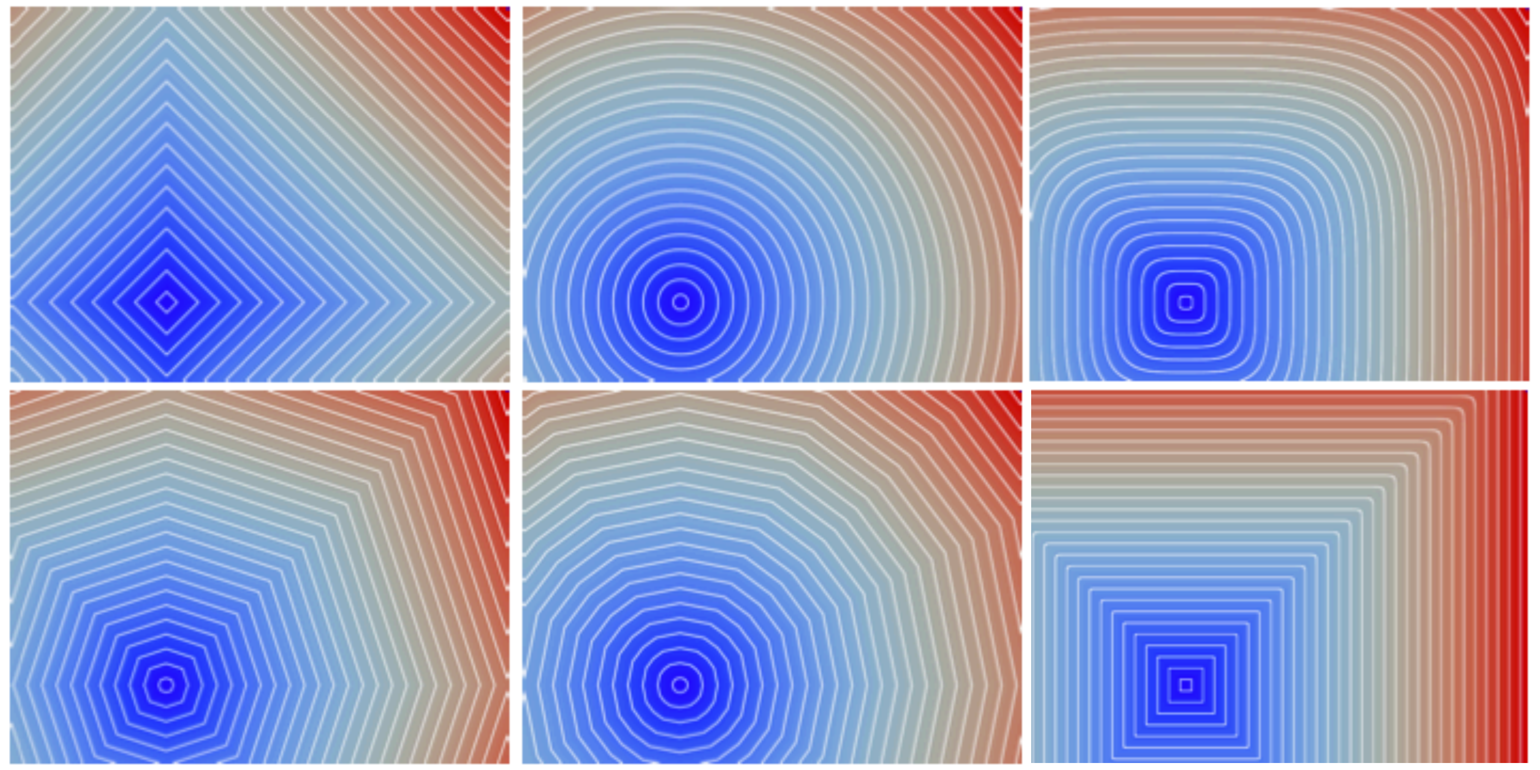
The algorithm is correct:

- for any dimension
- for any metric with axis symmetric unit ball (e.g. any l_p)
- on any toroidal nD domains

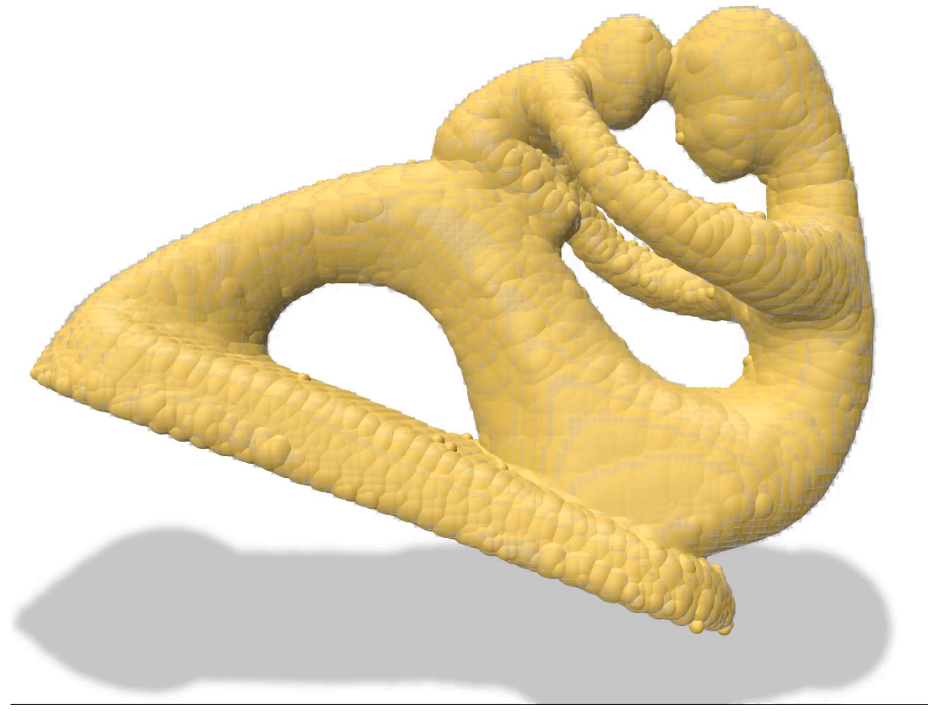
Exact and linear in time w.r.t. the number of grid points $O(d \cdot n^d)$ for l_2

$O(d^2 \cdot \log(p) \cdot \log(n) \cdot n^d)$ for exact l_p ($p \in \mathbb{Z}^+$), $O(d \cdot n^d)$ approx.

Trivial multithread / GPU / out-of-core implementations



Same techniques and computational costs for: [C. et al 07]



Separable approaches

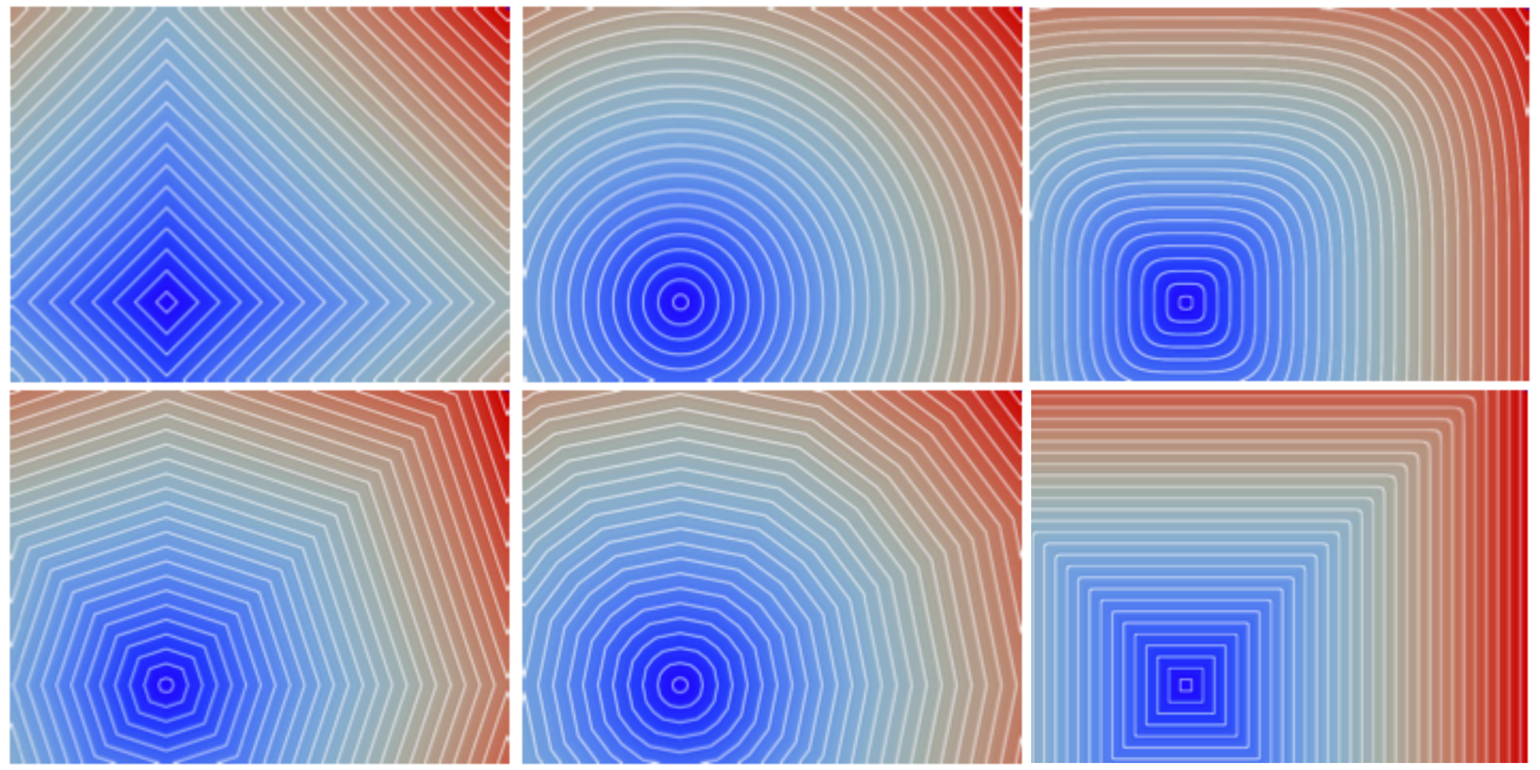
The algorithm is correct:

- for any dimension
- for any metric with axis symmetric unit ball (e.g. any l_p)
- on any toroidal nD domains

Exact and linear in time w.r.t. the number of grid points $O(d \cdot n^d)$ for l_2

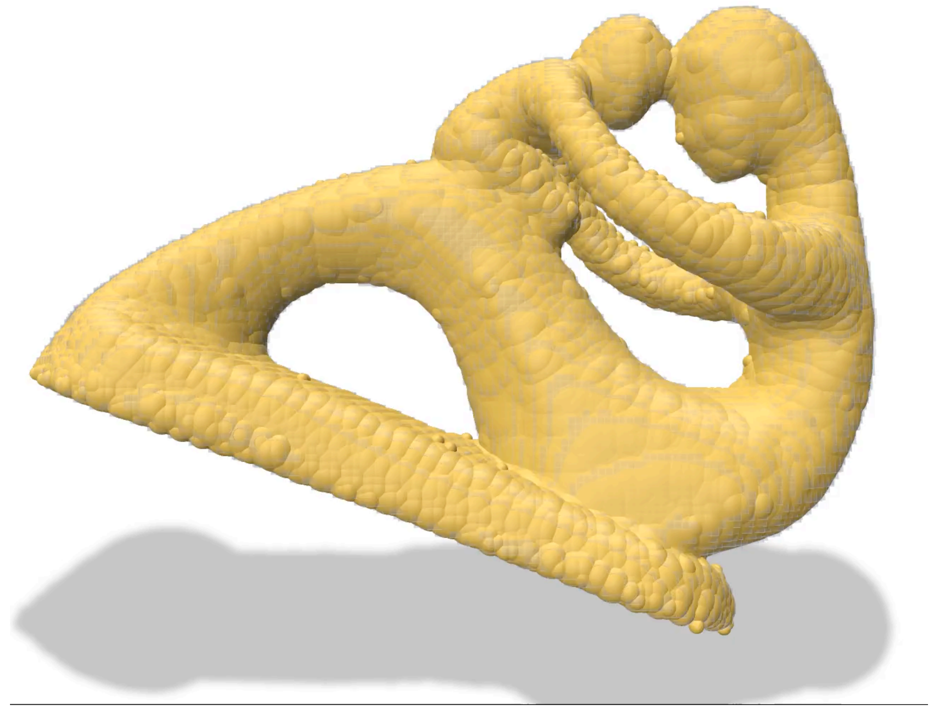
$O(d^2 \cdot \log(p) \cdot \log(n) \cdot n^d)$ for exact l_p ($p \in \mathbb{Z}^+$), $O(d \cdot n^d)$ approx.

Trivial multithread / GPU / out-of-core implementations

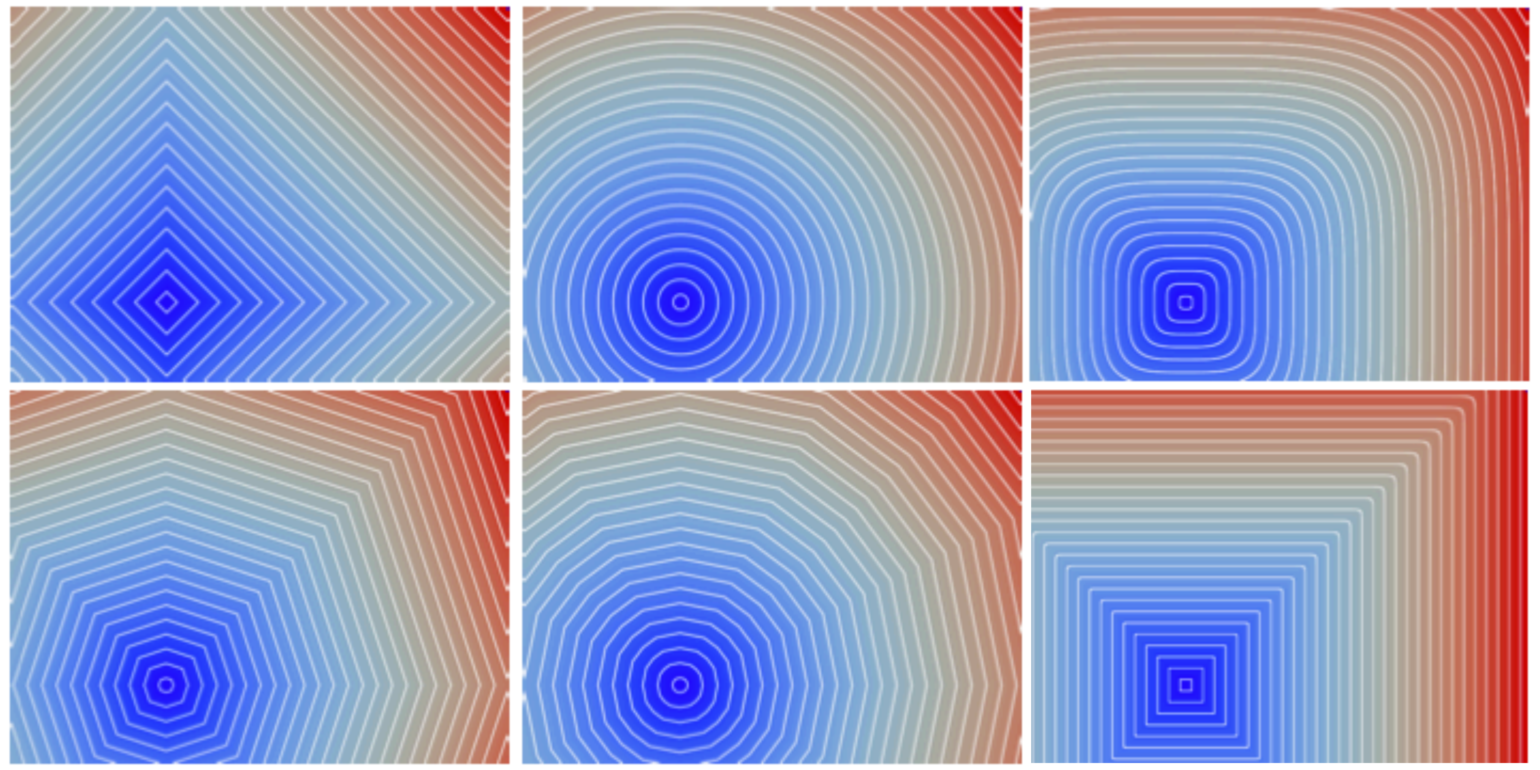


Same techniques and computational costs for: [C. et al 07]

- Power diagram / power maps construction



Separable approaches



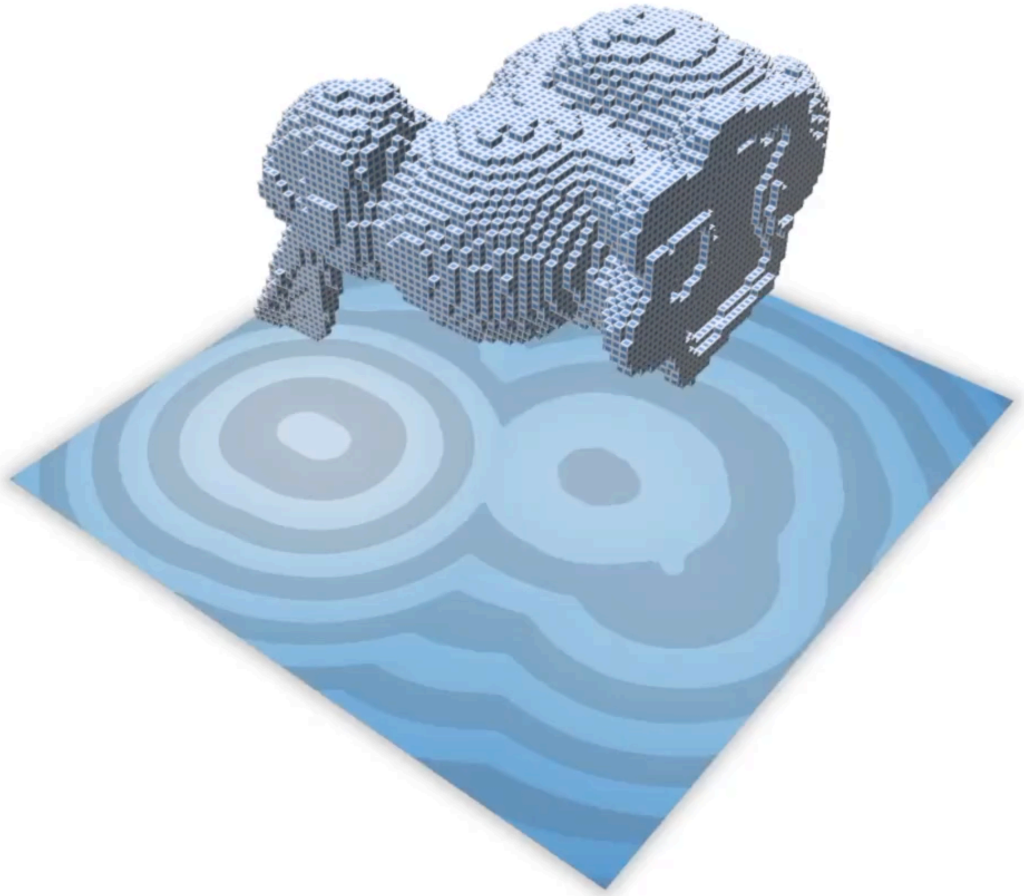
The algorithm is correct:

- for any dimension
- for any metric with axis symmetric unit ball (e.g. any l_p)
- on any toroidal nD domains

Exact and linear in time w.r.t. the number of grid points $O(d \cdot n^d)$ for l_2

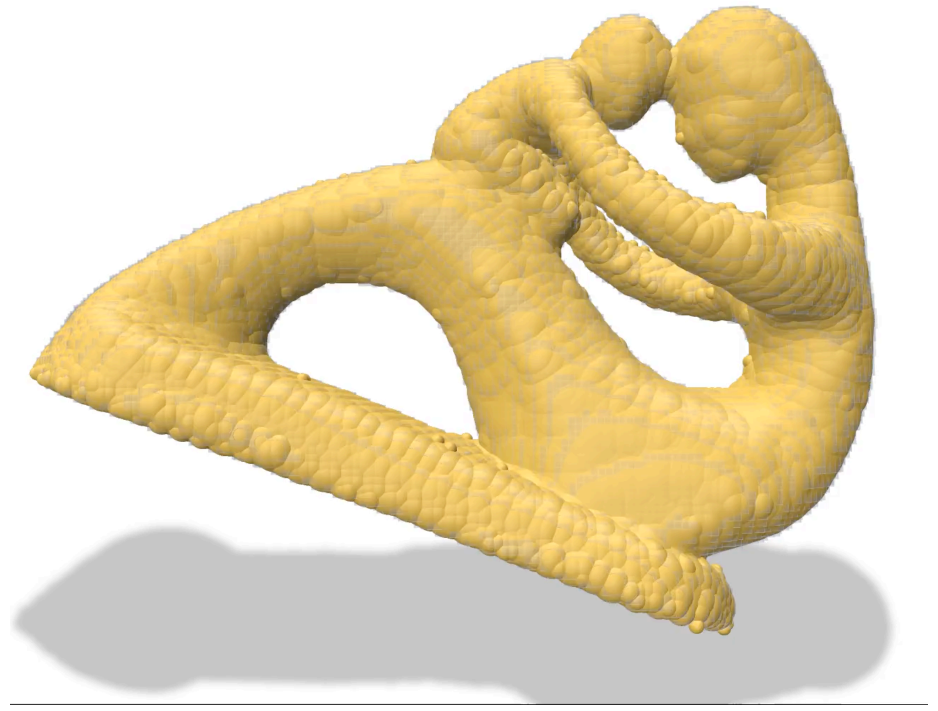
$O(d^2 \cdot \log(p) \cdot \log(n) \cdot n^d)$ for exact l_p $(p \in \mathbb{Z}^+)$, $O(d \cdot n^d)$ approx.

Trivial multithread / GPU / out-of-core implementations



Same techniques and computational costs for: [C. et al 07]

- Power diagram / power maps construction
- Discrete Medial Axis extraction (aka non-empty inner power cells)



Separable approaches

The algorithm is correct:

- for any dimension
- for any metric with axis symmetric unit ball (e.g. any l_p)
- on any toroidal nD domains

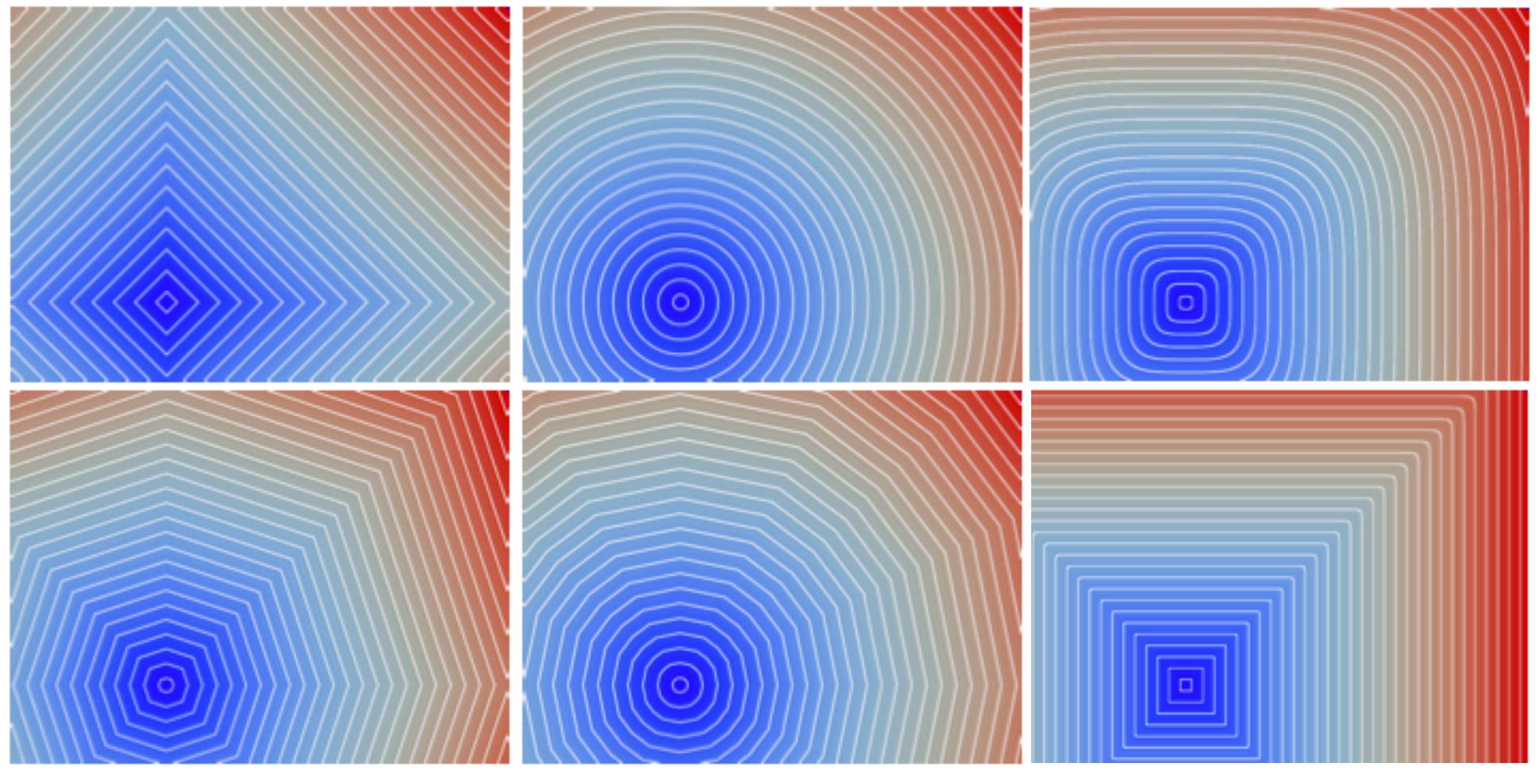
Exact and linear in time w.r.t. the number of grid points $O(d \cdot n^d)$ for l_2

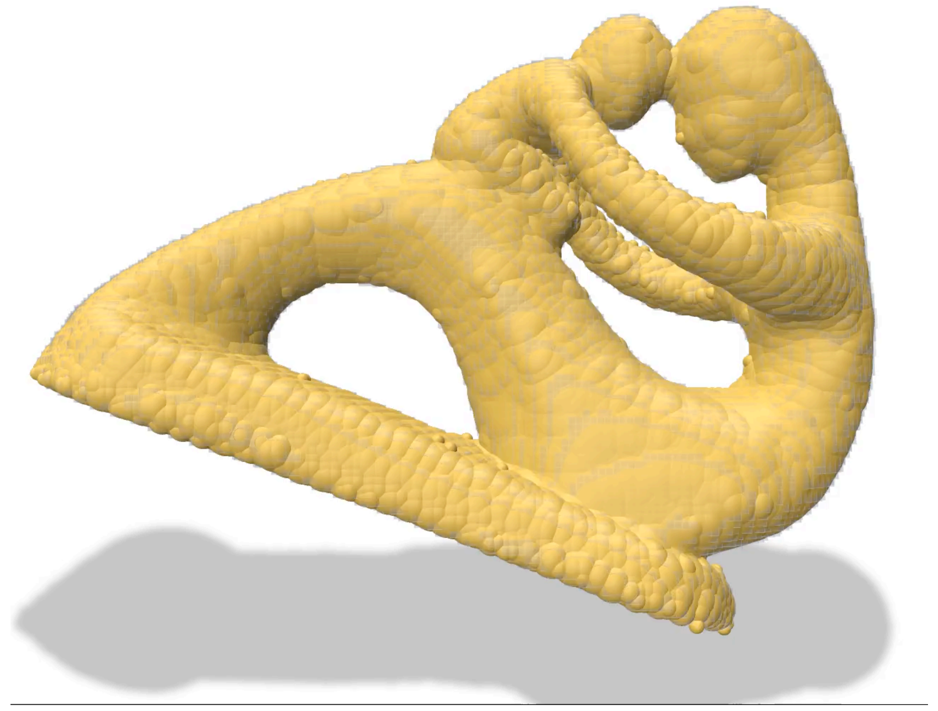
$O(d^2 \cdot \log(p) \cdot \log(n) \cdot n^d)$ for exact l_p ($p \in \mathbb{Z}^+$), $O(d \cdot n^d)$ approx.

Trivial multithread / GPU / out-of-core implementations

Same techniques and computational costs for: [C. et al 07]

- Power diagram / power maps construction
- Discrete Medial Axis extraction (aka non-empty inner power cells)
- Reverse reconstruction (balls→shape)





Separable approaches

The algorithm is correct:

- for any dimension
- for any metric with axis symmetric unit ball (e.g. any l_p)
- on any toroidal nD domains

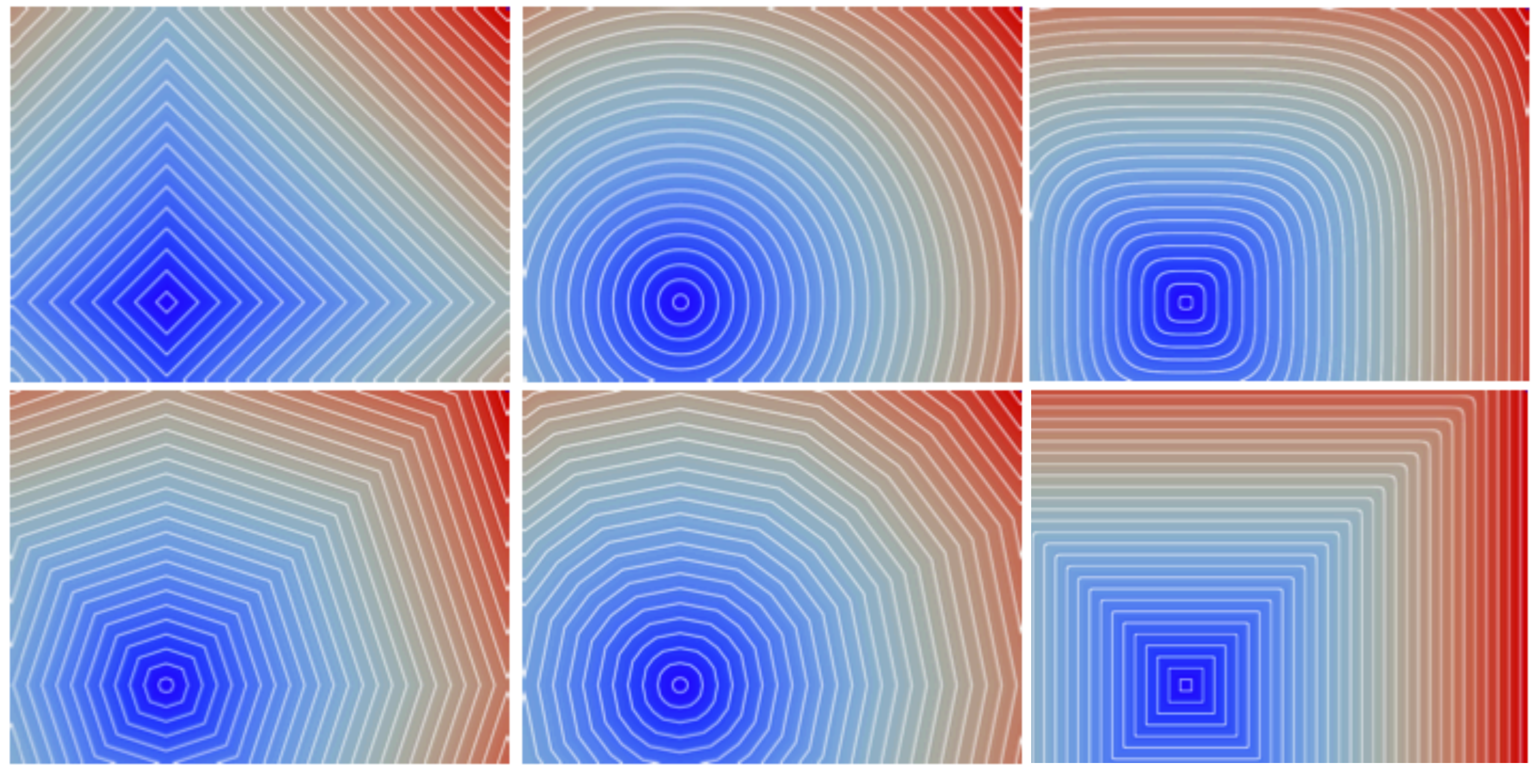
Exact and linear in time w.r.t. the number of grid points $O(d \cdot n^d)$ for l_2

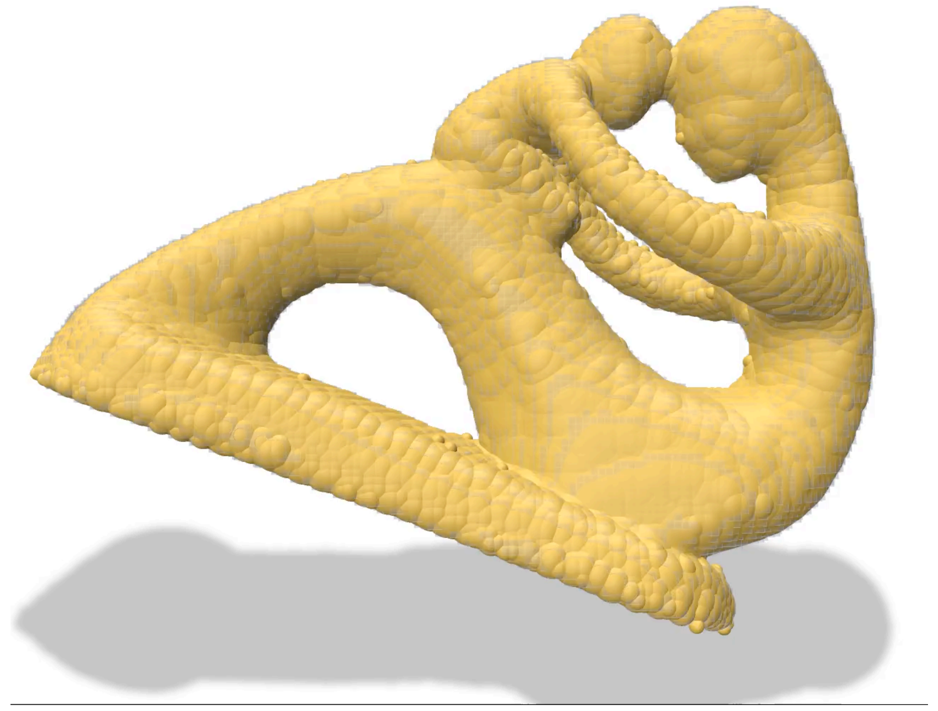
$O(d^2 \cdot \log(p) \cdot \log(n) \cdot n^d)$ for exact l_p ($p \in \mathbb{Z}^+$), $O(d \cdot n^d)$ approx.

Trivial multithread / GPU / out-of-core implementations

Same techniques and computational costs for: [C. et al 07]

- Power diagram / power maps construction
- Discrete Medial Axis extraction (aka non-empty inner power cells)
- Reverse reconstruction (balls→shape)





Separable approaches

The algorithm is correct:

- for any dimension
- for any metric with axis symmetric unit ball (e.g. any l_p)
- on any toroidal nD domains

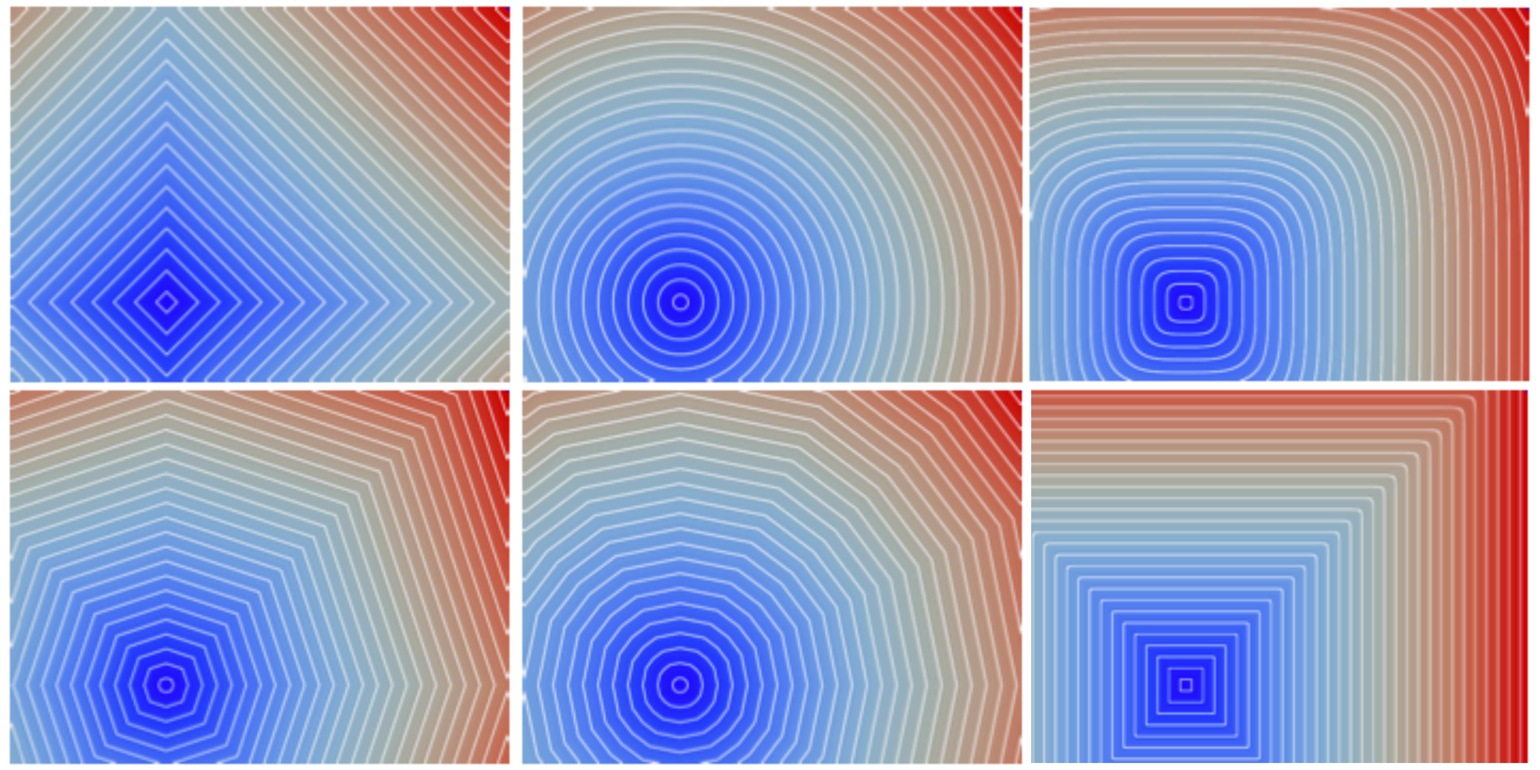
Exact and linear in time w.r.t. the number of grid points $O(d \cdot n^d)$ for l_2

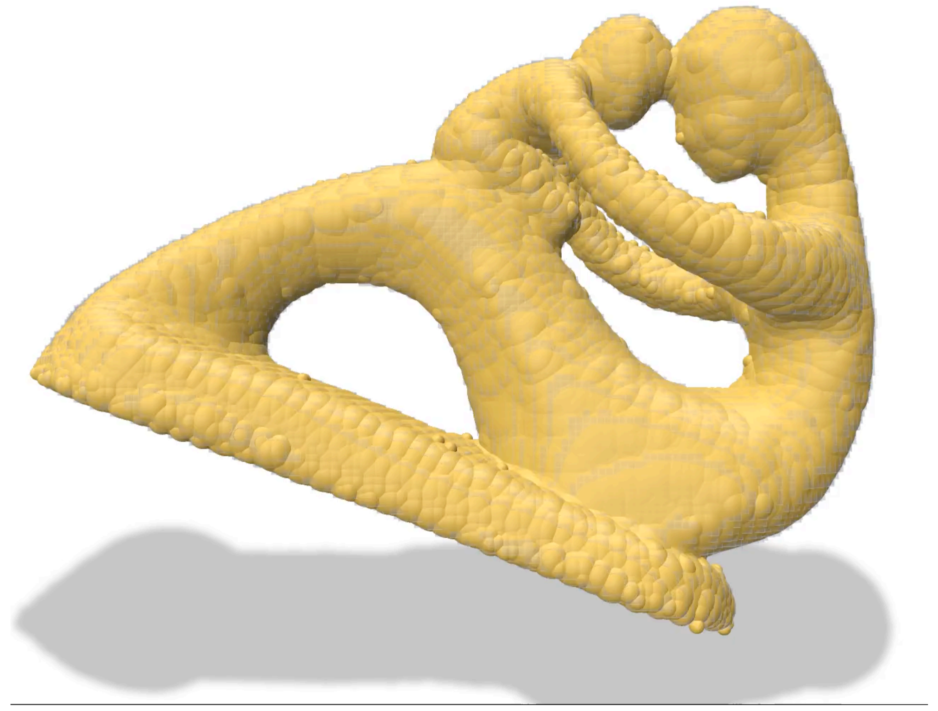
$O(d^2 \cdot \log(p) \cdot \log(n) \cdot n^d)$ for exact l_p ($p \in \mathbb{Z}^+$), $O(d \cdot n^d)$ approx.

Trivial multithread / GPU / out-of-core implementations

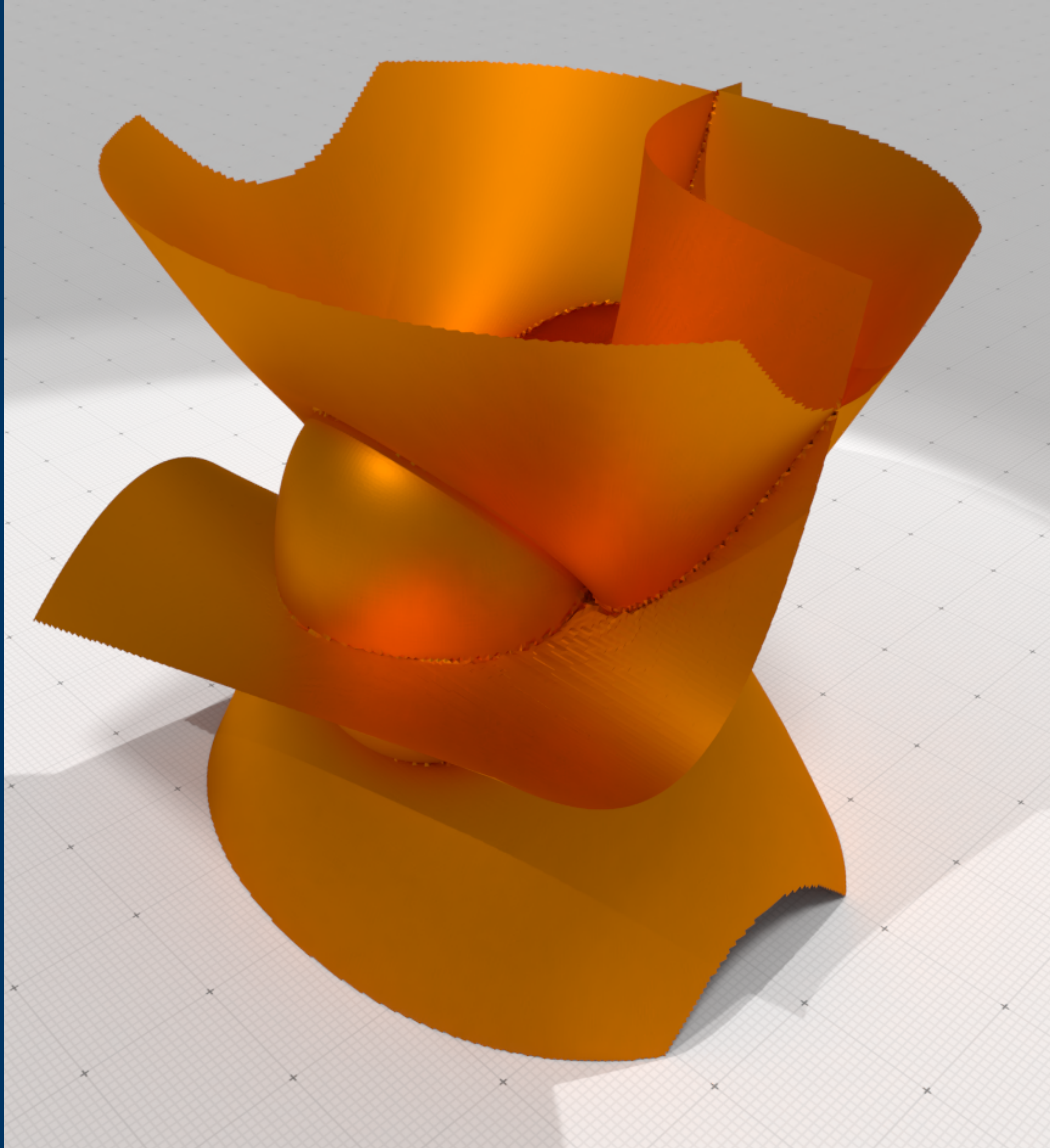
Same techniques and computational costs for: [C. et al 07]

- Power diagram / power maps construction
- Discrete Medial Axis extraction (aka non-empty inner power cells)
- Reverse reconstruction (balls→shape)



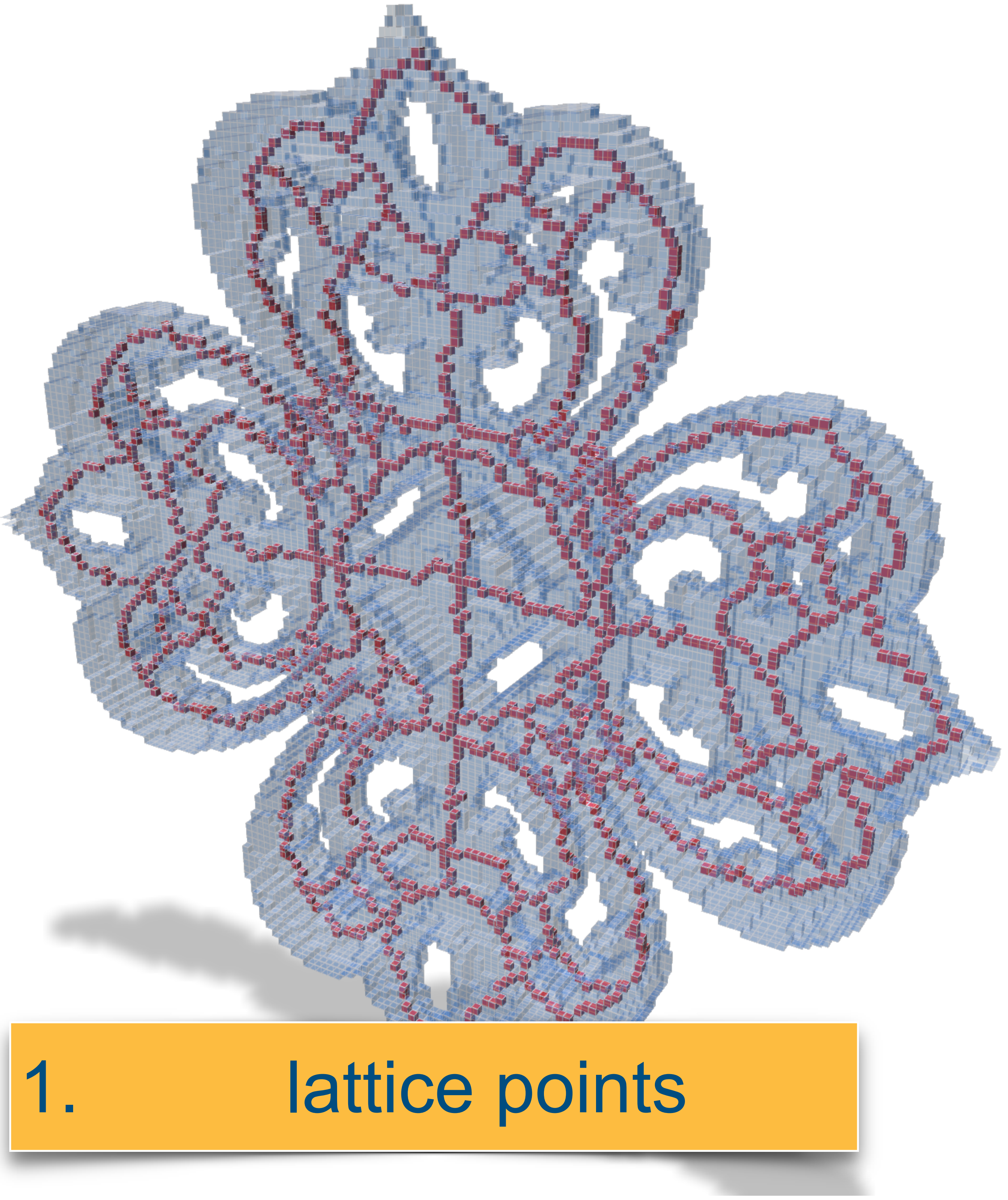


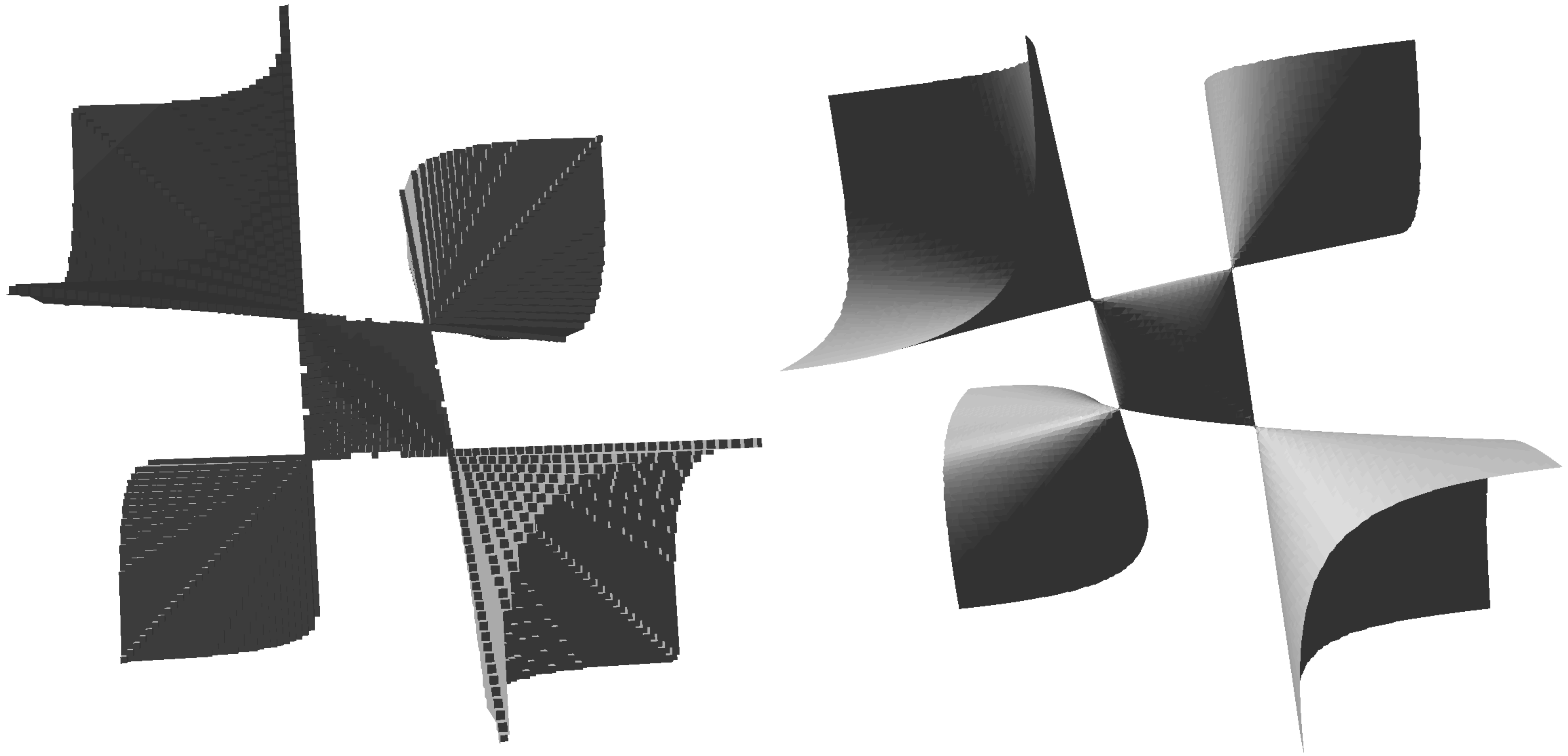
topology in $\mathbb{Z}^d$



Before geometry : topological models for $\mathbb{Z}^d$

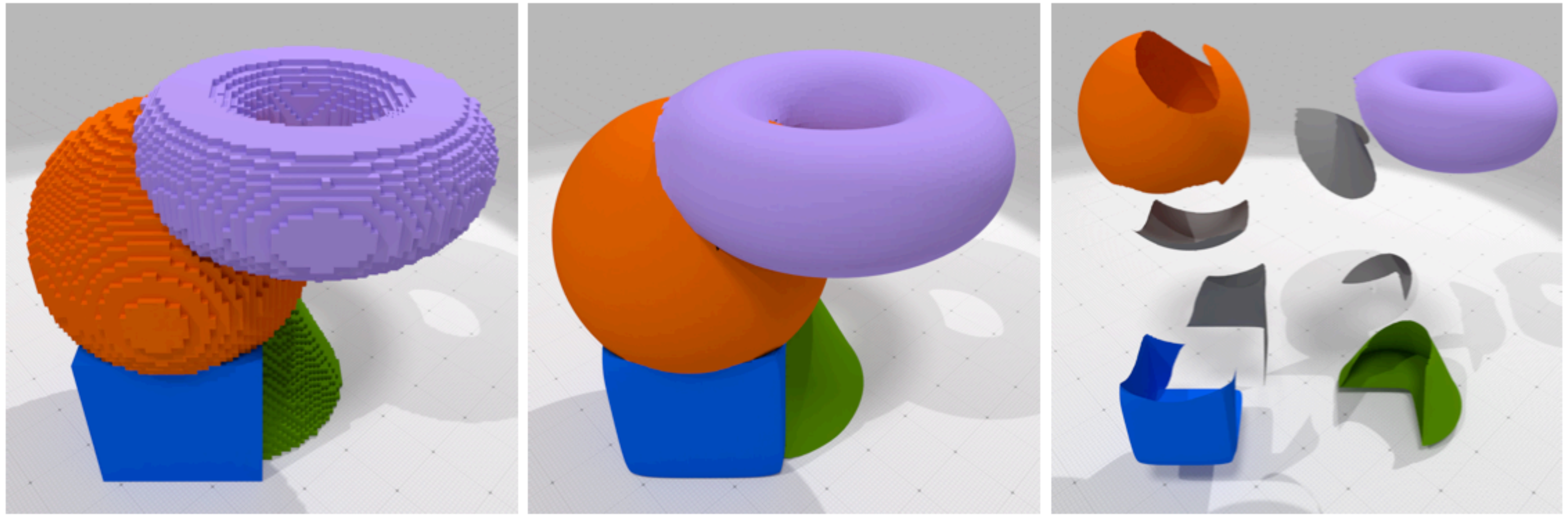
How to represent volumes, boundaries, curves, surfaces, partitions ?



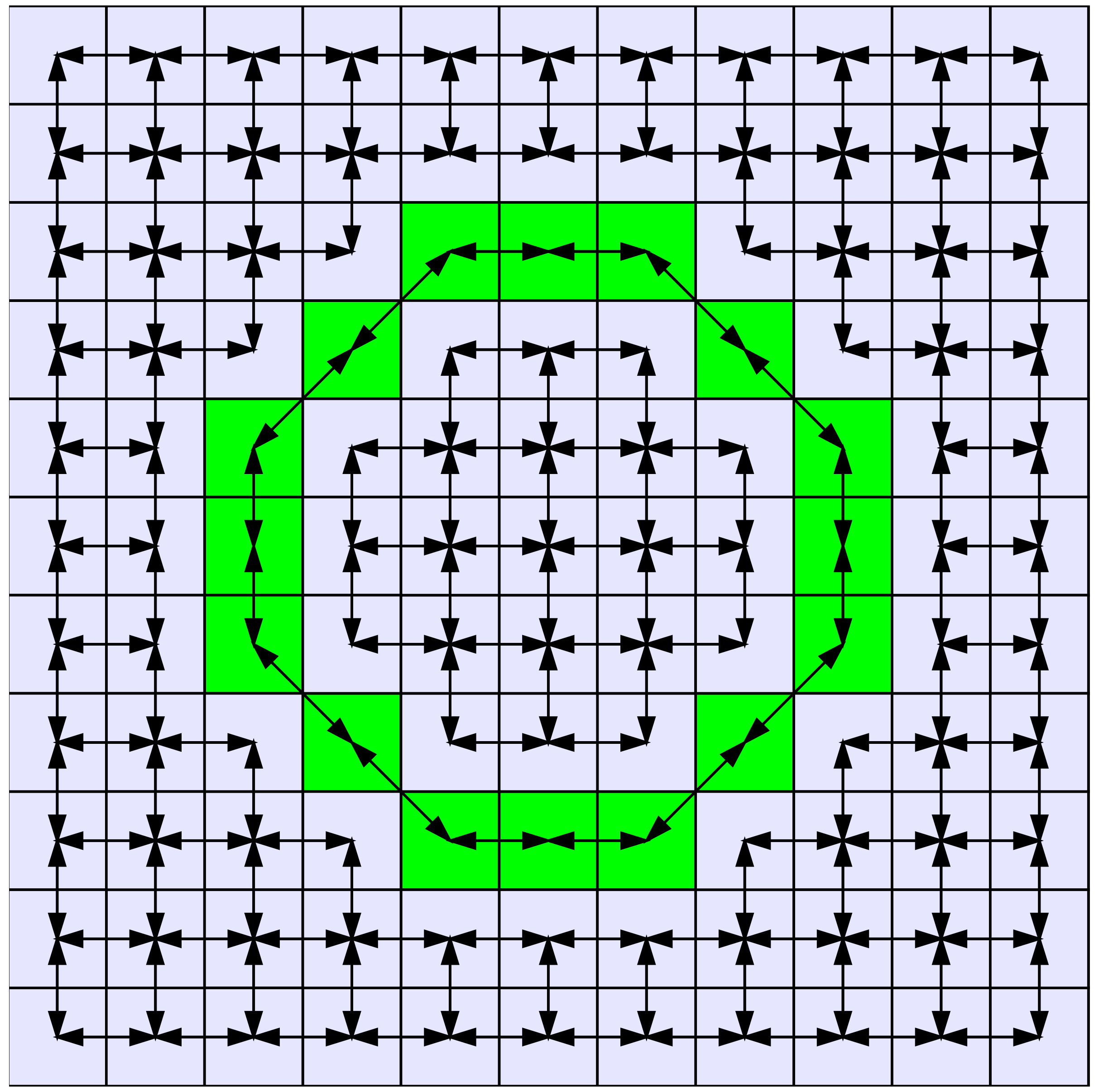


1. lattice points

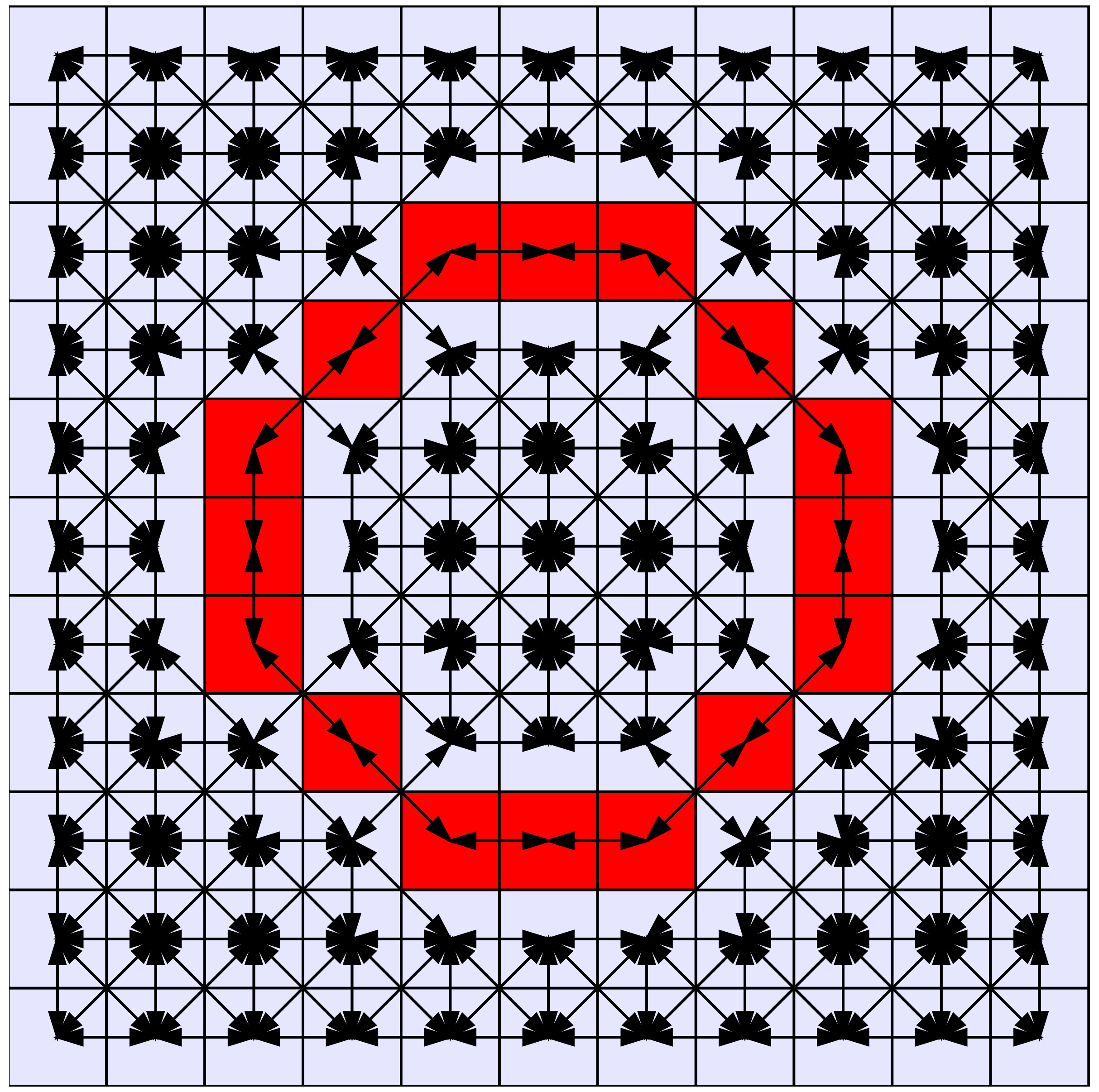
2. cubical complexes



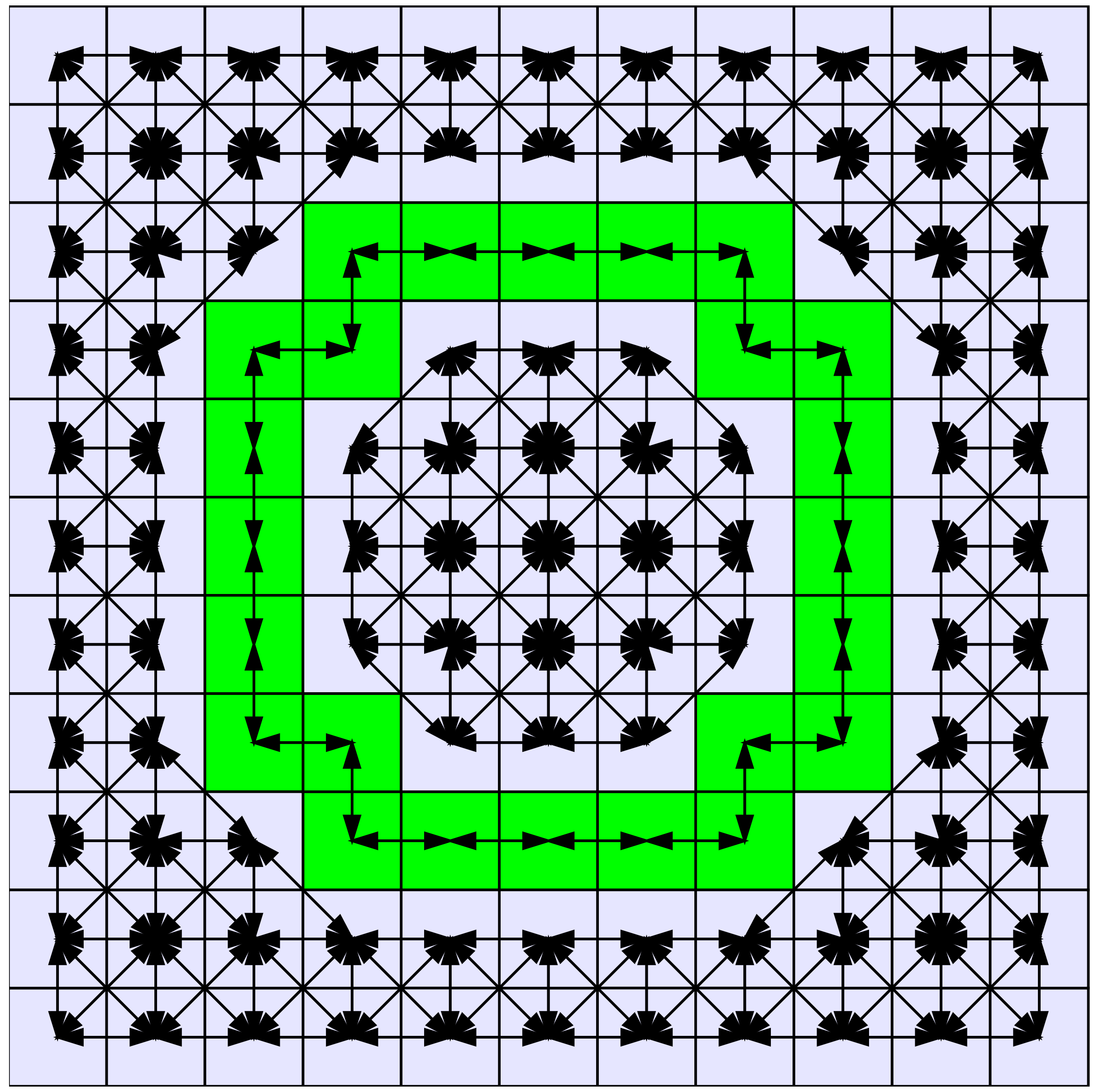
Digital topology



(8,4)-topology



(8,8)-topology



(4,8)-topology

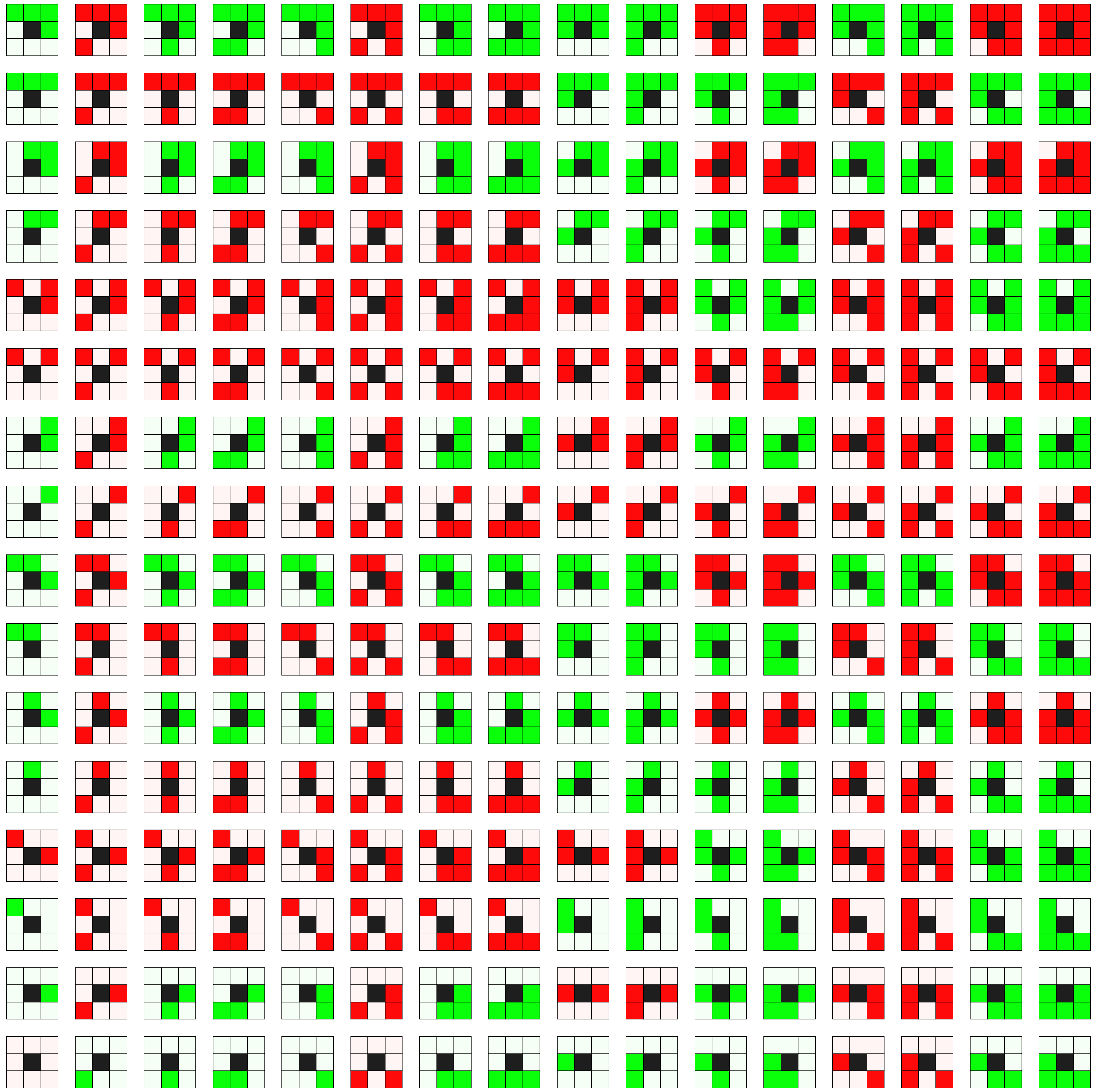
Good adjacencies for object/background

- Jordan separation theorem
- consistence borders and interior components
- definition of surfaces in $\mathbb{Z}^d$

Homotopy equivalence

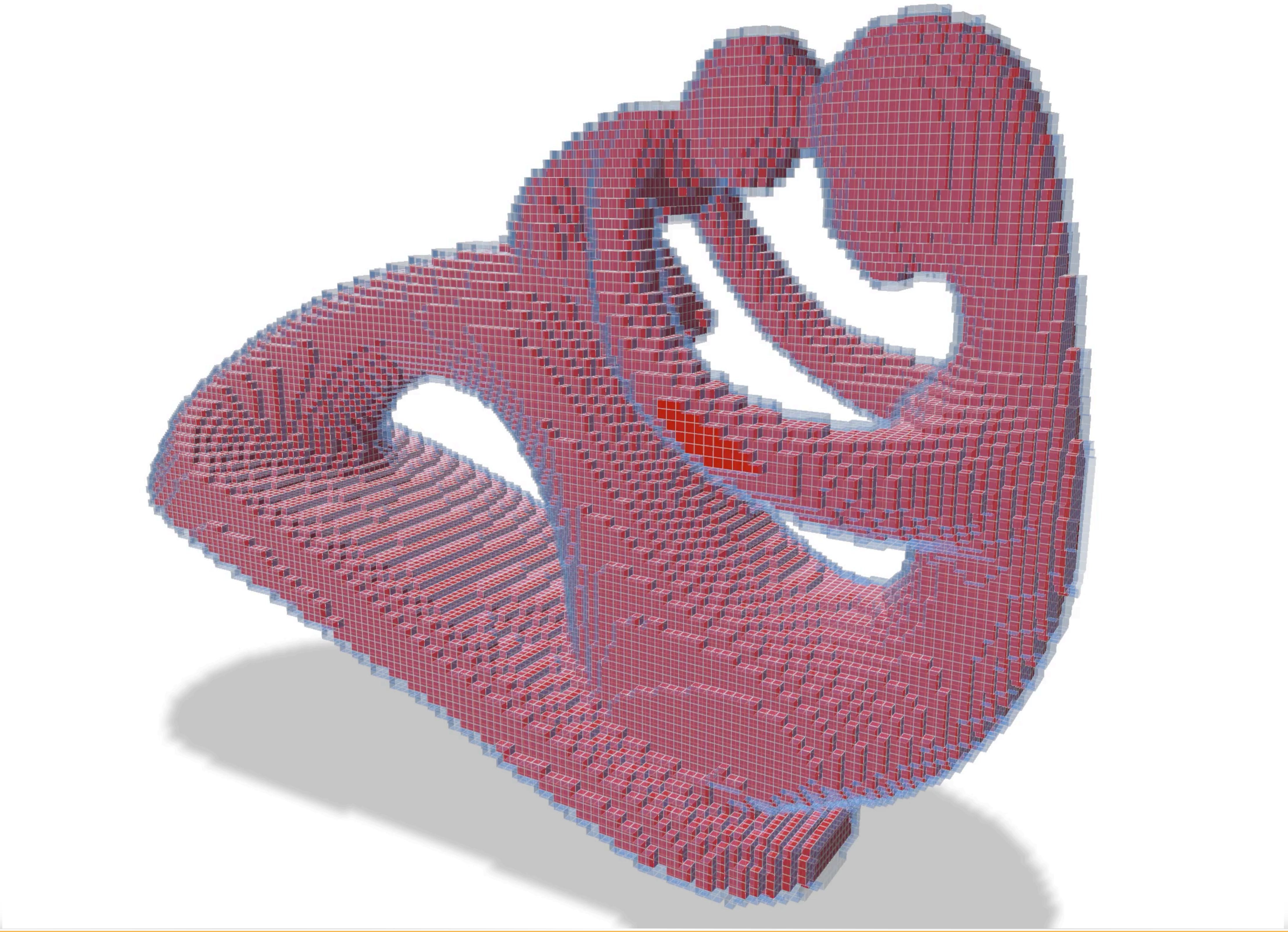
Homotopy equivalence

Topology invariance: simple points



(8,4)-topology

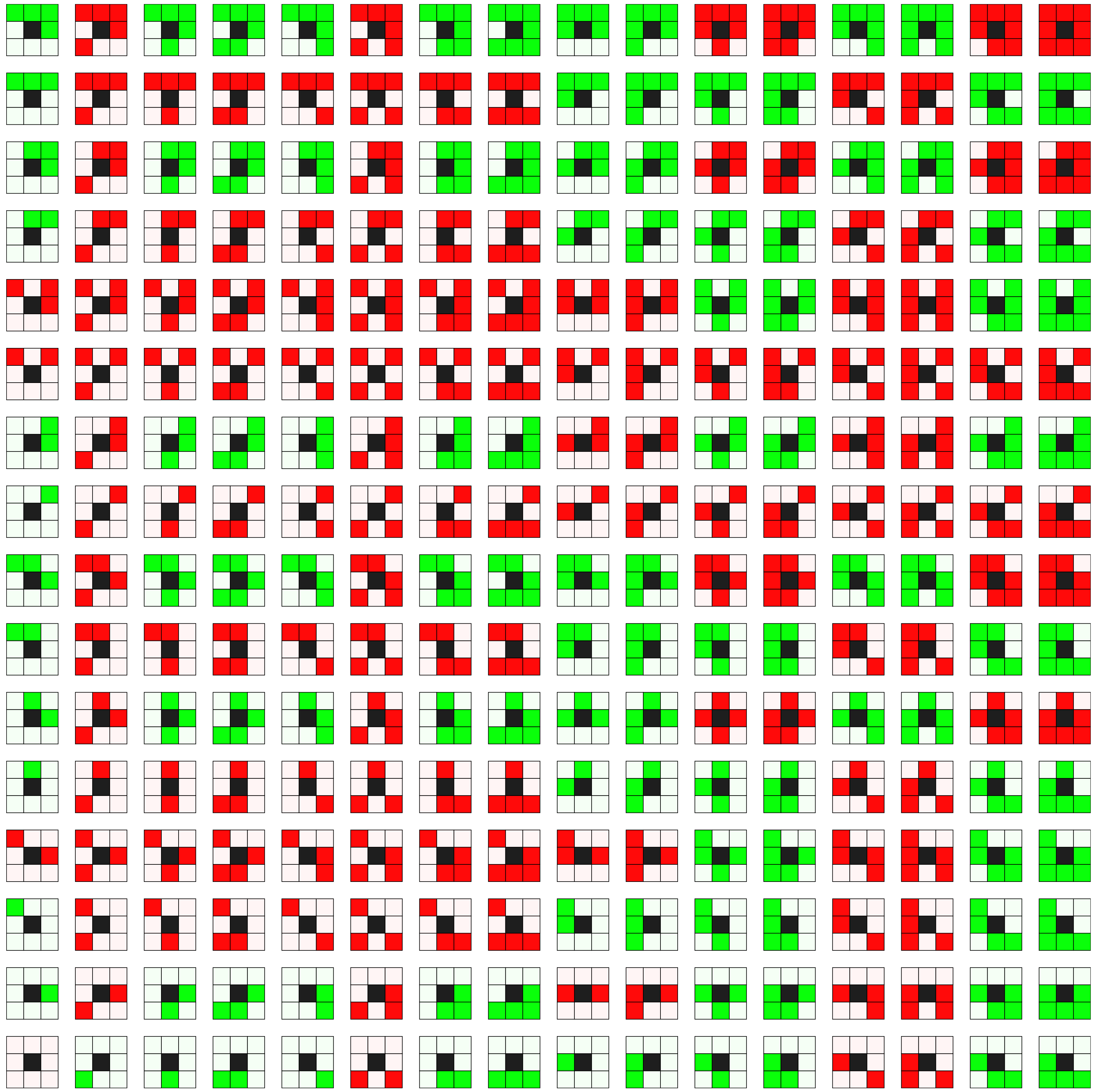
locally keep connected components



Simple points: points whose removal preserves topology

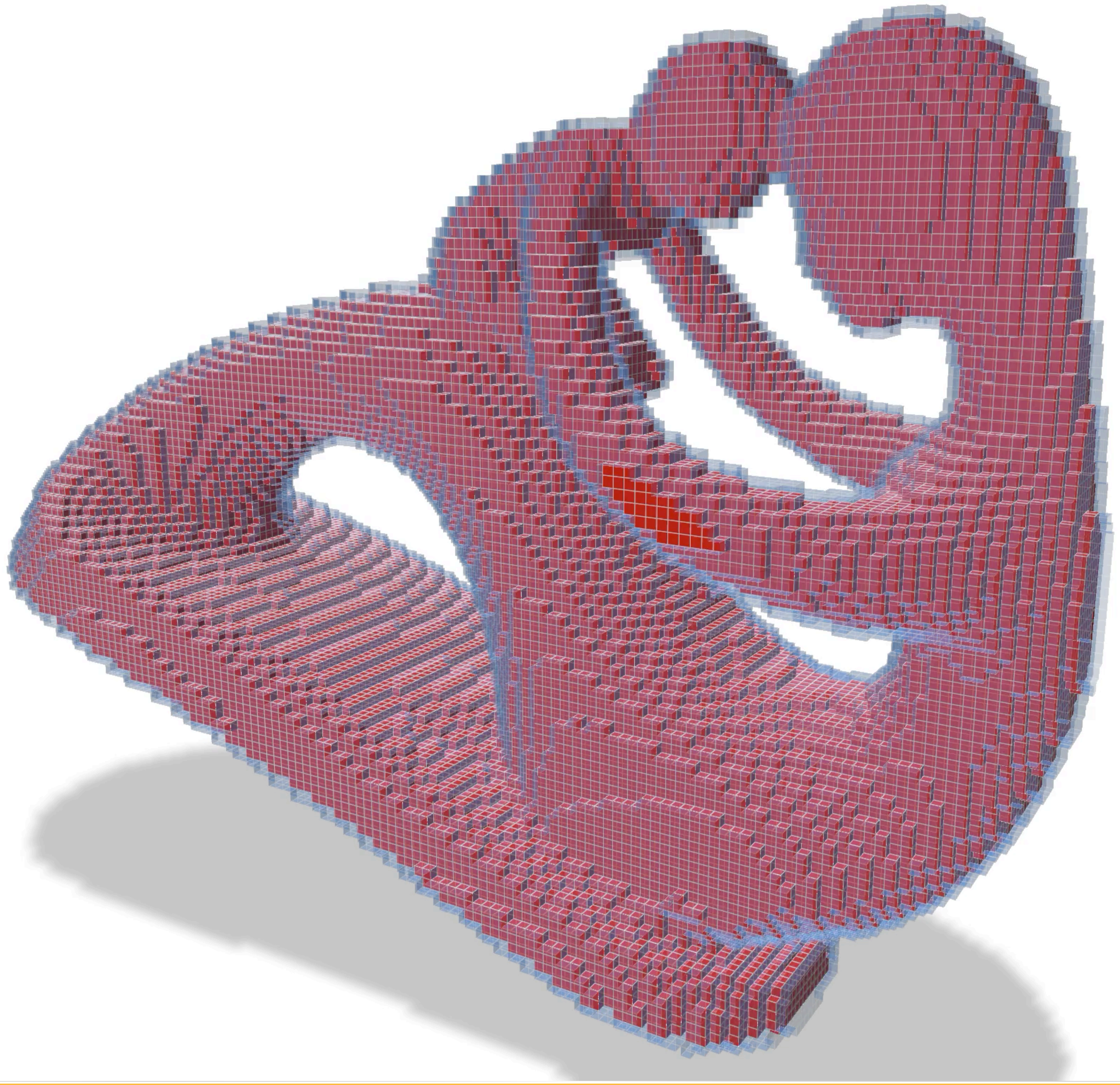
- digital topology invariance of object and background
- very fast: look-up tables in 2D and 3D
- useful for skeleton extraction / coupled with medial axis

Topology invariance: simple points



(8,4)-topology

locally keep connected components



Simple points: points whose removal preserves topology

- digital topology invariance of object and background
- very fast: look-up tables in 2D and 3D
- useful for skeleton extraction / coupled with medial axis

hands on…

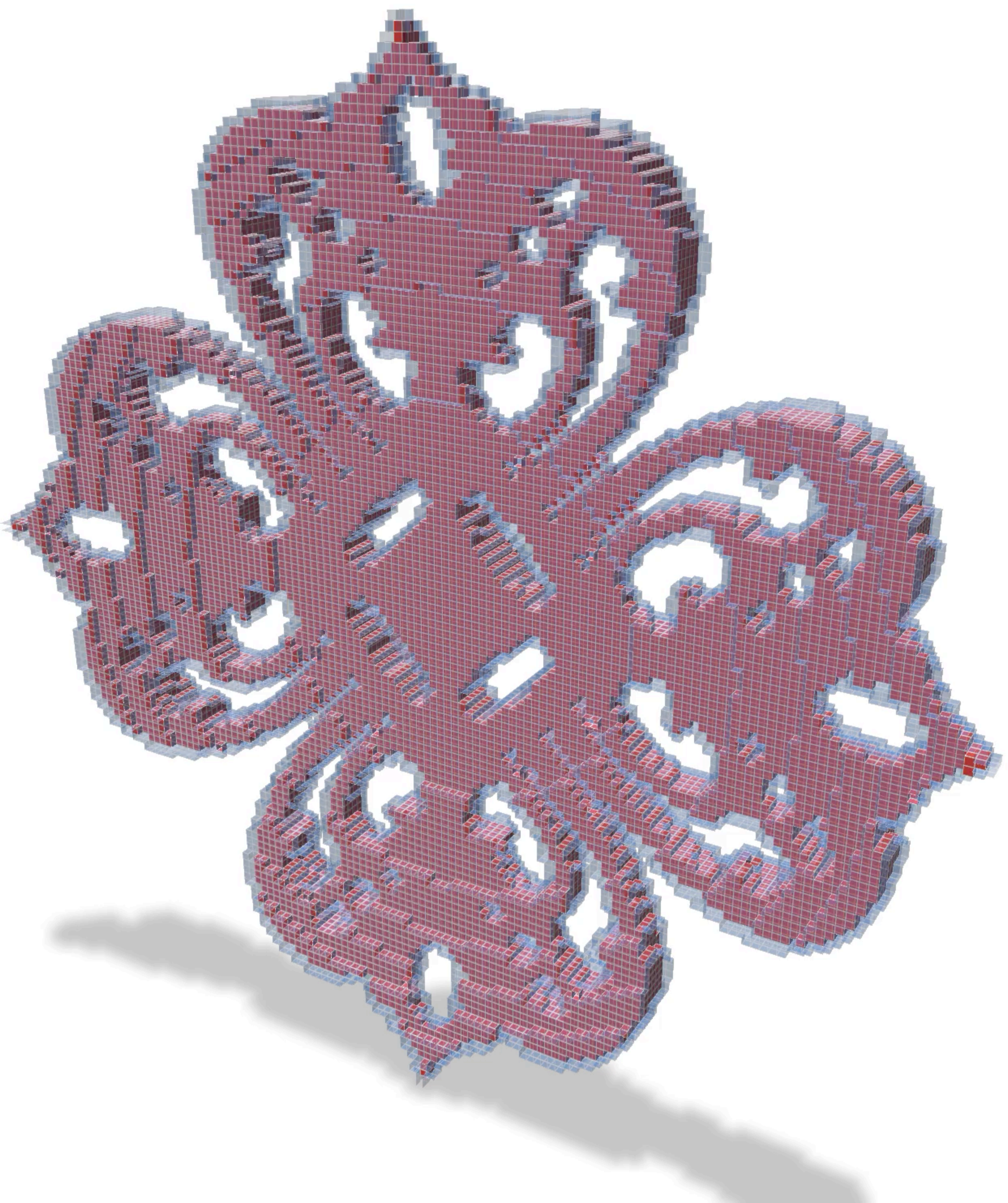
```cpp
// Build object with digital topology
const auto K = SH3::getKSpace( binary_image );
Domain domain( K.lowerBound(), K.upperBound() );
Z3i::DigitalSet voxel_set( domain );
for ( auto p : domain )
  if ( (*binary_image)( p ) ) voxel_set.insertNew( p );
the_object = CountedPtr< Z3i::Object26_6 >( new Z3i::Object26_6( dt26_6, voxel_set ) );
the_object->setTable(functions::loadTable<3>(simplicity::tableSimple26_6));
```

Create object with (26,6) topology from binary image

```cpp
// Removes a peel of simple points onto voxel object.
bool oneStep( CountedPtr< Z3i::Object26_6 > object )
{
  DigitalSet & S = object->pointSet();
  std::queue< Point > Q;
  for ( auto&& p : S )
    if ( object->isSimple( p ) )
      Q.push( p );
  int nb_simple = 0;
  while ( ! Q.empty() )
    {
      const auto p = Q.front();
      Q.pop();
      if ( object->isSimple( p ) )
        {
          S.erase( p );
          binary_image->setValue( p, false );
          ++nb_simple;
        }
    }
  trace.info() << "Removed " << nb_simple << " / " << S.size()
               << " points." << std::endl;
  registerDigitalSurface( binary_image, "Thinned object" );
  return nb_simple == 0;
}
```

Queue simple points

Remove simple points



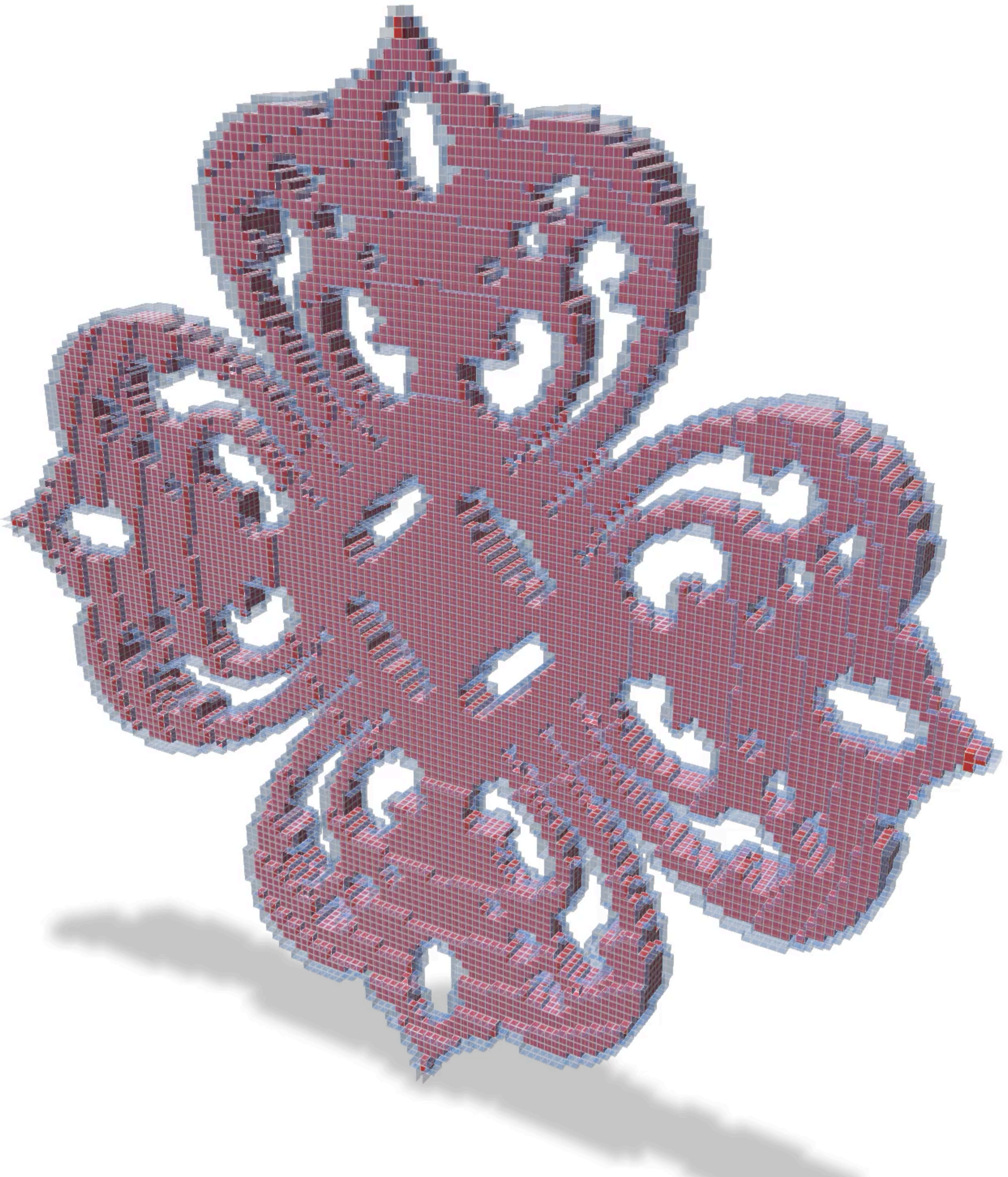
Create object with (26,6) topology from binary image

```
// Build object with digital topology
const auto K = SH3::getKSpace( binary_image );
Domain domain( K.lowerBound(), K.upperBound() );
Z3i::DigitalSet voxel_set( domain );
for ( auto p : domain )
  if ( (*binary_image)( p ) ) voxel_set.insertNew( p );
the_object = CountedPtr< Z3i::Object26_6 >( new Z3i::Object26_6( dt26_6, voxel_set ) );
the_object→setTable(functions::loadTable<3>(simplicity::tableSimple26_6));
```

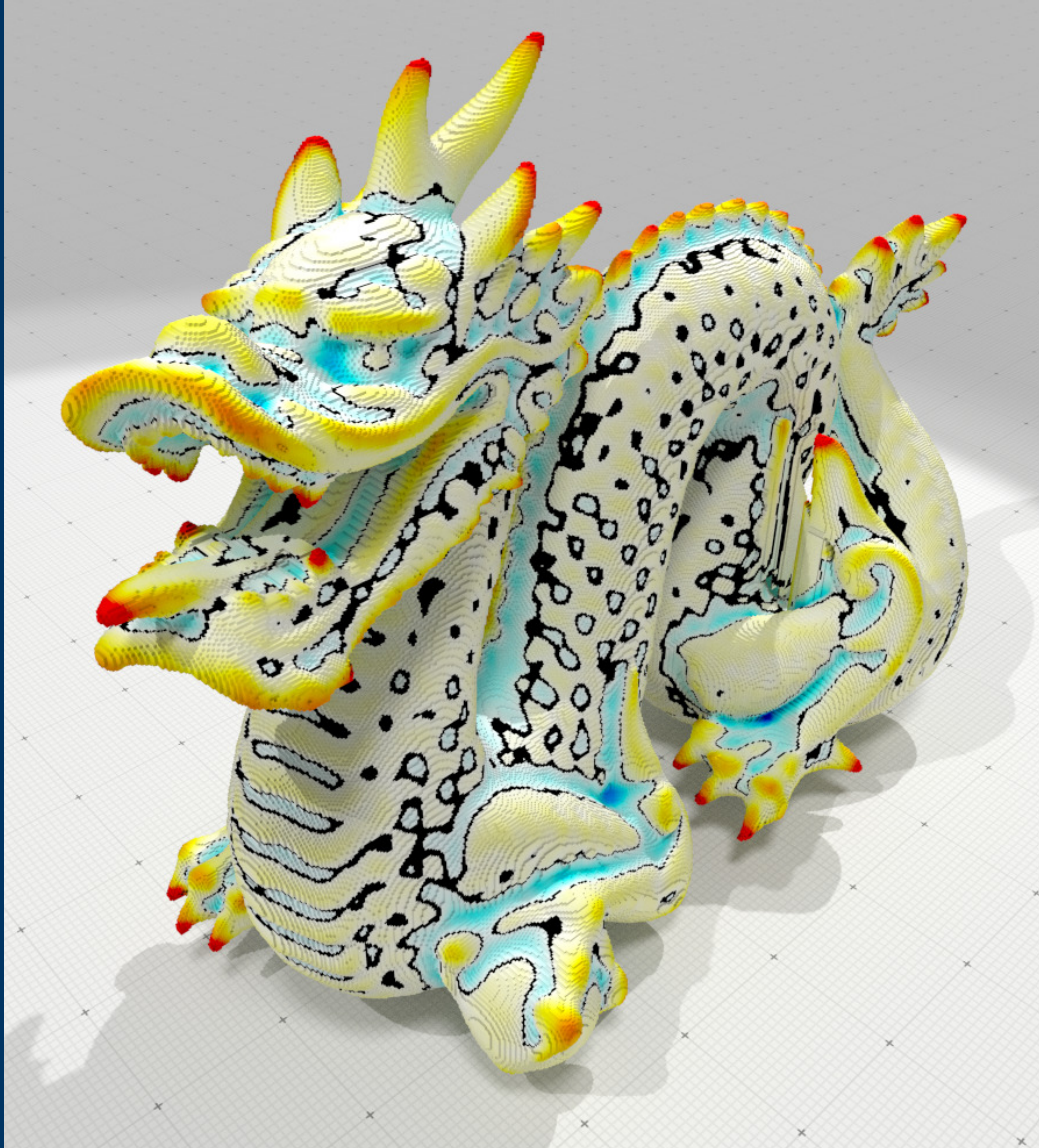
Queue simple points

Remove simple points

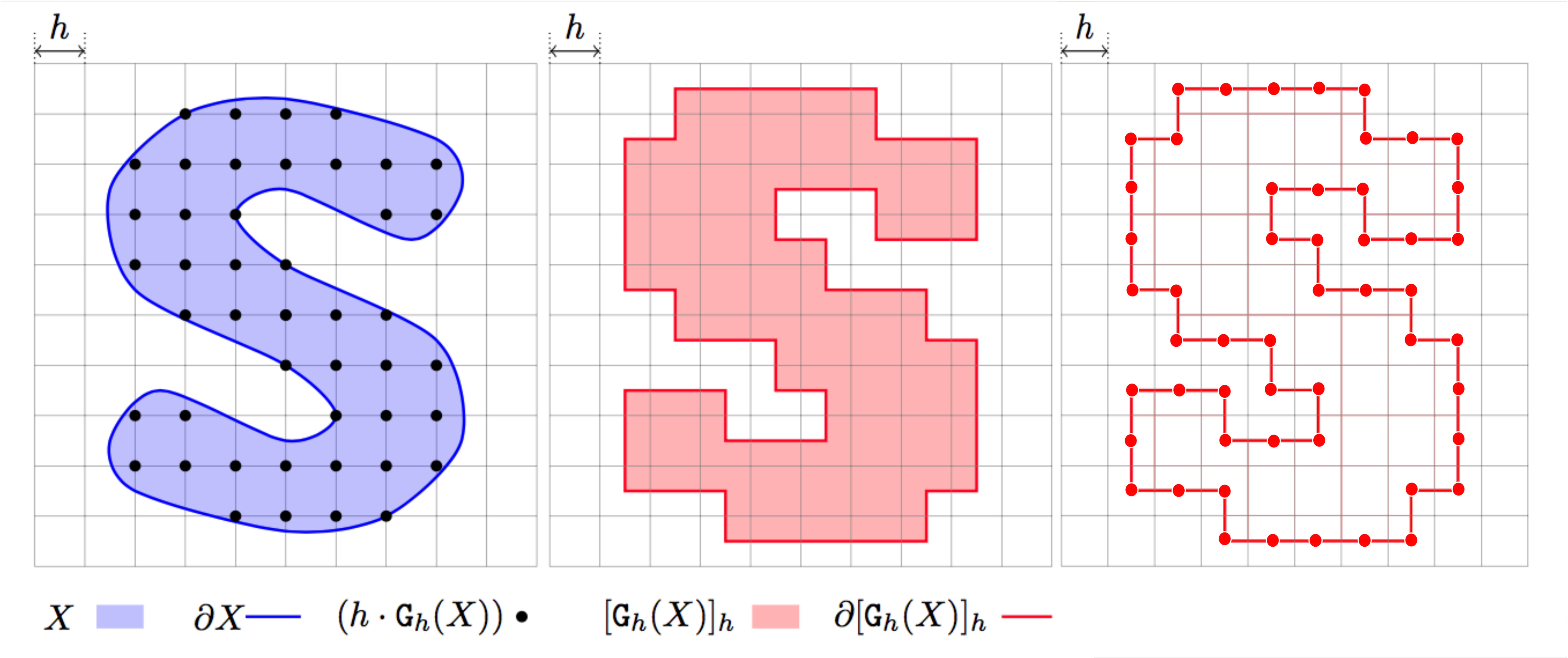
```
// Removes a peel of simple points onto voxel object.
bool oneStep( CountedPtr< Z3i::Object26_6 > object )
{
  DigitalSet & S = object→pointSet();
  std::queue< Point > Q;
  for ( auto&& p : S )
    if ( object→isSimple( p ) )
      Q.push( p );
  int nb_simple = 0;
  while ( ! Q.empty() )
    {
      const auto p = Q.front();
      Q.pop();
      if ( object→isSimple( p ) )
        {
          S.erase( p );
          binary_image→setValue( p, false );
          ++nb_simple;
        }
    }
  trace.info() << "Removed " << nb_simple << " / " << S.size()
               << " points." << std::endl;
  registerDigitalSurface( binary_image, "Thinned object" );
  return nb_simple == 0;
}
```



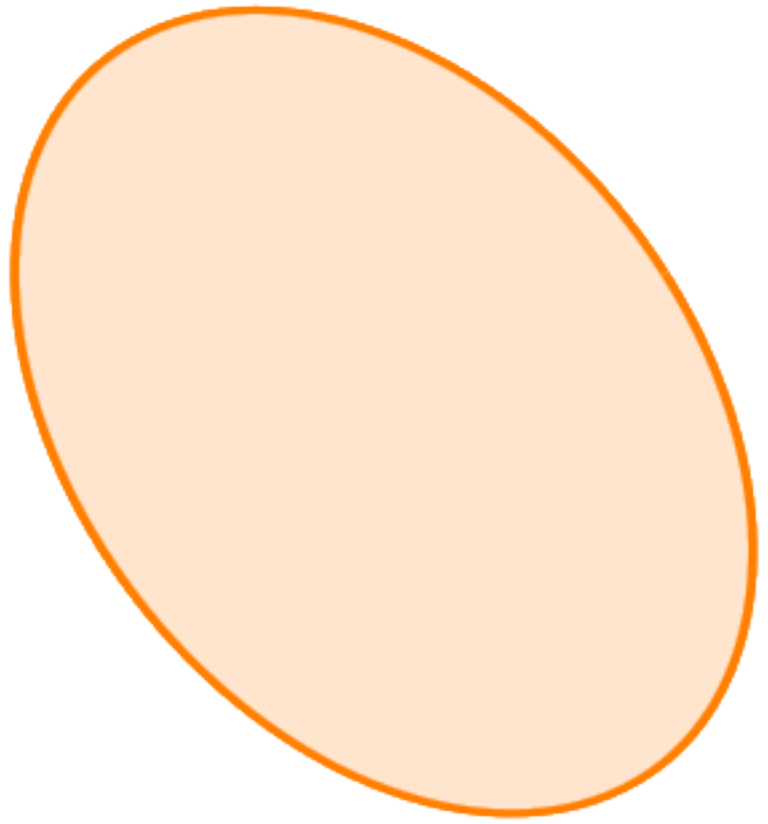
digital surface geometry

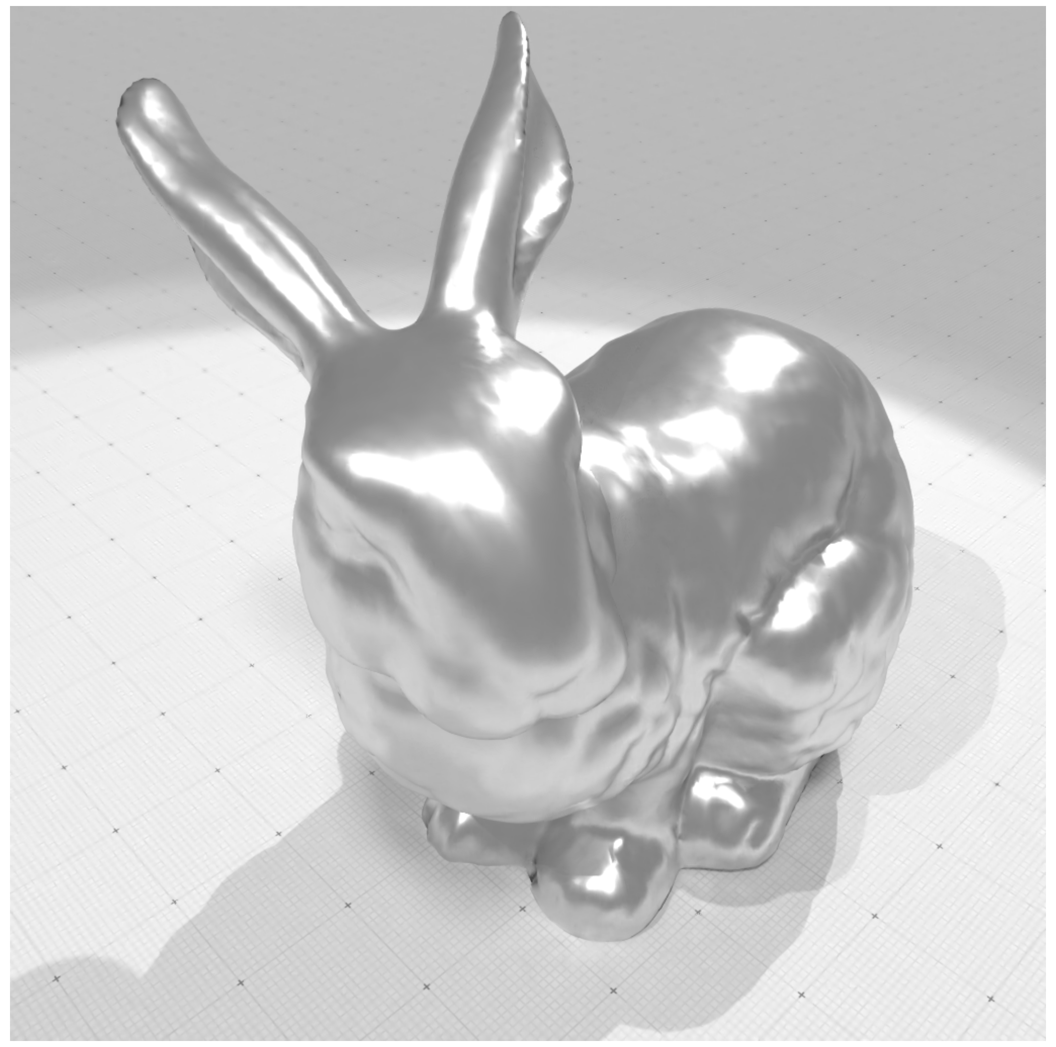


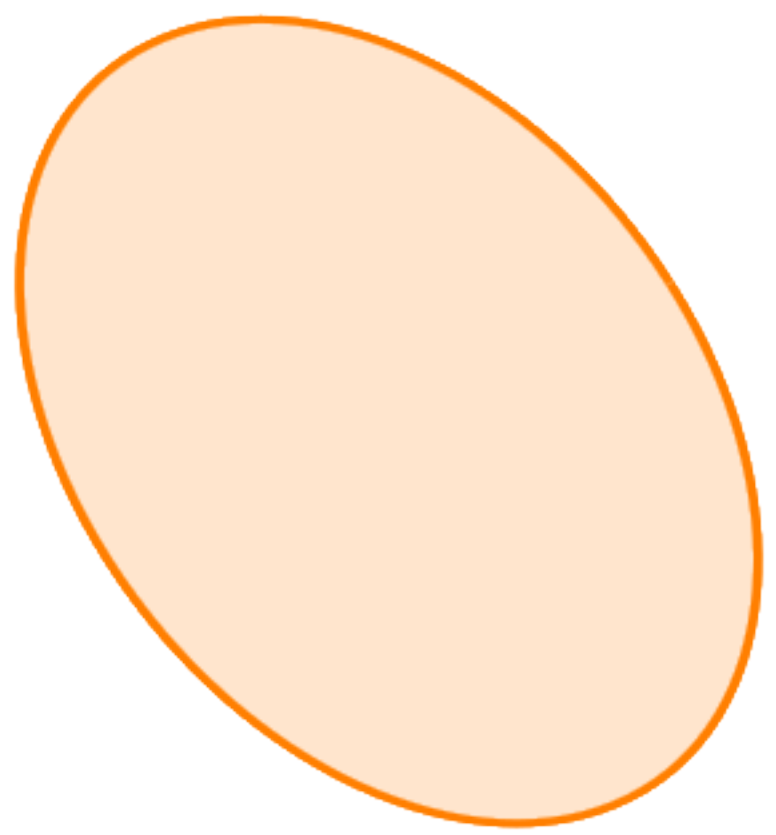
Linking continuous and digital geometry : Gauss digitization with gridstep h

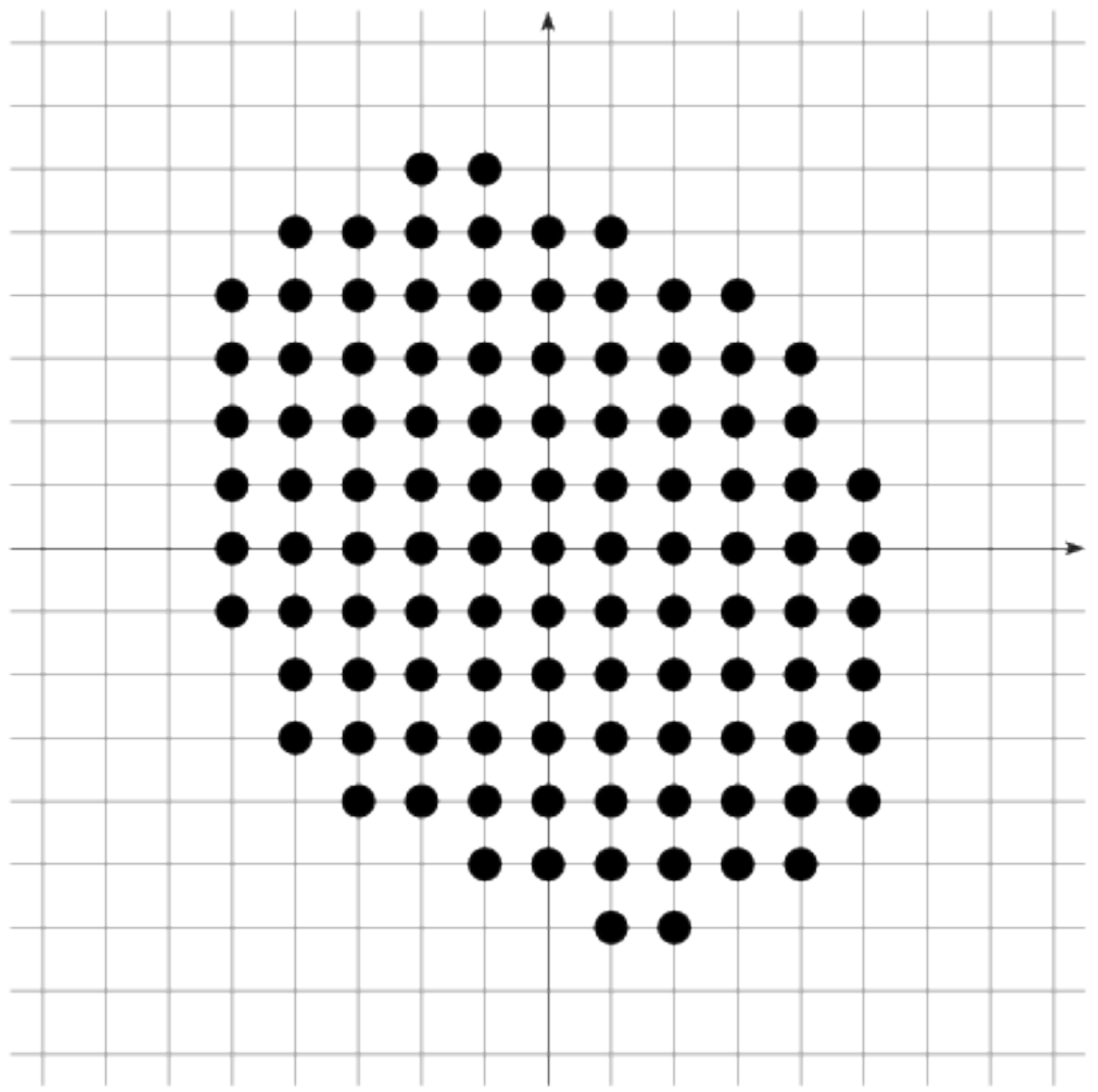


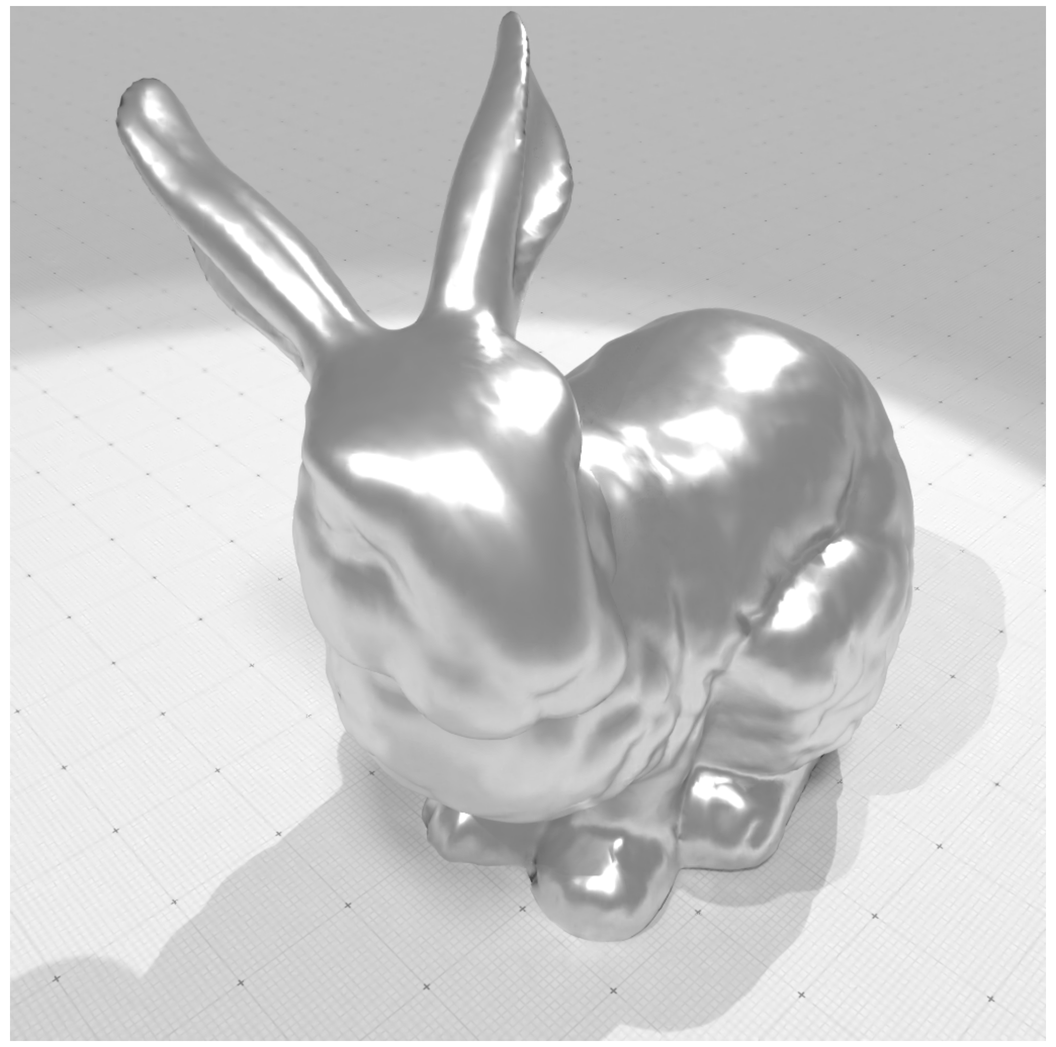
« digitization » « voxelization » « digitized surface »

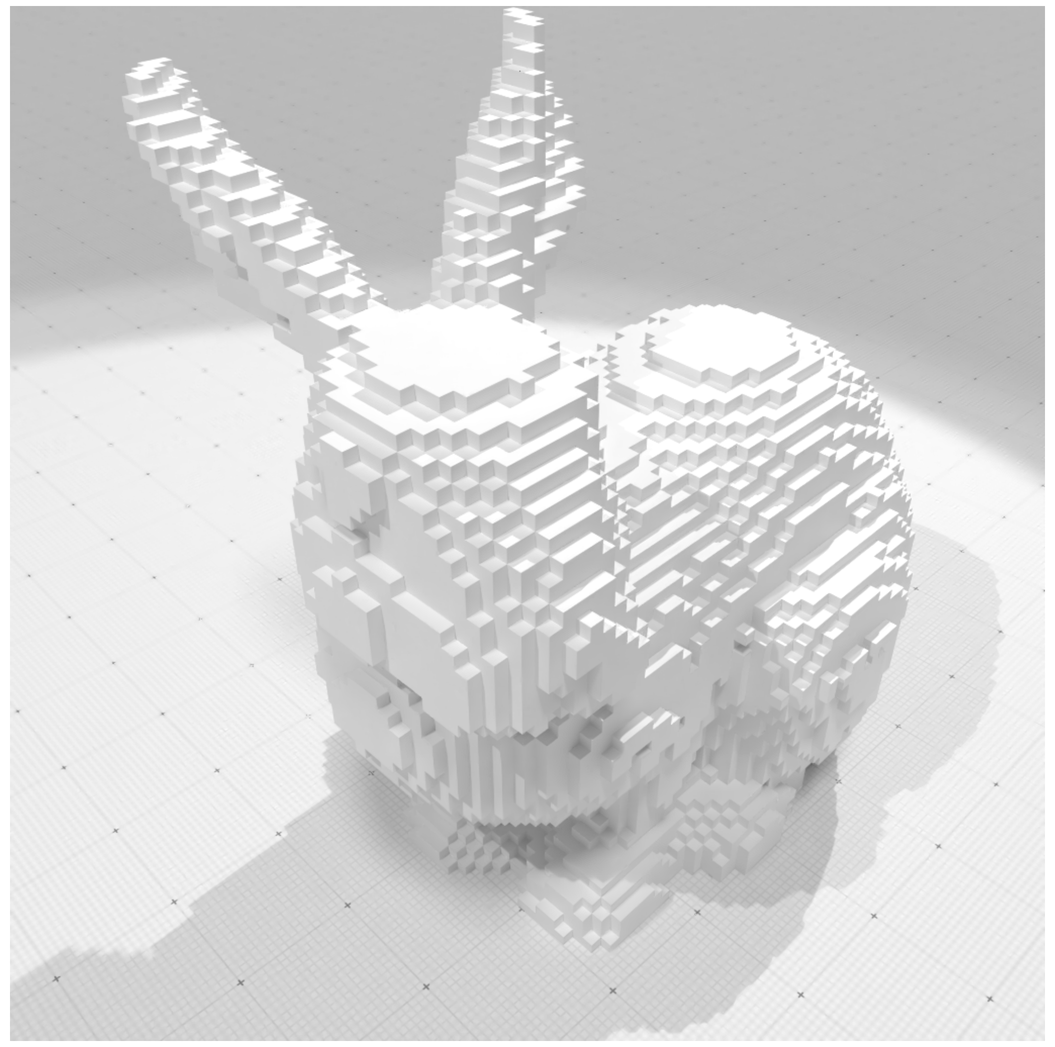


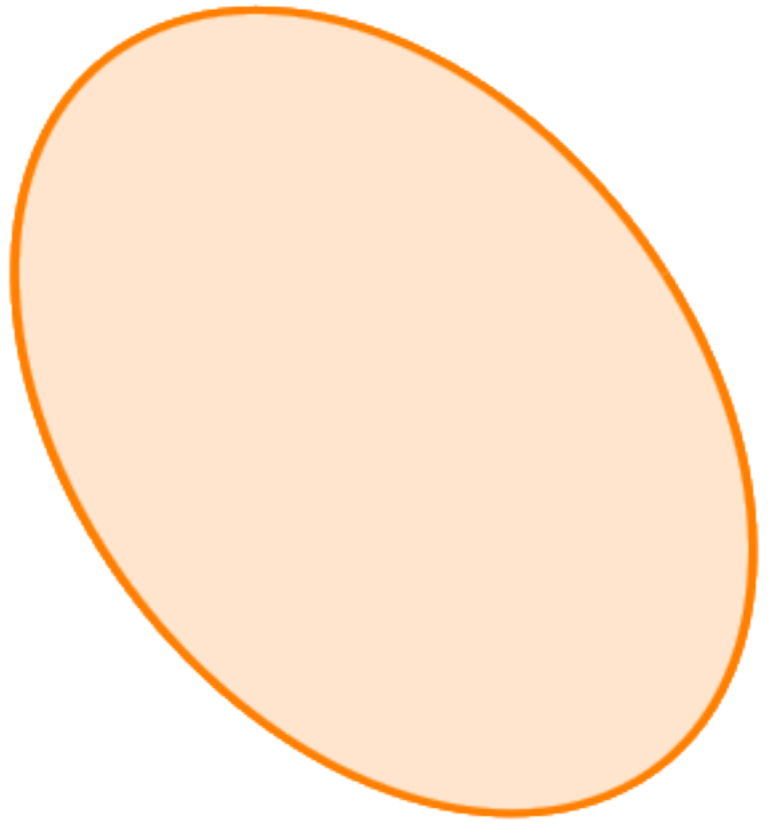
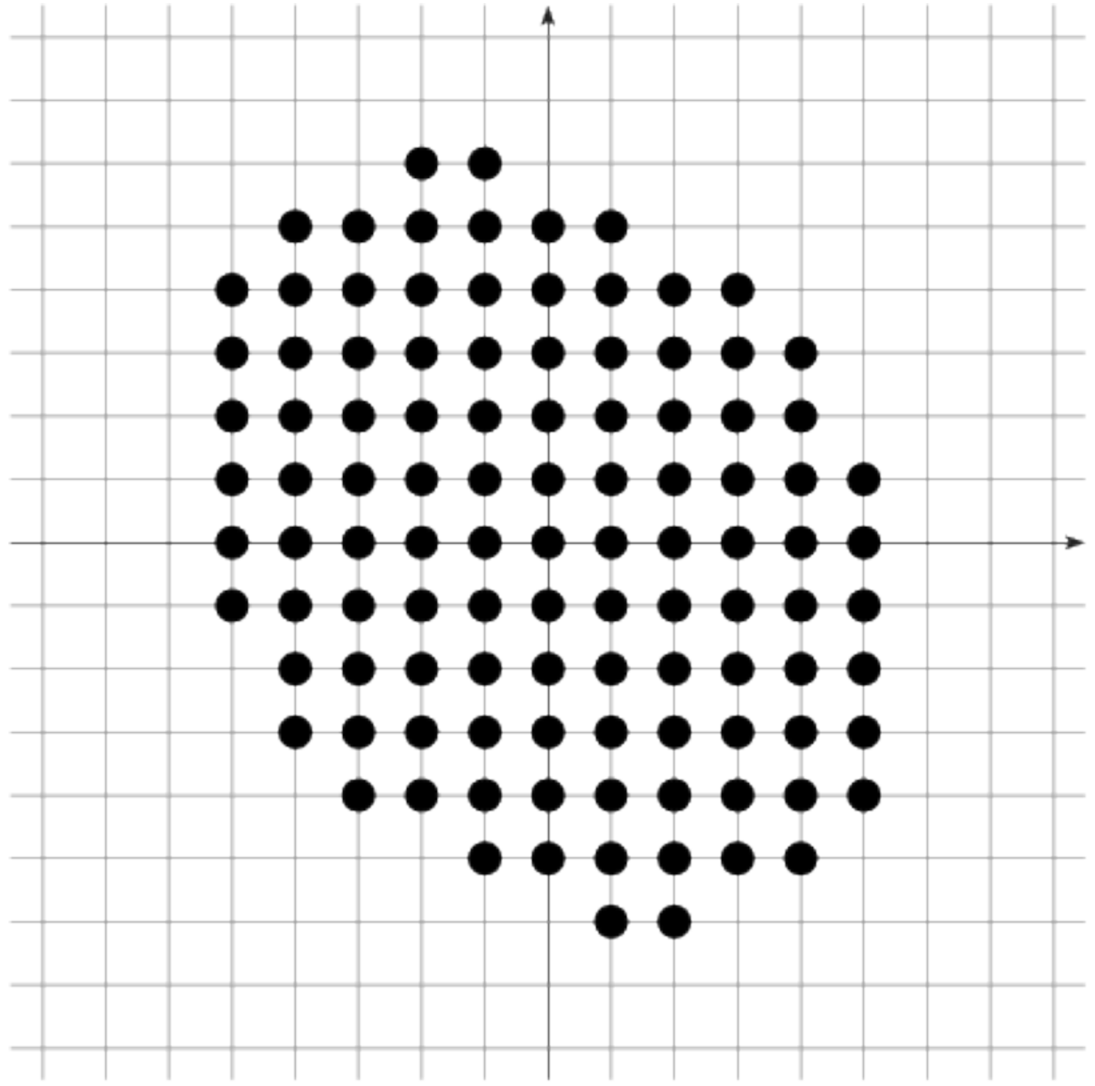
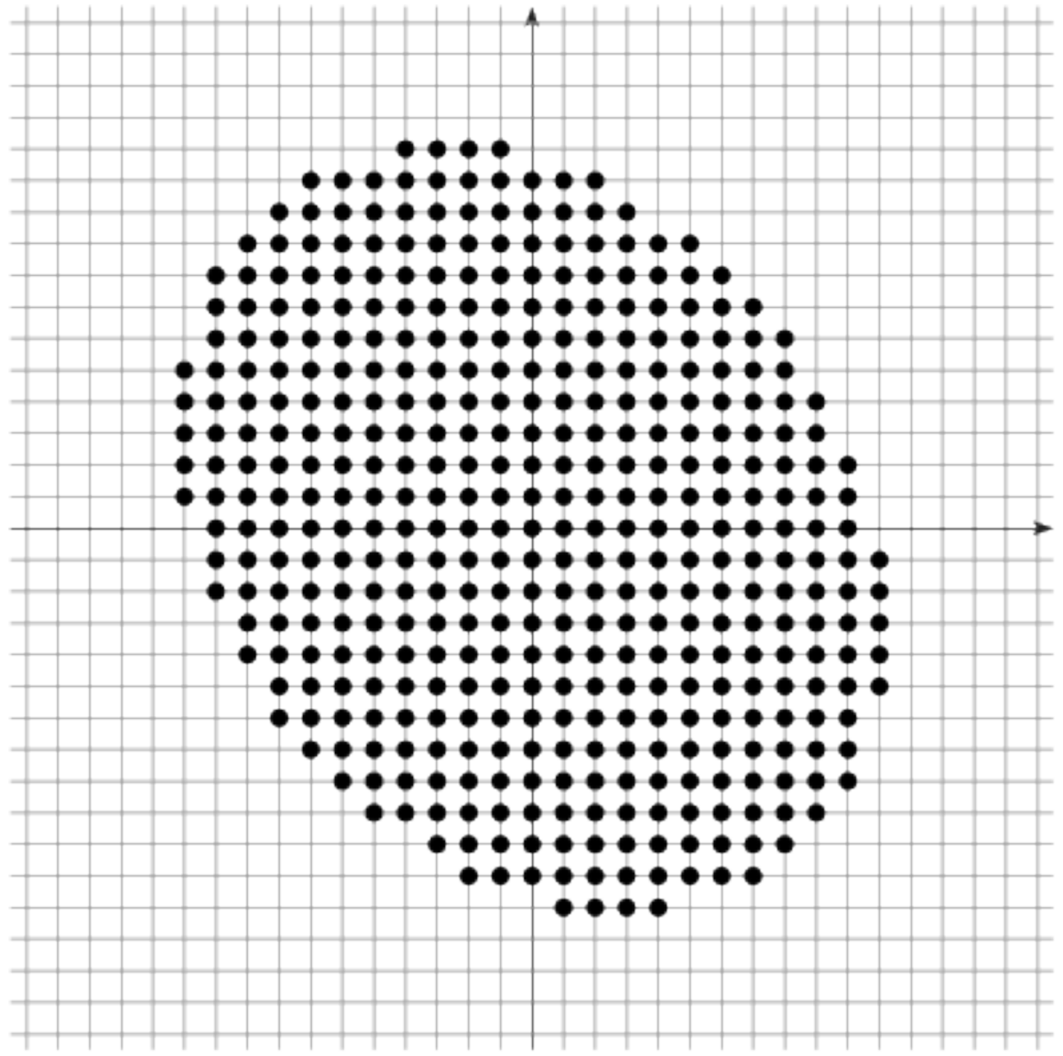
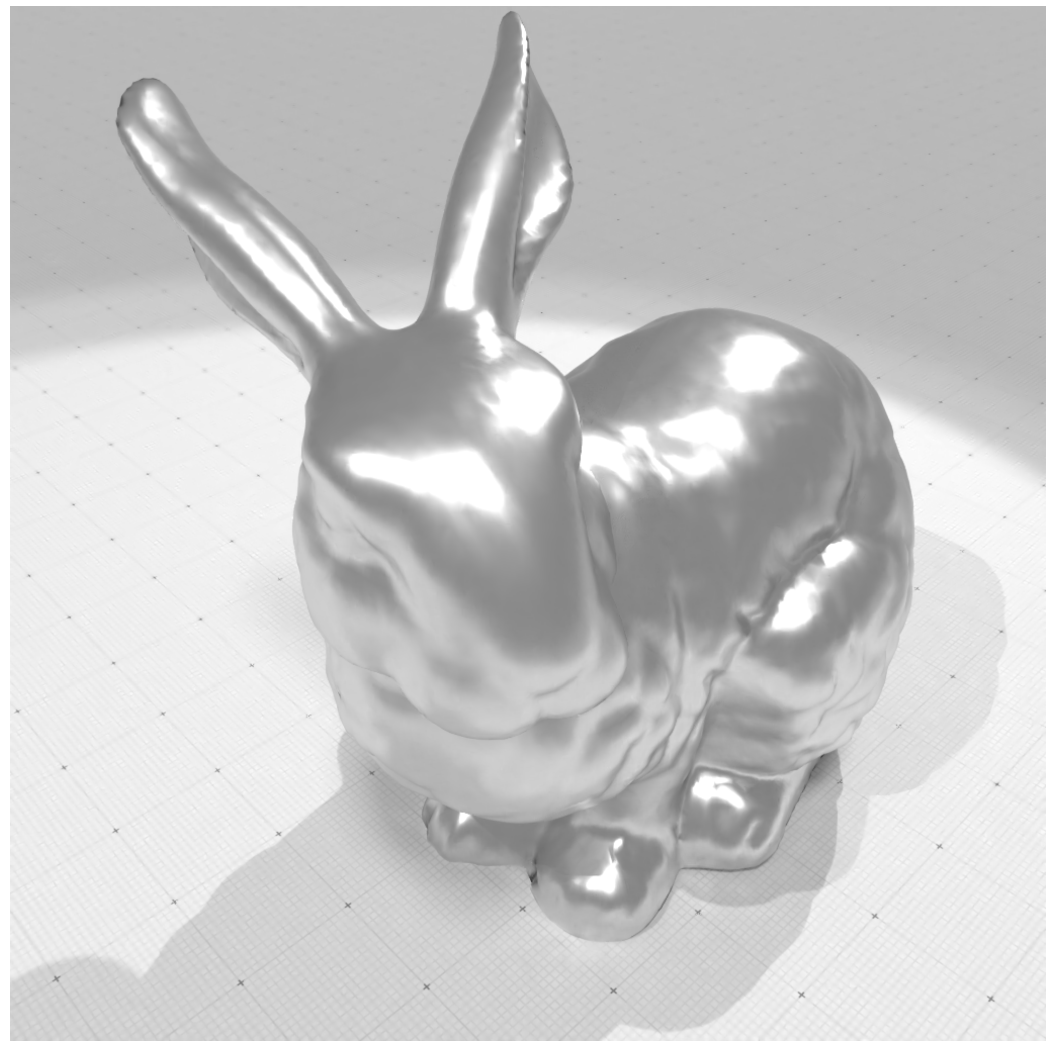
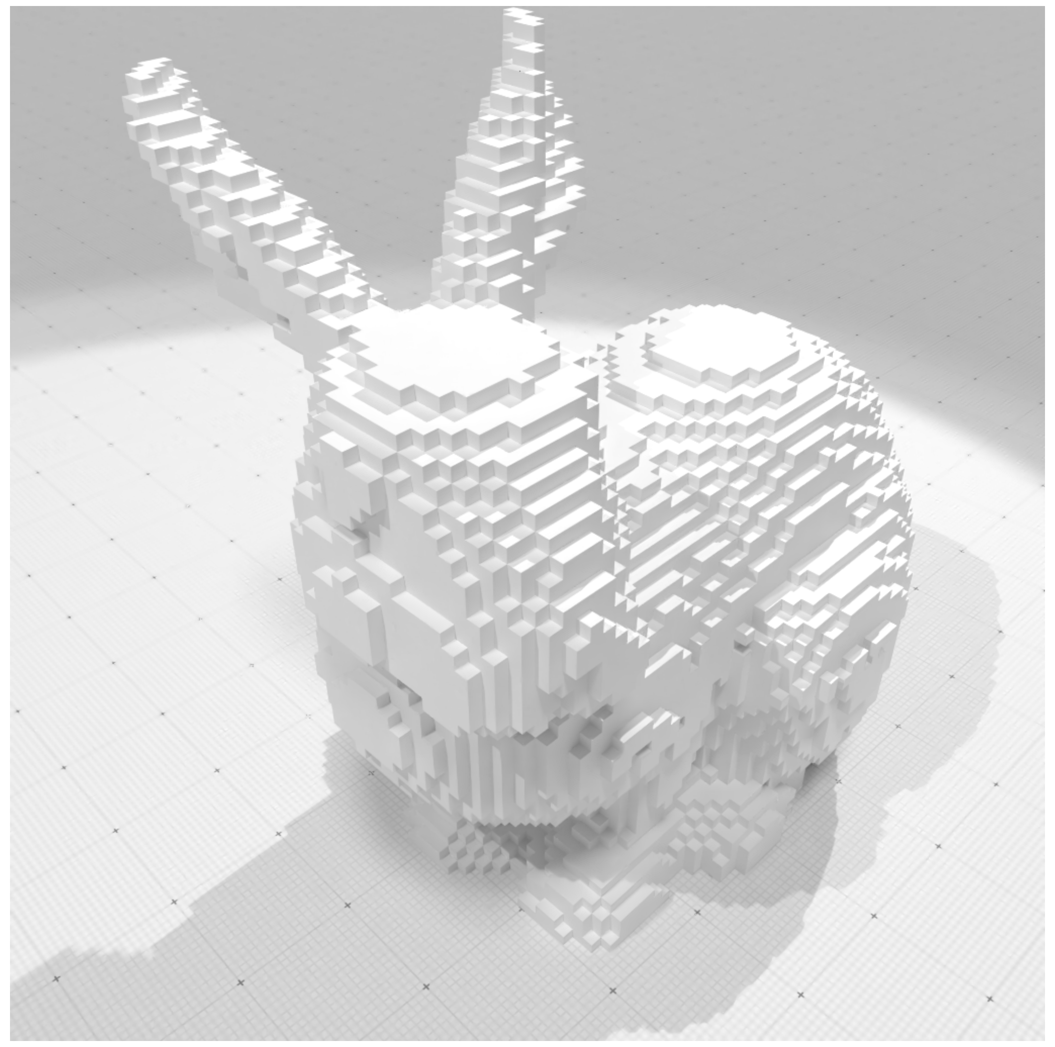
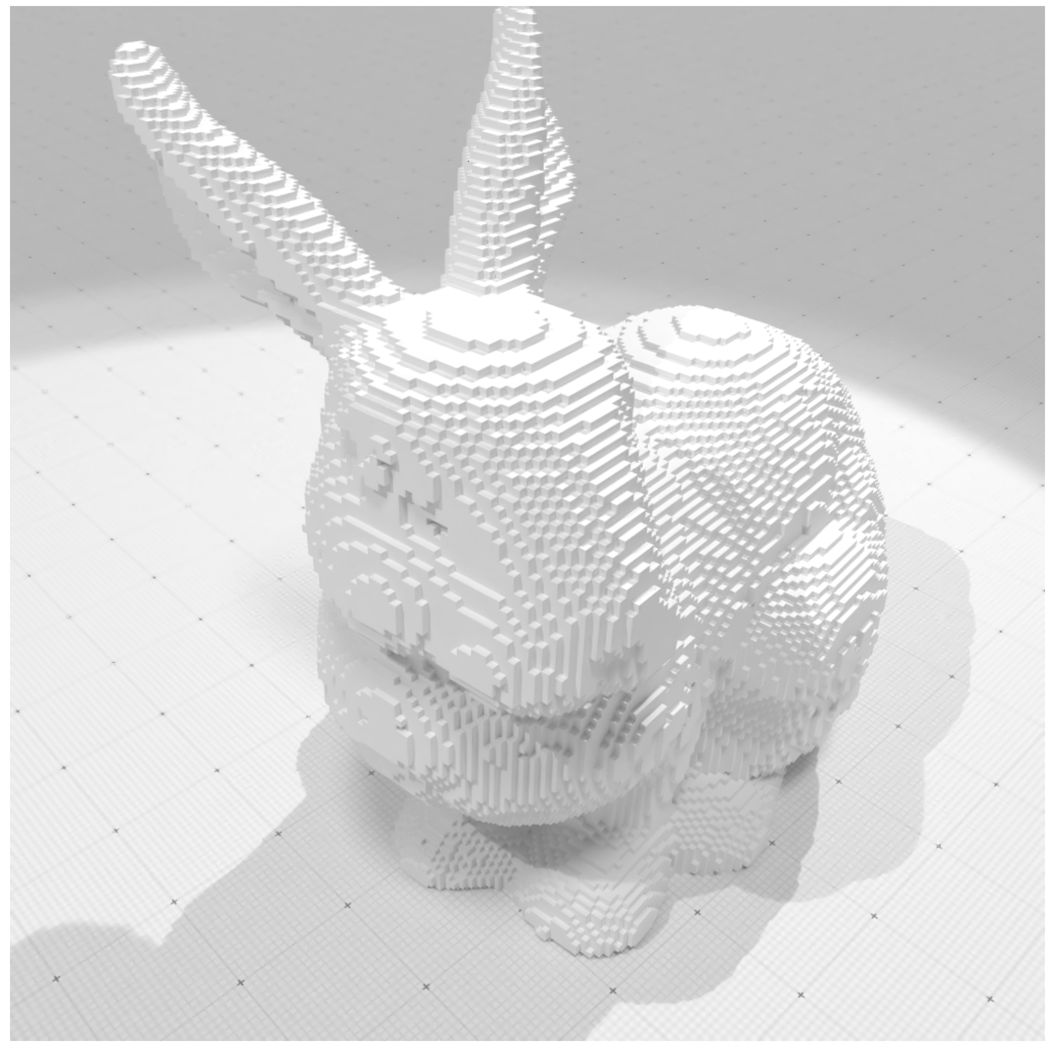


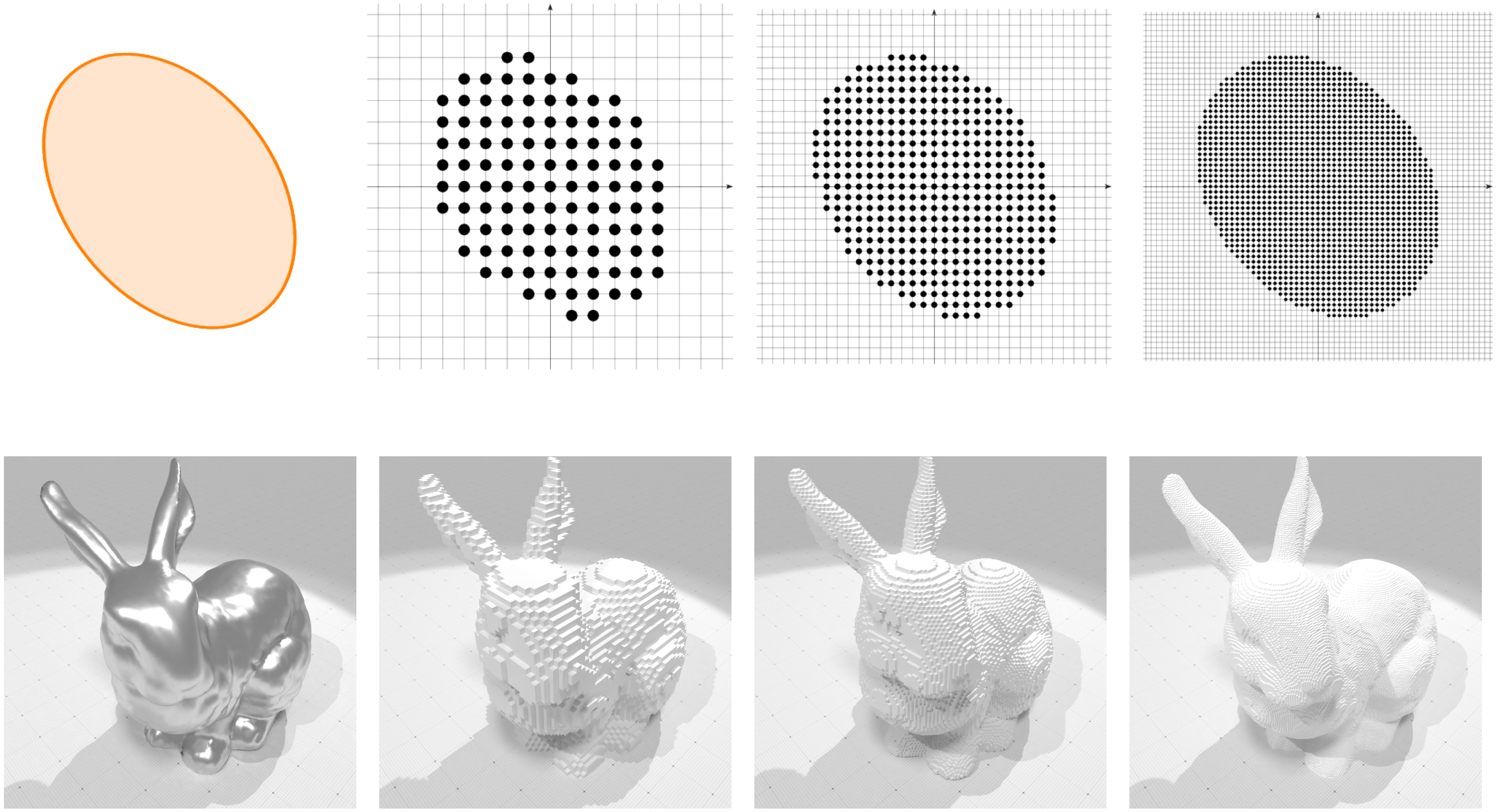










Volume estimation

$$h^d \cdot |X_h| \underset{h \to 0}{\longrightarrow} Vol(X)$$

$O(h)$ convergence speed

If X is strictly C^3-convex: $O\left(h^{\frac{15}{11}+\epsilon}\right)$

Multigrid convergence

For digitization process G, the discrete geometric estimator $\hat{E}$ is multigrid convergent to the geometric quantity E for the family of shapes $\mathbb{X}$, iff, for any $X \in \mathbb{X}$, there exists a grid step $h_X > 0$, such that :

$$\hat{E}(G_h(X), h) \text{ is defined for any } 0 < h < h_X,$$
$$|\hat{E}(G_h(X), h) - E(X)| < \tau_X(h)$$

where the speed of convergence $\tau_X(h)$ has null limit when $h \to 0$.

(Typically area, perimeter, integrals)

Multigrid convergence (local version)

For digitization process G, the local discrete geometric estimator $\hat{E}$ is multigrid convergent to the geometric quantity E for the family of shapes $\mathbb{X}$, iff, for any $X \in \mathbb{X}$, there exists a grid step $h_X > 0$, such that :

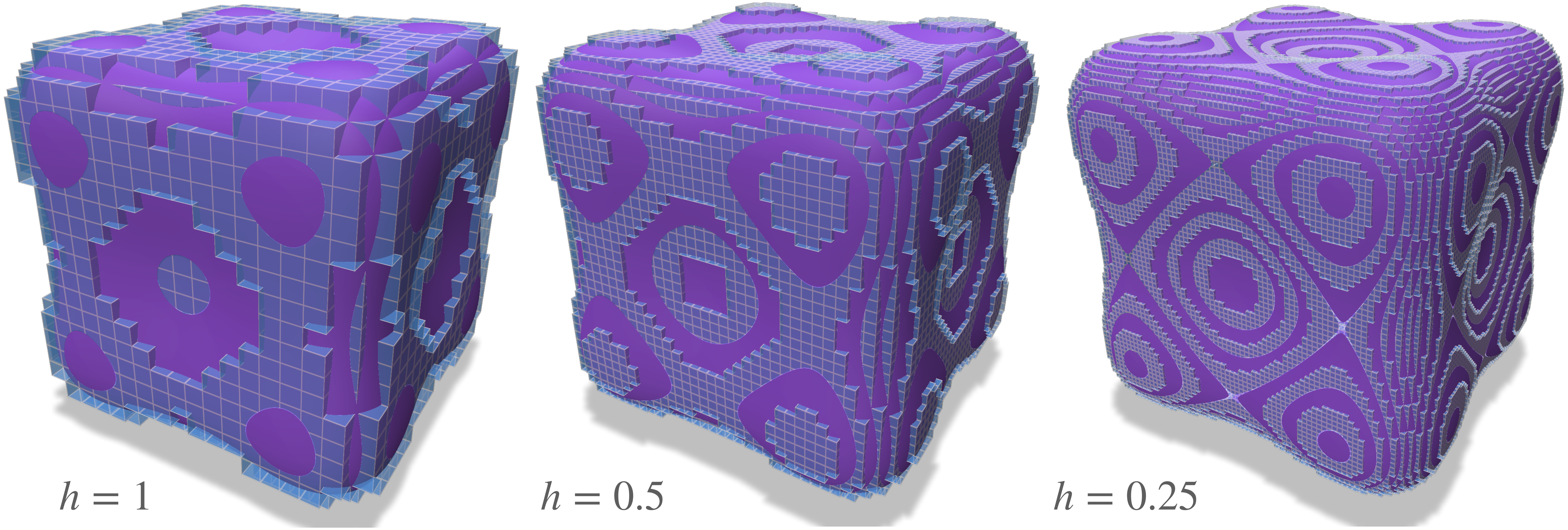
$\hat{E}(G_h(X), \hat{x}, h)$ is defined for any $\hat{x} \in \partial[G_h(X)]_h$ with $0 < h < h_X$,

for any $x \in \partial X$, for any $\hat{x} \in \partial[G_h(X)]_h$ with $\|x - \hat{x}\|_\infty \leq h$, $|\hat{E}(G_h(X), \hat{x}, h) - E(X, x)| < \tau_X(h)$

where the speed of convergence $\tau_X(h)$ has null limit when $h \to 0$.

(Typically normal direction, curvatures, ...)

Hausdorff closeness of digitized shapes

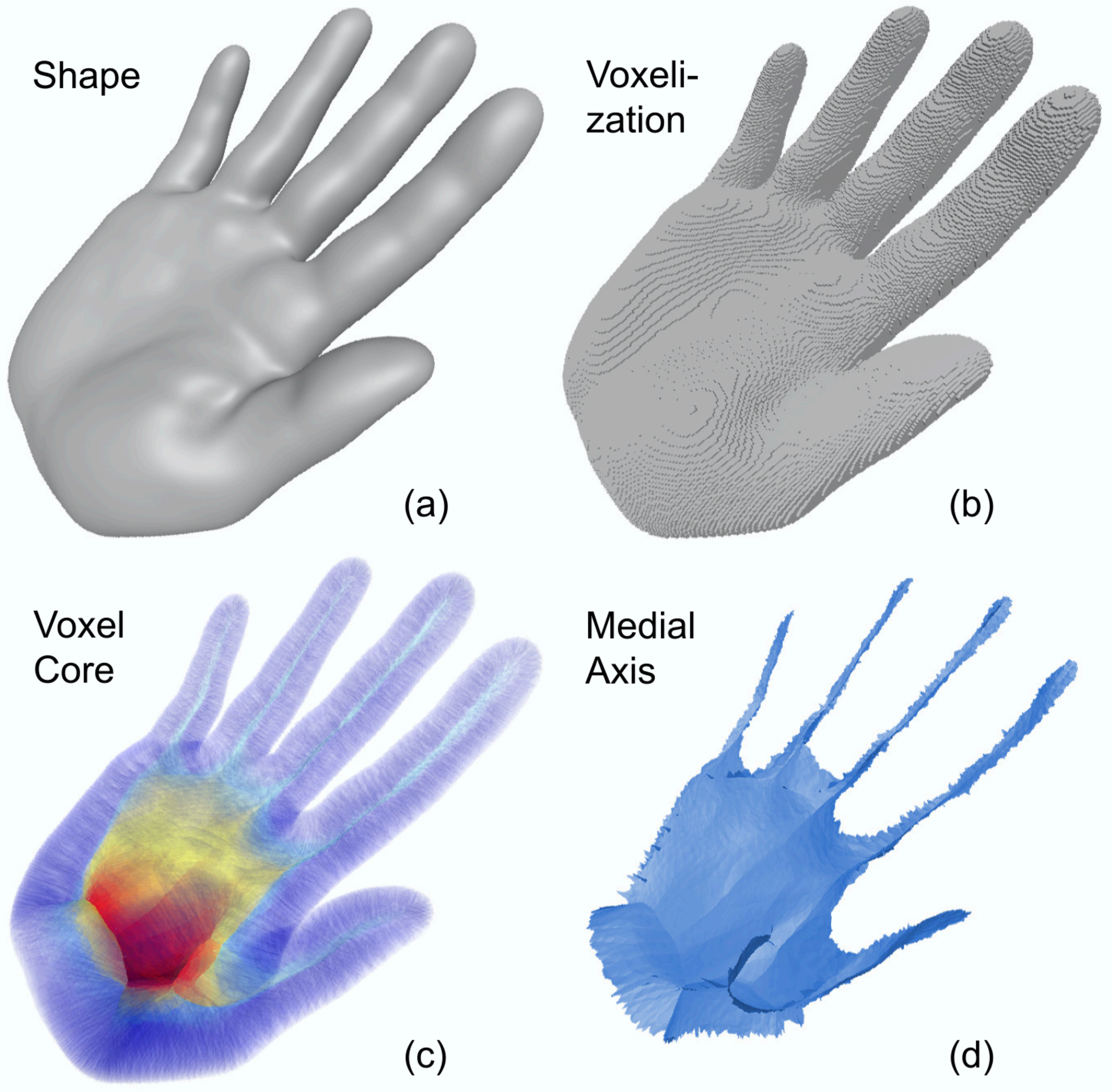


For any compact domain $X \in \mathbb{R}^d$ *such that* ∂X *has positive reach, and its digitization* $X_h := [G_h(X)]_h$ *on a grid with grid-step h, then* $d_H(\partial X, \partial X_h) \leq \sqrt{d}/2h$ *for small enough h*

[LT16]

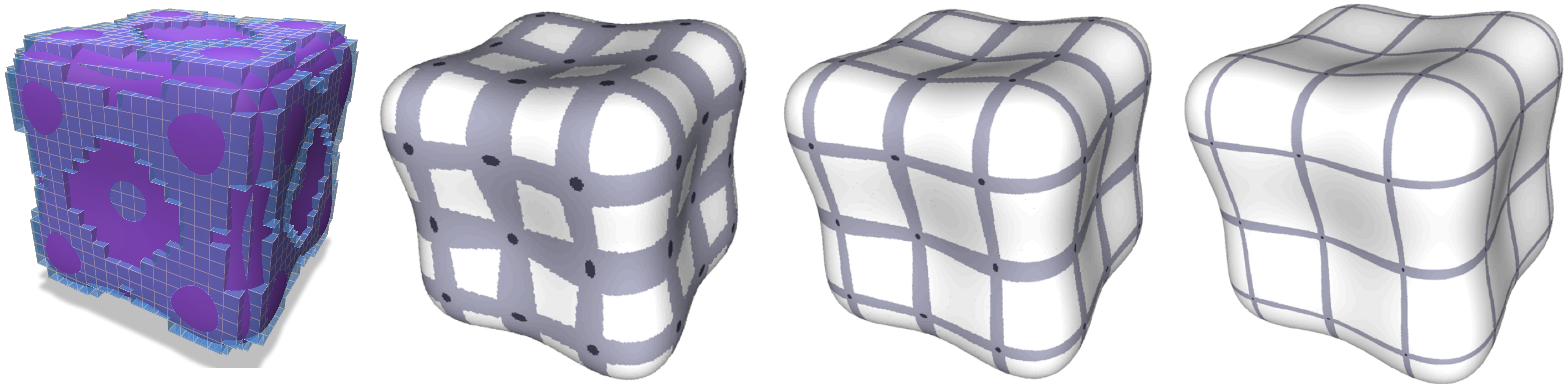
Homotopy equivalence

For a compact shape X with positive reach ρ, for $h < \frac{2\sqrt{3}}{3}\rho$, the set X and its voxelization $[G_h(X)]_h$ are homotopy equivalent.
Its voxel core is also homotopy equivalent.



[YLJ2018]

Bijectivity of projection and manifoldness



$h = 0.1$ $h = 0.05$ $h = 0.025$

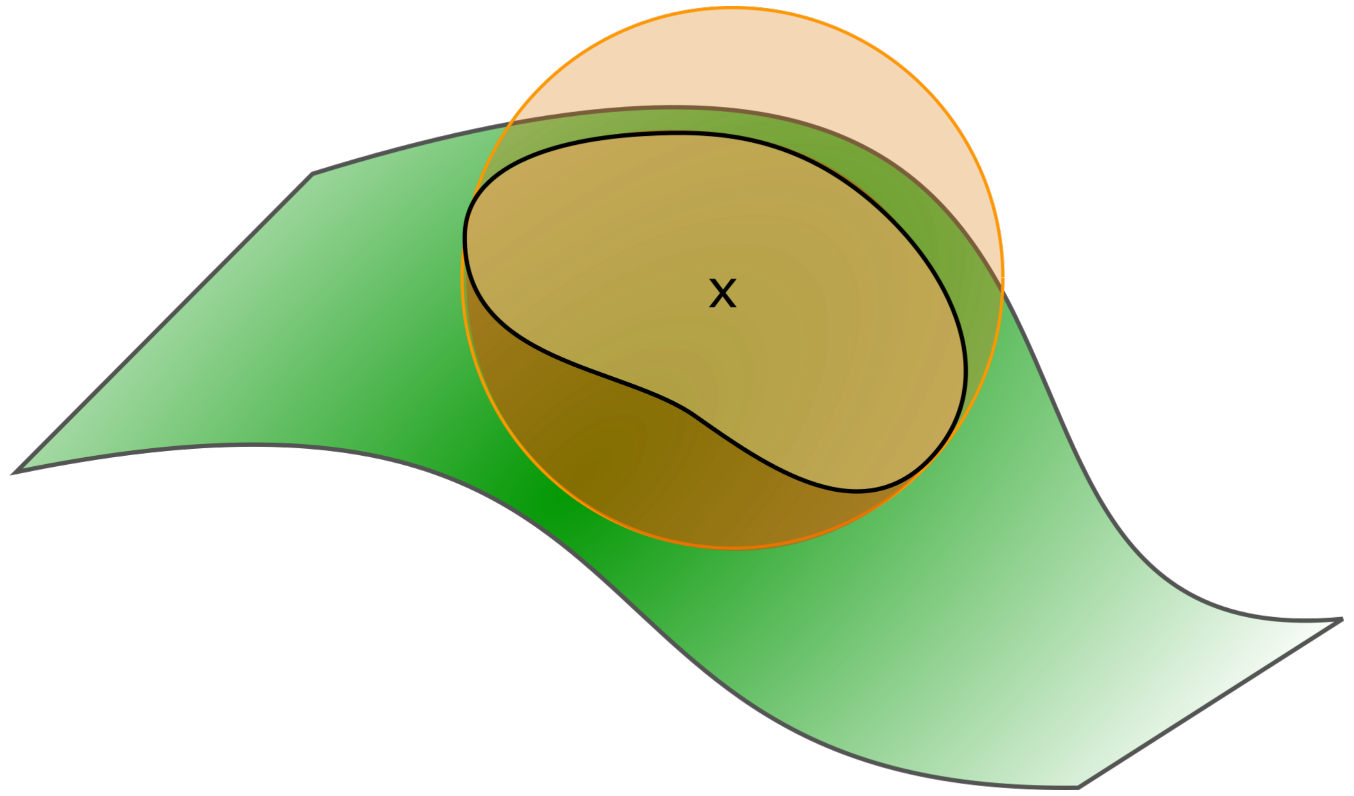
[LT16]

If X has positive reach,
the size of the non-injective part of projection
$\pi_X : \partial X_h \to \partial X$ tends to zero as $h \to 0$.
(light gray + dark gray zones $\approx O(h)$)

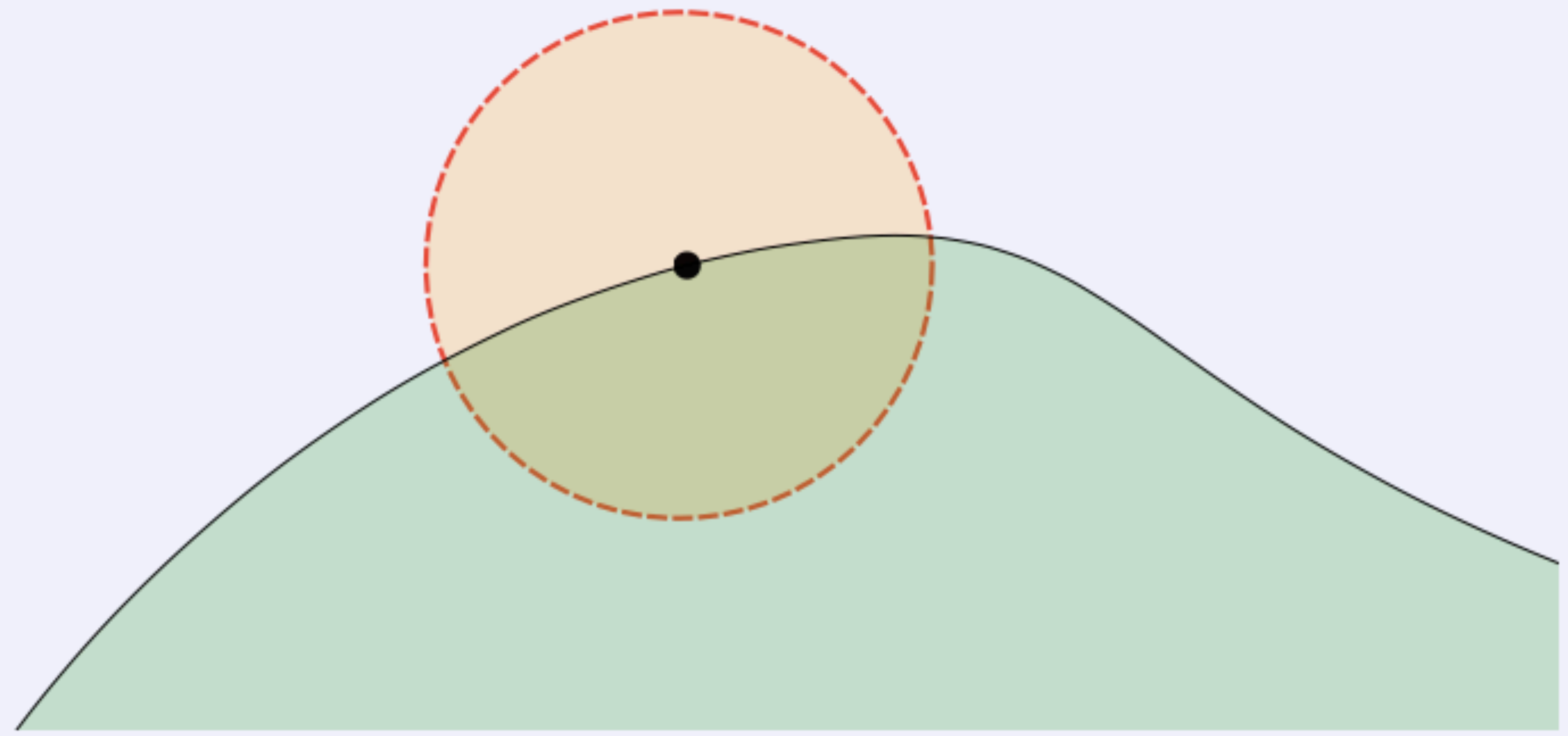
[LT16]

If X has positive reach,
the size of the non-manifoldness part of ∂X_h
tends quickly to zero as $h \to 0$.
(dark gray zones $\approx O(h^2)$)

Normal vector and curvatures estimation



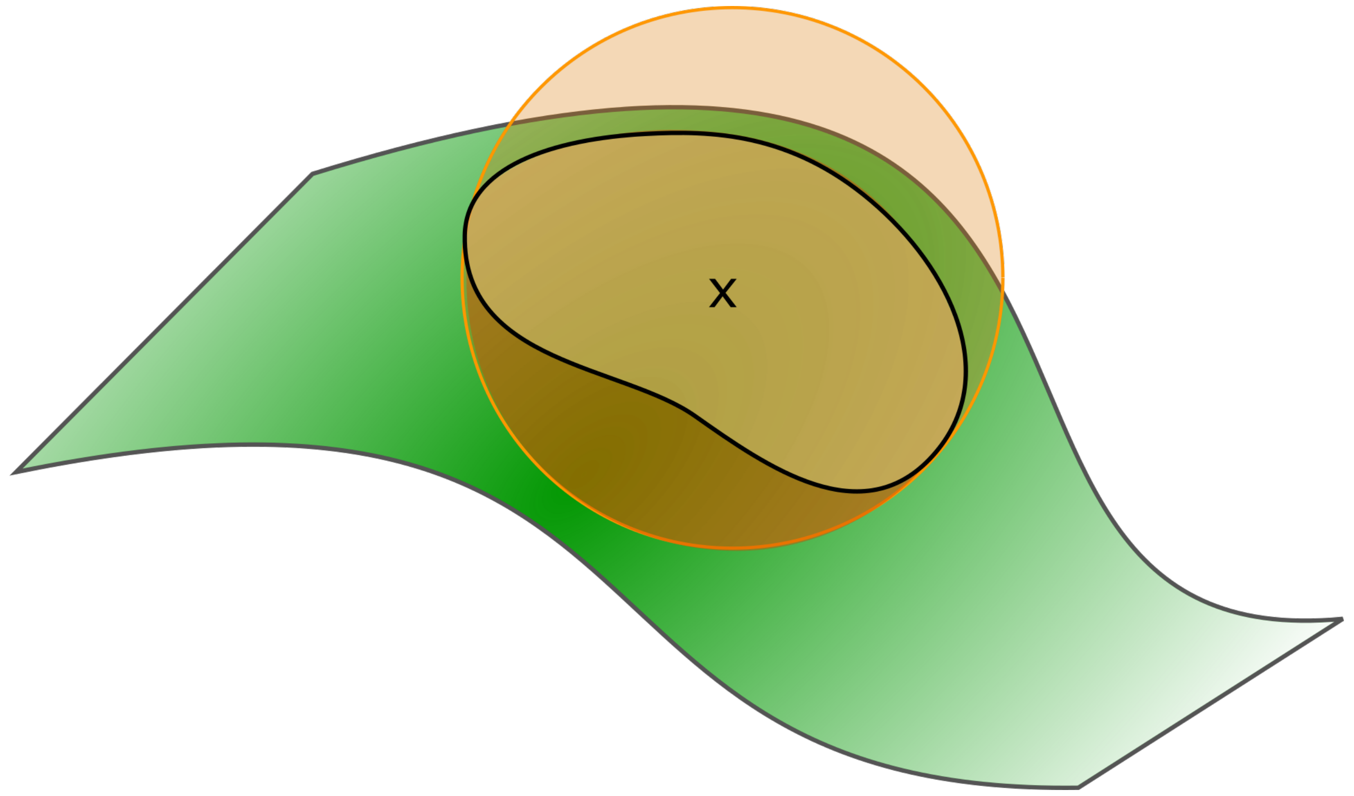
- Integral Invariants : analyzing set $B_R(x) \cap X$ gives normal vector, principal directions and curvatures [Pottmann et al. 2007]

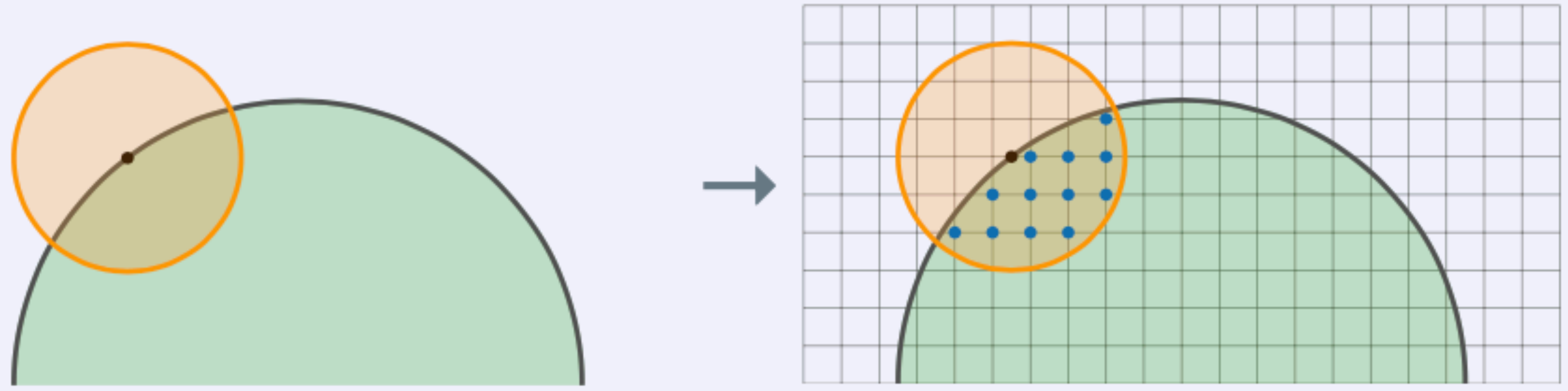


$$\kappa(M,\mathbf{x}) := \underbrace{\frac{3\pi}{2R} - \frac{3\cdot A_R(M,\mathbf{x})}{R^3}}_{\kappa^R(M,\mathbf{x})} + O(R)$$ [Pottmann et al. 2007]

Normal vector and curvatures estimation

- Integral Invariants : analyzing set $B_R(x) \cap X$ gives normal vector, principal directions and curvatures [Pottmann et al. 2007]



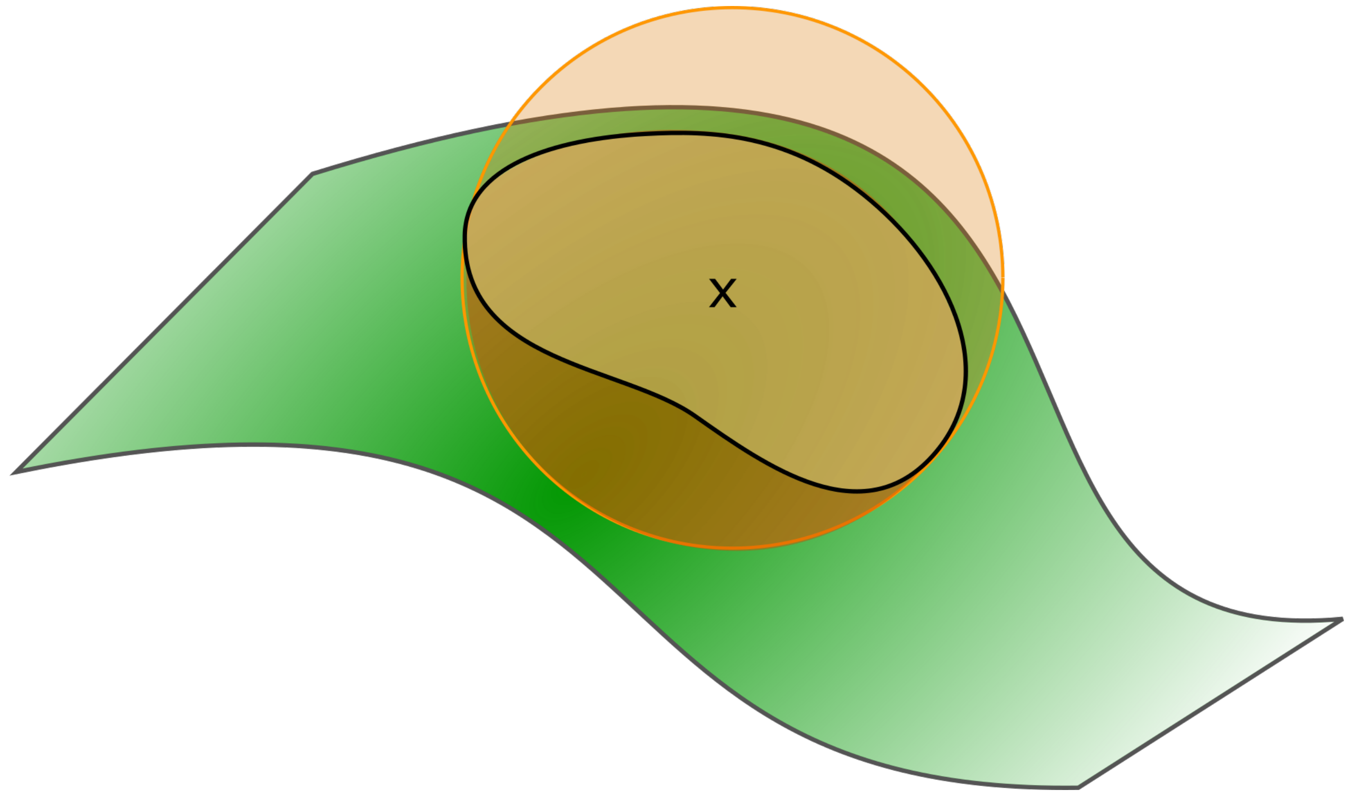


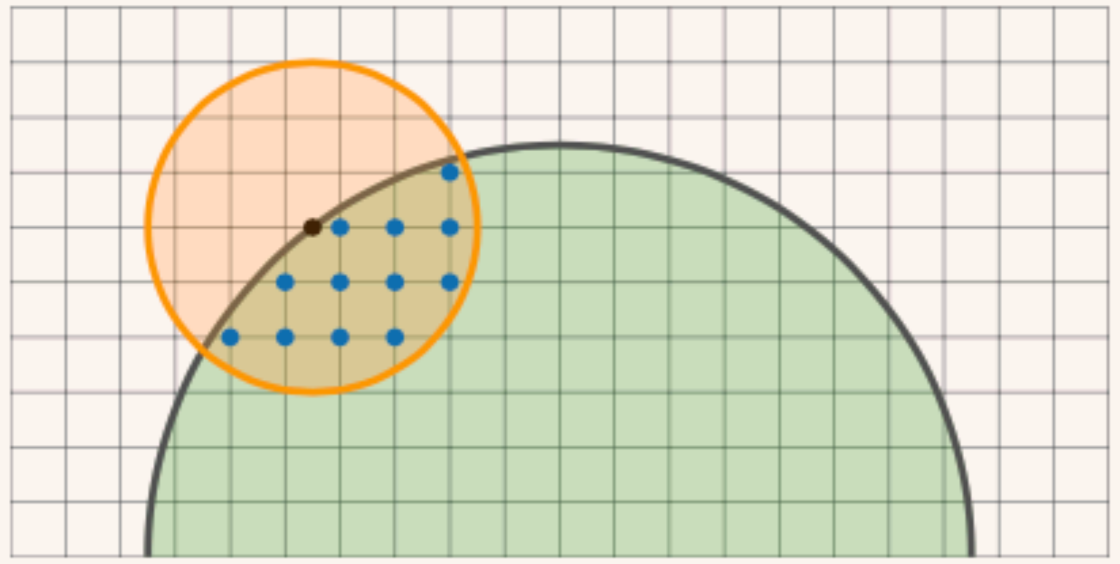
$$A_R(M, \mathbf{x}) \rightarrow \widehat{\mathrm{Area}}\,(B_{R/h}(\mathbf{x}/h) \cap \mathbb{G}_h(M))$$

[Gauss]

Normal vector and curvatures estimation

- Integral Invariants : analyzing set $B_R(x) \cap X$ gives normal vector, principal directions and curvatures [Pottmann et al. 2007]

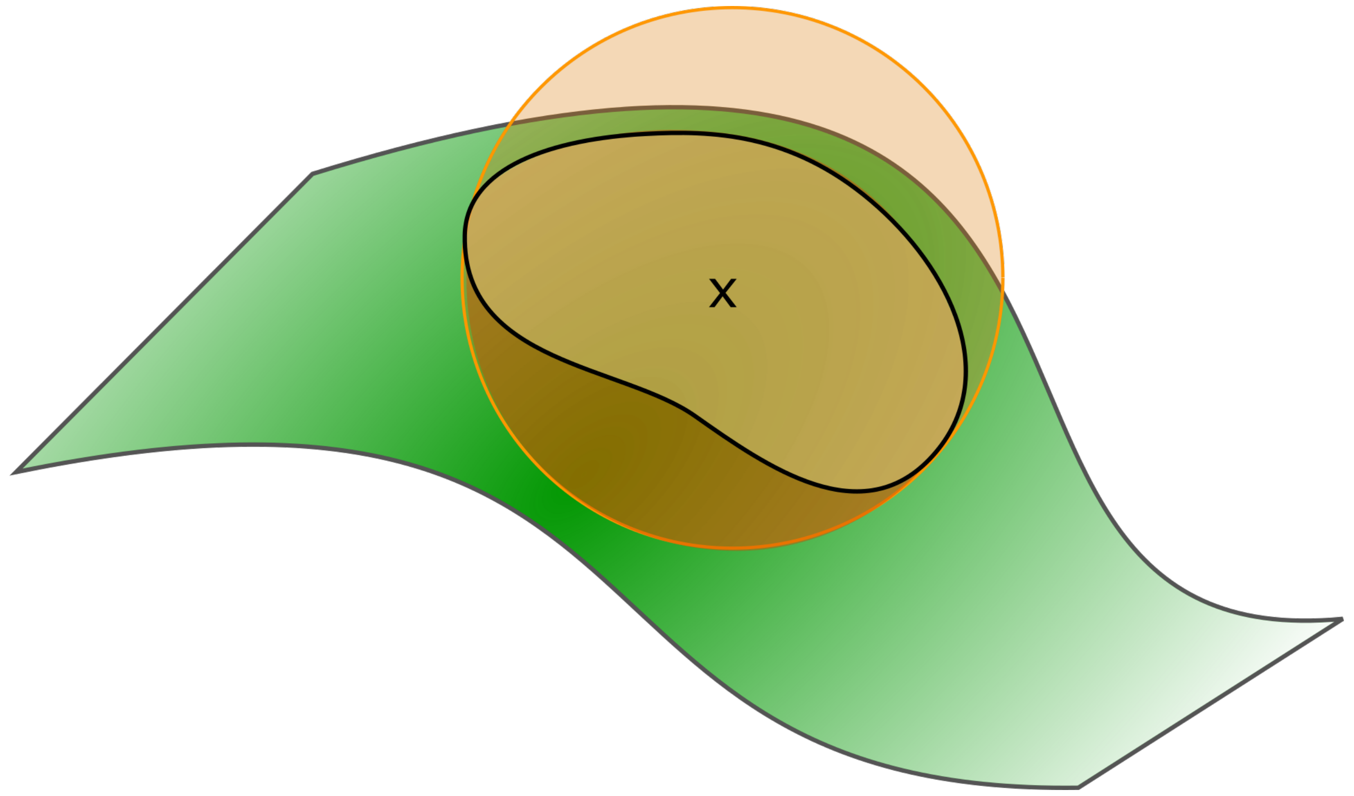




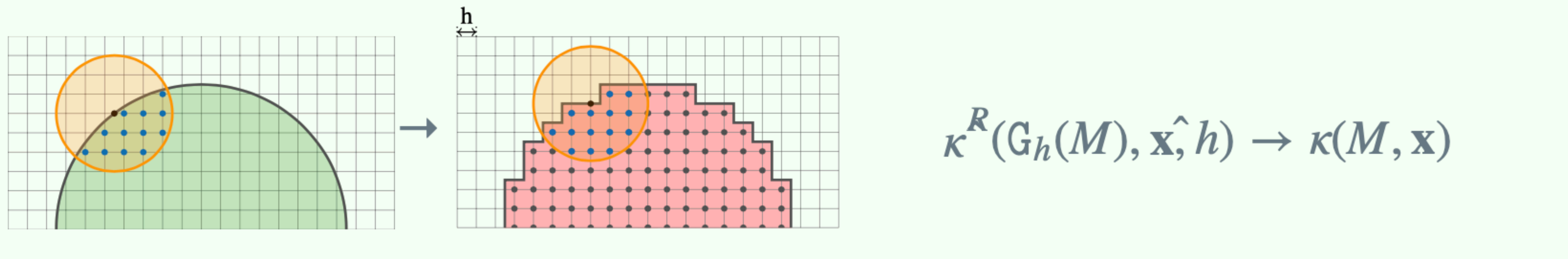
+ [Pottmann et al. 2007]

$\kappa^R(\mathrm{G}_h(M), \mathbf{x}, h)$

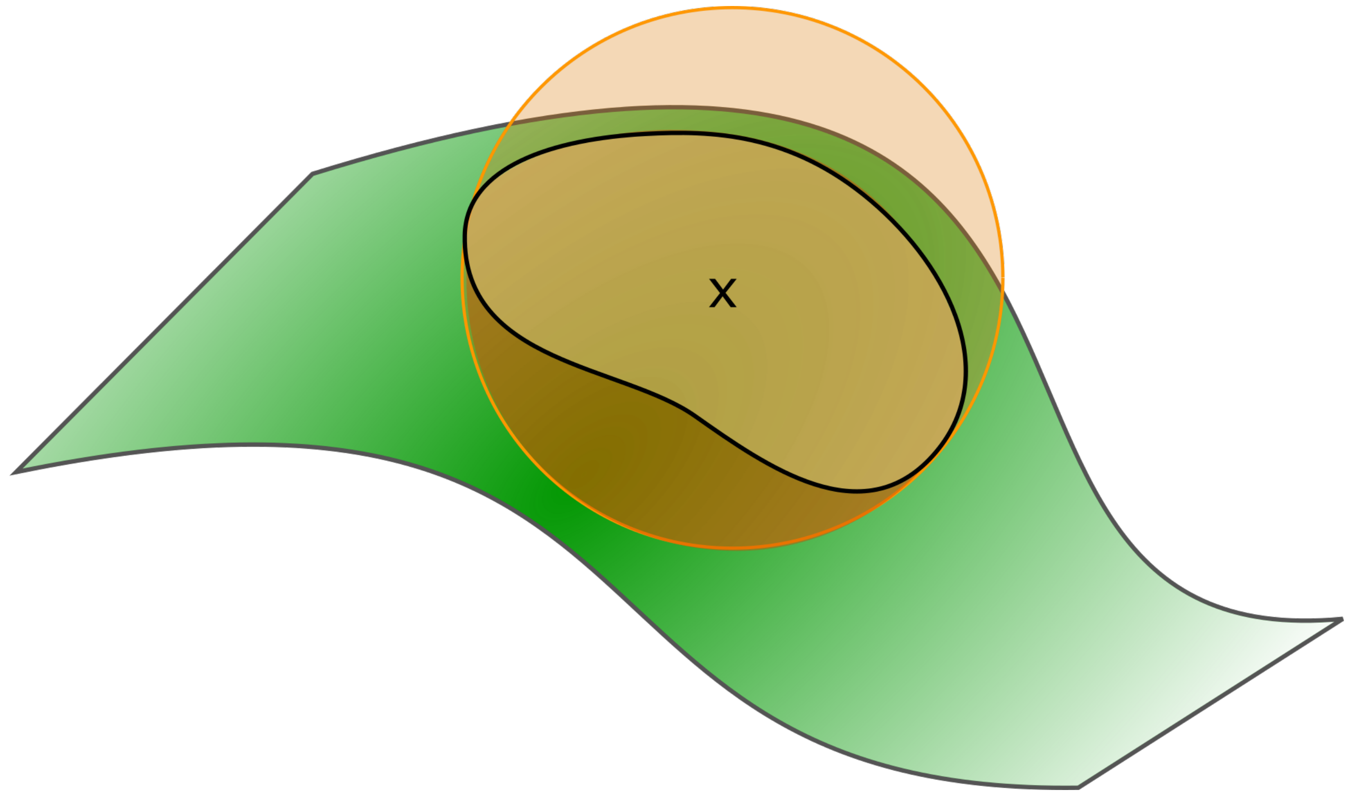
Normal vector and curvatures estimation



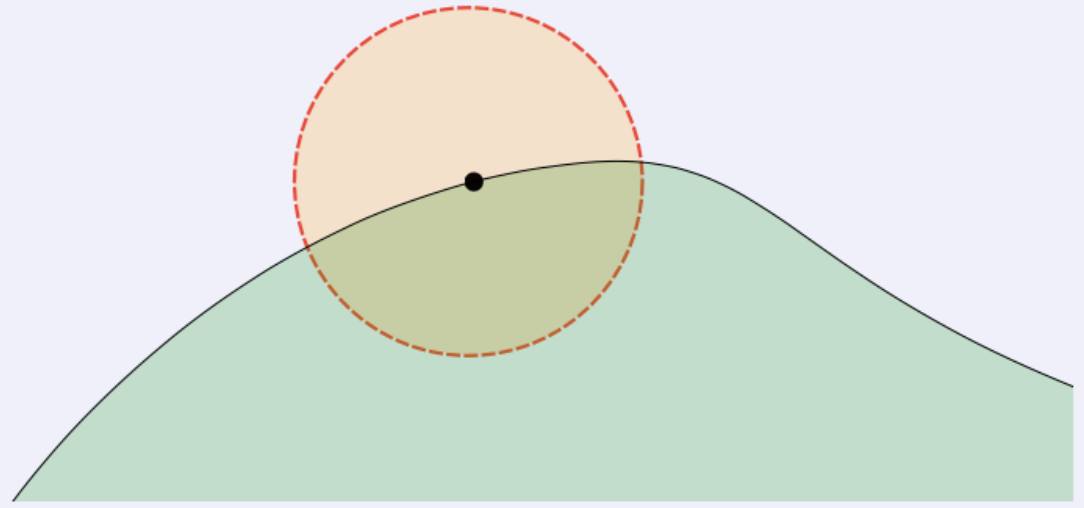
- Integral Invariants : analyzing set $B_R(x) \cap X$ gives normal vector, principal directions and curvatures [Pottmann et al. 2007]



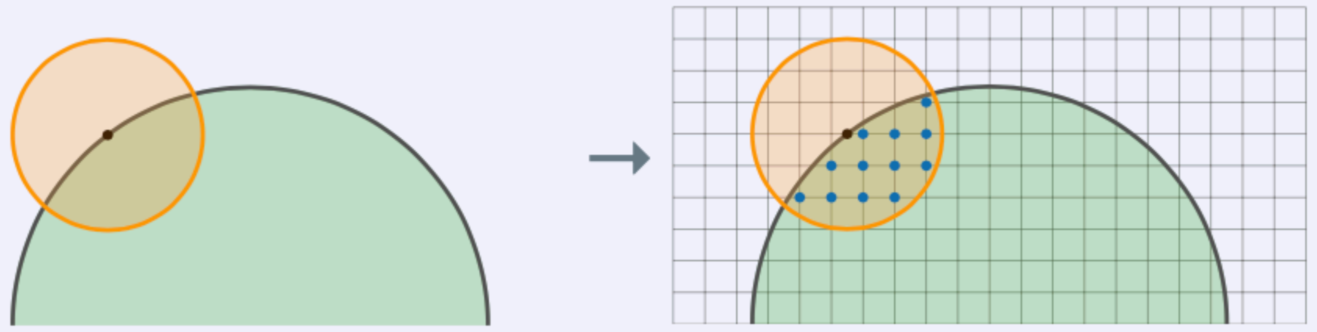
Normal vector and curvatures estimation



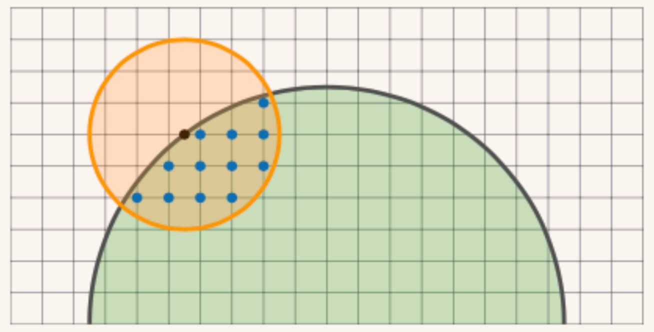
- Integral Invariants : analyzing set $B_R(x) \cap X$ gives normal vector, principal directions and curvatures [Pottmann et al. 2007]



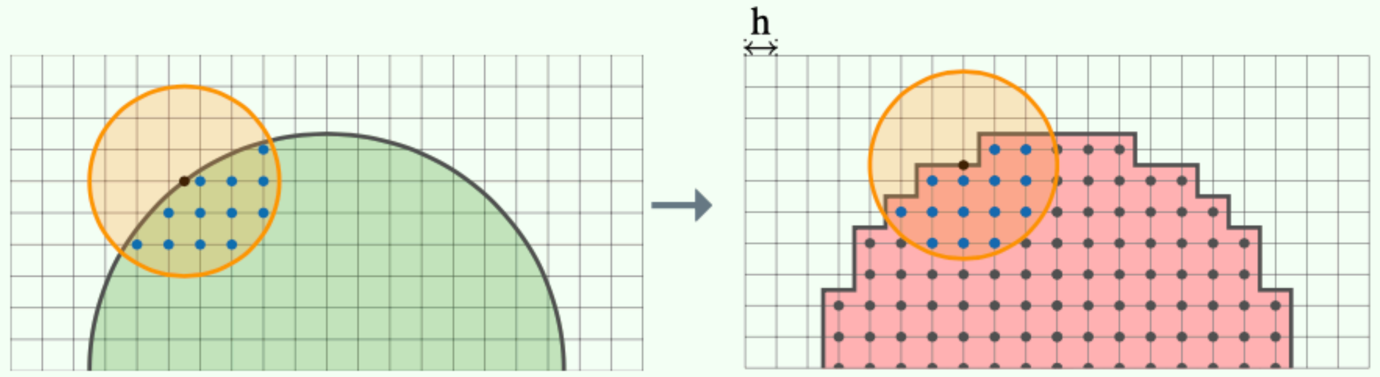
$$\kappa(M,\mathbf{x}) := \underbrace{\frac{3\pi}{2R} - \frac{3 \cdot A_R(M,\mathbf{x})}{R^3}}_{\kappa^R(M,\mathbf{x})} + O(R)$$ [Pottmann et al. 2007]



$$A_R(M,\mathbf{x}) \to \widehat{\mathrm{Area}}\,(B_{R/h}(\mathbf{x}/h) \cap \mathrm{G}_h(M))$$



+ [Pottmann et al. 2007] $\quad \hat{\kappa}^R(\mathrm{G}_h(M), \mathbf{x}, h)$



$$\hat{\kappa}^R(\mathrm{G}_h(M), \hat{\mathbf{x}}, h) \to \kappa(M,\mathbf{x})$$

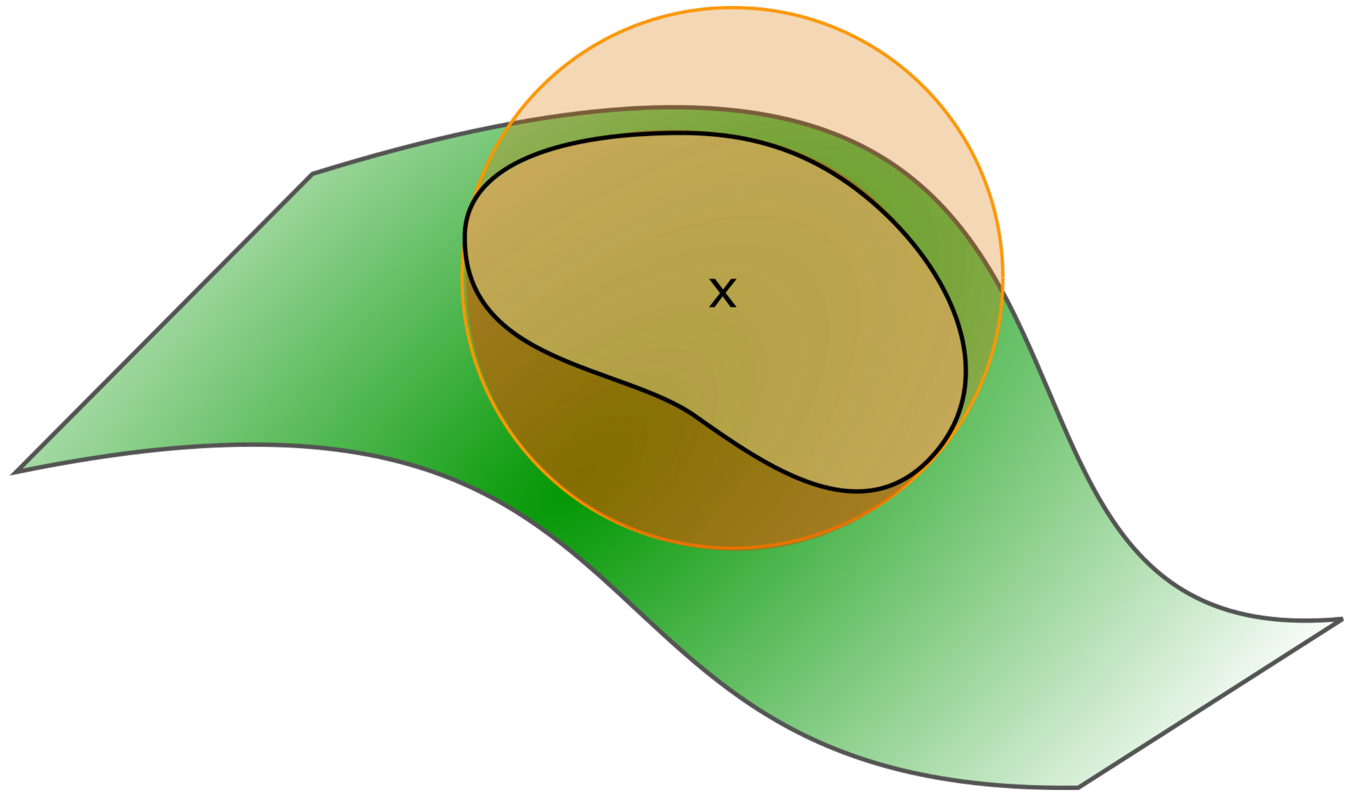
[C., Levallois, Lachaud]

Let M be a convex shape in $\mathbb{R}^2$ with a C^3 bounded positive curvature boundary.

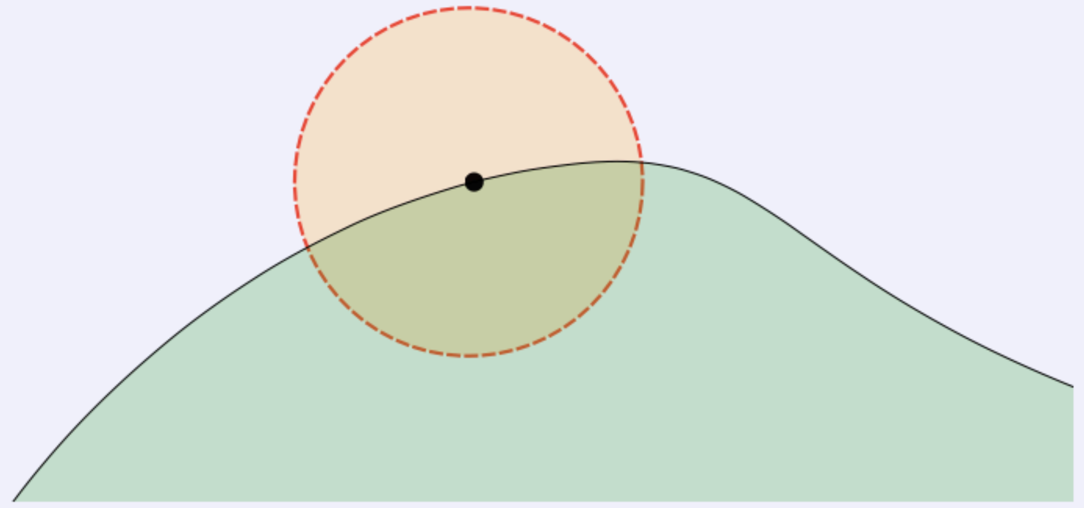
$$\forall \mathbf{x} \in \partial M, \forall \hat{\mathbf{x}} \in \partial[\mathrm{G}_h(M)]_h, \|\hat{x} - x\|_\infty \leq h \Rightarrow$$

$$|\hat{\kappa}^R(\mathrm{G}_h(M), \hat{\mathbf{x}}, h) - \kappa(M,\mathbf{x})| = O(R) + O\left(\frac{h^\beta}{R^{1+\beta}}\right) + O\left(\frac{h^{\alpha'}}{R^2}\right) + O\left(h^{\alpha'}\right) + O\left(\frac{h^{2\alpha'}}{R^2}\right)$$

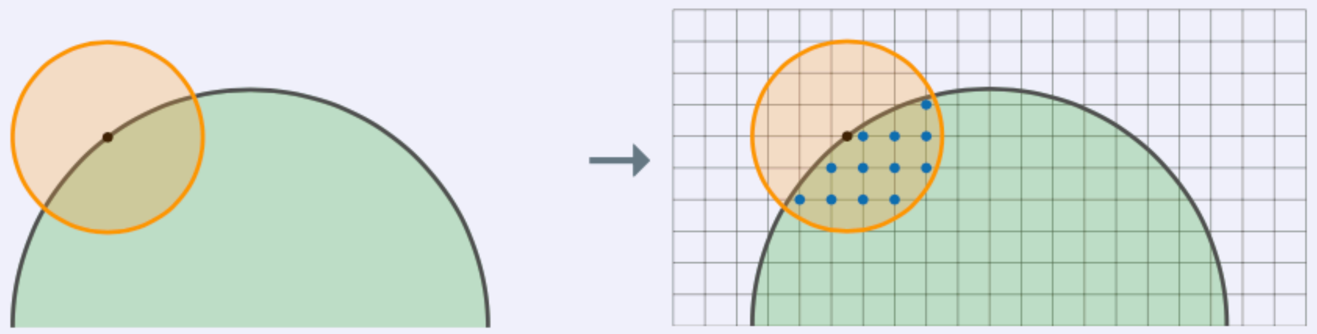
Normal vector and curvatures estimation



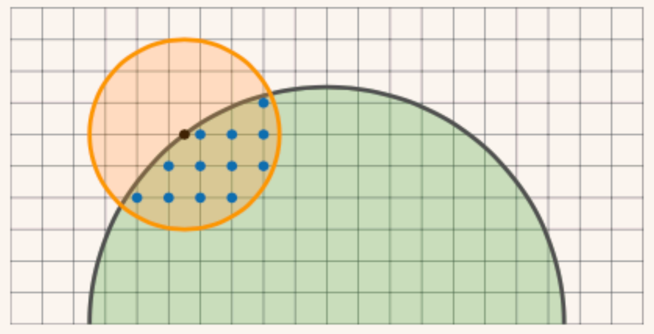
- Integral Invariants : analyzing set $B_R(x) \cap X$ gives normal vector, principal directions and curvatures [Pottmann et al. 2007]



$$\kappa(M,\mathbf{x}) := \underbrace{\frac{3\pi}{2R} - \frac{3 \cdot A_R(M,\mathbf{x})}{R^3}}_{\kappa^R(M,\mathbf{x})} + O(R)$$ [Pottmann et al. 2007]

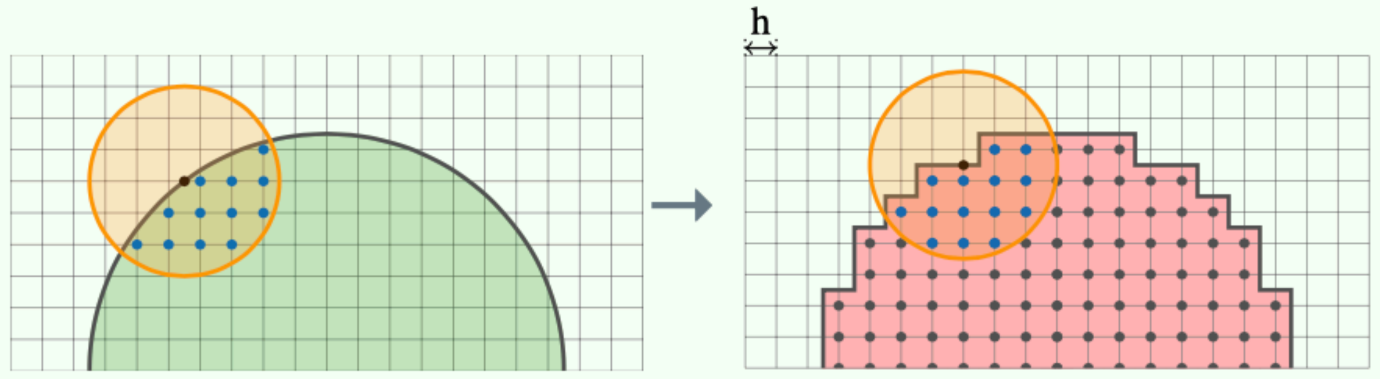


$$A_R(M,\mathbf{x}) \to \widehat{\text{Area}}\left(B_{R/h}(\mathbf{x}/h) \cap \mathrm{G}_h(M)\right)$$



+ [Pottmann et al. 2007]

$$\hat{\kappa}^R(\mathrm{G}_h(M), \mathbf{x}, h)$$



$$\hat{\kappa}^R(\mathrm{G}_h(M), \hat{\mathbf{x}}, h) \to \kappa(M,\mathbf{x})$$

[C., Levallois, Lachaud]

Let M be a convex shape in $\mathbb{R}^2$ with a C^3 bounded positive curvature boundary.

$$\forall \mathbf{x} \in \partial M, \forall \hat{\mathbf{x}} \in \partial[\mathrm{G}_h(M)]_h, \|\hat{x} - x\|_\infty \leq h \Rightarrow$$

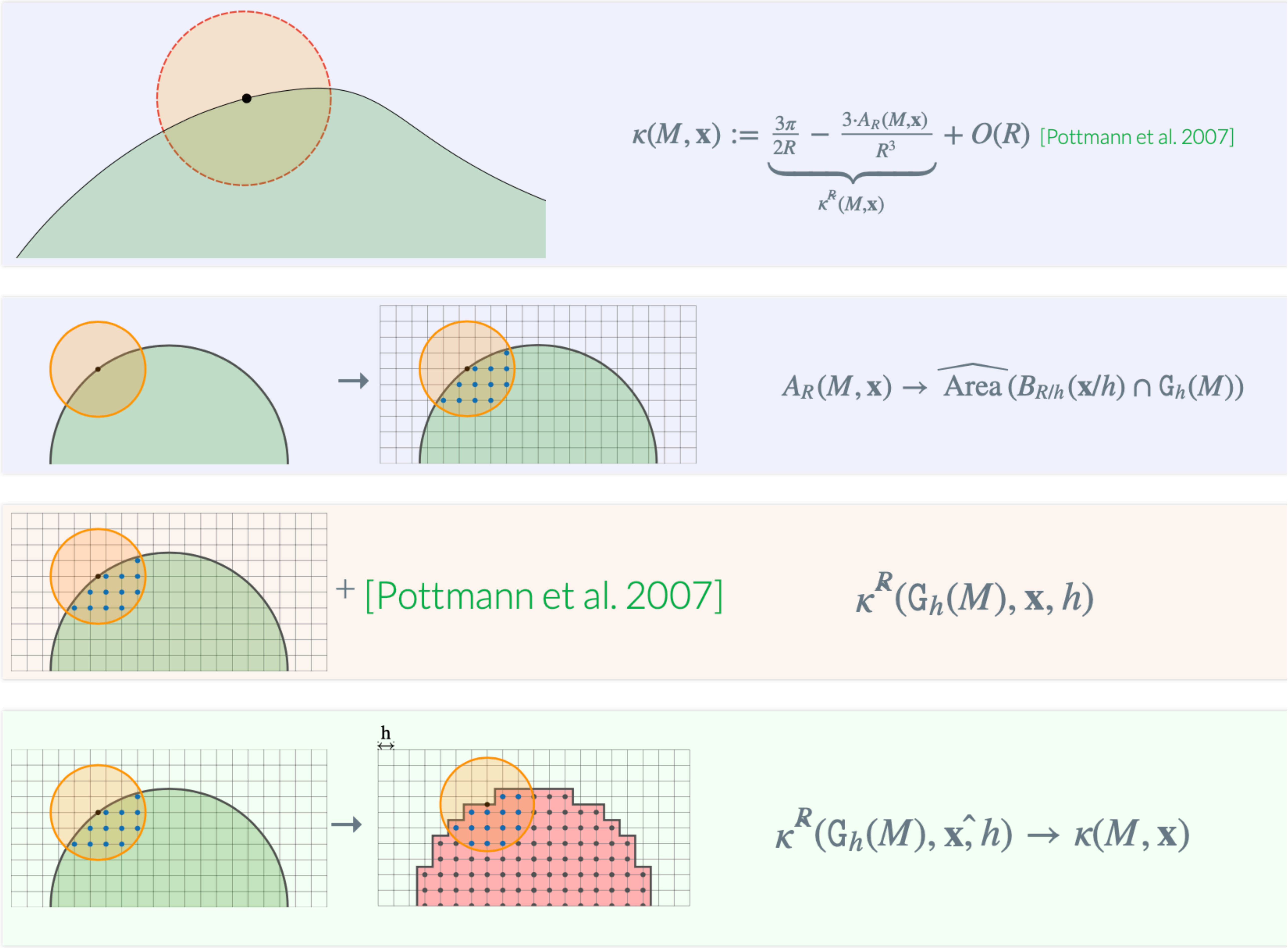
$$|\hat{\kappa}^R(\mathrm{G}_h(M), \hat{\mathbf{x}}, h) - \kappa(M,\mathbf{x})| = O(R) + O\left(\frac{h^\beta}{R^{1+\beta}}\right) + O\left(\frac{h^{\alpha'}}{R^2}\right) + O\left(h^{\alpha'}\right) + O\left(\frac{h^{2\alpha'}}{R^2}\right)$$

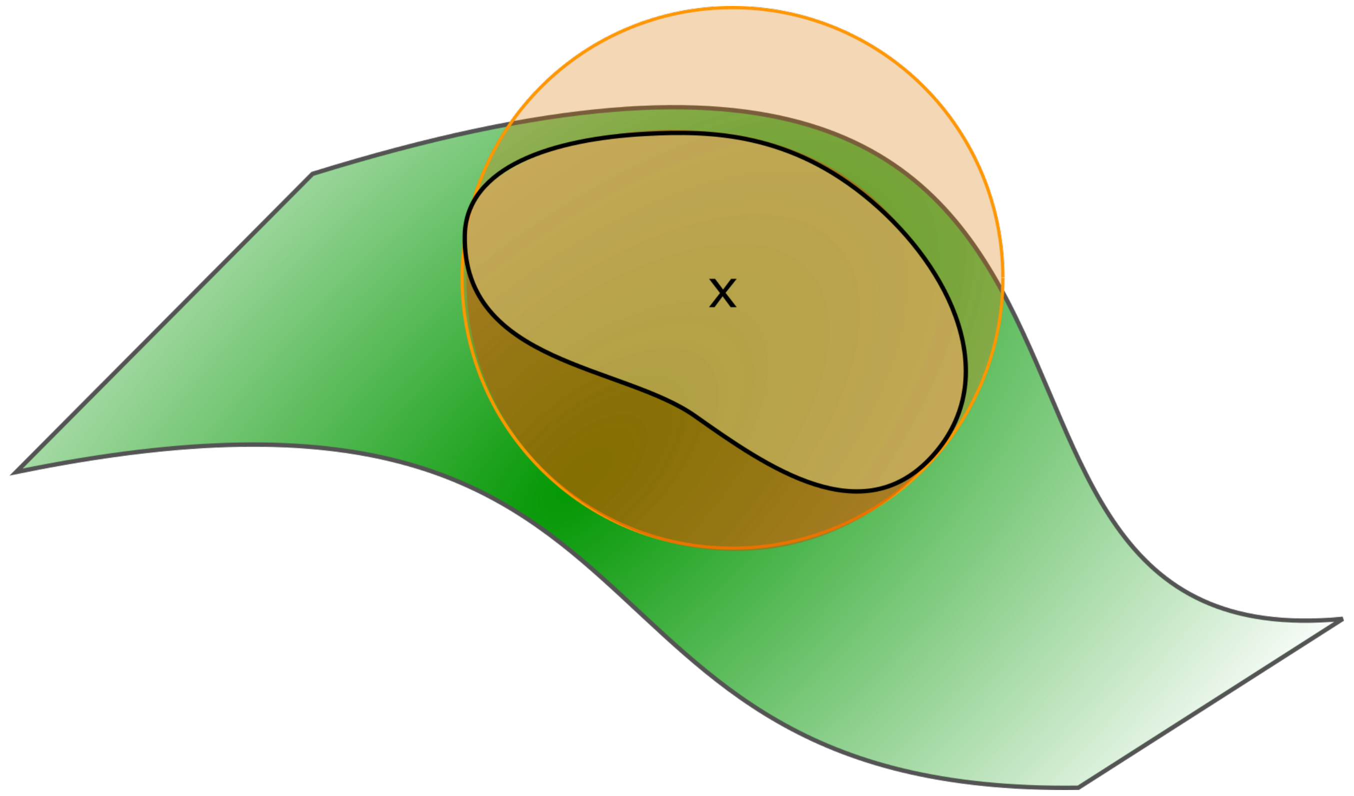
With optimal radius $R = O(h^{\frac{1}{3}})$, then :

- normals $\left\| \hat{\mathbf{n}}(G_h(M), \xi(x), h)) - \mathbf{n}(M,x) \right\| \leq C \cdot h^{\frac{2}{3}}$
- mean curvature $\left\| \hat{\kappa}(M_h, \xi(x))) - \kappa(M,x) \right\|_2 \leq C \cdot h^{\frac{1}{3}}$
- … [CLL2014], [LCL2017]

Normal vector and curvatures estimation

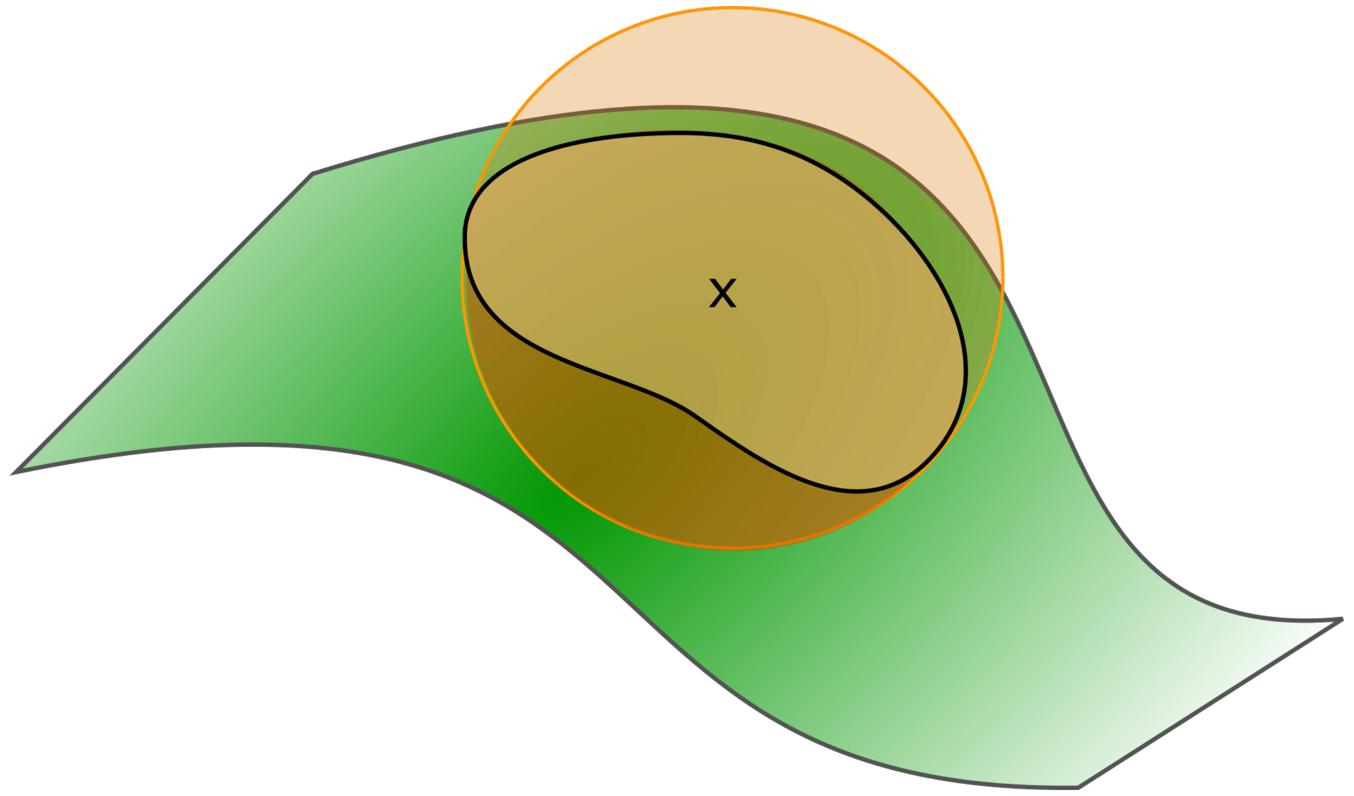
- Integral Invariants : analyzing set $B_R(x) \cap X$ gives normal vector, principal directions and curvatures [Pottmann et al. 2007]

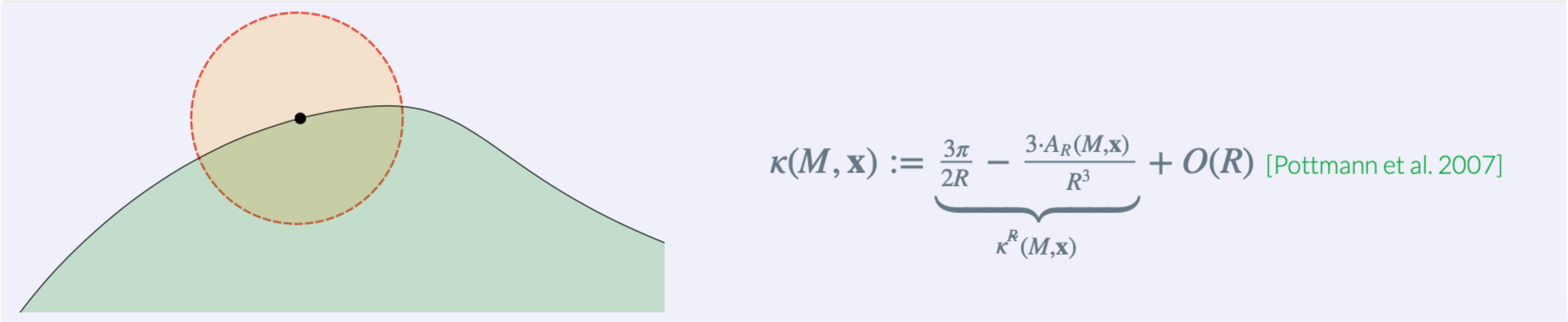


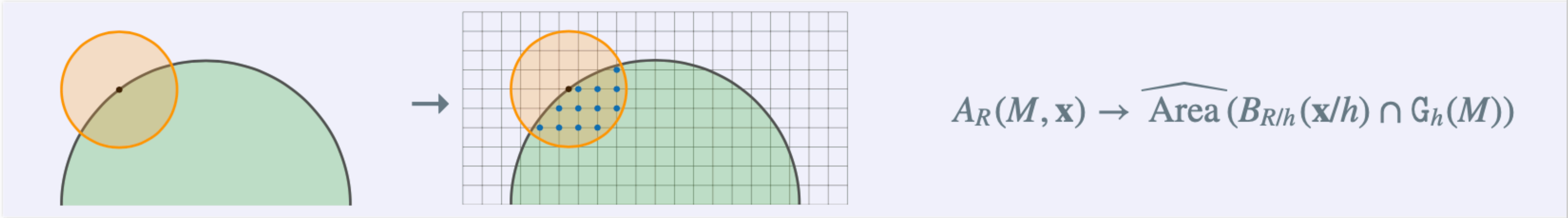


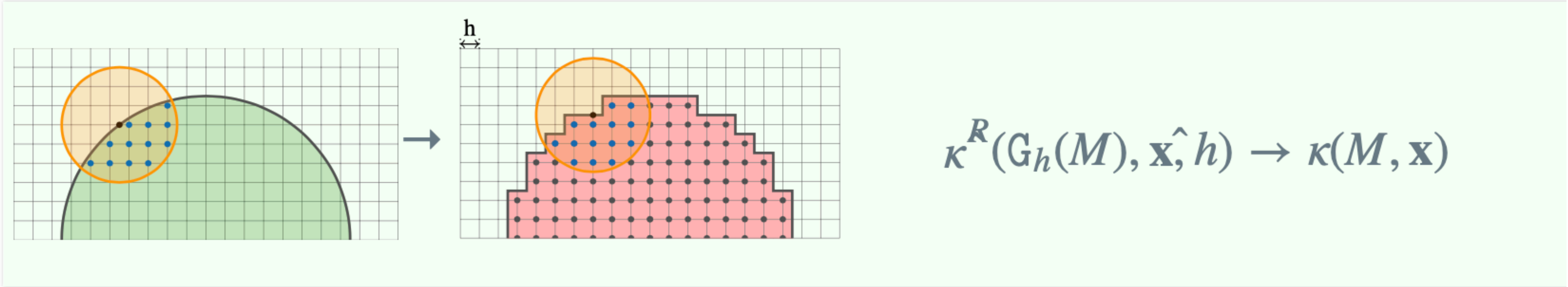
Normal vector and curvatures estimation

- Integral Invariants : analyzing set $B_R(x) \cap X$ gives normal vector, principal directions and curvatures [Pottmann et al. 2007]



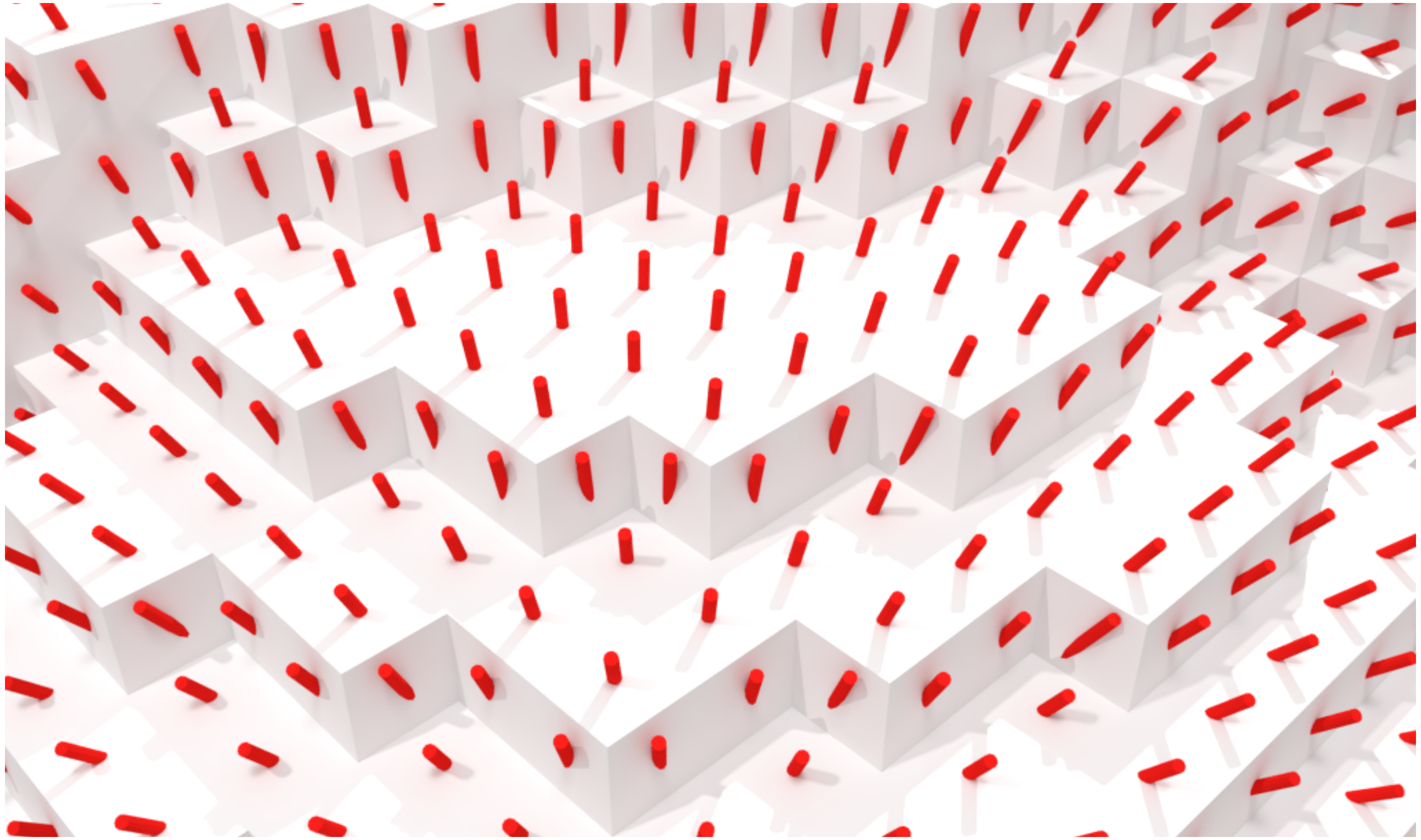


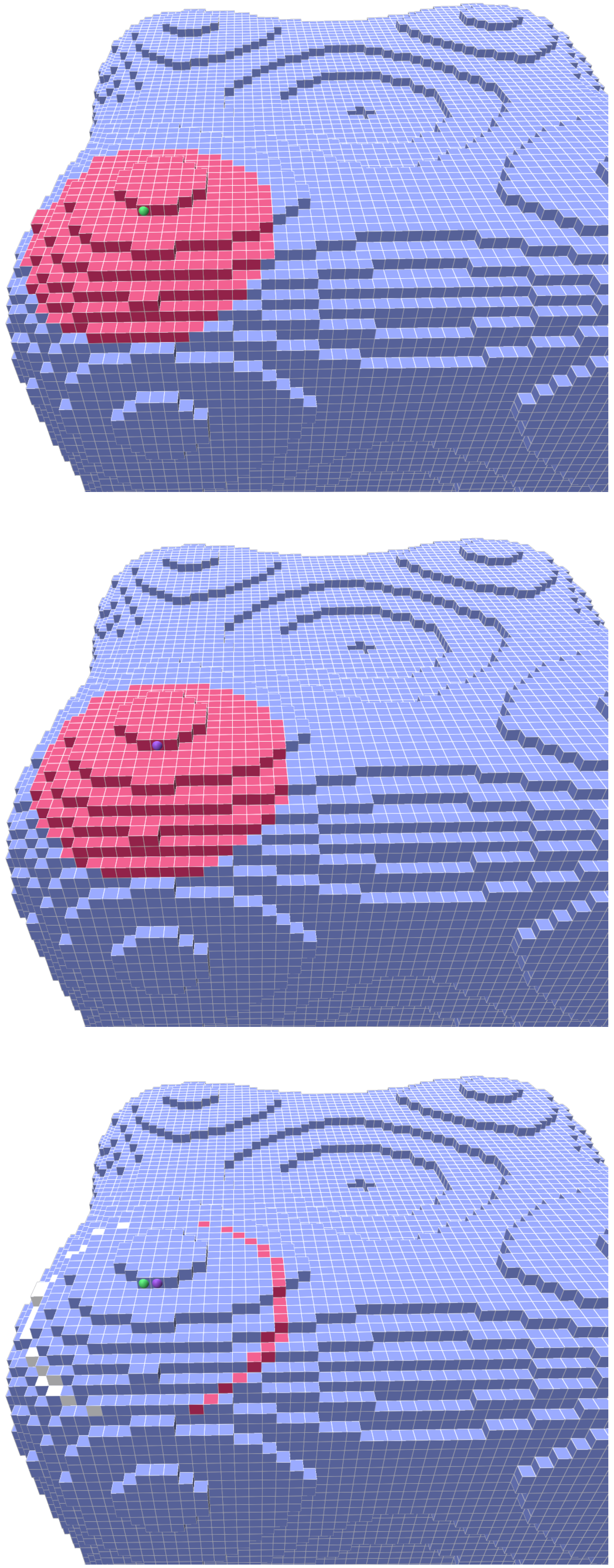




Curvature tensor: covariance matrix instead of the volume of $B_R(x) \cap X$ + eigenvalues / eigenvectors

Normal vector field estimation

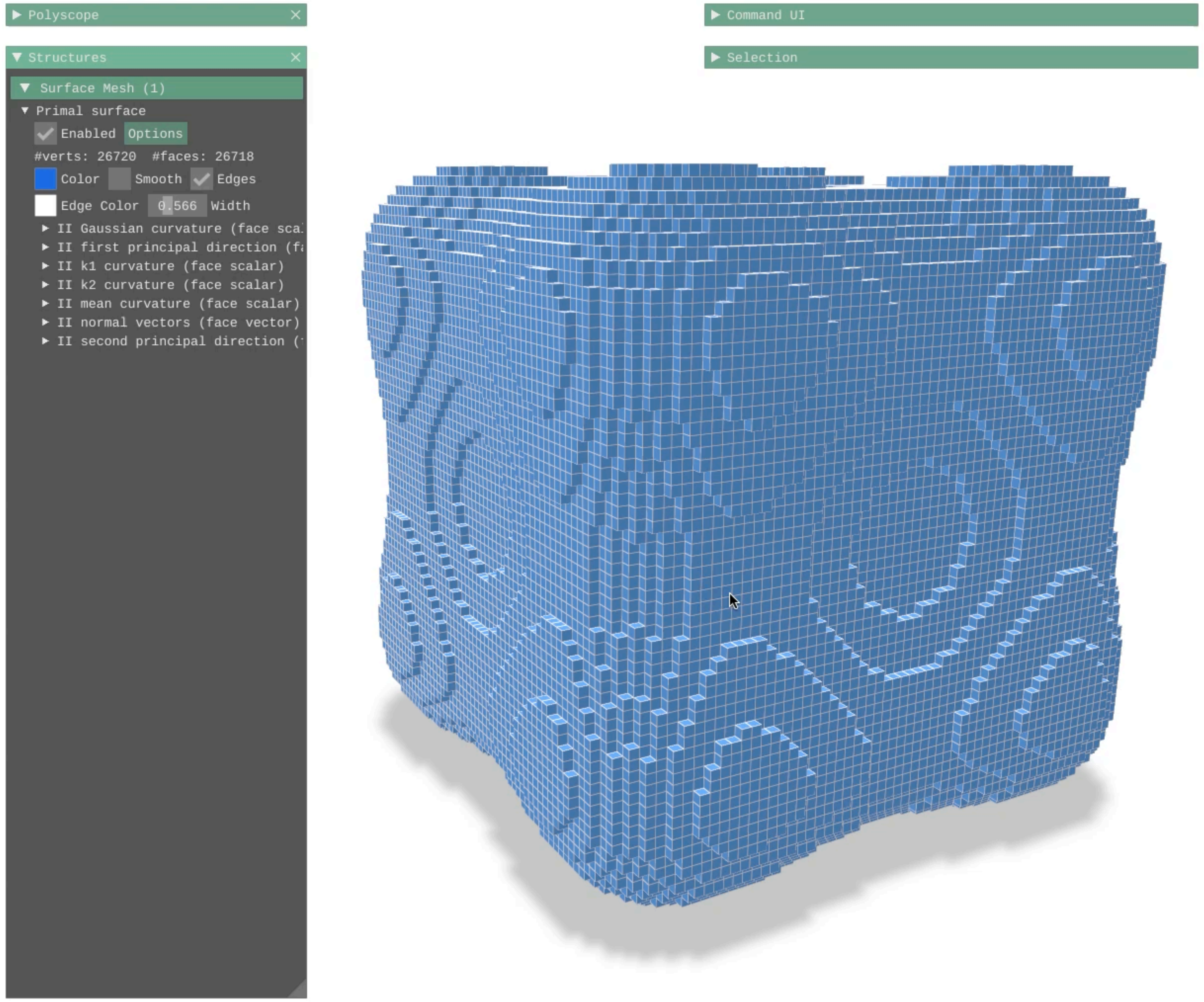




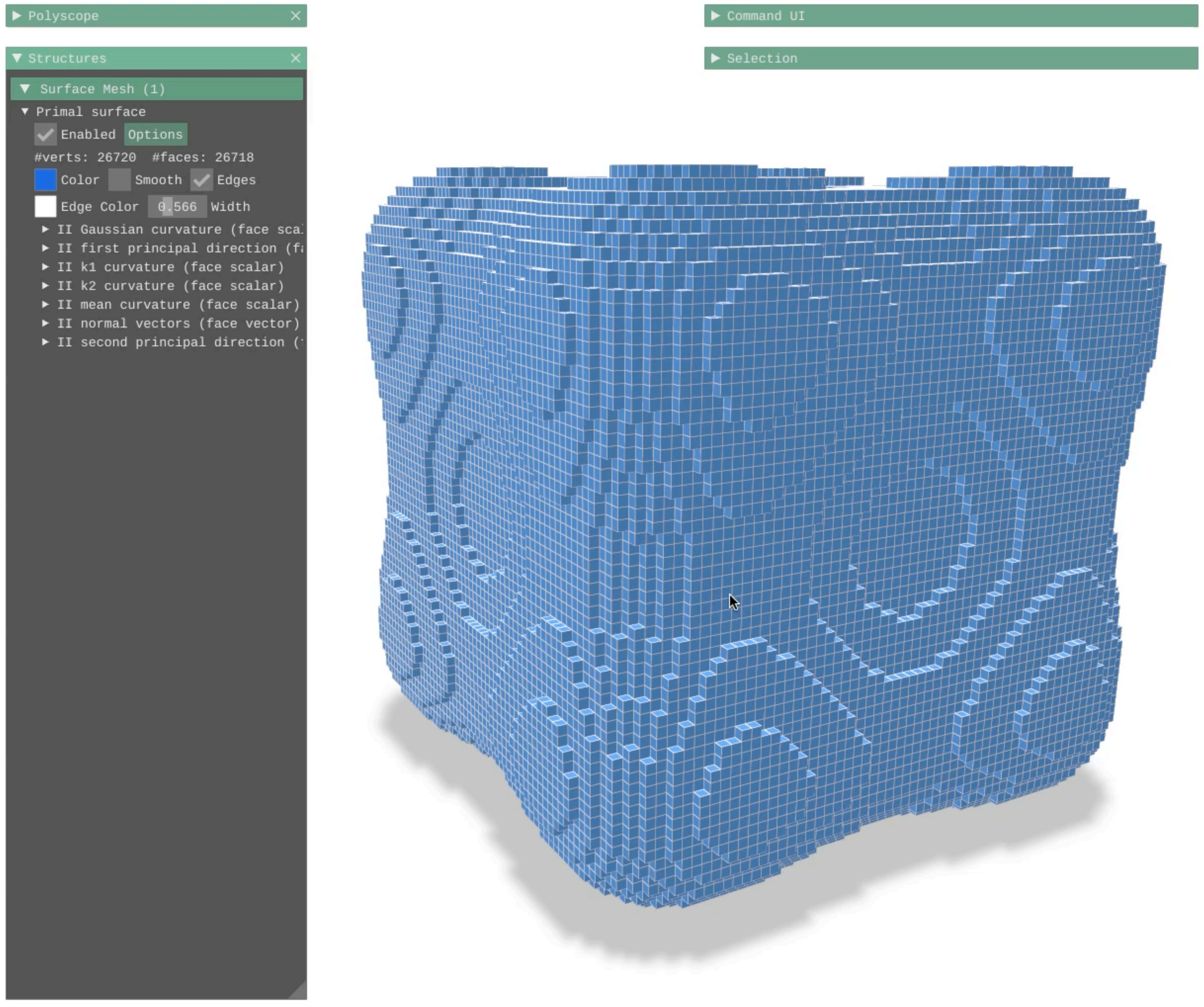
Incremental computation : estimate at y nearby x only requires preceding result + looking at points within $B_R(y) \ominus B_R(x)$

hands on…

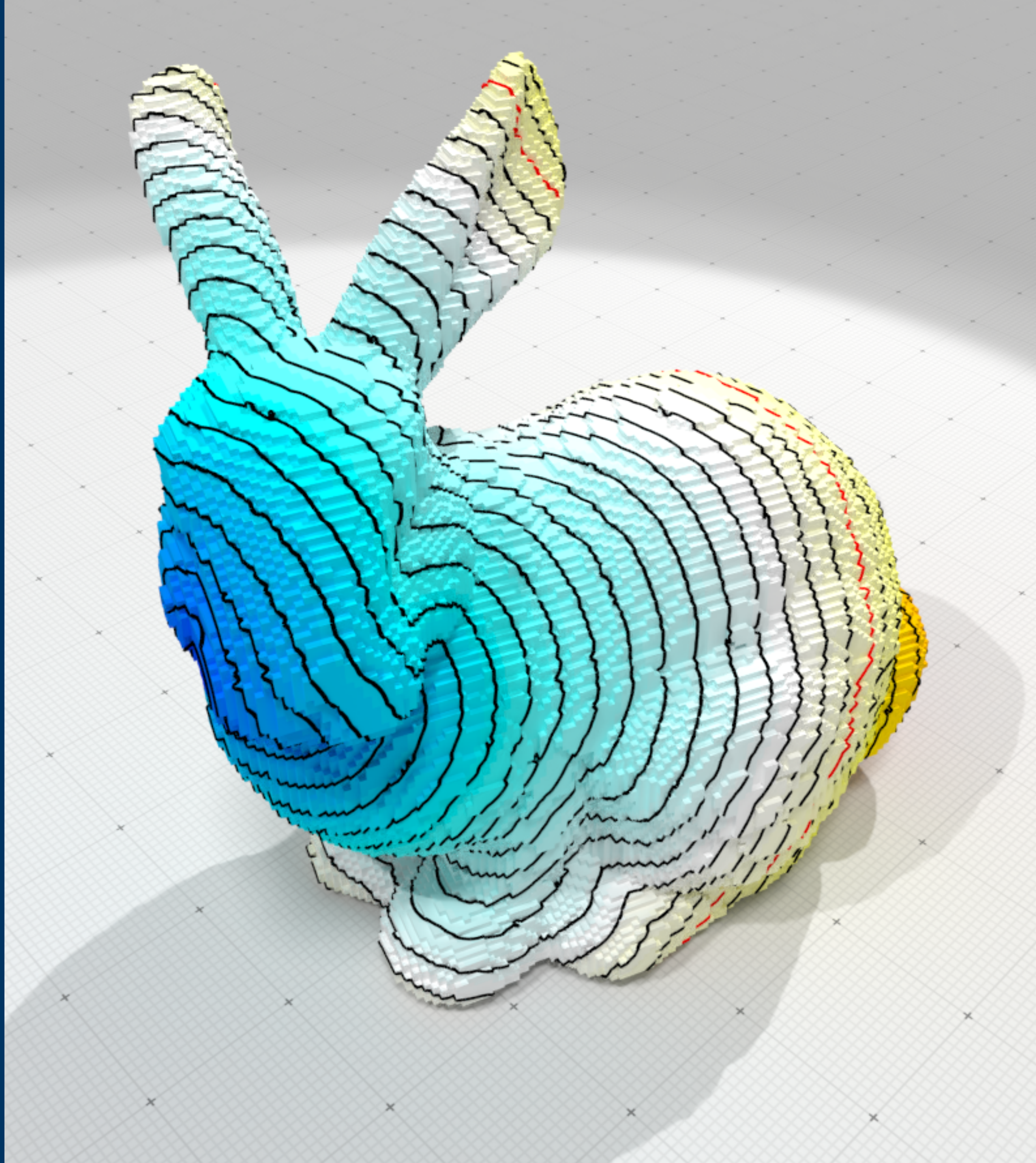
```cpp
void oneStepAll(double h)
{
  auto params = SH3::defaultParameters() | SHG3::defaultParameters() |  SHG3::parametersGeometryEstimation();
  params( "polynomial", "goursat" )( "gridstep", h );
  auto implicit_shape  = SH3::makeImplicitShape3D  ( params );
  auto digitized_shape = SH3::makeDigitizedImplicitShape3D( implicit_shape, params );
  auto K               = SH3::getKSpace( params );
  auto binary_image    = SH3::makeBinaryImage( digitized_shape, params );
  auto surface         = SH3::makeDigitalSurface( binary_image, K, params );
  auto embedder        = SH3::getCellEmbedder( K );
  SH3::Cell2Index c2i;
  auto surfels         = SH3::getSurfelRange( surface, params );
  auto primalSurface   = SH3::makePrimalPolygonalSurface(c2i, surface);

  //Need to convert the faces
  std::vector<std::vector<std::size_t>> faces;
  for(auto &face: primalSurface->allFaces())
    faces.push_back(primalSurface->verticesAroundFace( face ));
  auto digsurf = polyscope::registerSurfaceMesh("Primal surface", primalSurface->positions(), faces);
  digsurf->rescaleToUnit(); digsurf->setEdgeWidth(h*h);  digsurf->setEdgeColor({1.,1.,1.});

  //Computing some differential quantities
  params("r-radius", 5*std::pow(h,-2.0/3.0));
  auto Mcurv     = SHG3::getIIMeanCurvatures(binary_image, surfels, params);
  auto normalsII = SHG3::getIINormalVectors(binary_image, surfels, params);
  auto KTensor   = SHG3::getIIPrincipalCurvaturesAndDirections(binary_image, surfels, params);  //Recomputing...

  std::vector<double> Gcurv(surfels.size()),k1(surfels.size()),k2(surfels.size());
  std::vector<RealVector> d1(surfels.size()),d2(surfels.size());
  auto i=0;
  for(auto &t: KTensor) //AOS->SOA
  {
    k1[i]    = std::get<0>(t);
    k2[i]    = std::get<1>(t);
    d1[i]    = std::get<2>(t);
    d2[i]    = std::get<3>(t);
    Gcurv[i] = k1[i]*k2[i];
    ++i;
  }

  //Attaching quantities
  digsurf->addFaceVectorQuantity("II normal vectors", normalsII, polyscope::VectorType::AMBIENT);
  digsurf->addFaceScalarQuantity("II mean curvature", Mcurv);
  digsurf->addFaceScalarQuantity("II Gaussian curvature", Gcurv);
  digsurf->addFaceScalarQuantity("II k1 curvature", k1);
  digsurf->addFaceScalarQuantity("II k2 curvature", k2);
  digsurf->addFaceVectorQuantity("II first principal direction", d1, polyscope::VectorType::AMBIENT);
  digsurf->addFaceVectorQuantity("II second principal direction", d2, polyscope::VectorType::AMBIENT);
}
```



```cpp
void oneStepAll(double h)
{
  auto params = SH3::defaultParameters() | SHG3::defaultParameters() |  SHG3::parametersGeometryEstimation();
  params( "polynomial", "goursat" )( "gridstep", h );
  auto implicit_shape  = SH3::makeImplicitShape3D  ( params );
  auto digitized_shape = SH3::makeDigitizedImplicitShape3D( implicit_shape, params );
  auto K               = SH3::getKSpace( params );
  auto binary_image    = SH3::makeBinaryImage( digitized_shape, params );
  auto surface         = SH3::makeDigitalSurface( binary_image, K, params );
  auto embedder        = SH3::getCellEmbedder( K );
  SH3::Cell2Index c2i;
  auto surfels         = SH3::getSurfelRange( surface, params );
  auto primalSurface   = SH3::makePrimalPolygonalSurface(c2i, surface);

  //Need to convert the faces
  std::vector<std::vector<std::size_t>> faces;
  for(auto &face: primalSurface->allFaces())
    faces.push_back(primalSurface->verticesAroundFace( face ));
  auto digsurf = polyscope::registerSurfaceMesh("Primal surface", primalSurface->positions(), faces);
  digsurf->rescaleToUnit(); digsurf->setEdgeWidth(h*h);  digsurf->setEdgeColor({1.,1.,1.});

  //Computing some differential quantities
  params("r-radius", 5*std::pow(h,-2.0/3.0));
  auto Mcurv     = SHG3::getIIMeanCurvatures(binary_image, surfels, params);
  auto normalsII = SHG3::getIINormalVectors(binary_image, surfels, params);
  auto KTensor   = SHG3::getIIPrincipalCurvaturesAndDirections(binary_image, surfels, params);  //Recomputing...

  std::vector<double> Gcurv(surfels.size()),k1(surfels.size()),k2(surfels.size());
  std::vector<RealVector> d1(surfels.size()),d2(surfels.size());
  auto i=0;
  for(auto &t: KTensor) //AOS->SOA
  {
    k1[i]    = std::get<0>(t);
    k2[i]    = std::get<1>(t);
    d1[i]    = std::get<2>(t);
    d2[i]    = std::get<3>(t);
    Gcurv[i] = k1[i]*k2[i];
    ++i;
  }

  //Attaching quantities
  digsurf->addFaceVectorQuantity("II normal vectors", normalsII, polyscope::VectorType::AMBIENT);
  digsurf->addFaceScalarQuantity("II mean curvature", Mcurv);
  digsurf->addFaceScalarQuantity("II Gaussian curvature", Gcurv);
  digsurf->addFaceScalarQuantity("II k1 curvature", k1);
  digsurf->addFaceScalarQuantity("II k2 curvature", k2);
  digsurf->addFaceVectorQuantity("II first principal direction", d1, polyscope::VectorType::AMBIENT);
  digsurf->addFaceVectorQuantity("II second principal direction", d2, polyscope::VectorType::AMBIENT);
}
```



advanced digital surface geometry processing



$$\dot{u} = \Delta u \qquad u(0) = u_0$$

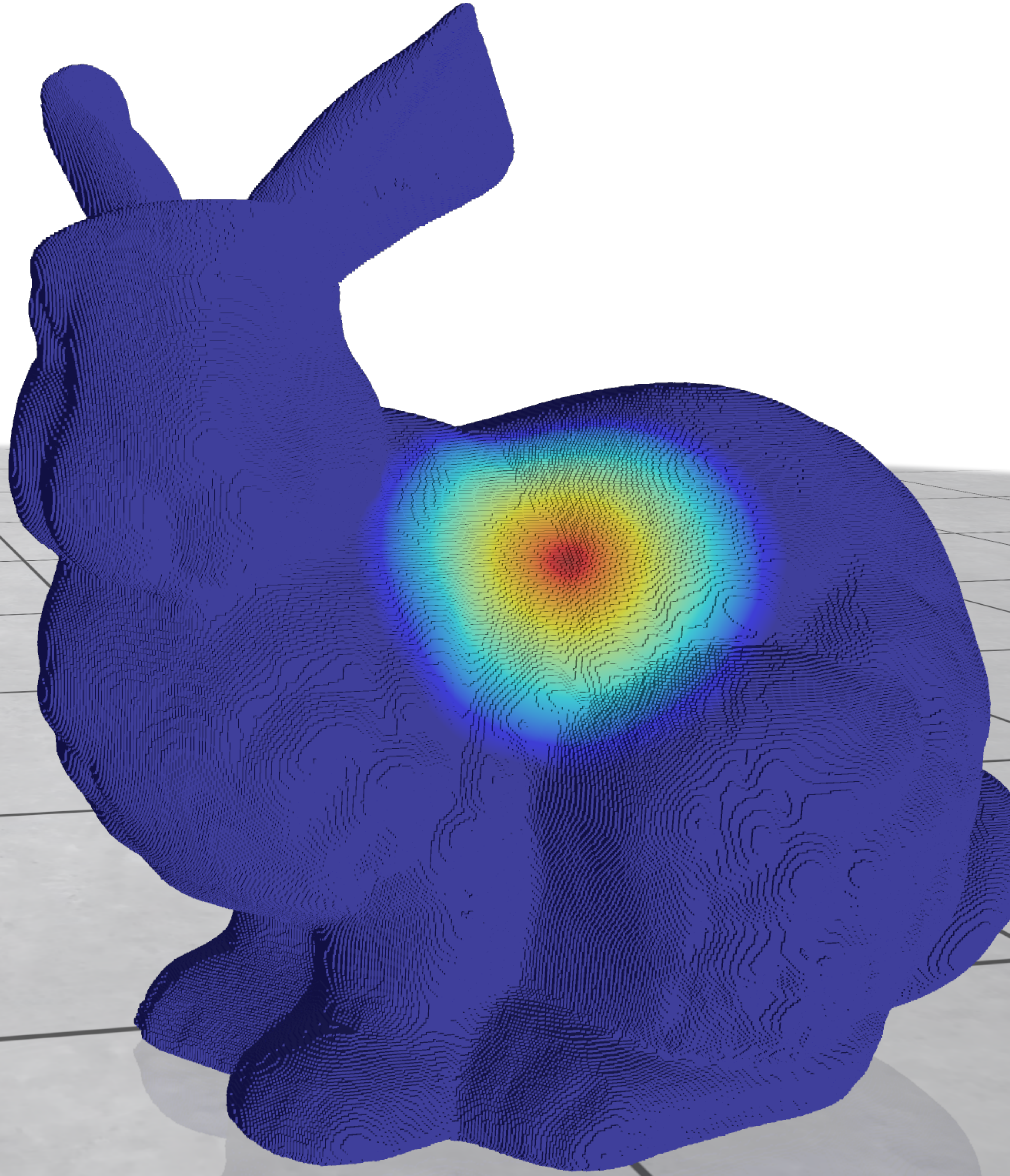
$\dot{u} = \Delta u \qquad u(0) = u_0$

u

$$\dot{u} = \Delta u \qquad u(0) = u_0$$

u

∇u

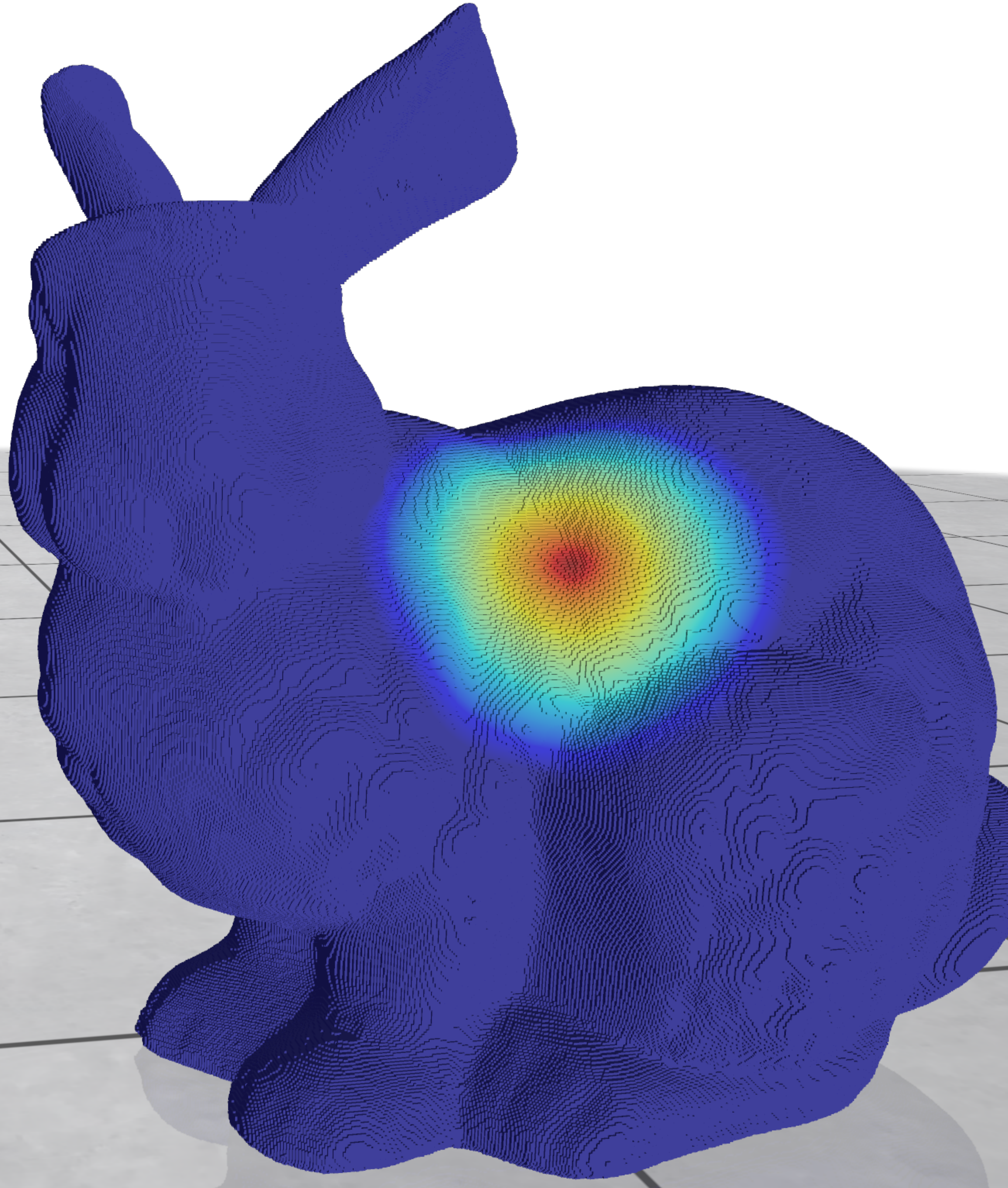


$\dot{u} = \Delta u \qquad u(0) = u_0$

u

∇u

$\operatorname{div} \vec{F}$



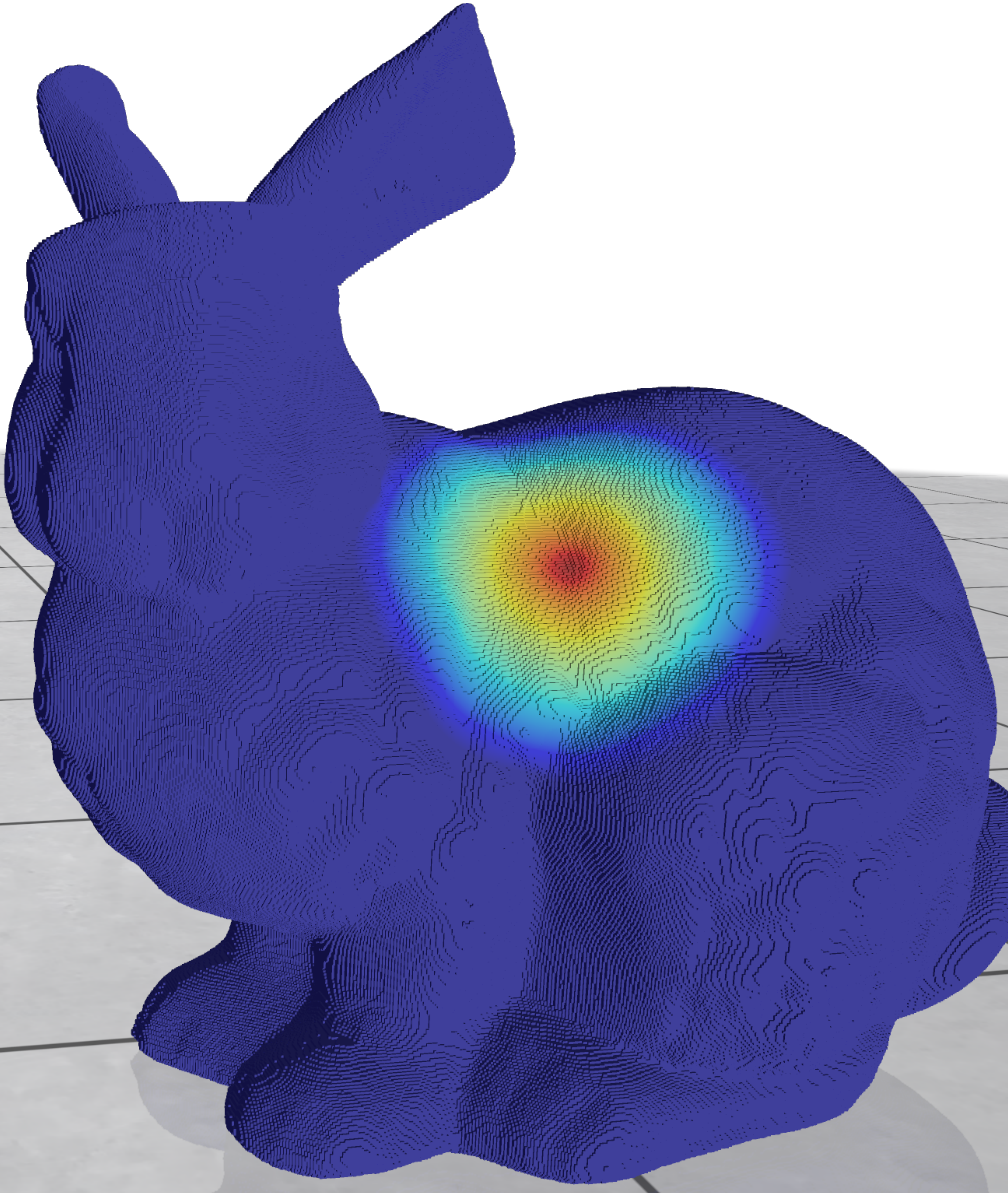
$\dot{u} = \Delta u \qquad u(0) = u_0$

u

∇u

$\text{div}\ \overrightarrow{F}$

$\text{curl}\ \overrightarrow{F}$



$\dot{u} = \Delta u \qquad u(0) = u_0$

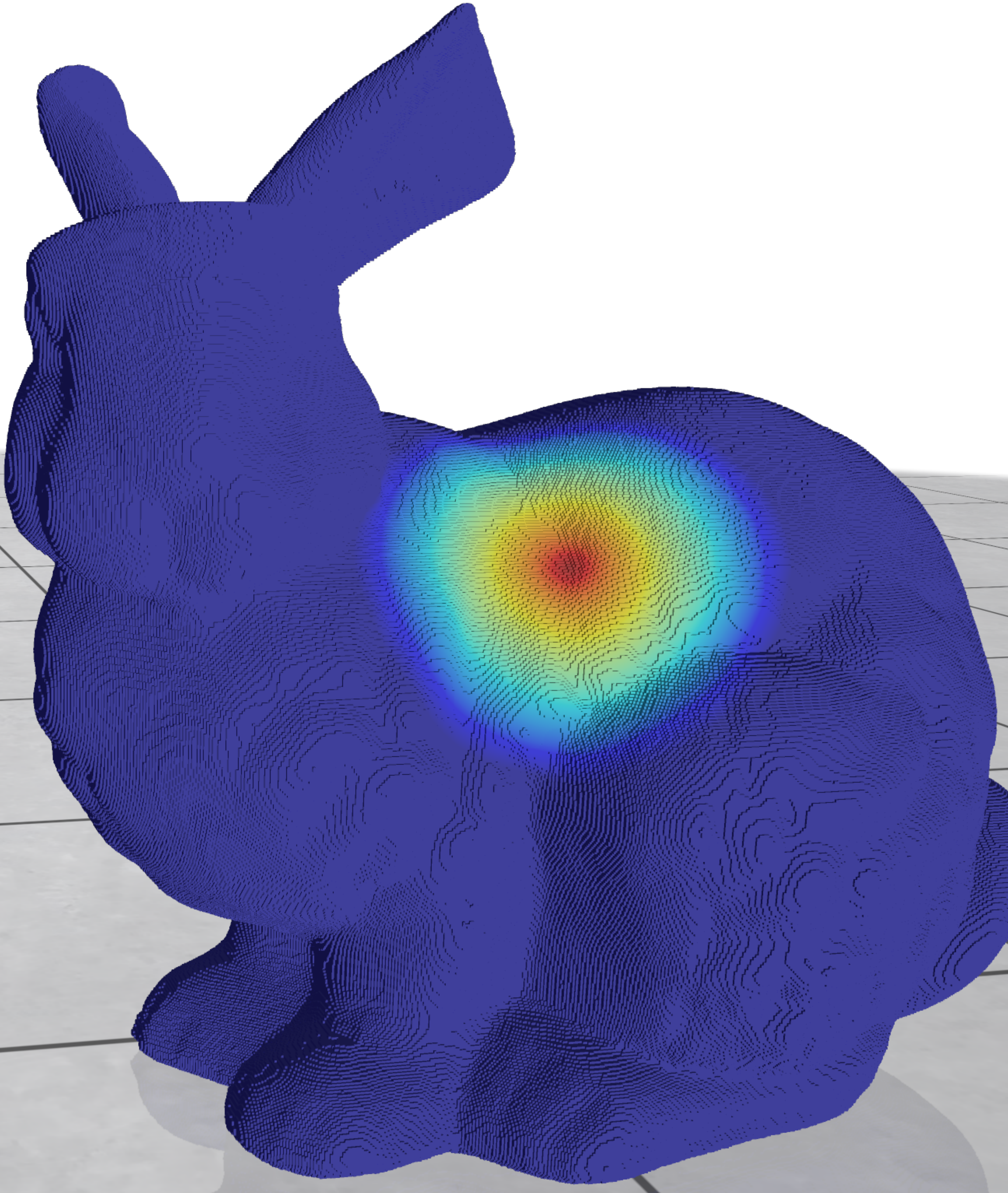
u

∇u

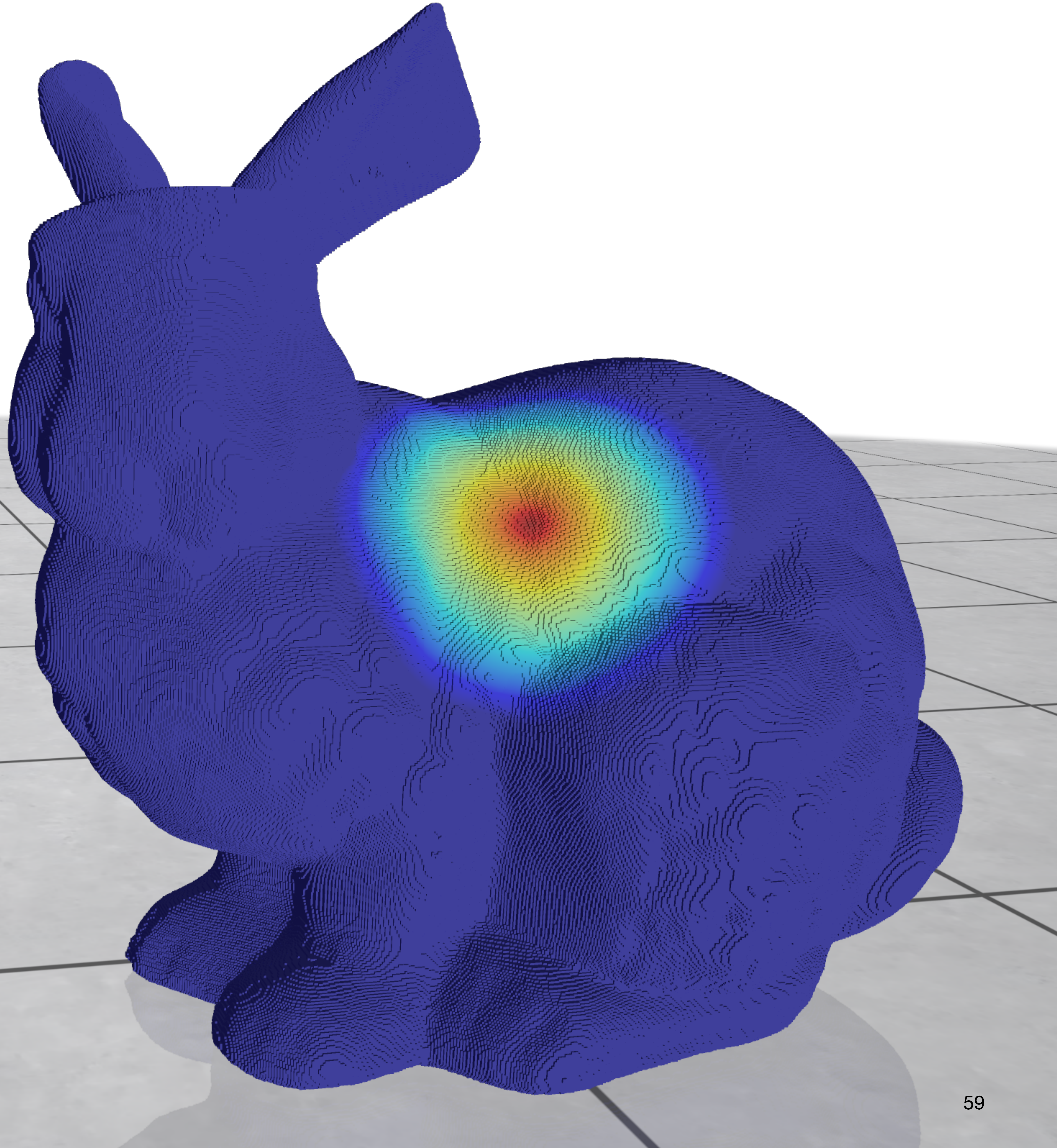
$\operatorname{div} \overrightarrow{F}$

$\operatorname{curl} \overrightarrow{F}$

$\Delta u := \operatorname{div} \nabla u$



$\dot{u} = \Delta u \qquad u(0) = u_0$



u

∇u

$\operatorname{div} \vec{F}$

$\operatorname{curl} \vec{F}$

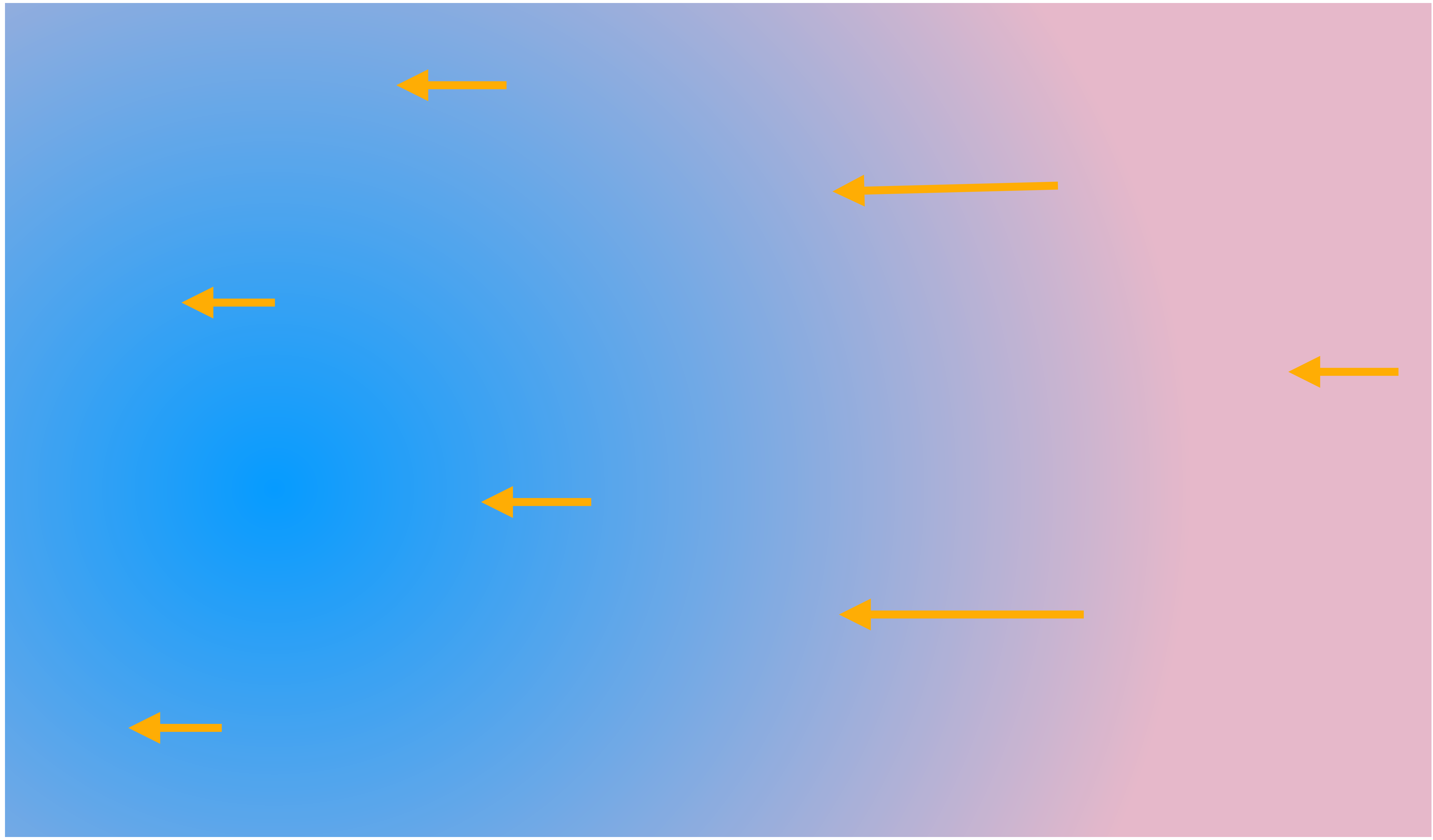
$\Delta u := \operatorname{div} \nabla u$

$\Delta u = g$

Calculus on a continuous setting

$$f : \mathbb{R}^2 \to \mathbb{R}$$
$$(x, y) \mapsto f(x, y)$$

Calculus on a continuous setting

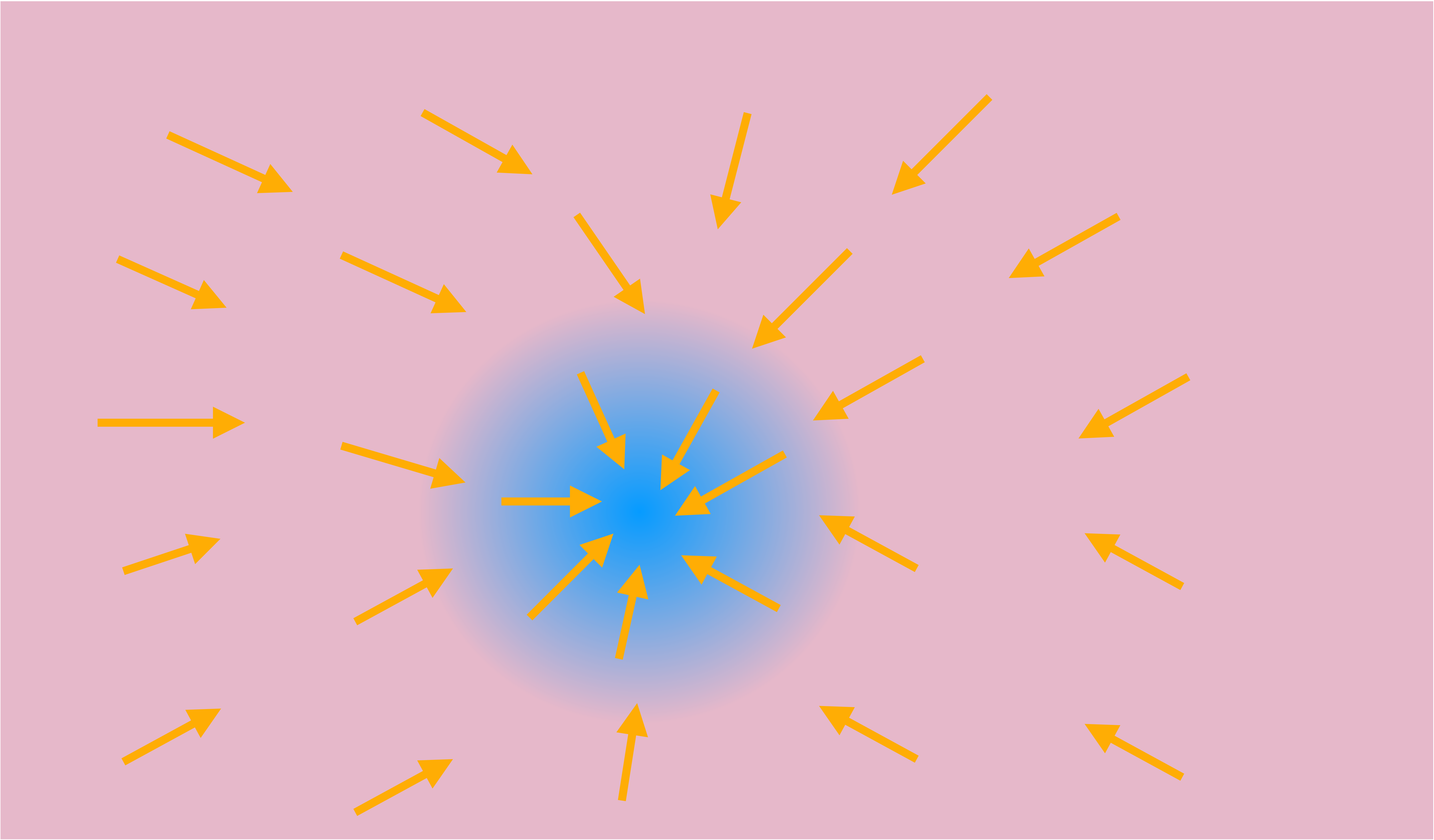


$$f : \mathbb{R}^2 \to \mathbb{R}$$

$$(x, y) \mapsto f(x, y)$$

$$\nabla f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right)^t$$

Calculus on a continuous setting

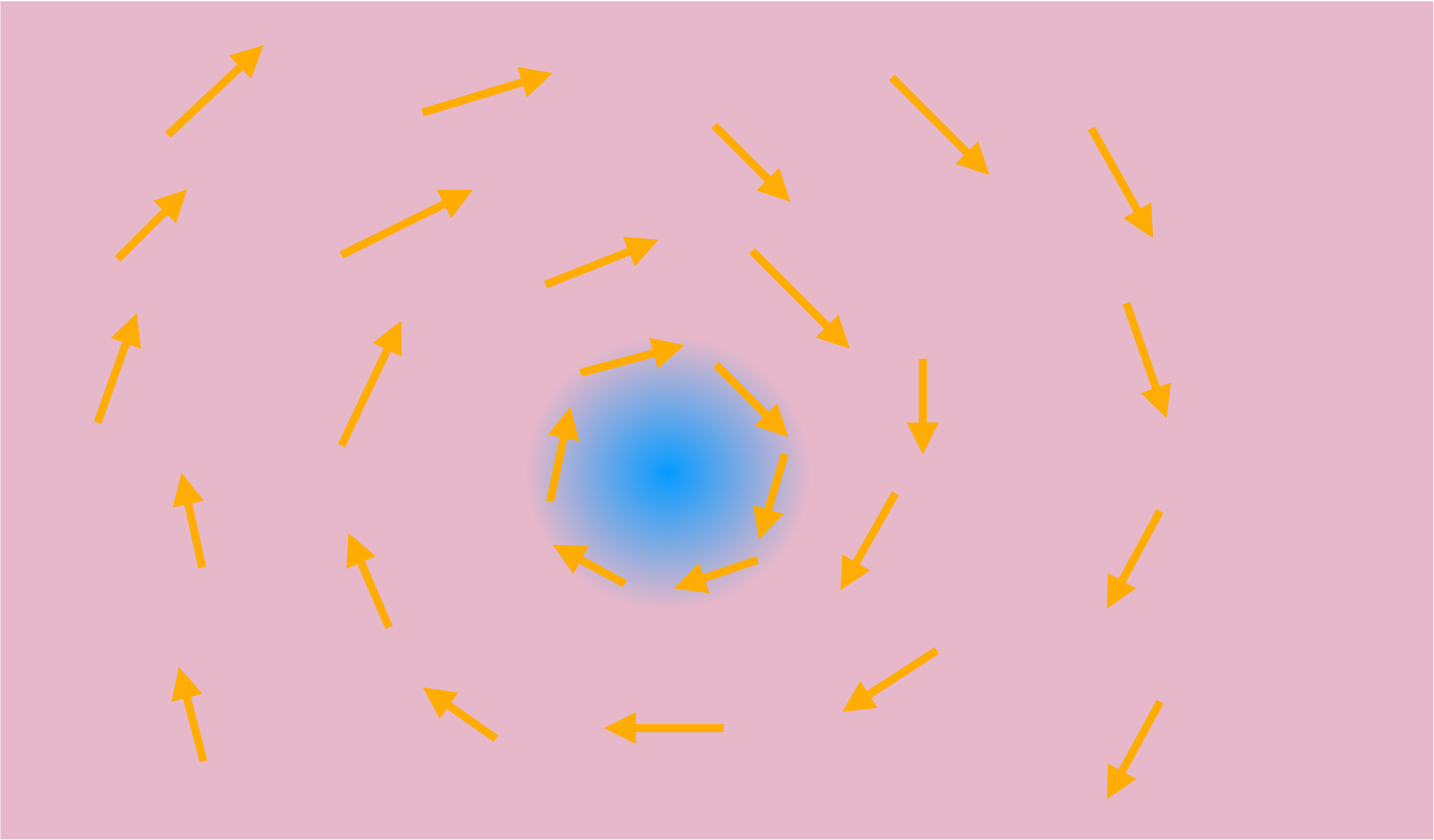


$$f : \mathbb{R}^2 \to \mathbb{R}$$
$$(x, y) \mapsto f(x, y)$$

$$\nabla f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right)^t$$

$$\text{div}\, F = \frac{\partial F_x}{\partial x} + \frac{\partial F_y}{\partial y}$$

Calculus on a continuous setting



$$f : \mathbb{R}^2 \to \mathbb{R}$$

$$(x, y) \mapsto f(x, y)$$

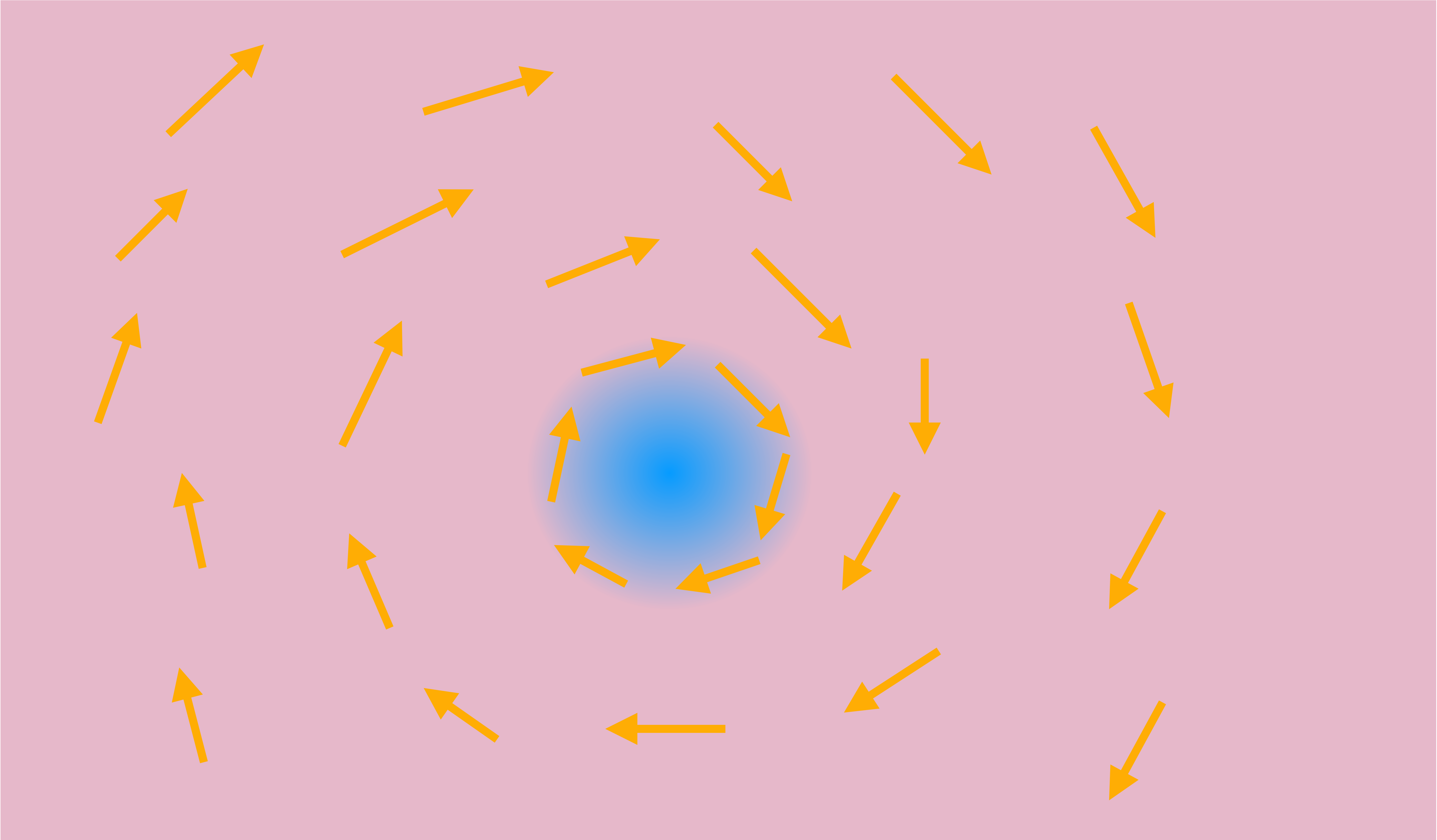
$$\nabla f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right)^t$$

$$\operatorname{div} F = \frac{\partial F_x}{\partial x} + \frac{\partial F_y}{\partial y}$$

$$\operatorname{curl} F = -\frac{\partial F_y}{\partial x} + \frac{\partial F_x}{\partial y}$$

$$= -\operatorname{div} JF$$

Calculus on a continuous setting



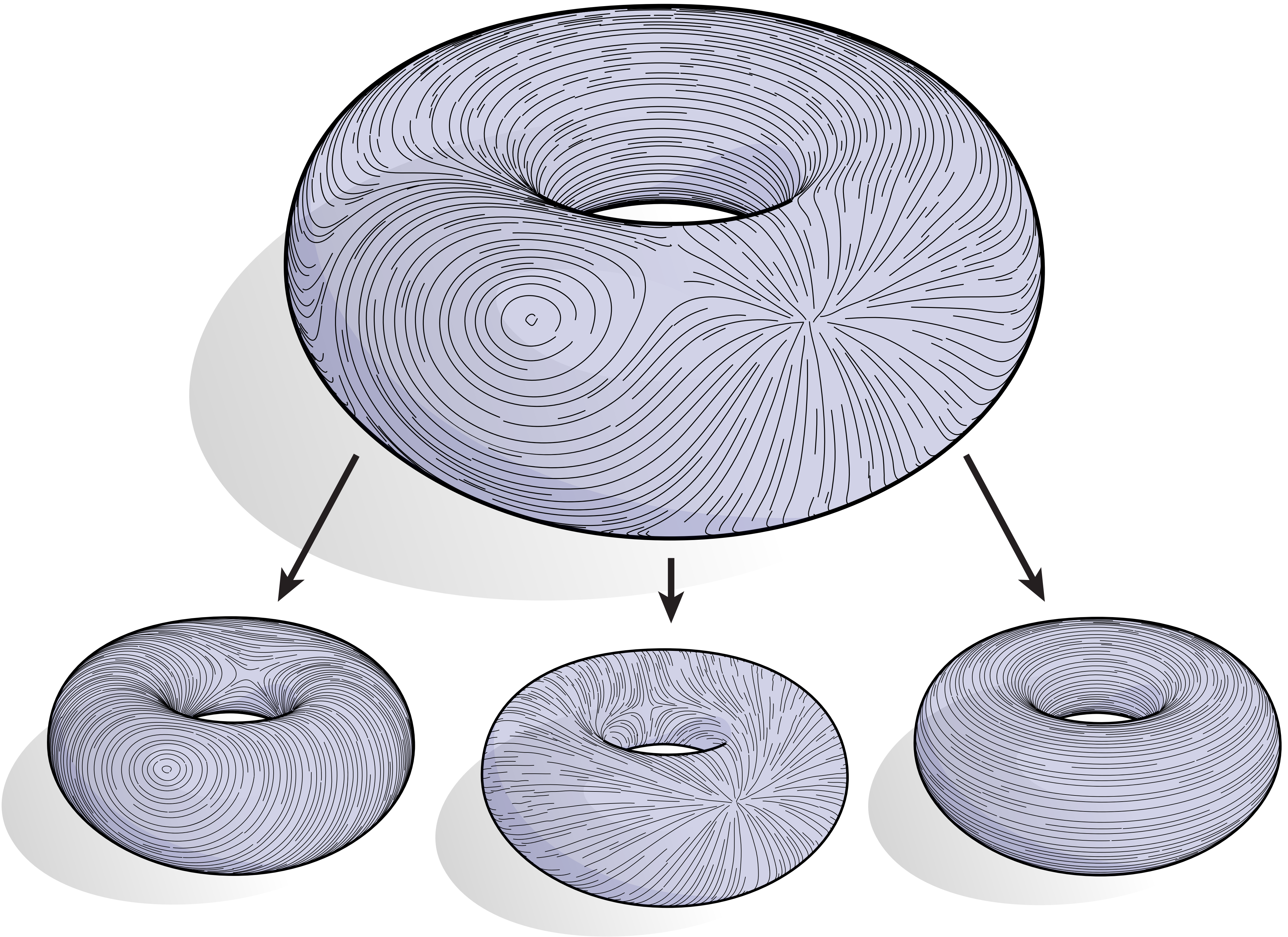
$$f : \mathbb{R}^2 \to \mathbb{R}$$
$$(x, y) \mapsto f(x, y)$$

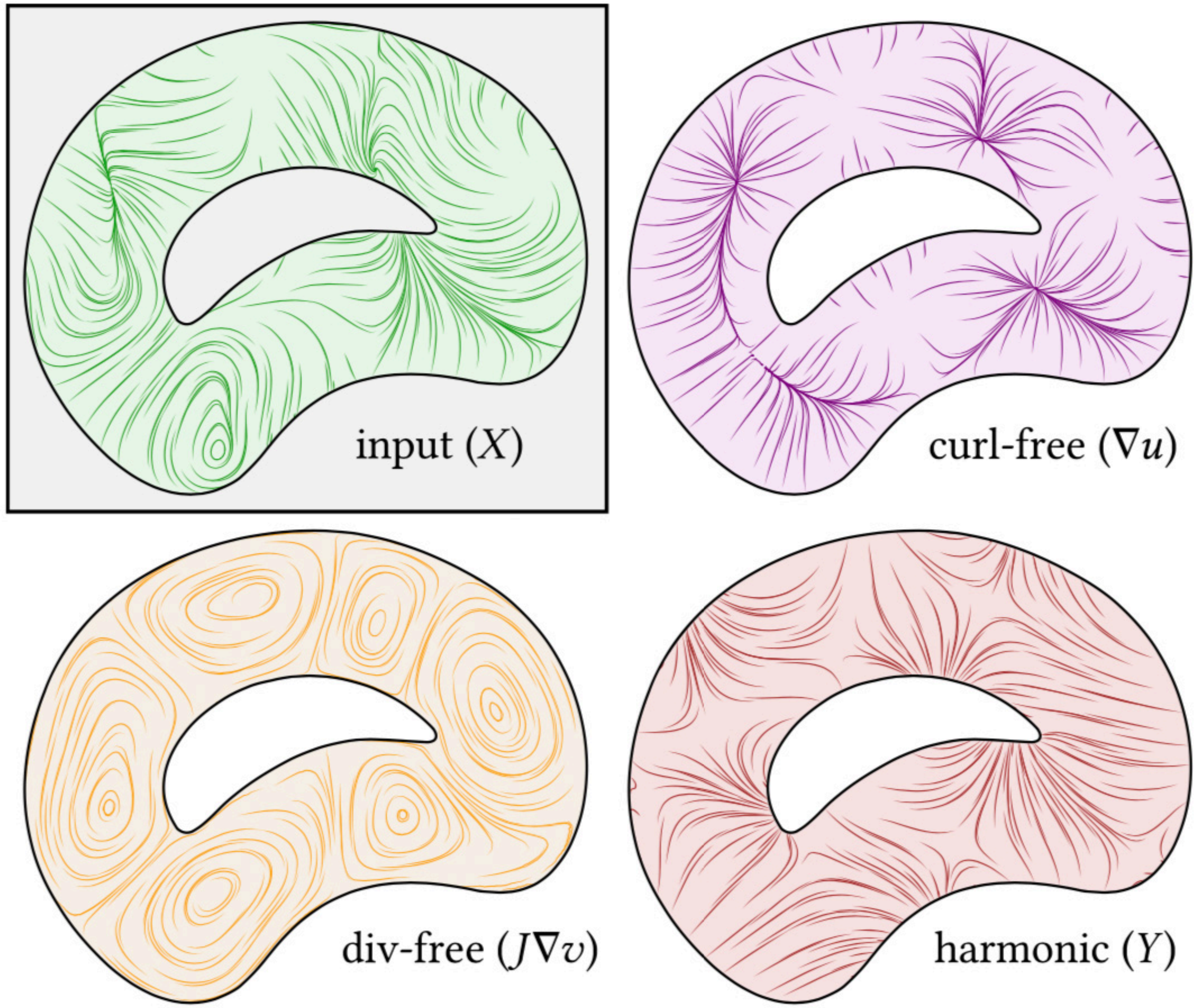
$$\nabla f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right)^t$$

$$\operatorname{div} F = \frac{\partial F_x}{\partial x} + \frac{\partial F_y}{\partial y}$$

$$\operatorname{curl} F = -\frac{\partial F_y}{\partial x} + \frac{\partial F_x}{\partial y}$$
$$= -\operatorname{div} JF$$

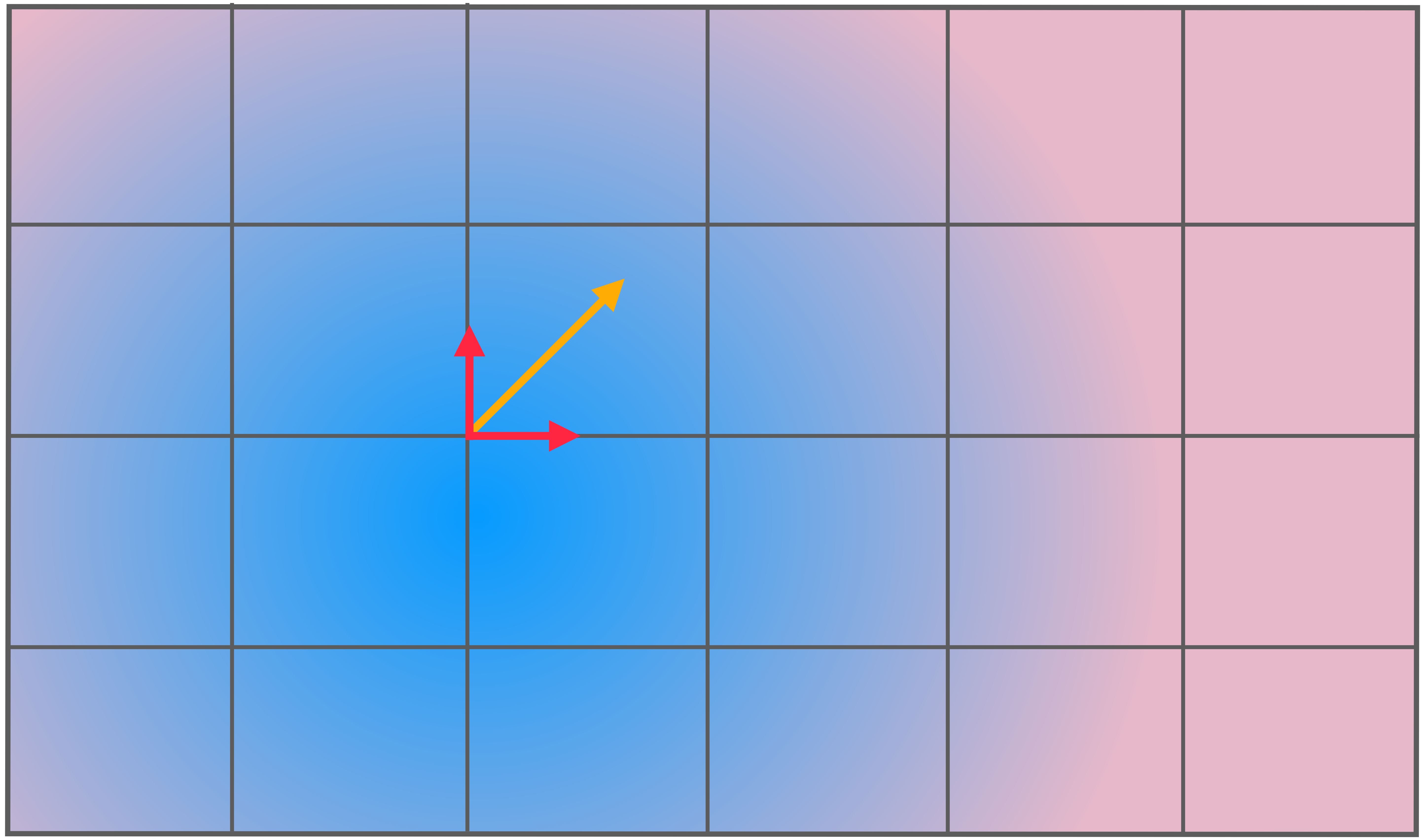
$$\operatorname{curl} \nabla f = 0 \qquad \Delta f = \operatorname{div} \nabla f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \qquad \int_{\partial\Omega} F \cdot ds = \iint_{\Omega} \operatorname{curl} F \cdot d\omega \ \ldots$$





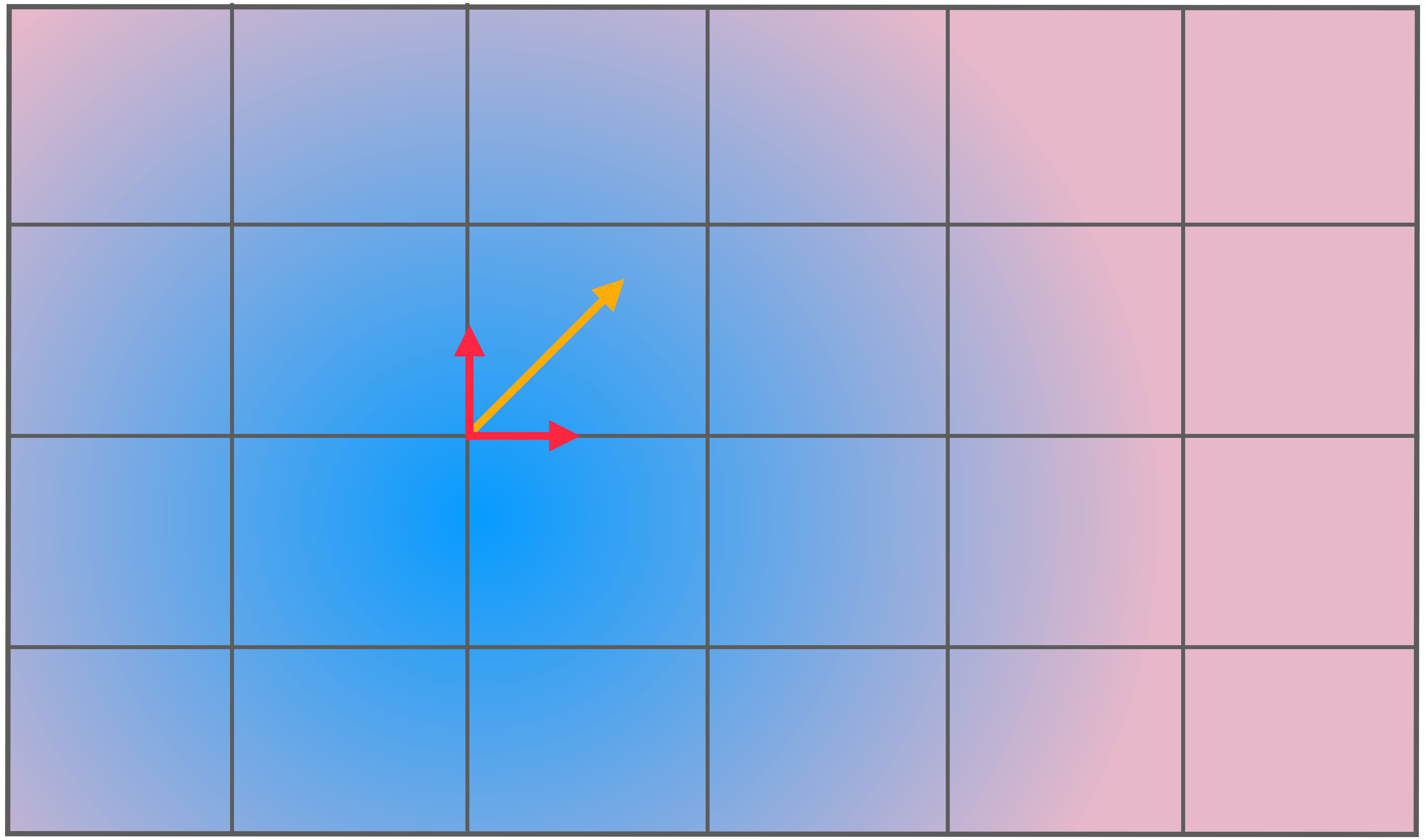
$$\mathscr{X} = \text{Image(grad)} \oplus \text{Image(J grad)} \oplus \mathscr{H}$$

Discrete setting: regular grid



$$\nabla^h f := \left(\frac{f(x+h) - f(x)}{h}, \frac{f(y+h) - f(y)}{h} \right)^t$$

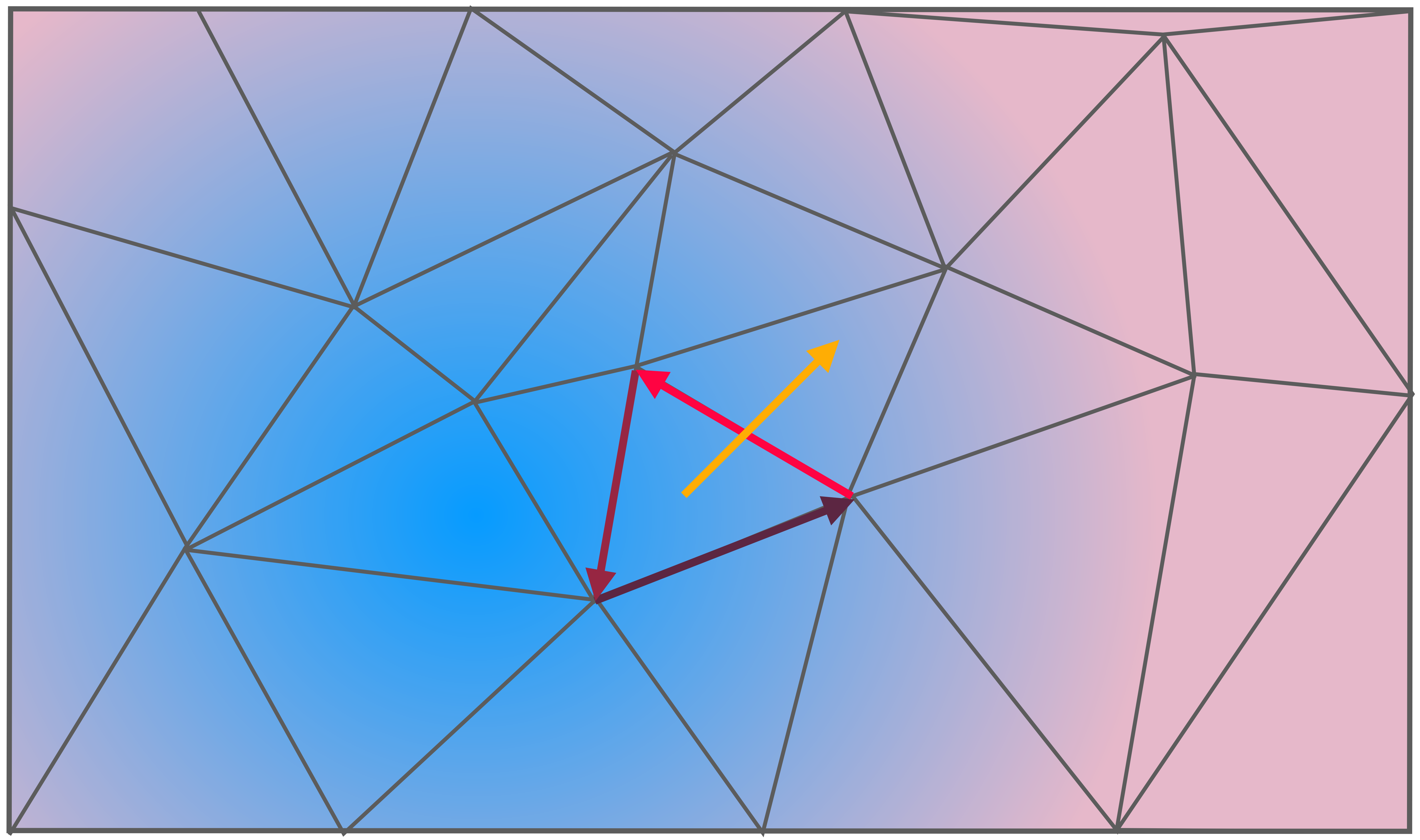
Discrete setting: regular grid



$$\nabla^h f := \left(\frac{f(x+h) - f(x)}{h}, \frac{f(y+h) - f(y)}{h} \right)^t$$

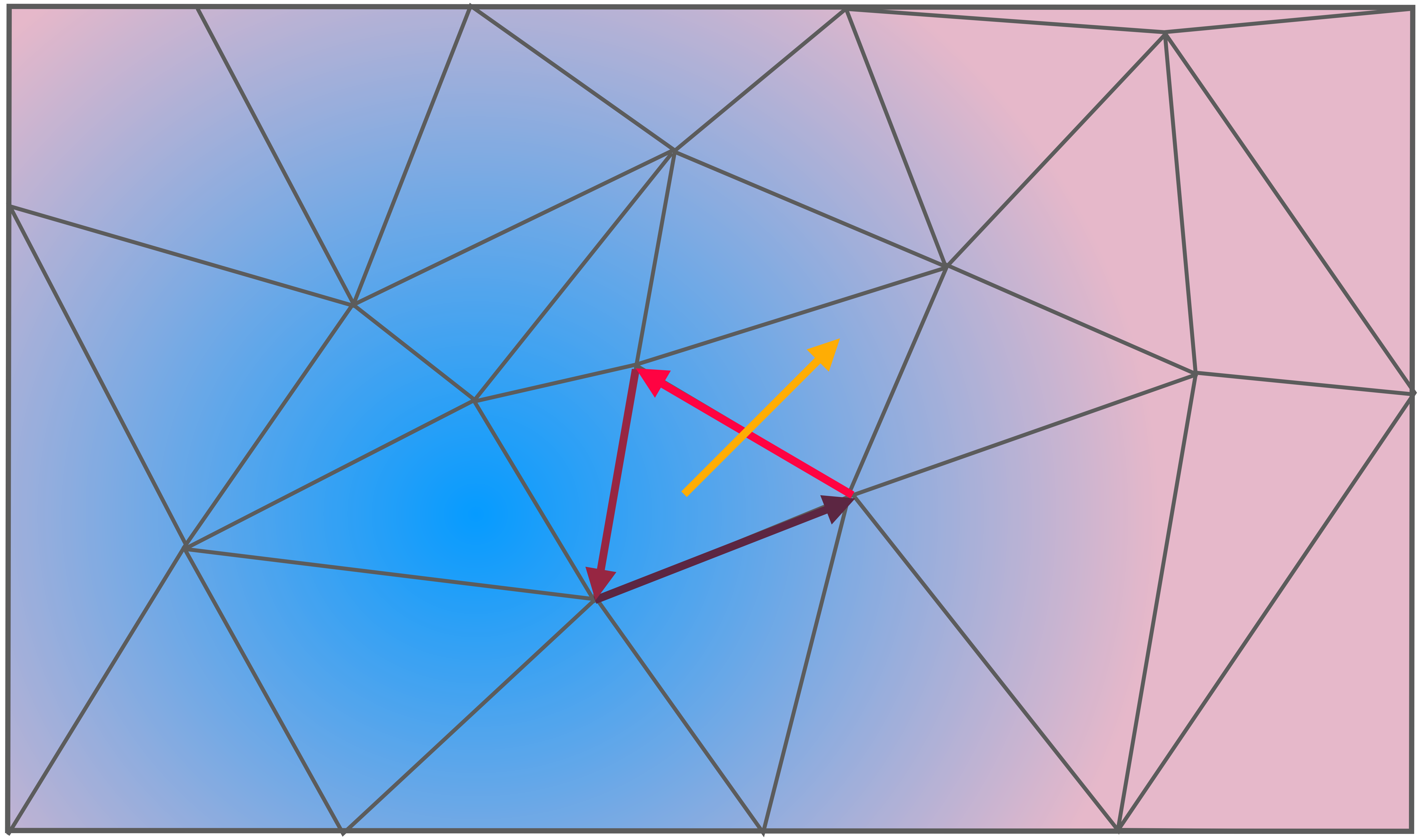
$$\Delta^h f := \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} + \frac{f(y+h) - 2f(y) + f(y-h)}{h^2}$$

Discrete setting: triangular surfaces



$$f(p) = f_i\phi_i + f_j\phi_j + f_k\phi_k$$

Discrete setting: triangular surfaces

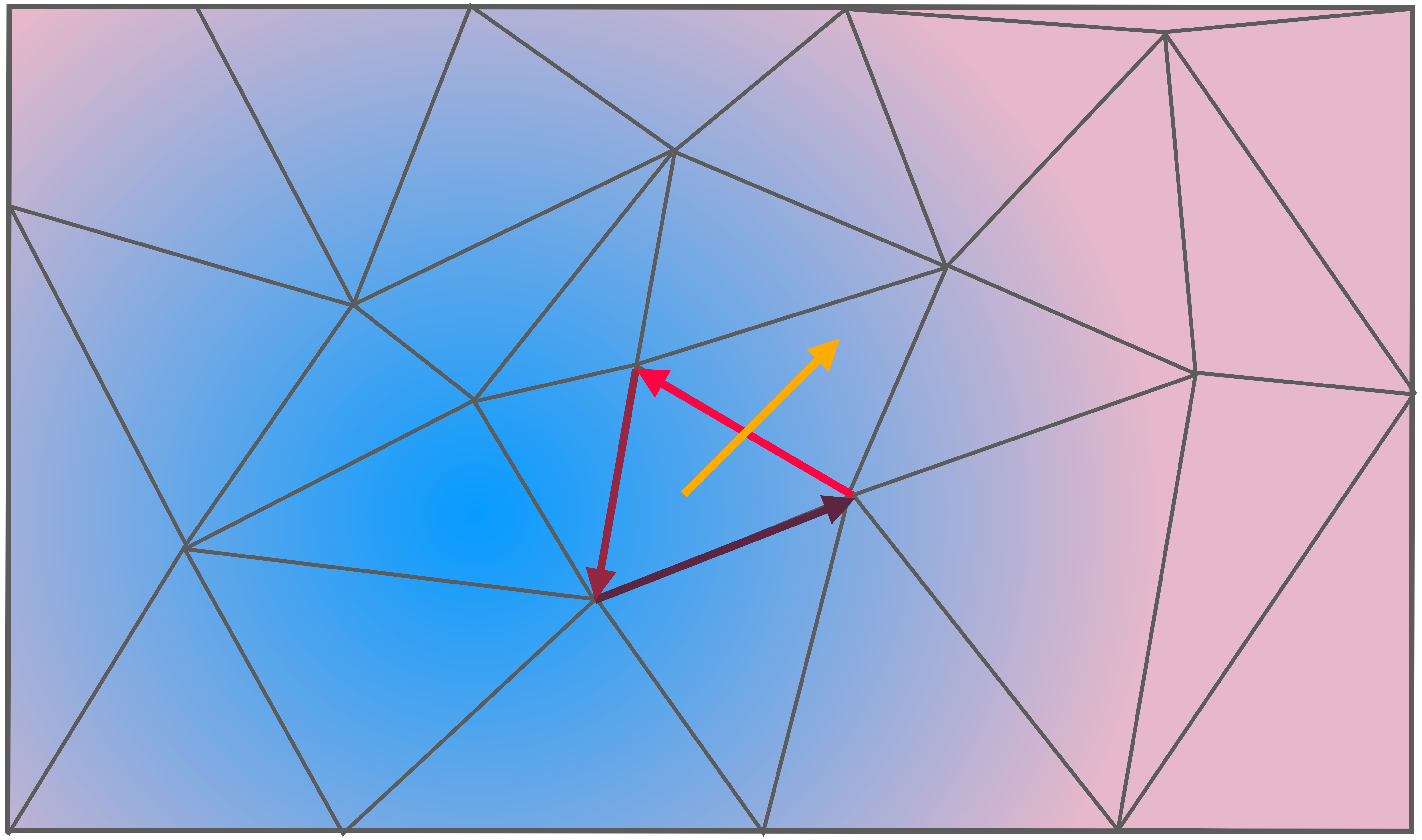


$$f(p) = f_i\phi_i + f_j\phi_j + f_k\phi_k$$

$$\nabla f(p) = f_i \nabla \phi_i + f_j \nabla \phi_j + f_k \nabla \phi_k$$

$$\nabla \phi_i := \frac{1}{2a_{t_{ijk}}} \left(\vec{n}_{ijk} \times \vec{e}_{jk}\right)$$

Discrete setting: triangular surfaces



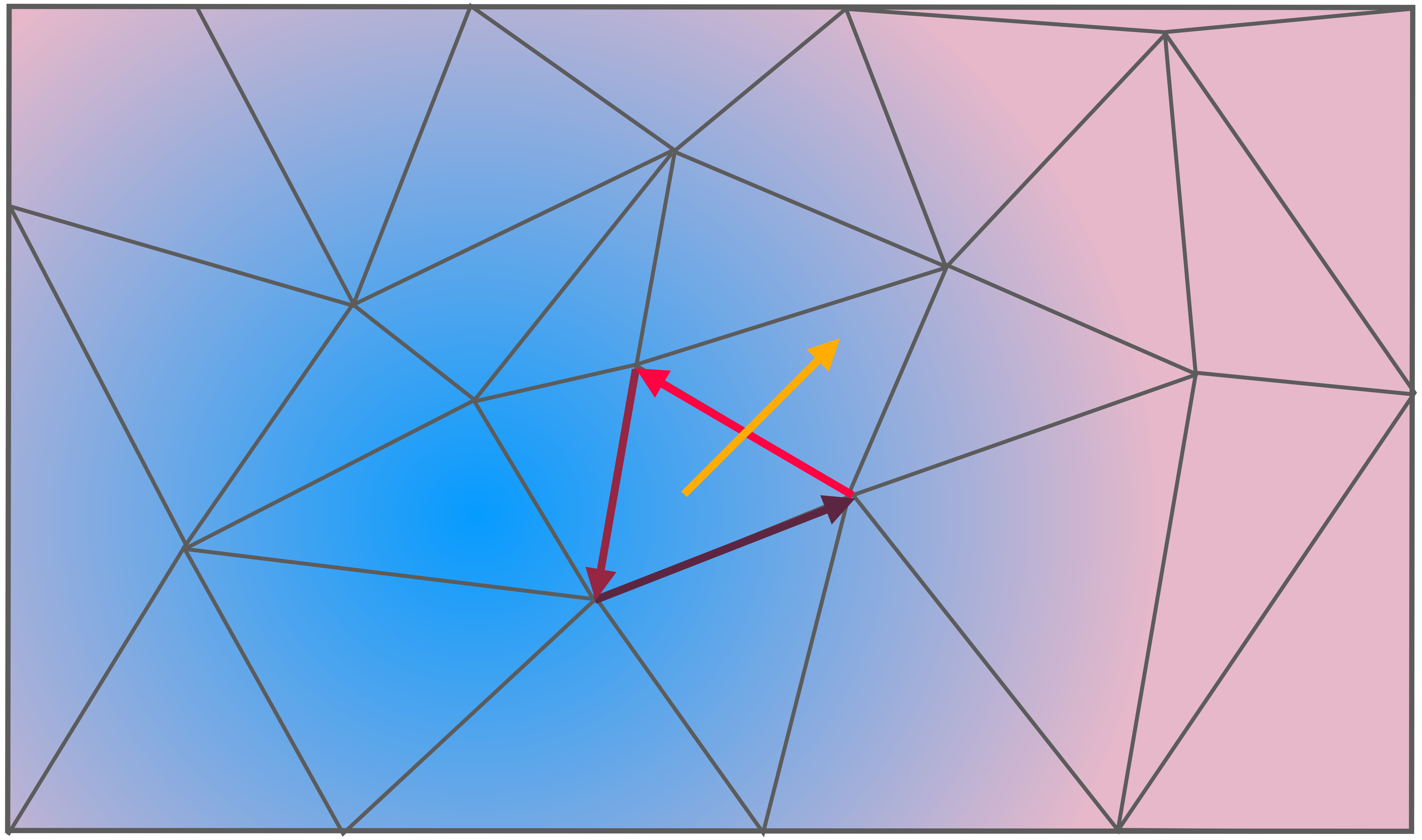
$$f(p) = f_i\phi_i + f_j\phi_j + f_k\phi_k$$

$$\nabla f(p) = f_i \nabla \phi_i + f_j \nabla \phi_j + f_k \nabla \phi_k$$

$$\nabla \phi_i := \frac{1}{2a_{t_{ijk}}} \left(\overrightarrow{n}_{ijk} \times \overrightarrow{e}_{jk} \right)$$

$$\mathrm{div}(U)_i = -\sum_{t_{ijk} \in v_i} \overrightarrow{u}_{ijk} \cdot \left(\overrightarrow{n}_{ijk} \times \overrightarrow{e}_{jk} \right)$$

Discrete setting: triangular surfaces



$$f(p) = f_i\phi_i + f_j\phi_j + f_k\phi_k$$

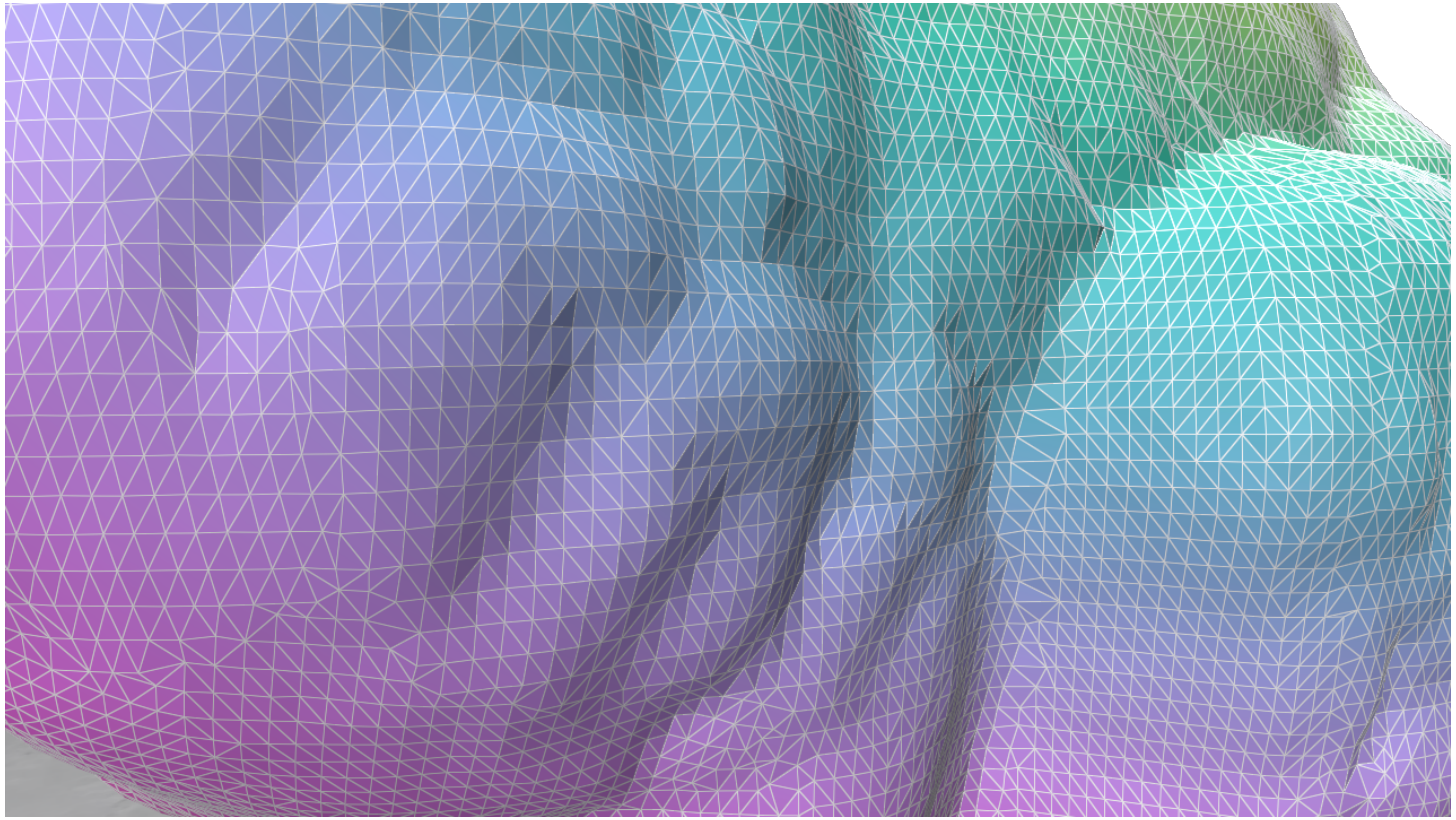
$$\nabla f(p) = f_i \nabla \phi_i + f_j \nabla \phi_j + f_k \nabla \phi_k$$

$$\nabla \phi_i := \frac{1}{2a_{t_{ijk}}} \left(\vec{n}_{ijk} \times \vec{e}_{jk}\right)$$

$$\mathrm{div}(U)_i = -\sum_{t_{ijk}\in v_i} \vec{u}_{ijk} \cdot (\vec{n}_{ijk} \times \vec{e}_{jk})$$

$$\mathrm{curl}(U)_i = \sum_{t_{ijk}\in v_i} \vec{u}_{ijk} \cdot \vec{e}_{jk}$$

Discrete setting: triangular surfaces



$$f(p) = f_i\phi_i + f_j\phi_j + f_k\phi_k$$

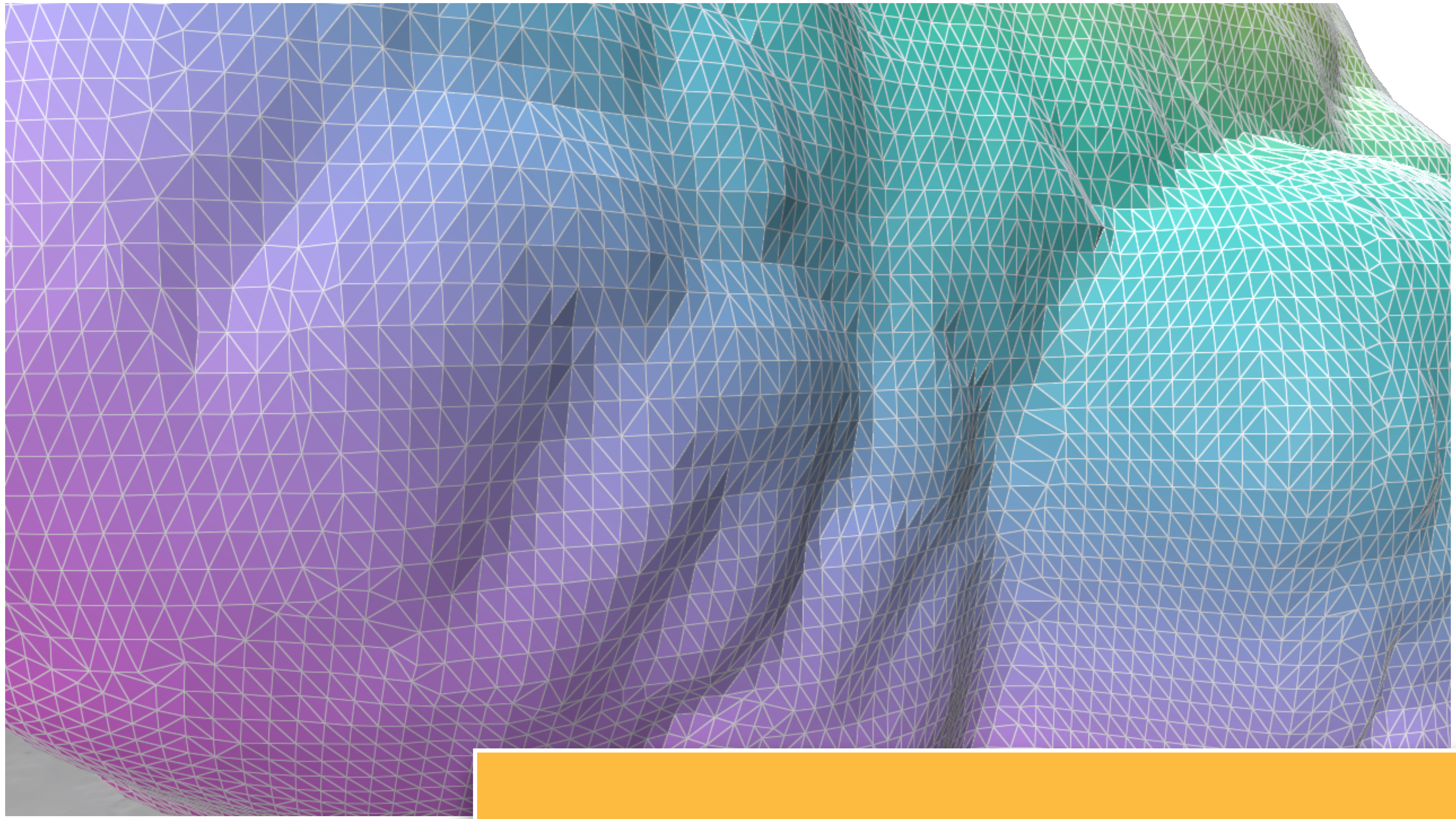
$$\nabla f(p) = f_i \nabla \phi_i + f_j \nabla \phi_j + f_k \nabla \phi_k$$

$$\nabla \phi_i := \frac{1}{2a_{t_{ijk}}} \left(\overrightarrow{n}_{ijk} \times \overrightarrow{e}_{jk}\right)$$

$$\text{div}(U)_i = -\sum_{t_{ijk} \in v_i} \overrightarrow{u}_{ijk} \cdot (\overrightarrow{n}_{ijk} \times \overrightarrow{e}_{jk})$$

$$\text{curl}(U)_i = \sum_{t_{ijk} \in v_i} \overrightarrow{u}_{ijk} \cdot \overrightarrow{e}_{jk}$$

Discrete setting: triangular surfaces



$$f(p) = f_i\phi_i + f_j\phi_j + f_k\phi_k$$

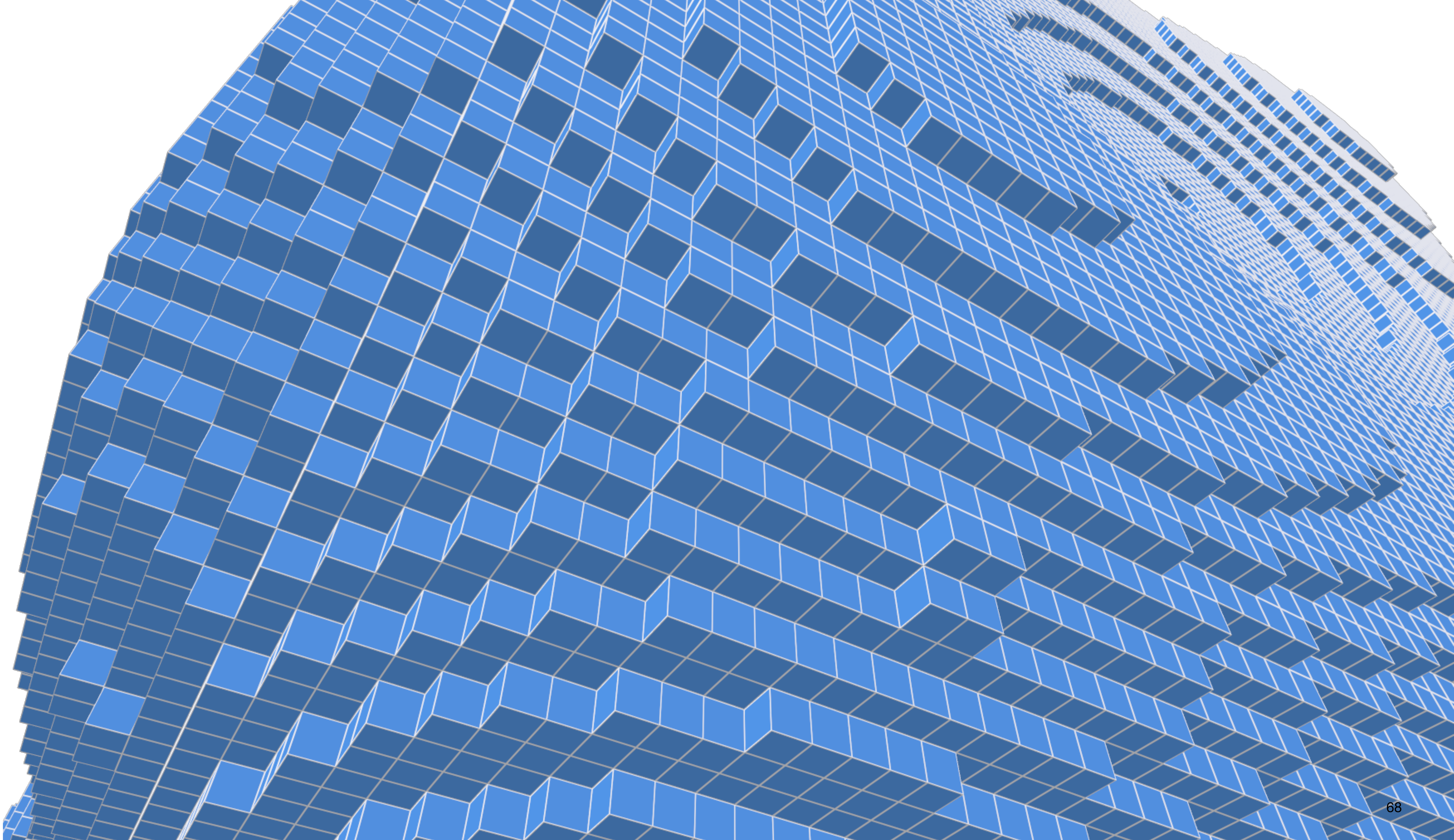
$$\nabla f(p) = f_i \nabla \phi_i + f_j \nabla \phi_j + f_k \nabla \phi_k$$

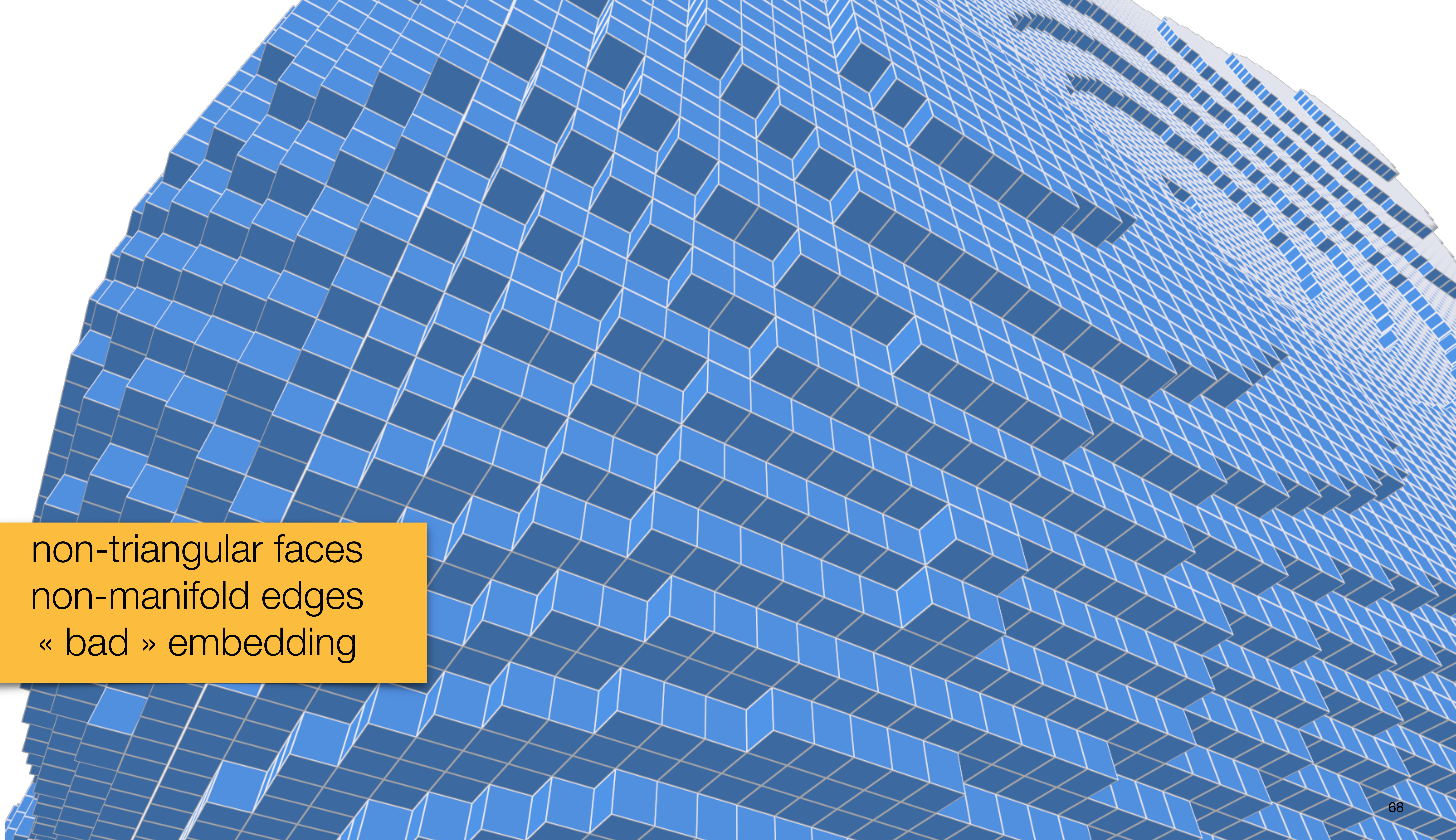
$$\nabla \phi_i := \frac{1}{2a_{t_{ijk}}} \left(\vec{n}_{ijk} \times \vec{e}_{jk}\right)$$

$$\mathrm{div}(U)_i = - \sum_{t_{ijk} \in v_i} \vec{u}_{ijk} \cdot (\vec{n}_{ijk} \times \vec{e}_{jk})$$

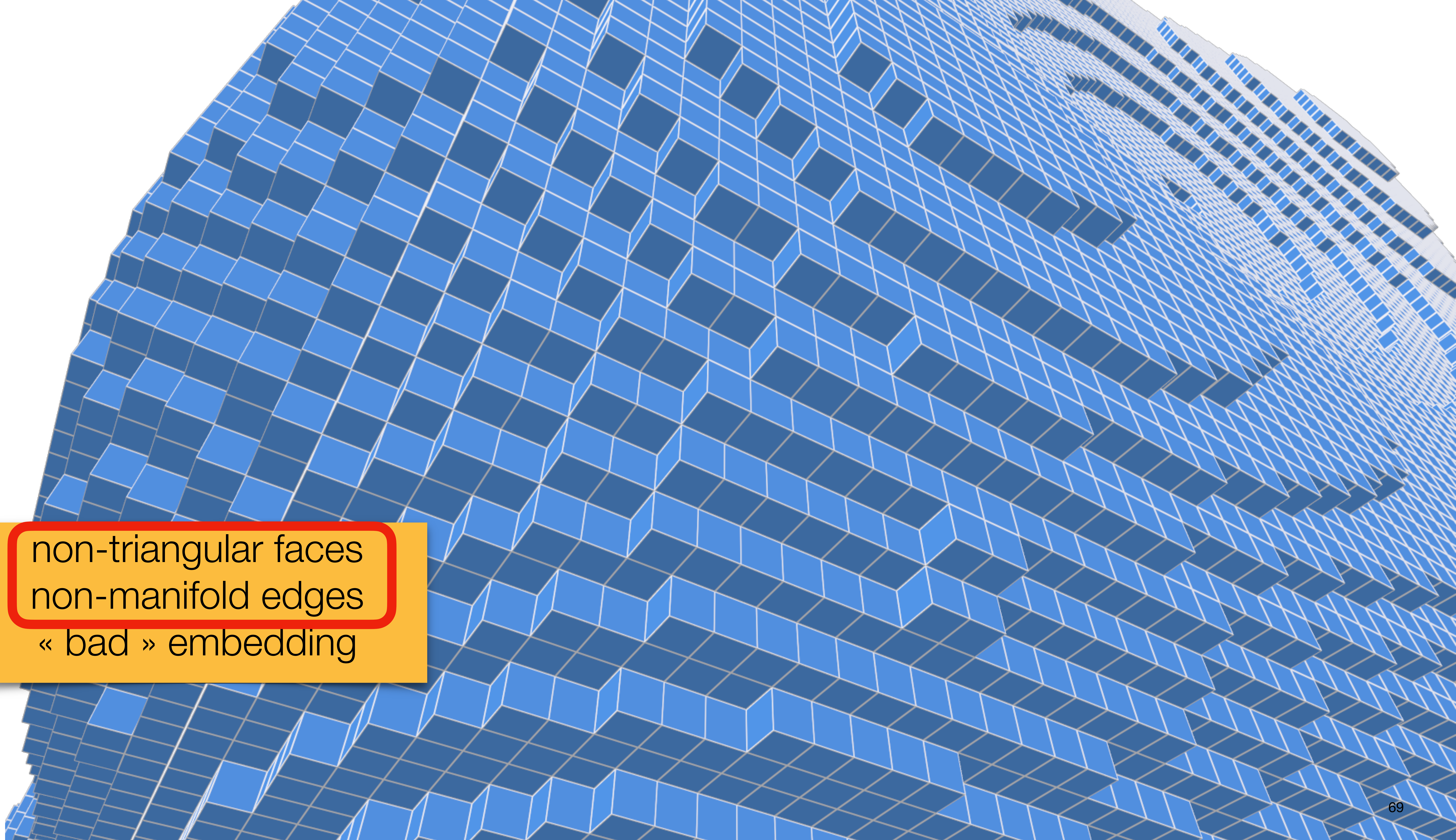
$$\sum_{\in v_i} \vec{u}_{ijk} \cdot \vec{e}_{jk}$$

Discrete exterior Calculus, FEM, VEM, FVM…

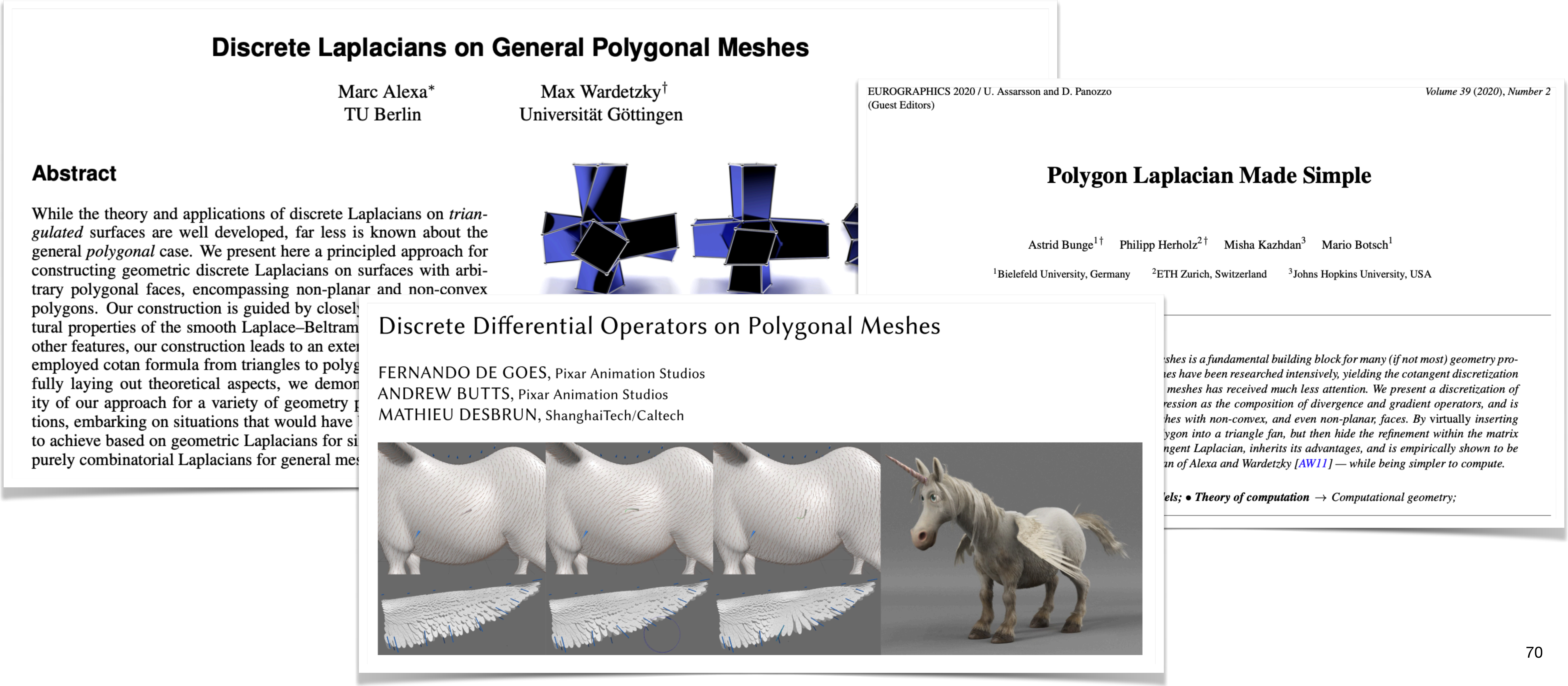




non-triangular faces
non-manifold edges
« bad » embedding


non-triangular faces
non-manifold edges
« bad » embedding

Calculus on polygonal meshes



Discrete Laplacians on General Polygonal Meshes

Marc Alexa* TU Berlin
Max Wardetzky† Universität Göttingen

Abstract

While the theory and applications of discrete Laplacians on *triangulated* surfaces are well developed, far less is known about the general *polygonal* case. We present here a principled approach for constructing geometric discrete Laplacians on surfaces with arbitrary polygonal faces, encompassing non-planar and non-convex polygons. Our construction is guided by closely
tural properties of the smooth Laplace–Beltram
other features, our construction leads to an exter
employed cotan formula from triangles to polyg
fully laying out theoretical aspects, we demon
ity of our approach for a variety of geometry p
tions, embarking on situations that would have
to achieve based on geometric Laplacians for si
purely combinatorial Laplacians for general mes

EUROGRAPHICS 2020 / U. Assarsson and D. Panozzo (Guest Editors)
Volume 39 (2020), Number 2

Polygon Laplacian Made Simple

Astrid Bunge[1]† Philipp Herholz[2]† Misha Kazhdan[3] Mario Botsch[1]

[1]Bielefeld University, Germany [2]ETH Zurich, Switzerland [3]Johns Hopkins University, USA

shes is a fundamental building block for many (if not most) geometry pro-
es have been researched intensively, yielding the cotangent discretization
meshes has received much less attention. We present a discretization of
ression as the composition of divergence and gradient operators, and is
hes with non-convex, and even non-planar, faces. By virtually inserting
ygon into a triangle fan, but then hide the refinement within the matrix
ngent Laplacian, inherits its advantages, and is empirically shown to be
an of Alexa and Wardetzky [AW11] — while being simpler to compute.

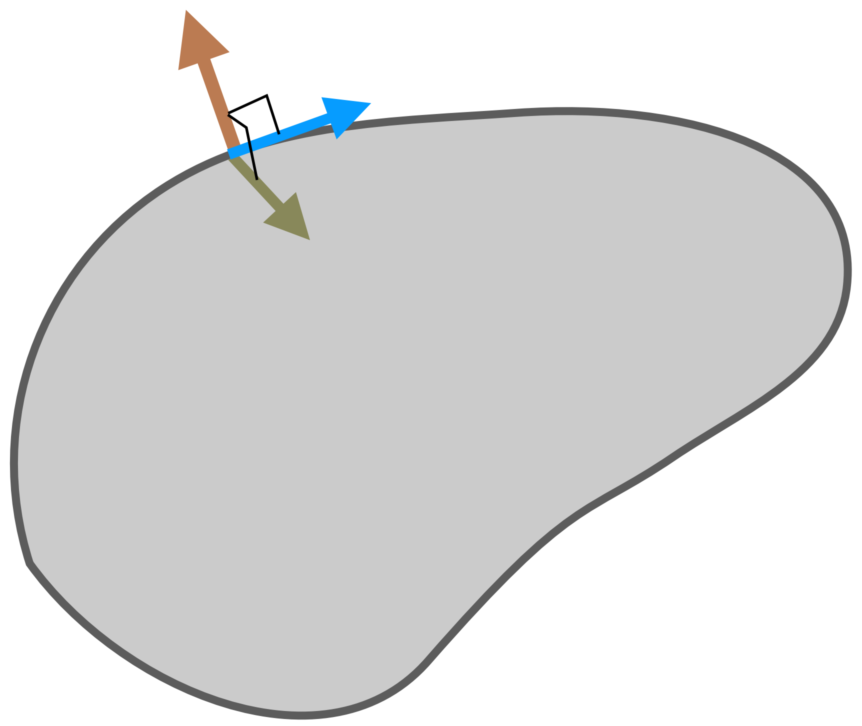
els; • Theory of computation → Computational geometry;

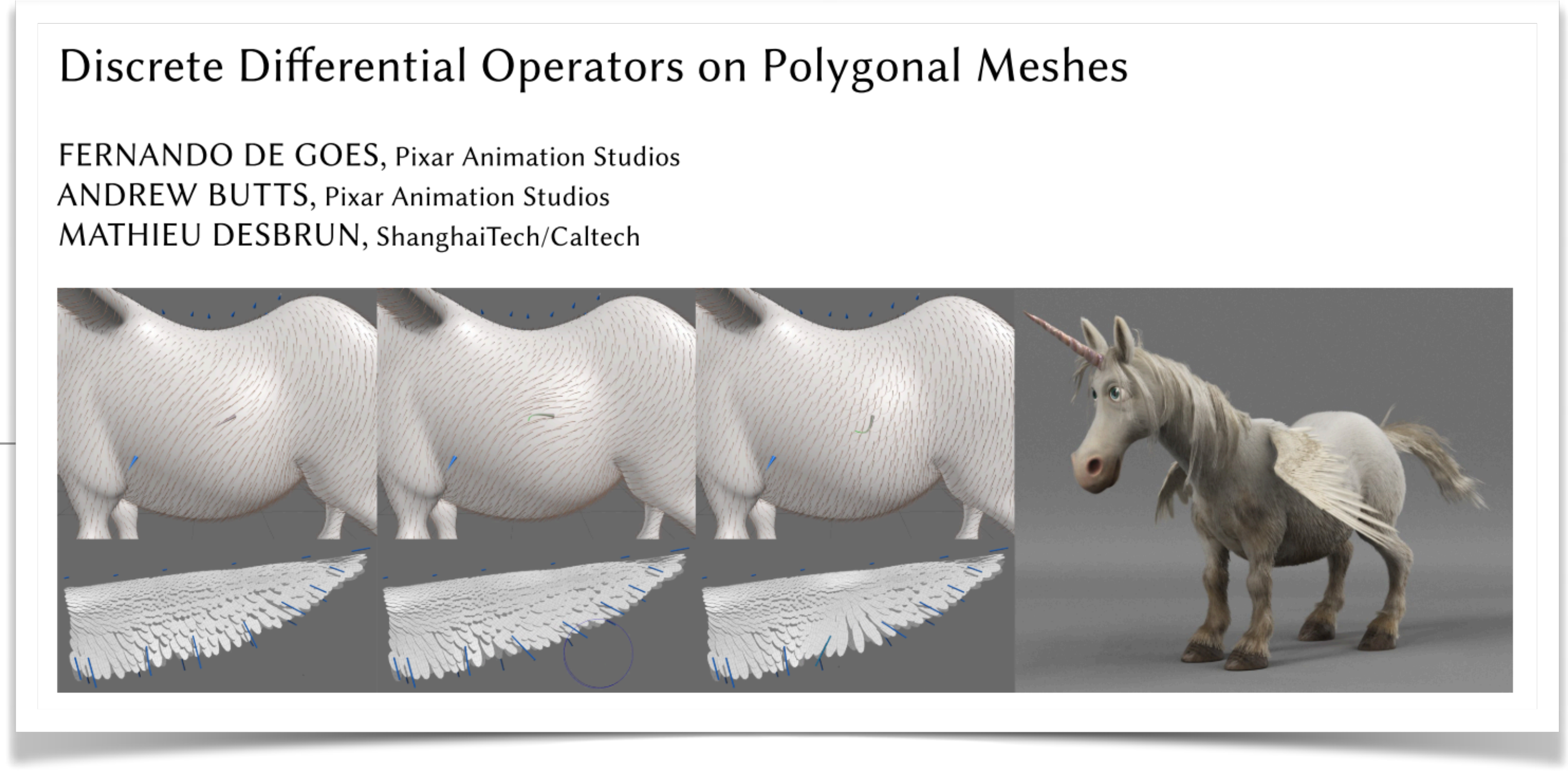
Discrete Differential Operators on Polygonal Meshes

FERNANDO DE GOES, Pixar Animation Studios
ANDREW BUTTS, Pixar Animation Studios
MATHIEU DESBRUN, ShanghaiTech/Caltech

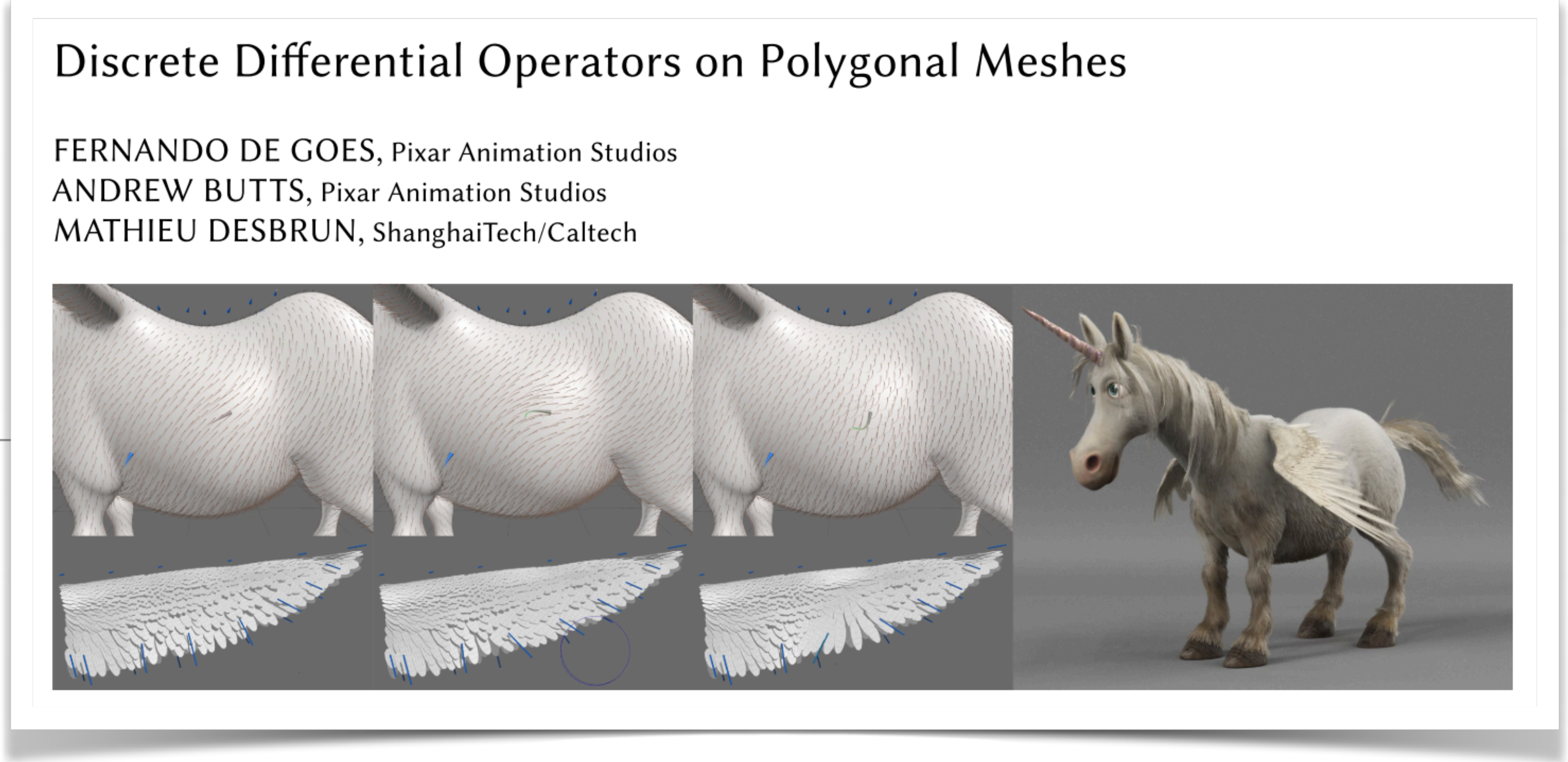
Step 1 Gradient

$$\nabla^{\perp}\phi(x) = (n(x) \times \nabla\phi(x)) = [n(x)]_{\times}\nabla\phi(x)$$



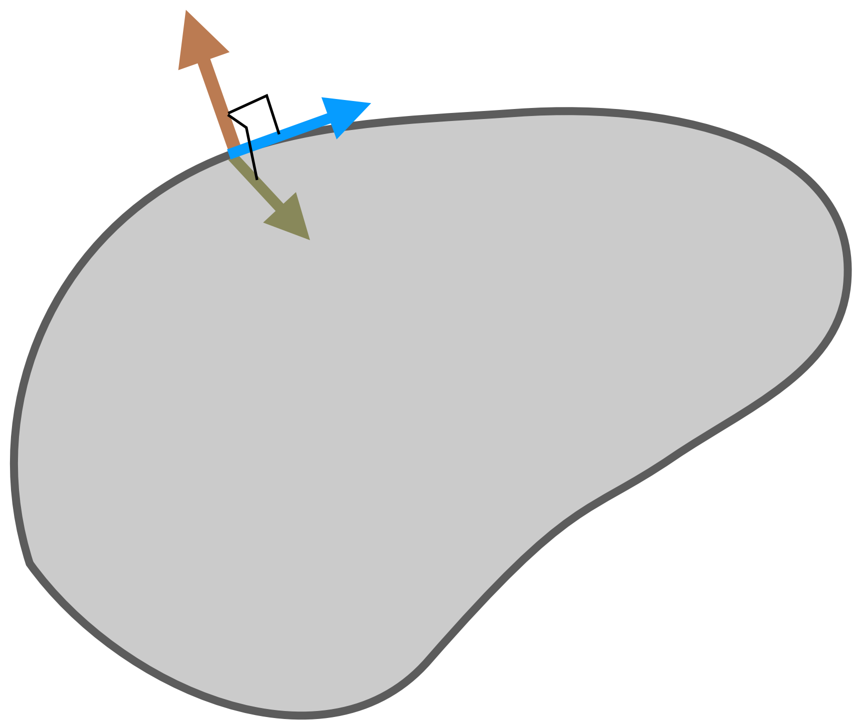


Step 1 Gradient


Discrete Differential Operators on Polygonal Meshes

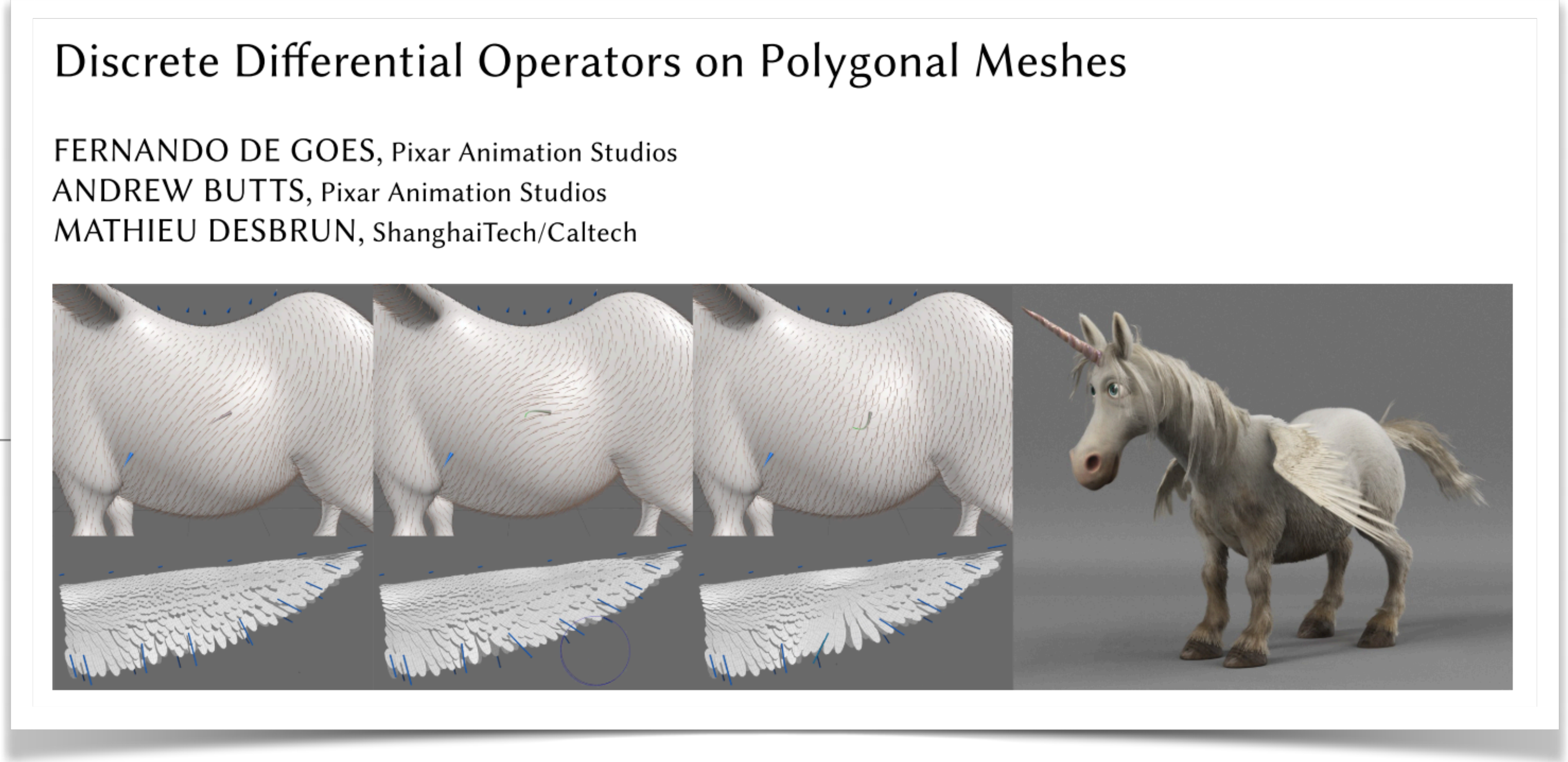
FERNANDO DE GOES, Pixar Animation Studios
ANDREW BUTTS, Pixar Animation Studios
MATHIEU DESBRUN, ShanghaiTech/Caltech

$$\nabla^{\perp}\phi(x) = (n(x) \times \nabla\phi(x)) = [n(x)]_{\times}\nabla\phi(x)$$



$$\int_f \nabla\phi(x)dx = \oint_{\partial f} \phi(x)(t(x) \times n(x))dx$$

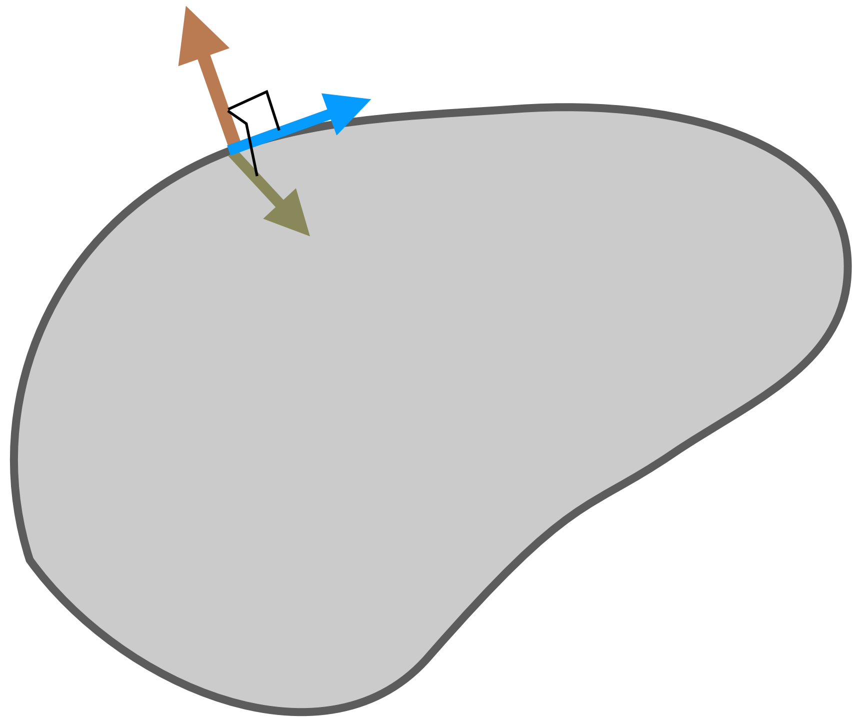
(Stokes' theorem)


Discrete Differential Operators on Polygonal Meshes

FERNANDO DE GOES, Pixar Animation Studios
ANDREW BUTTS, Pixar Animation Studios
MATHIEU DESBRUN, ShanghaiTech/Caltech

Step 1 Gradient

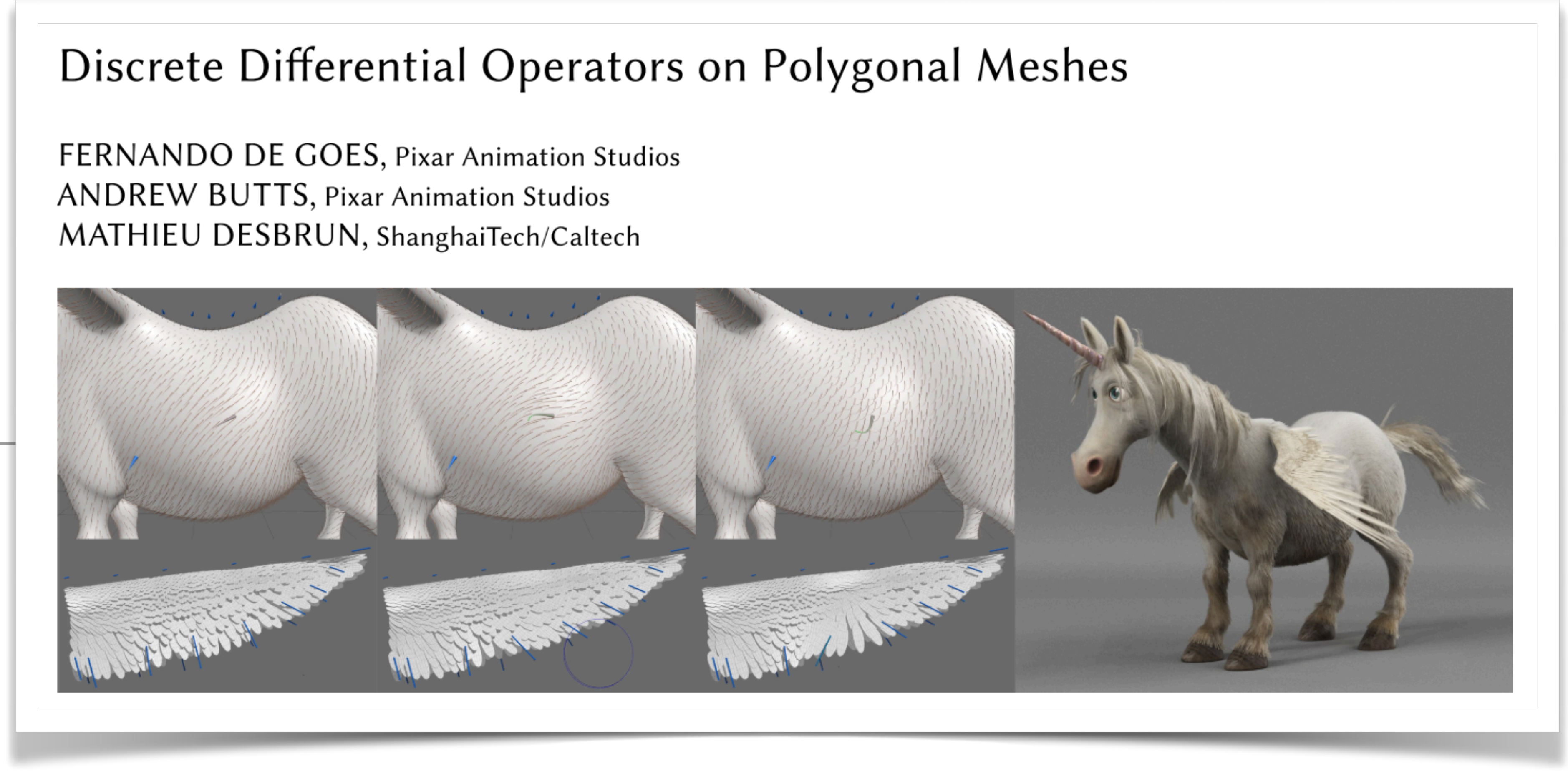
$$\nabla^{\perp}\phi(x) = (n(x) \times \nabla\phi(x)) = [n(x)]_{\times}\nabla\phi(x)$$



$$\int_f \nabla\phi(x)dx = \oint_{\partial f} \phi(x)(t(x) \times n(x))dx$$ (Stokes' theorem)

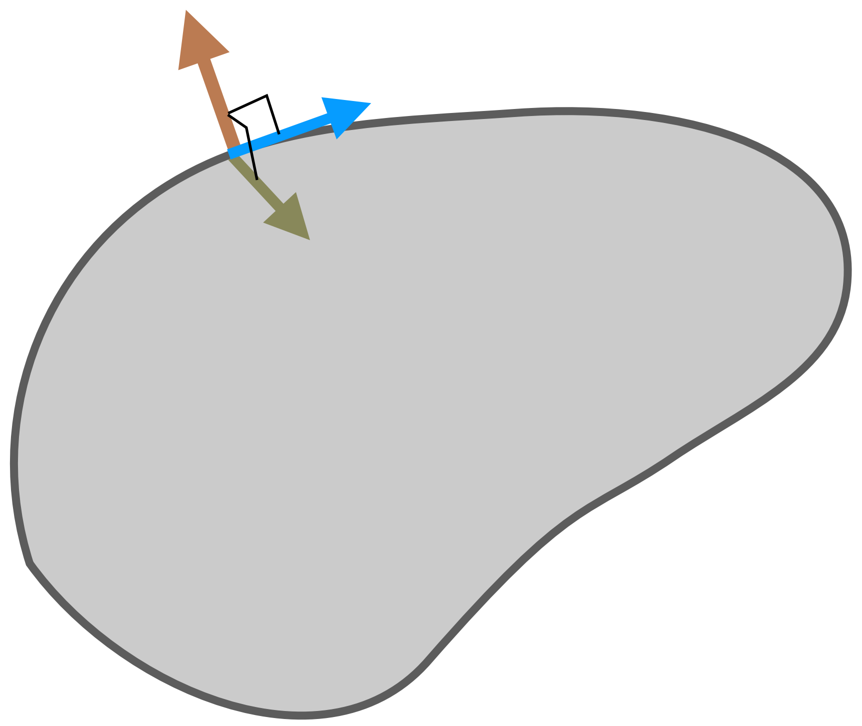
$$\int_f \nabla^{\perp}\phi(x)dx = \oint_{\partial f} \phi(x)t(x)dx$$

Step 1 Gradient


Discrete Differential Operators on Polygonal Meshes

FERNANDO DE GOES, Pixar Animation Studios
ANDREW BUTTS, Pixar Animation Studios
MATHIEU DESBRUN, ShanghaiTech/Caltech

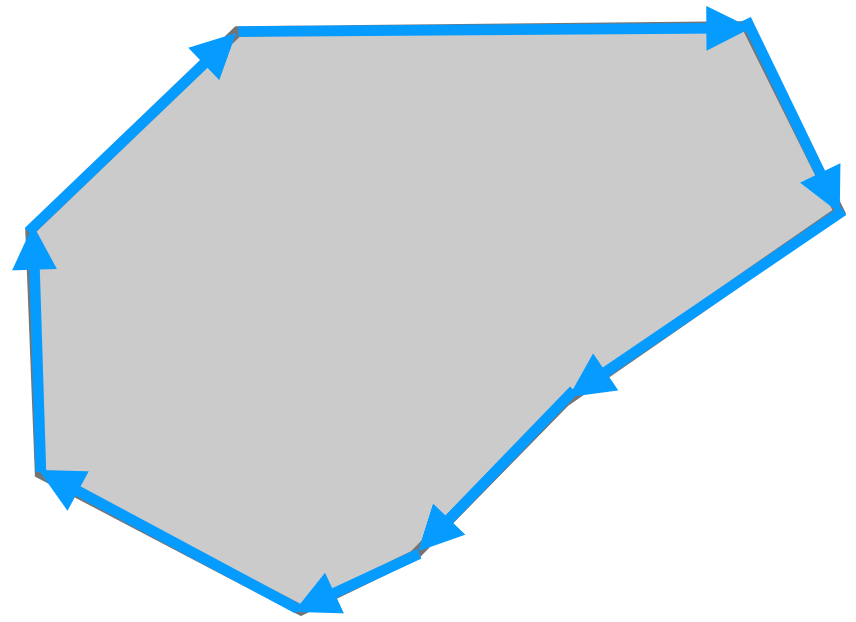
$$\nabla^{\perp}\phi(x) = (n(x) \times \nabla\phi(x)) = [n(x)]_{\times}\nabla\phi(x)$$



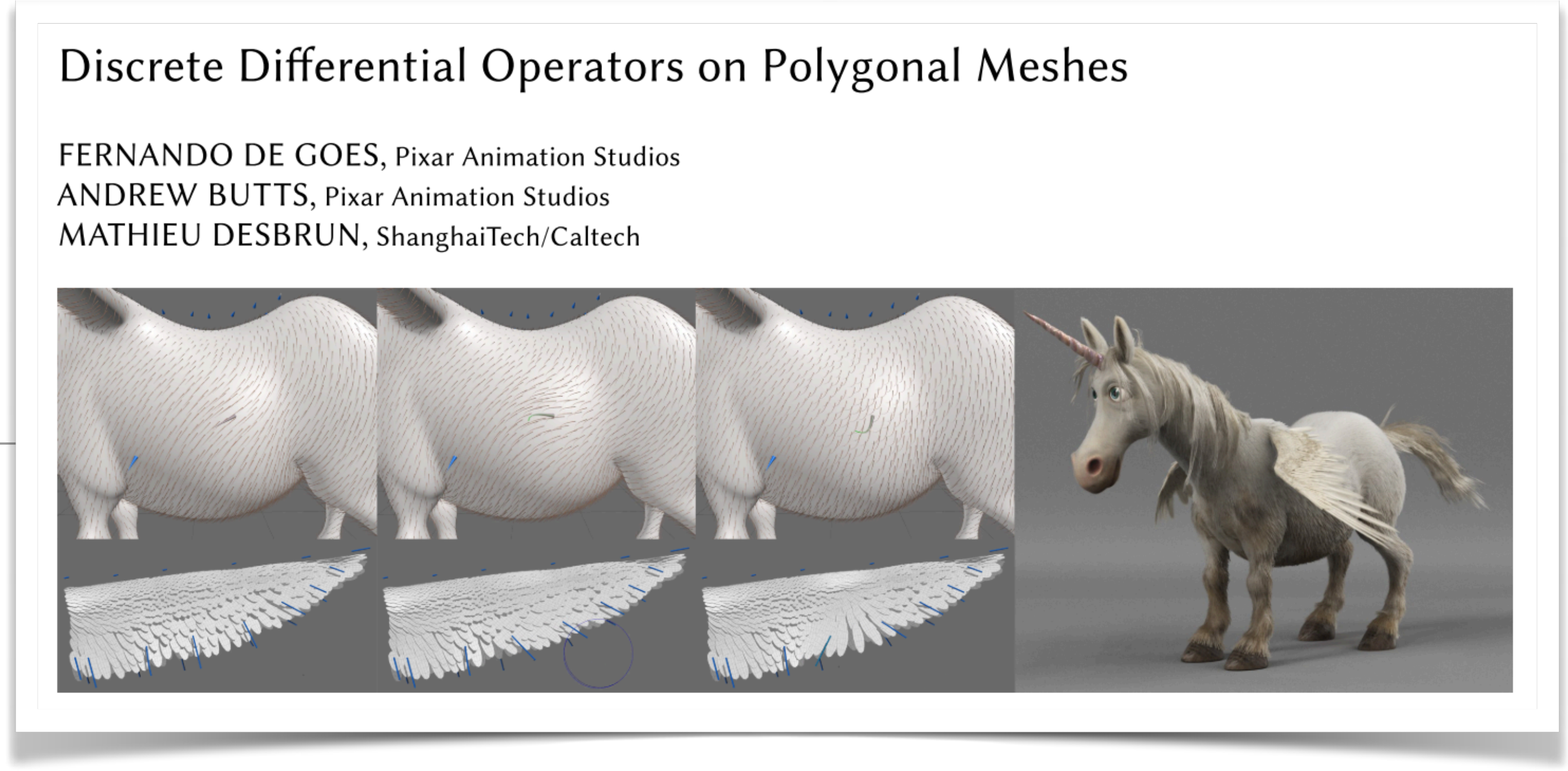
$$\int_f \nabla\phi(x)dx = \oint_{\partial f} \phi(x)(t(x) \times n(x))dx$$

(Stokes' theorem)

$$\int_f \nabla^{\perp}\phi(x)dx = \oint_{\partial f} \phi(x)t(x)dx$$

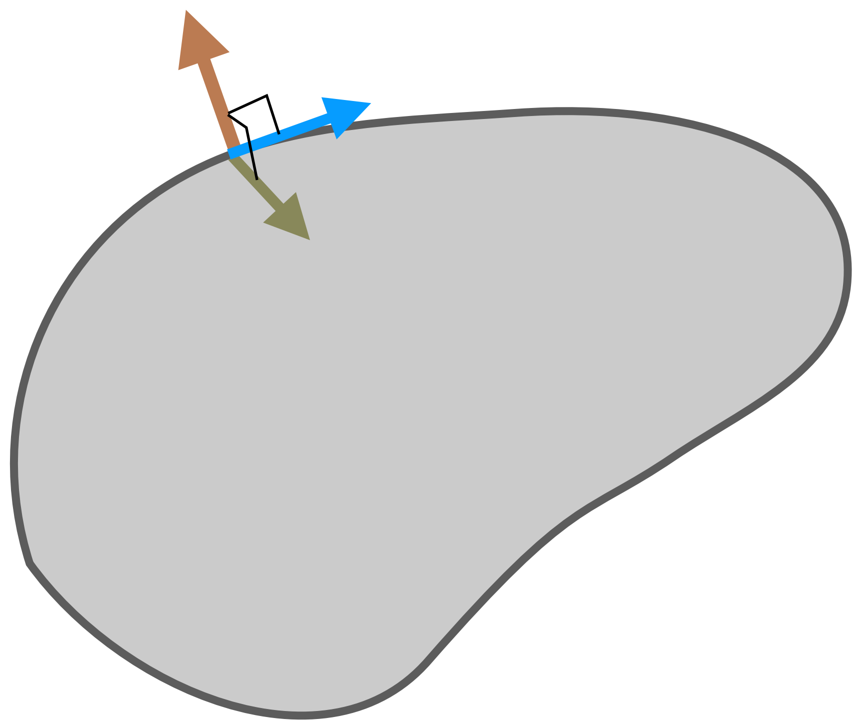


Step 1 Gradient


Discrete Differential Operators on Polygonal Meshes

FERNANDO DE GOES, Pixar Animation Studios
ANDREW BUTTS, Pixar Animation Studios
MATHIEU DESBRUN, ShanghaiTech/Caltech

$$\nabla^{\perp}\phi(x) = (n(x) \times \nabla\phi(x)) = [n(x)]_{\times}\nabla\phi(x)$$

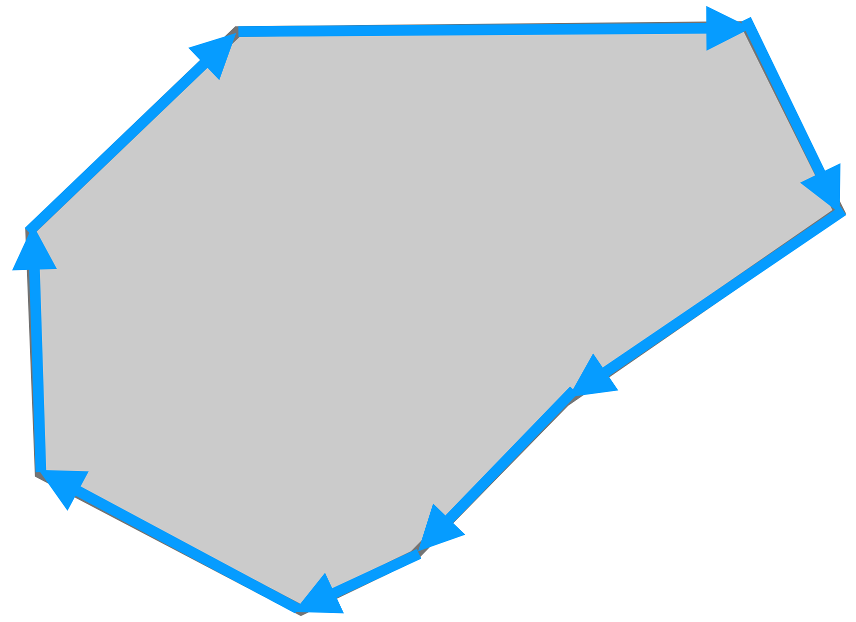


$$\int_f \nabla\phi(x)dx = \oint_{\partial f} \phi(x)(t(x) \times n(x))dx$$

(Stokes' theorem)

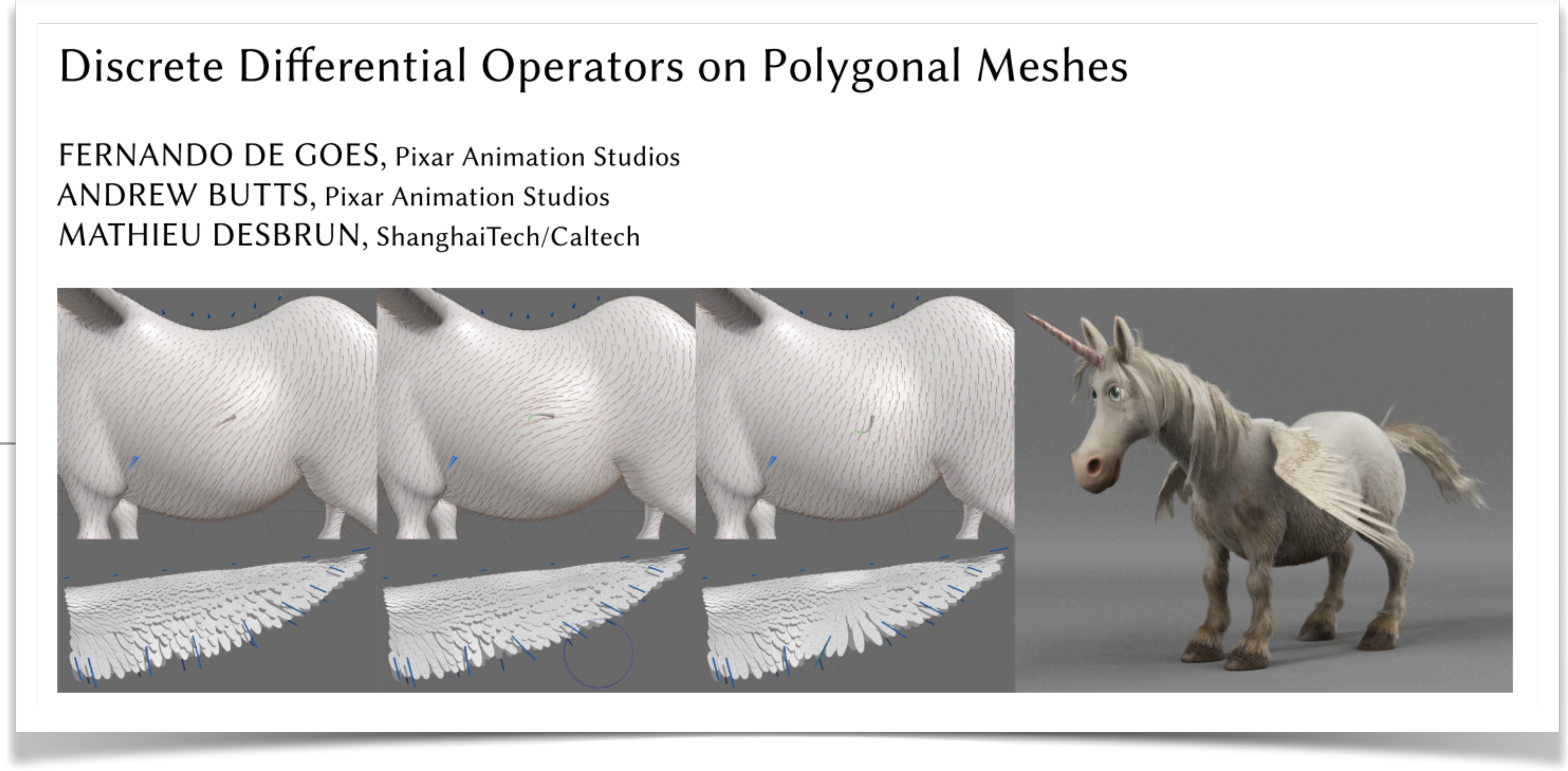
$$\int_f \nabla^{\perp}\phi(x)dx = \oint_{\partial f} \phi(x)t(x)dx$$

$$\sum_{edges}$$

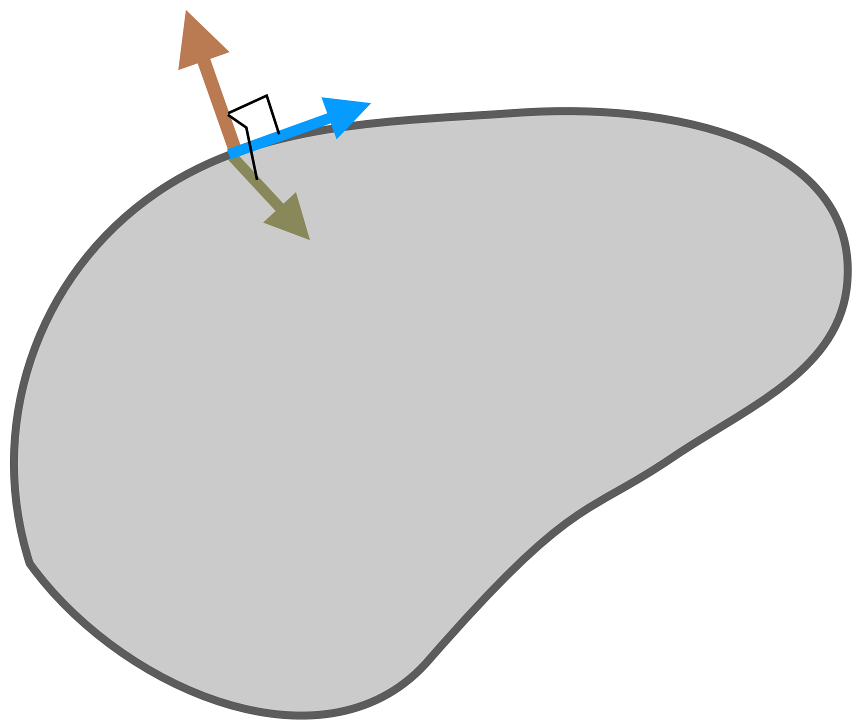


Step 1 Gradient

$$\nabla^{\perp}\phi(x) = (n(x) \times \nabla\phi(x)) = [n(x)]_{\times}\nabla\phi(x)$$


Discrete Differential Operators on Polygonal Meshes

FERNANDO DE GOES, Pixar Animation Studios
ANDREW BUTTS, Pixar Animation Studios
MATHIEU DESBRUN, ShanghaiTech/Caltech

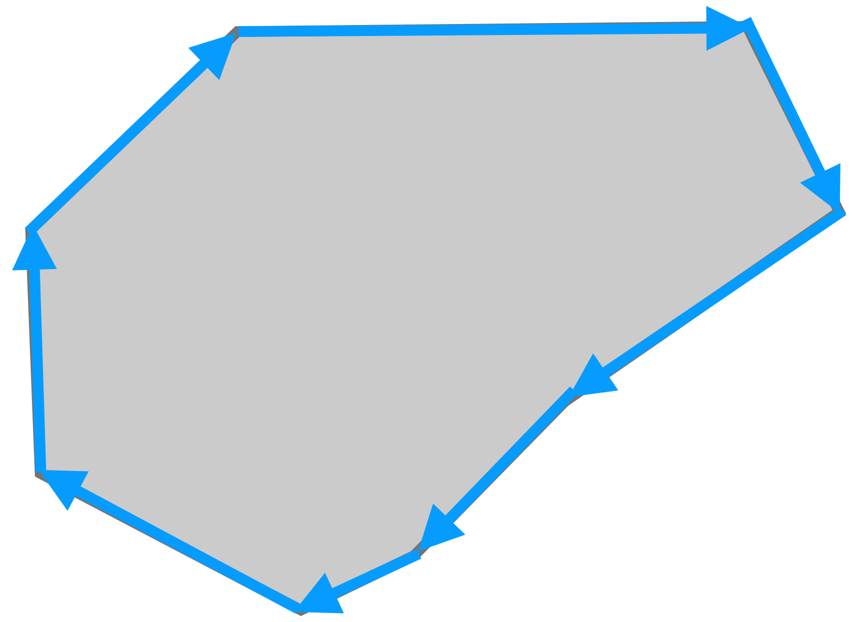


$$\int_f \nabla\phi(x)dx = \oint_{\partial f} \phi(x)(t(x) \times n(x))dx$$ (Stokes' theorem)

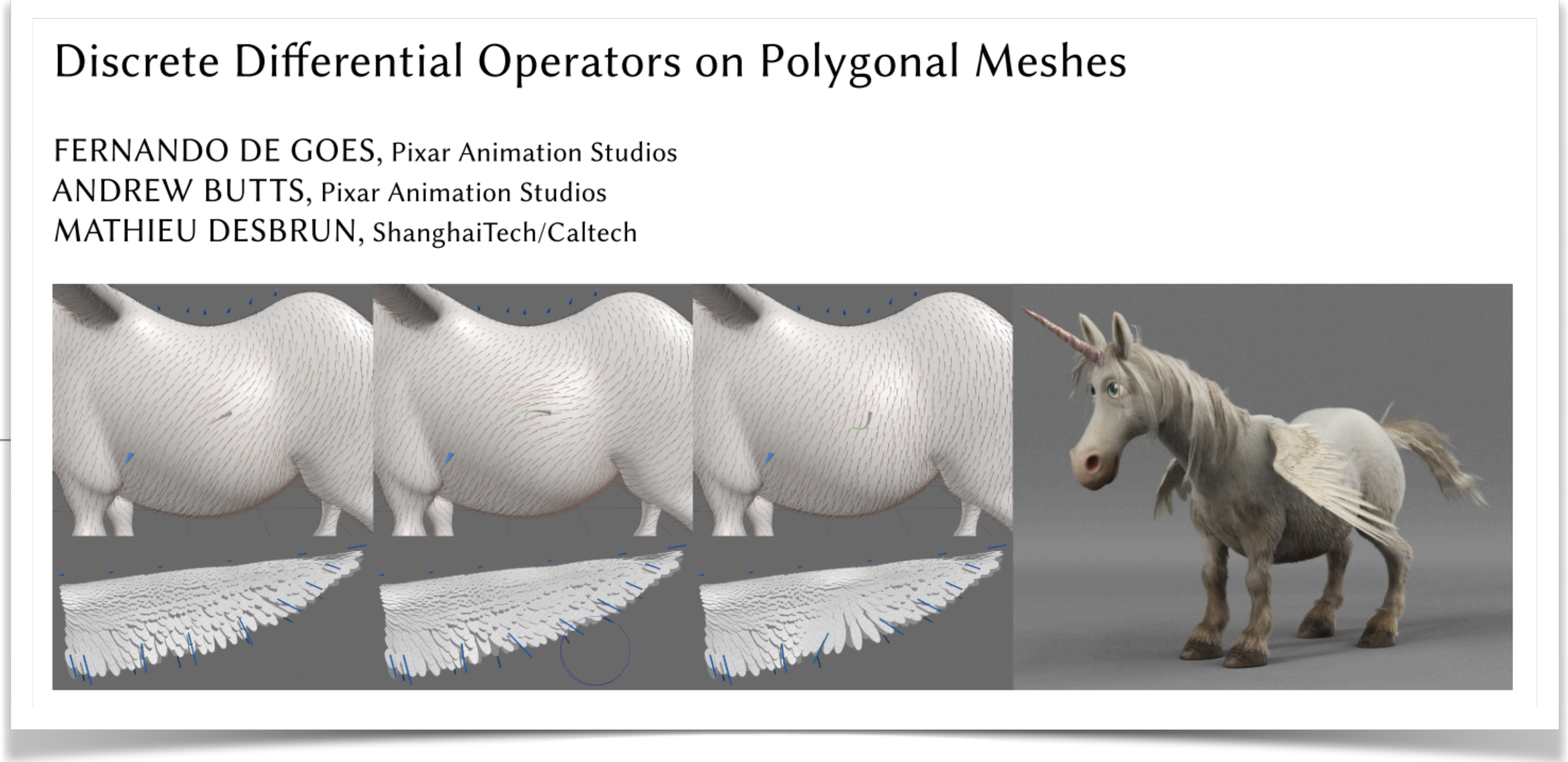
$$\int_f \nabla^{\perp}\phi(x)dx = \oint_{\partial f} \phi(x)t(x)dx$$

$\oint_{\partial f} \to \sum_{edges}$

$t(x)dx \to (x_{i+1} - x_i)$

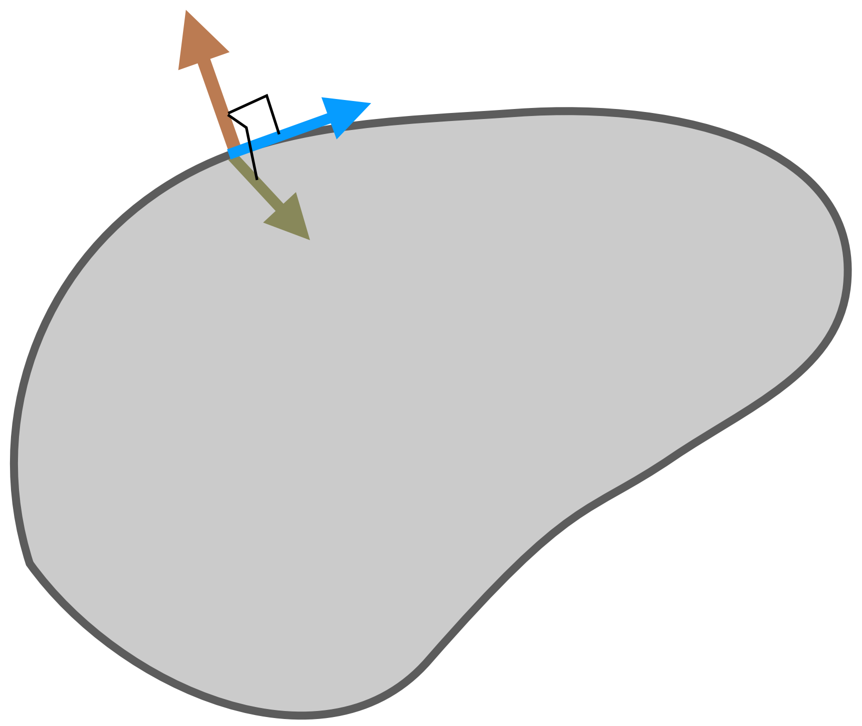


Step 1 Gradient


Discrete Differential Operators on Polygonal Meshes

FERNANDO DE GOES, Pixar Animation Studios
ANDREW BUTTS, Pixar Animation Studios
MATHIEU DESBRUN, ShanghaiTech/Caltech

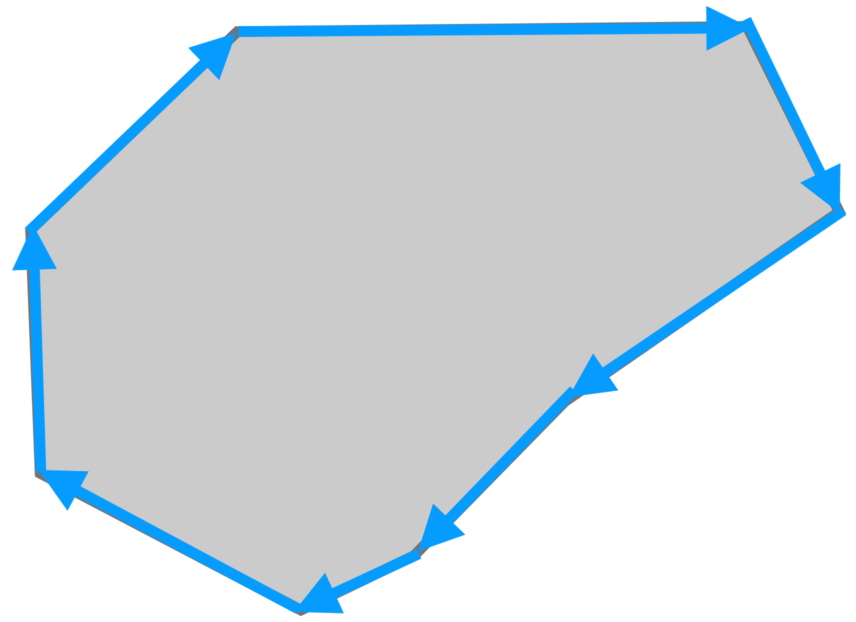
$$\nabla^{\perp}\phi(x) = (n(x) \times \nabla\phi(x)) = [n(x)]_{\times}\nabla\phi(x)$$



$$\int_f \nabla\phi(x)dx = \oint_{\partial f} \phi(x)(t(x) \times n(x))dx$$ (Stokes' theorem)

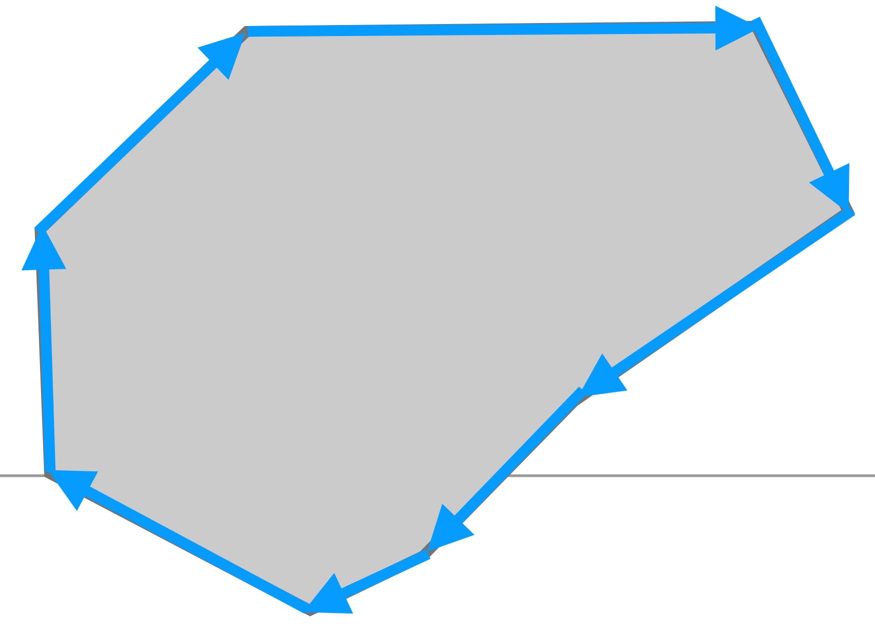
$$\int_f \nabla^{\perp}\phi(x)dx = \oint_{\partial f} \phi(x)t(x)dx$$

$\sum_{edges}$ $\qquad \phi\left(\frac{x_{i+1} + x_i}{2}\right)$ $\qquad (x_{i+1} - x_i)$

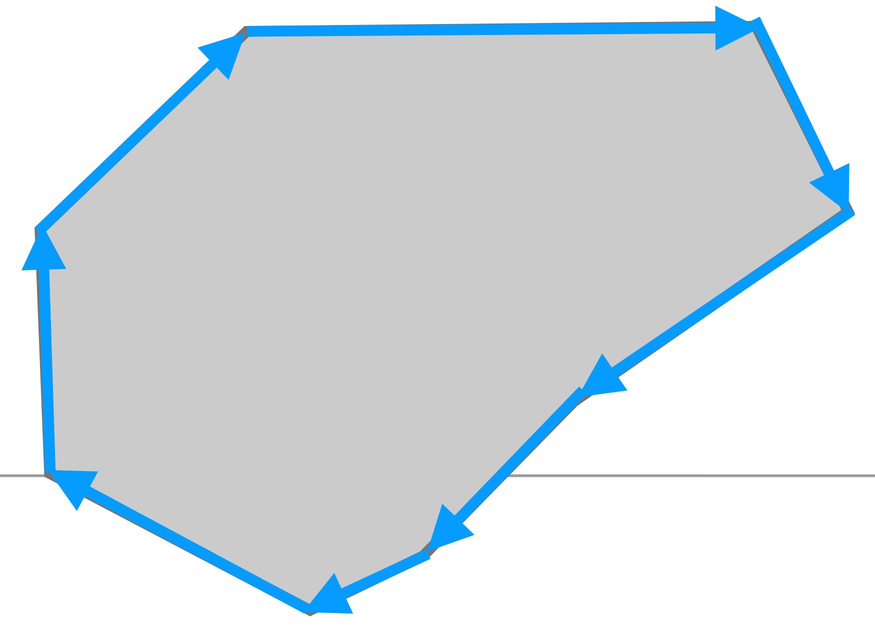


For $\phi_f = sX_f + 1_s r$, we want $G_f\phi_s = s$

Matrix form of per face operators

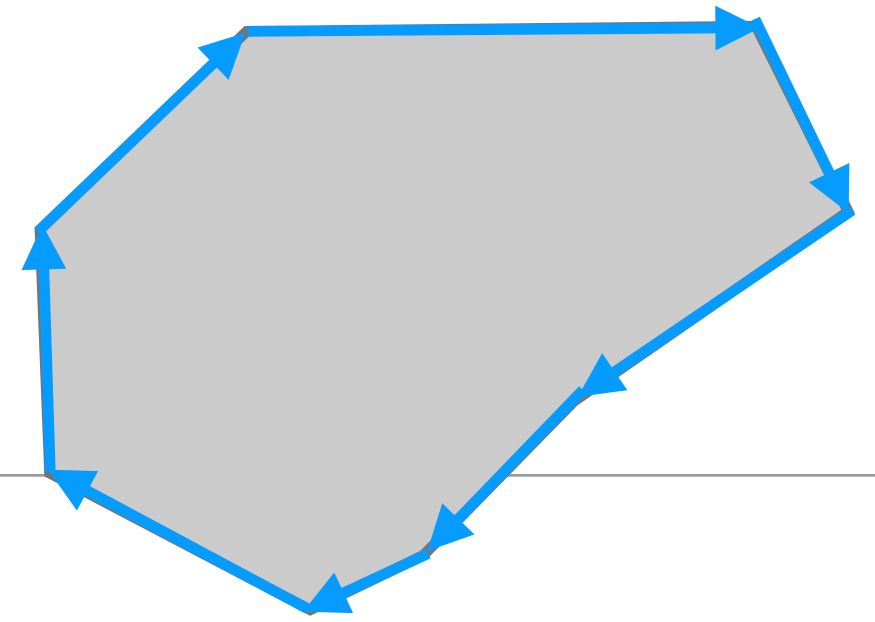


Matrix form of per face operators



$$\phi_f = [\phi(v_1)\ldots\phi(v_{n_f})]^t$$

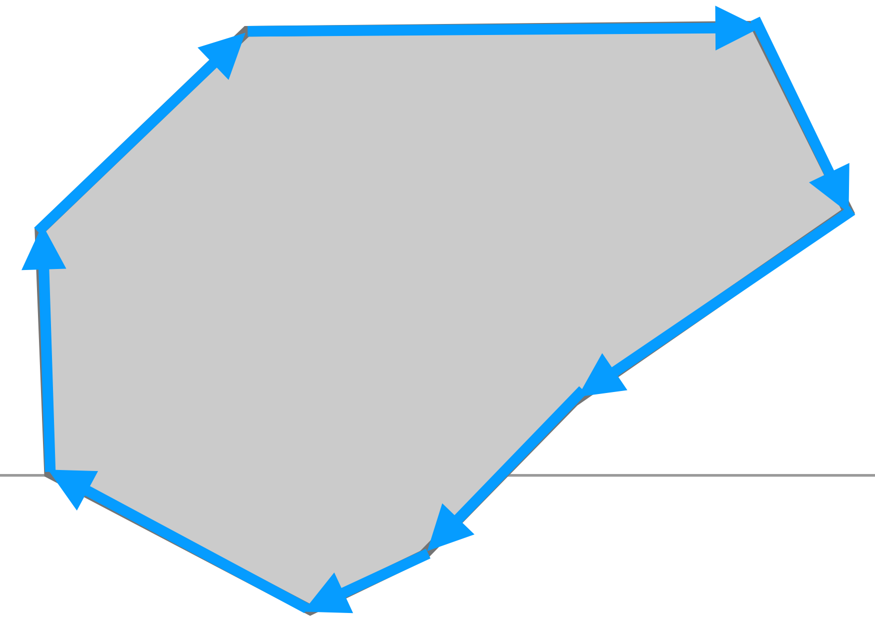
Matrix form of per face operators



$$\phi_f = [\phi(v_1)\ldots\phi(v_{n_f})]^t$$

$$\mathbf{G}_f^{\perp} := \mathbf{E}_f^t \mathbf{A}_f$$

Matrix form of per face operators

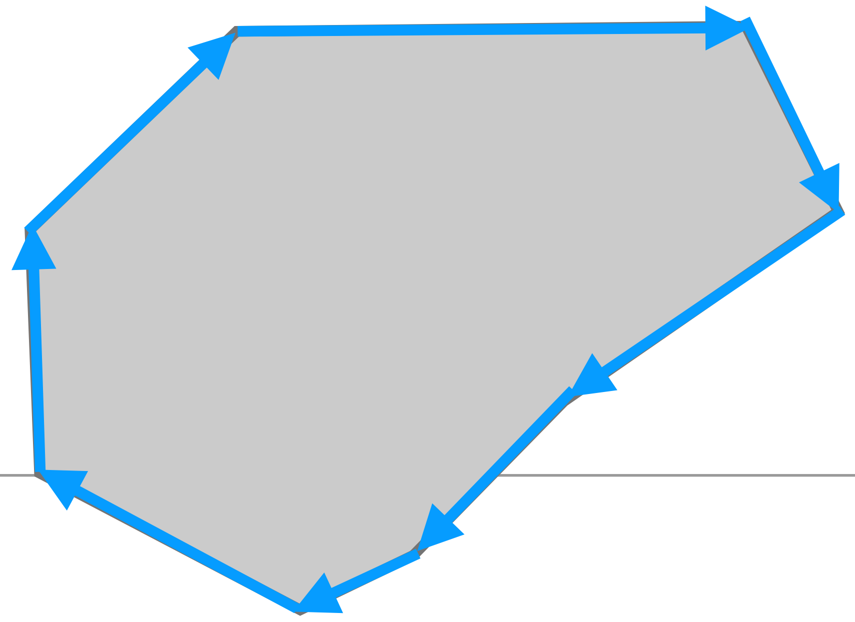


$$\phi_f = [\phi(v_1)\dots\phi(v_{n_f})]^t$$

$$\mathbf{G}_f^{\perp} := \mathbf{E}_f^t \mathbf{A}_f$$

Symbol	Meaning	Definition
n_f	Number of vertices	$v_1, \dots, v_{n_f} \in f$
$\mathbf{X}_f$	Vertex positions	$\mathbf{X}_f = [\mathbf{x}_{v_1} \dots \mathbf{x}_{v_{n_f}}]^t \in \mathbb{R}^{n_f \times 3}$
$\mathbf{D}_f$	Difference operator	$\mathbf{D}_f^{i,i+1} = 1,\ \mathbf{D}_f^{i,i} = -1$
$\mathbf{A}_f$	Average operator	$\mathbf{A}_f^{i,i+1} = \mathbf{A}_f^{i,i} = 1/2$
$\mathbf{E}_f$	Edge vectors	$\mathbf{E}_f = \mathbf{D}_f \mathbf{X}_f$
$\mathbf{B}_f$	Edge midpoints	$\mathbf{B}_f = \mathbf{A}_f \mathbf{X}_f$
$\mathbf{c}_f$	Face center	$\mathbf{c}_f = \mathbf{X}_f^t \mathbf{1}_f / n_f$
$\mathbf{a}_f$	Polygonal vector area	$\mathbf{a}_f = 1/2 \sum_{v_i \in f} \mathbf{x}_{v_i} \times \mathbf{x}_{v_{i+1}}$
a_f	Area of polygonal face	$a_f = \lvert \mathbf{a}_f \rvert$
$\mathbf{n}_f$	Normal of polygonal face	$\mathbf{n}_f = \mathbf{a}_f / a_f$
$\mathbf{h}_f$	Vertex heights for polygonal face	$\mathbf{h}_f = (\mathbf{X}_f - \mathbf{1}_f \mathbf{c}_f^t)\mathbf{n}_f$

Matrix form of per face operators



$$\phi_f = [\phi(v_1)\ldots\phi(v_{n_f})]^t$$

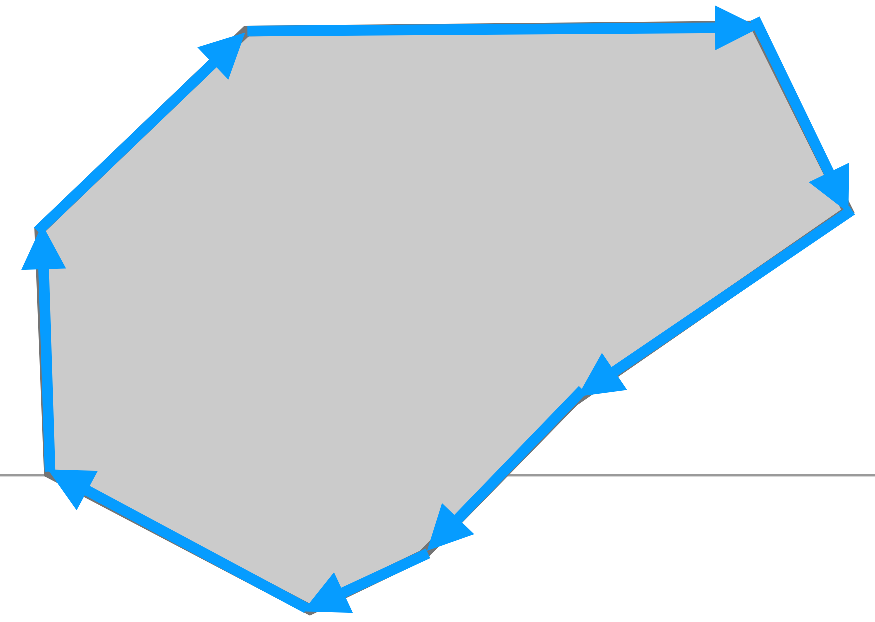
$$\mathbf{G}_f^{\perp} := \mathbf{E}_f^t \mathbf{A}_f$$

$$\mathbf{G}_f := -\frac{1}{a_f}[\mathbf{n}_f]\mathbf{E}_f^t \mathbf{A}_f$$

$3 \times n$ *matrix*

Symbol	Meaning	Definition
n_f	Number of vertices	$v_1, \ldots, v_{n_f} \in f$
$\mathbf{X}_f$	Vertex positions	$\mathbf{X}_f = [\mathbf{x}_{v_1} \ldots \mathbf{x}_{v_{n_f}}]^t \in \mathbb{R}^{n_f \times 3}$
$\mathbf{D}_f$	Difference operator	$\mathbf{D}_f^{i,i+1} = 1,\ \mathbf{D}_f^{i,i} = -1$
$\mathbf{A}_f$	Average operator	$\mathbf{A}_f^{i,i+1} = \mathbf{A}_f^{i,i} = 1/2$
$\mathbf{E}_f$	Edge vectors	$\mathbf{E}_f = \mathbf{D}_f \mathbf{X}_f$
$\mathbf{B}_f$	Edge midpoints	$\mathbf{B}_f = \mathbf{A}_f \mathbf{X}_f$
$\mathbf{c}_f$	Face center	$\mathbf{c}_f = \mathbf{X}_f^t \mathbf{1}_f / n_f$
$\mathbf{a}_f$	Polygonal vector area	$\mathbf{a}_f = 1/2 \sum_{v_i \in f} \mathbf{x}_{v_i} \times \mathbf{x}_{v_{i+1}}$
a_f	Area of polygonal face	$a_f = \lvert \mathbf{a}_f \rvert$
$\mathbf{n}_f$	Normal of polygonal face	$\mathbf{n}_f = \mathbf{a}_f / a_f$
$\mathbf{h}_f$	Vertex heights for polygonal face	$\mathbf{h}_f = (\mathbf{X}_f - \mathbf{1}_f \mathbf{c}_f^t)\mathbf{n}_f$

Matrix form of per face operators



$$\phi_f = [\phi(v_1)\ldots\phi(v_{n_f})]^t$$

$$\mathbf{G}_f^{\perp} := \mathbf{E}_f^t \mathbf{A}_f$$

$$\mathbf{G}_f := -\frac{1}{a_f}[\mathbf{n}_f]\mathbf{E}_f^t \mathbf{A}_f$$

$3 \times n$ *matrix*

Symbol	Meaning	Definition
n_f	Number of vertices	$v_1, \ldots, v_{n_f} \in f$
$\mathbf{X}_f$	Vertex positions	$\mathbf{X}_f = [\mathbf{x}_{v_1} \ldots \mathbf{x}_{v_{n_f}}]^t \in \mathbb{R}^{n_f \times 3}$
$\mathbf{D}_f$	Difference operator	$\mathbf{D}_f^{i,i+1} = 1,\ \mathbf{D}_f^{i,i} = -1$
$\mathbf{A}_f$	Average operator	$\mathbf{A}_f^{i,i+1} = \mathbf{A}_f^{i,i} = 1/2$
$\mathbf{E}_f$	Edge vectors	$\mathbf{E}_f = \mathbf{D}_f \mathbf{X}_f$
$\mathbf{B}_f$	Edge midpoints	$\mathbf{B}_f = \mathbf{A}_f \mathbf{X}_f$
$\mathbf{c}_f$	Face center	$\mathbf{c}_f = \mathbf{X}_f^t \mathbf{1}_f / n_f$
$\mathbf{a}_f$	Polygonal vector area	$\mathbf{a}_f = 1/2 \sum_{v_i \in f} \mathbf{x}_{v_i} \times \mathbf{x}_{v_{i+1}}$
a_f	Area of polygonal face	$a_f = \lvert\mathbf{a}_f\rvert$
$\mathbf{n}_f$	Normal of polygonal face	$\mathbf{n}_f = \mathbf{a}_f / a_f$
$\mathbf{h}_f$	Vertex heights for polygonal face	$\mathbf{h}_f = (\mathbf{X}_f - \mathbf{1}_f \mathbf{c}_f^t)\mathbf{n}_f$

$$G_f \phi_f : \quad 3 \times 1$$

Wrap up for polygonal non-convex / non-planar faces

Wrap up for polygonal non-convex / non-planar faces

- Per face, globally consistent, linear operators

Wrap up for polygonal non-convex / non-planar faces

- Per face, globally consistent, linear operators

$$\mathbf{G}_f = -\frac{1}{a_f}[\mathbf{n}_f]\,\mathbf{E}_f^t\mathbf{A}_f.$$

gradient

Wrap up for polygonal non-convex / non-planar faces

- Per face, globally consistent, linear operators

$$\mathbf{G}_f = -\frac{1}{a_f}[\mathbf{n}_f]\mathbf{E}_f^t\mathbf{A}_f.$$

gradient

$$\mathbf{V}_f = \mathbf{E}_f\left(\mathbf{I} - \mathbf{n}_f\mathbf{n}_f^t\right).$$

flat

Wrap up for polygonal non-convex / non-planar faces

- Per face, globally consistent, linear operators

$$\mathbf{G}_f = -\frac{1}{a_f}[\mathbf{n}_f]\mathbf{E}_f^t\mathbf{A}_f.$$

gradient

$$\mathbf{V}_f = \mathbf{E}_f\left(\mathbf{I} - \mathbf{n}_f\mathbf{n}_f^t\right).$$

flat

$$\mathbf{U}_f = \frac{1}{a_f}[\mathbf{n}_f](\mathbf{B}_f^t - \mathbf{c}_f\mathbf{1}_f^t).$$

sharp

Wrap up for polygonal non-convex / non-planar faces

- Per face, globally consistent, linear operators

$$\mathbf{G}_f = -\frac{1}{a_f}[\mathbf{n}_f]\mathbf{E}_f^t\mathbf{A}_f.$$

gradient

$$\mathbf{V}_f = \mathbf{E}_f\left(\mathbf{I} - \mathbf{n}_f\mathbf{n}_f^t\right).$$

flat

$$\mathbf{U}_f = \frac{1}{a_f}[\mathbf{n}_f](\mathbf{B}_f^t - \mathbf{c}_f\mathbf{1}_f^t).$$

sharp

$$\mathbf{P}_f = \mathbf{I} - \mathbf{V}_f\mathbf{U}_f.$$

projection

Wrap up for polygonal non-convex / non-planar faces

- Per face, globally consistent, linear operators

$$\mathbf{G}_f = -\frac{1}{a_f}[\mathbf{n}_f]\mathbf{E}_f^t\mathbf{A}_f.$$

gradient

$$\mathbf{V}_f = \mathbf{E}_f\left(\mathbf{I} - \mathbf{n}_f\mathbf{n}_f^t\right).$$

flat

$$\mathbf{U}_f = \frac{1}{a_f}[\mathbf{n}_f](\mathbf{B}_f^t - \mathbf{c}_f\mathbf{1}_f^t).$$

sharp

$$\mathbf{P}_f = \mathbf{I} - \mathbf{V}_f\mathbf{U}_f.$$

projection

$$\mathbf{M}_f = a_f\,\mathbf{U}_f^t\mathbf{U}_f + \lambda\,\mathbf{P}_f^t\mathbf{P}_f,$$

Inner prod. 1-form

Wrap up for polygonal non-convex / non-planar faces

- Per face, globally consistent, linear operators

$$\mathbf{G}_f = -\frac{1}{a_f}[\mathbf{n}_f]\mathbf{E}_f^t\mathbf{A}_f.$$

gradient

$$\mathbf{V}_f = \mathbf{E}_f\left(\mathbf{I} - \mathbf{n}_f\mathbf{n}_f^t\right).$$

flat

$$\mathbf{U}_f = \frac{1}{a_f}[\mathbf{n}_f](\mathbf{B}_f^t - \mathbf{c}_f\mathbf{1}_f^t).$$

sharp

$$\mathbf{P}_f = \mathbf{I} - \mathbf{V}_f\mathbf{U}_f.$$

projection

$$\mathbf{M}_f = a_f\,\mathbf{U}_f^t\mathbf{U}_f + \lambda\,\mathbf{P}_f^t\mathbf{P}_f,$$

Inner prod. 1-form

$$\mathbf{L}_f = \mathbf{D}_f^t\mathbf{M}_f\mathbf{D}_f.$$

Laplace-Beltrami

Wrap up for polygonal non-convex / non-planar faces

- Per face, globally consistent, linear operators

$$\mathbf{G}_f = -\frac{1}{a_f}[\mathbf{n}_f]\mathbf{E}_f^t\mathbf{A}_f.$$

gradient

$$\mathbf{V}_f = \mathbf{E}_f\left(\mathbf{I} - \mathbf{n}_f\mathbf{n}_f^t\right).$$

flat

$$\mathbf{U}_f = \frac{1}{a_f}[\mathbf{n}_f](\mathbf{B}_f^t - \mathbf{c}_f\mathbf{1}_f^t).$$

sharp

$$\mathbf{P}_f = \mathbf{I} - \mathbf{V}_f\mathbf{U}_f.$$

projection

$$\mathbf{M}_f = a_f\,\mathbf{U}_f^t\mathbf{U}_f + \lambda\,\mathbf{P}_f^t\mathbf{P}_f,$$

Inner prod. 1-form

$$\mathbf{L}_f = \mathbf{D}_f^t\mathbf{M}_f\mathbf{D}_f.$$

Laplace-Beltrami

Wrap up for polygonal non-convex / non-planar faces

- Per face, globally consistent, linear operators

$$\mathbf{G}_f = -\frac{1}{a_f}[\mathbf{n}_f]\mathbf{E}_f^t\mathbf{A}_f.$$

gradient

$$\mathbf{V}_f = \mathbf{E}_f\left(\mathbf{I} - \mathbf{n}_f\mathbf{n}_f^t\right).$$

flat

$$\mathbf{U}_f = \frac{1}{a_f}[\mathbf{n}_f](\mathbf{B}_f^t - \mathbf{c}_f\mathbf{1}_f^t).$$

sharp

$$\mathbf{P}_f = \mathbf{I} - \mathbf{V}_f\mathbf{U}_f.$$

projection

$$\mathbf{M}_f = a_f\,\mathbf{U}_f^t\mathbf{U}_f + \lambda\,\mathbf{P}_f^t\mathbf{P}_f,$$

Inner prod. 1-form

$$\mathbf{L}_f = \mathbf{D}_f^t\mathbf{M}_f\mathbf{D}_f.$$

Laplace-Beltrami

Wrap up for polygonal non-convex / non-planar faces

- Per face, globally consistent, linear operators

$$\mathbf{G}_f = -\frac{1}{a_f}[\mathbf{n}_f]\mathbf{E}_f^t\mathbf{A}_f.$$

gradient

$$\mathbf{V}_f = \mathbf{E}_f\left(\mathbf{I} - \mathbf{n}_f\mathbf{n}_f^t\right).$$

flat

$$\mathbf{U}_f = \frac{1}{a_f}[\mathbf{n}_f](\mathbf{B}_f^t - \mathbf{c}_f\mathbf{1}_f^t).$$

sharp

$$\mathbf{P}_f = \mathbf{I} - \mathbf{V}_f\mathbf{U}_f.$$

projection

$$\mathbf{M}_f = a_f\,\mathbf{U}_f^t\mathbf{U}_f + \lambda\,\mathbf{P}_f^t\mathbf{P}_f,$$

Inner prod. 1-form

$$\mathbf{L}_f = \mathbf{D}_f^t\mathbf{M}_f\mathbf{D}_f.$$

Laplace-Beltrami

Wrap up for polygonal non-convex / non-planar faces

- Per face, globally consistent, linear operators

$$\mathbf{G}_f = -\frac{1}{a_f}[\mathbf{n}_f]\mathbf{E}_f^t\mathbf{A}_f.$$

gradient

$$\mathbf{V}_f = \mathbf{E}_f\left(\mathbf{I} - \mathbf{n}_f\mathbf{n}_f^t\right).$$

flat

$$\mathbf{U}_f = \frac{1}{a_f}[\mathbf{n}_f](\mathbf{B}_f^t - \mathbf{c}_f\mathbf{1}_f^t).$$

sharp

$$\mathbf{P}_f = \mathbf{I} - \mathbf{V}_f\mathbf{U}_f.$$

projection

$$\mathbf{M}_f = a_f\,\mathbf{U}_f^t\mathbf{U}_f + \lambda\,\mathbf{P}_f^t\mathbf{P}_f,$$

Inner prod. 1-form

$$\mathbf{L}_f = \mathbf{D}_f^t\mathbf{M}_f\mathbf{D}_f.$$

Laplace-Beltrami

Wrap up for polygonal non-convex / non-planar faces

- Per face, globally consistent, linear operators

$$\mathbf{G}_f = -\frac{1}{a_f}[\mathbf{n}_f]\mathbf{E}_f^t\mathbf{A}_f.$$

gradient

$$\mathbf{V}_f = \mathbf{E}_f\left(\mathbf{I} - \mathbf{n}_f\mathbf{n}_f^t\right).$$

flat

$$\mathbf{U}_f = \frac{1}{a_f}[\mathbf{n}_f](\mathbf{B}_f^t - \mathbf{c}_f\mathbf{1}_f^t).$$

sharp

$$\mathbf{P}_f = \mathbf{I} - \mathbf{V}_f\mathbf{U}_f.$$

projection

$$\mathbf{M}_f = a_f\,\mathbf{U}_f^t\mathbf{U}_f + \lambda\,\mathbf{P}_f^t\mathbf{P}_f,$$

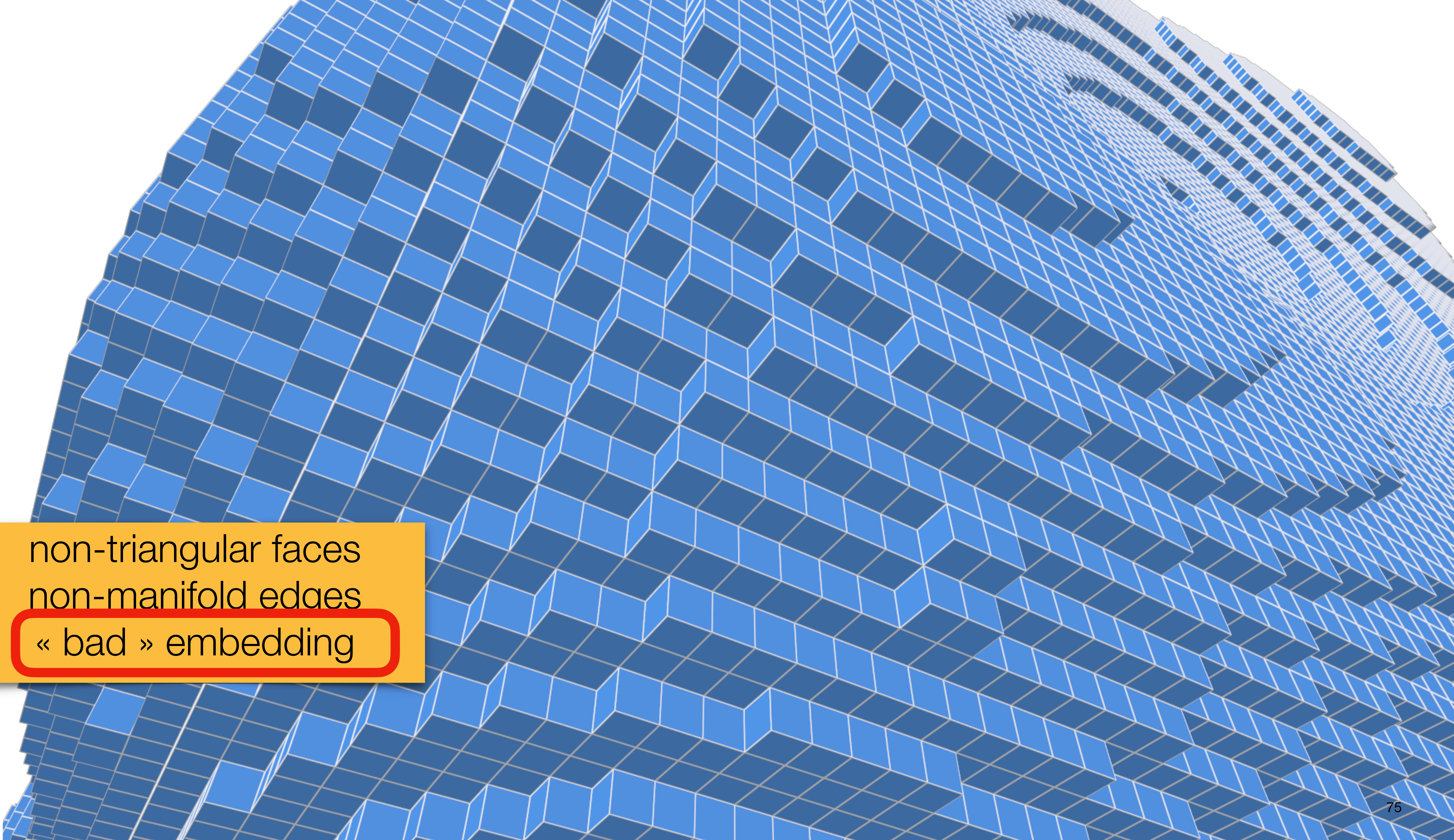
Inner prod. 1-form

$$\mathbf{L}_f = \mathbf{D}_f^t\mathbf{M}_f\mathbf{D}_f.$$

Laplace-Beltrami

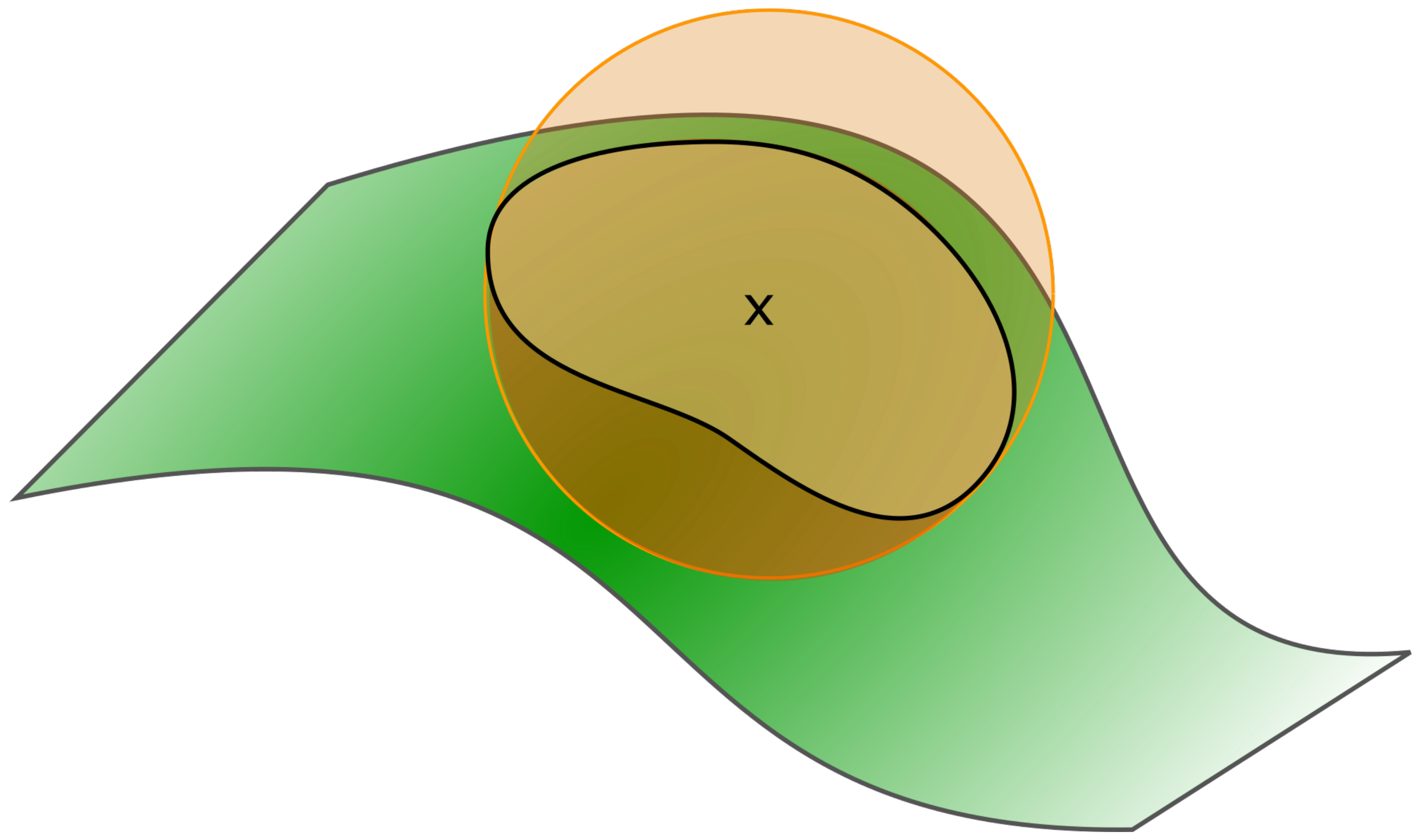
- …But… flat metric space from the mesh embedding

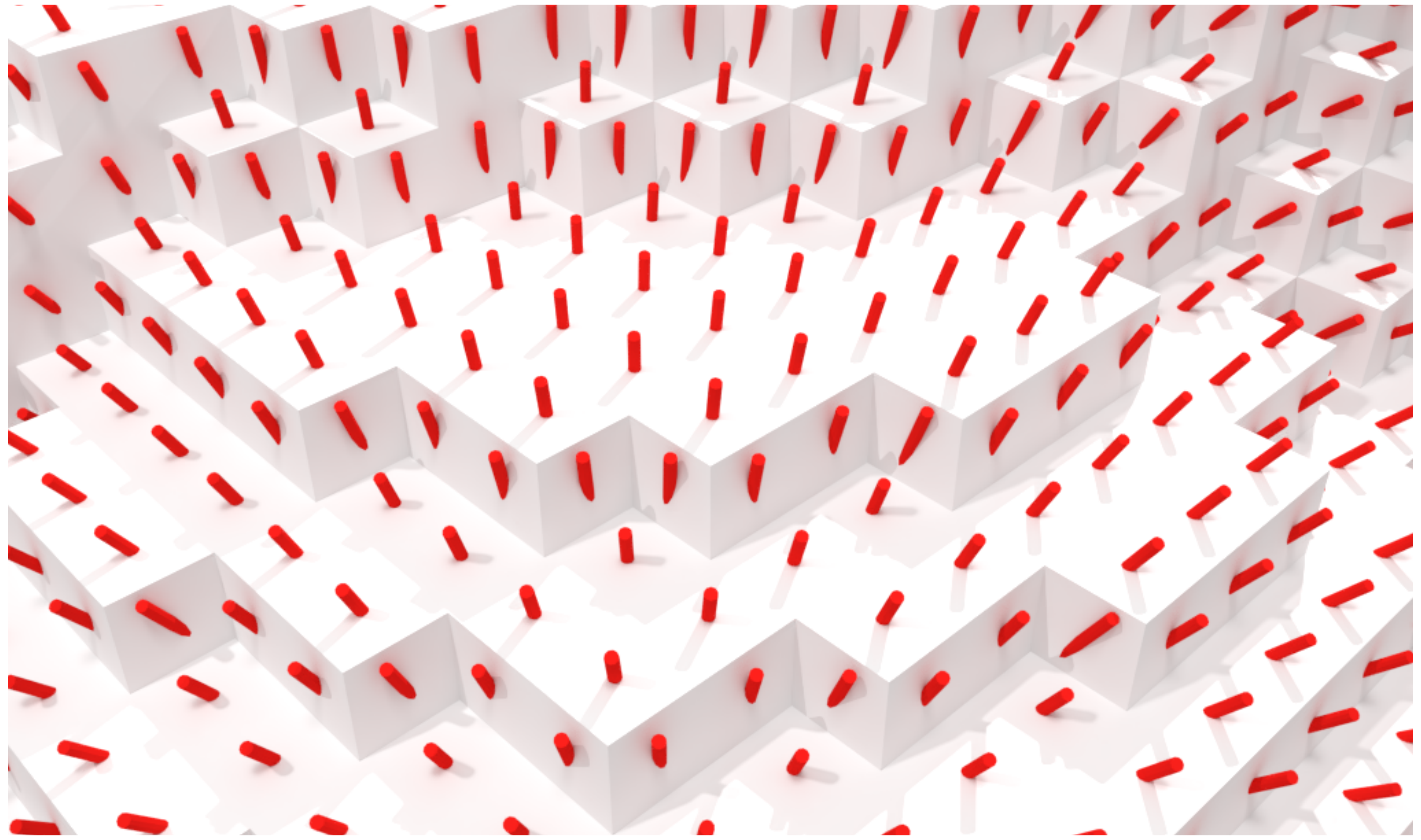
```
//Flat
Eigen::MatrixXd V(const Face f)
{
  return E(f)*( Eigen::MatrixXd::Identity(3,3) - normalFace(f)*normalFace(f).transpose());
}

//Edge midPoints
Eigen::MatrixXd B(const Face f)
{
  return A(f) * X(f);
}

//Centroids
Eigen::VectorXd centroid(const Face f)
{
  return 1/(double)f.degree() * X(f).transpose() * Eigen::VectorXd::Ones(f.degree());
}

//Sharp
Eigen::MatrixXd U(const Face f)
{
  return 1/areaFace(f) * bracket(normalFace(f)) *
        ( B(f).transpose() - centroid(f)* Eigen::VectorXd::Ones(f.degree()).transpose() );
}

//Projection
Eigen::MatrixXd P(const Face f)
{
  return Eigen::MatrixXd::Identity(f.degree(),f.degree()) - V(f)*U(f);
}

//Mass Matrix
Eigen::MatrixXd M(const Face f, const double lambda=1.0)
{
  return areaFace(f) * U(f).transpose()*U(f) + lambda * P(f).transpose()*P(f);
}
//weak Laplacian
Eigen::MatrixXd L(const Face f, const double lambda=1.0)
{
  return D(f).transpose() * M(f,lambda) * D(f);
}
```

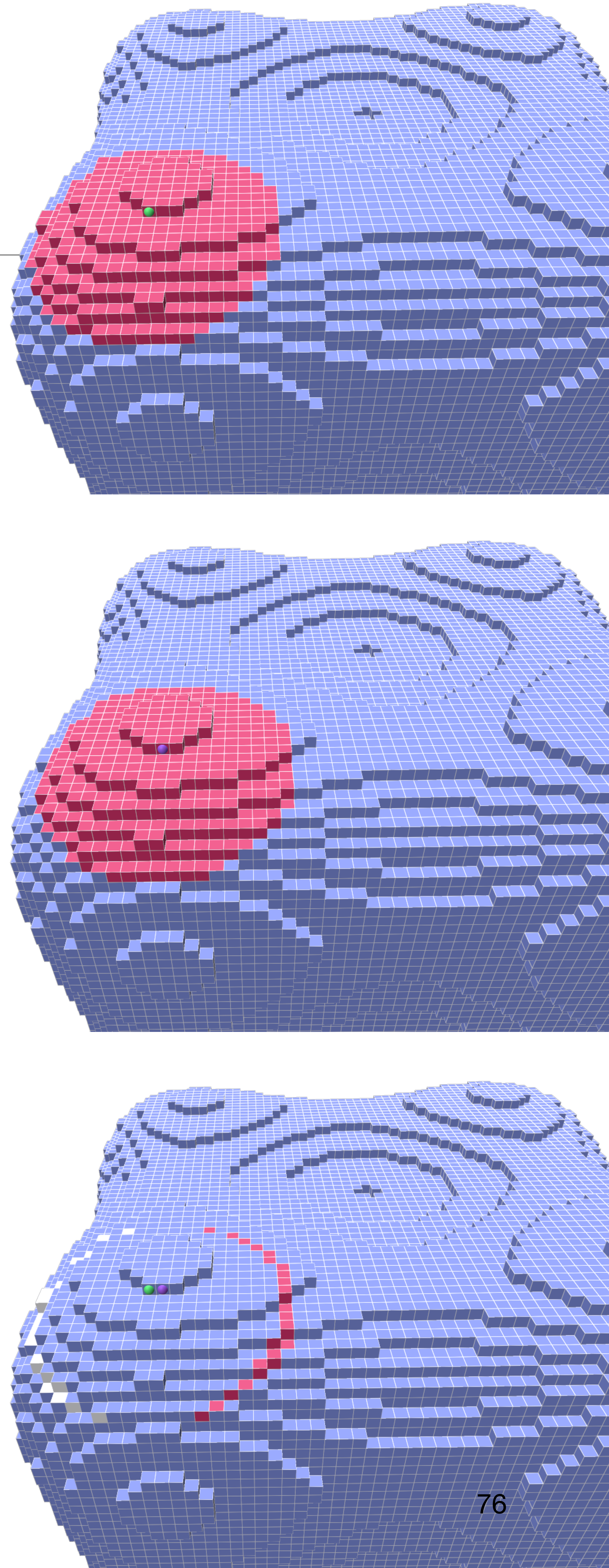

non-triangular faces
non-manifold edges
« bad » embedding

Normal Vector estimation from Integral Invariants

normal vectors from eigenvectors of the covariance matrix of $B_r(p) \cap X$





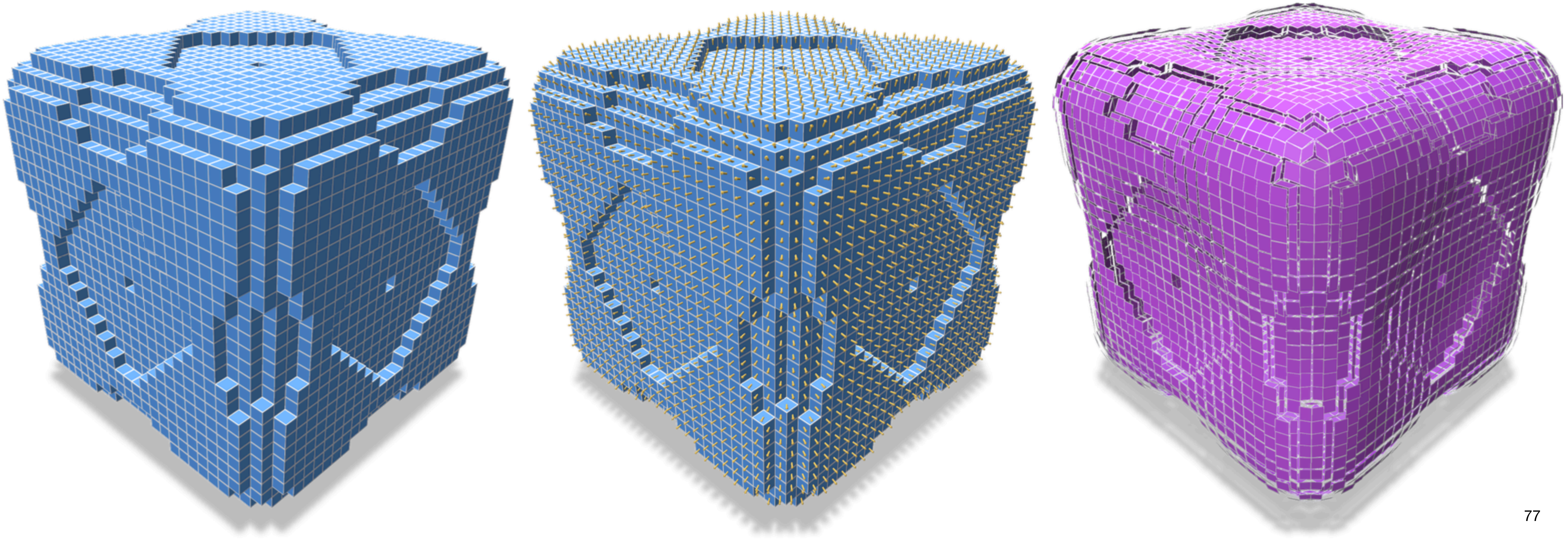


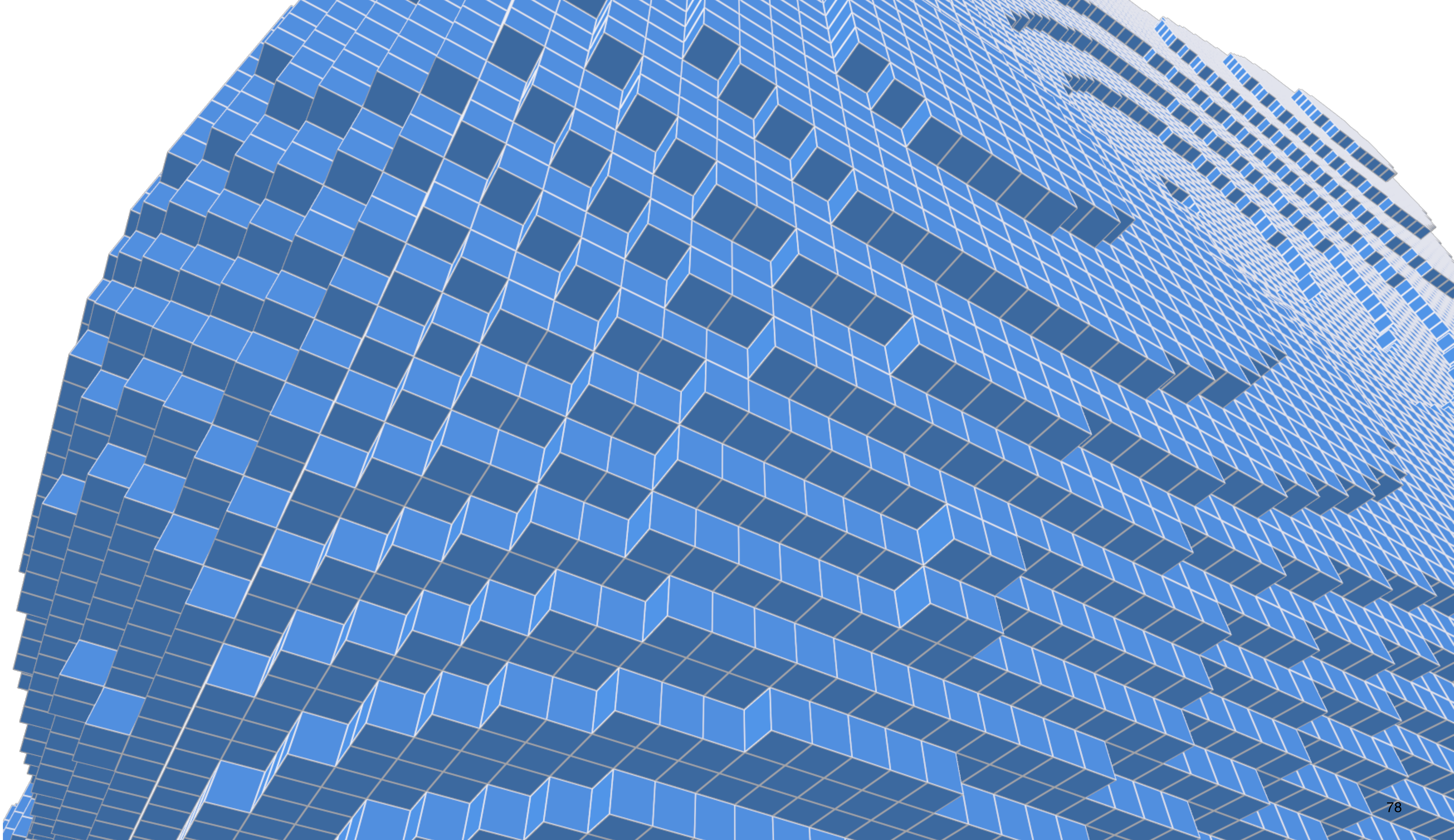
Fast computation, multigrid convergence properties [Lachaud et al 17]

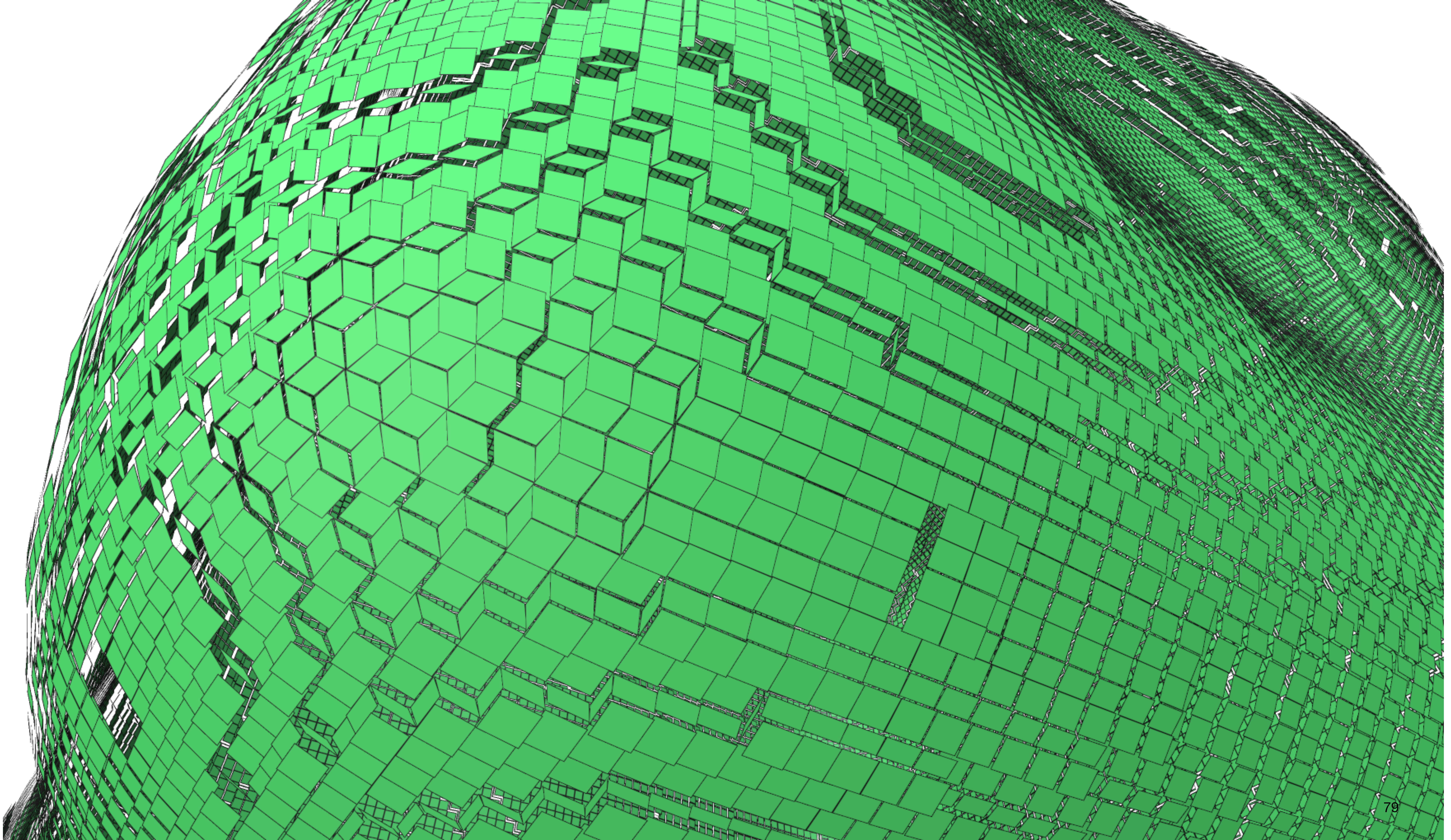
Implicit Projected Embedding

projection operator $\Pi_f := (\mathbf{I}_{3\times 3} - \mathbf{u}_f \mathbf{u}_f^t)$

« implicit » positions $\mathbf{X}_f^* := \mathbf{X}_f \Pi_f$

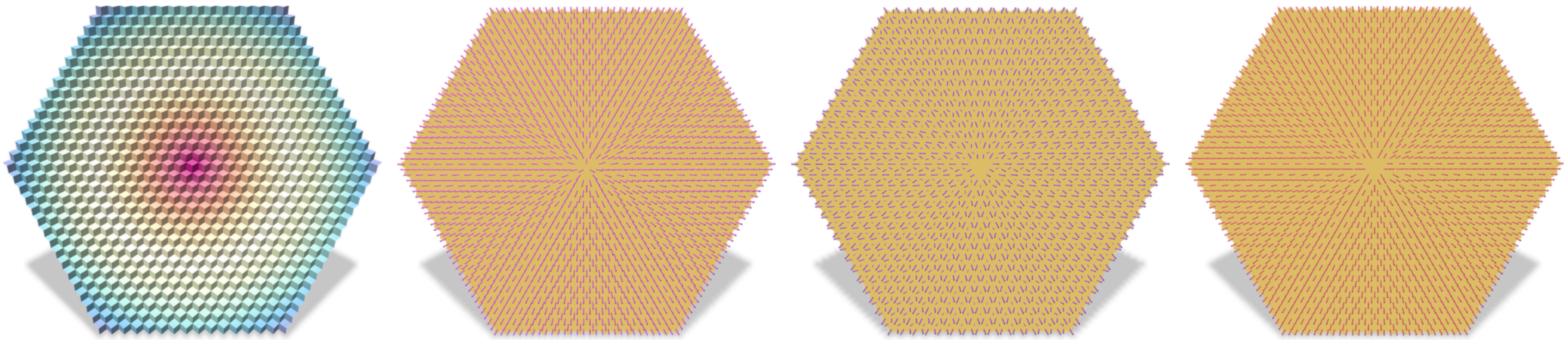




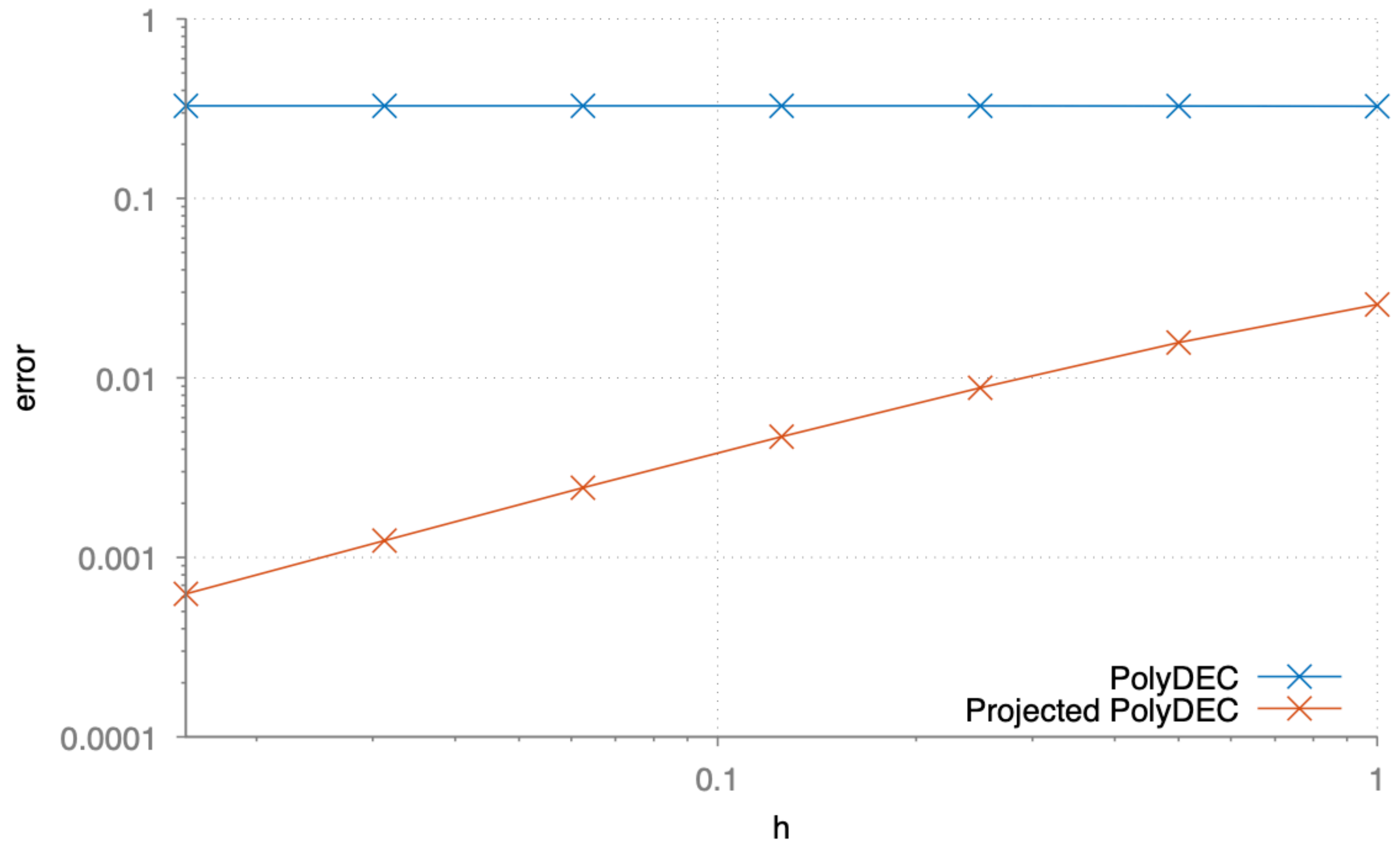


⇒ **Per face operators « à la » [de Goes et al] with Π_f correction**

Experimental validation: Gradient accuracy

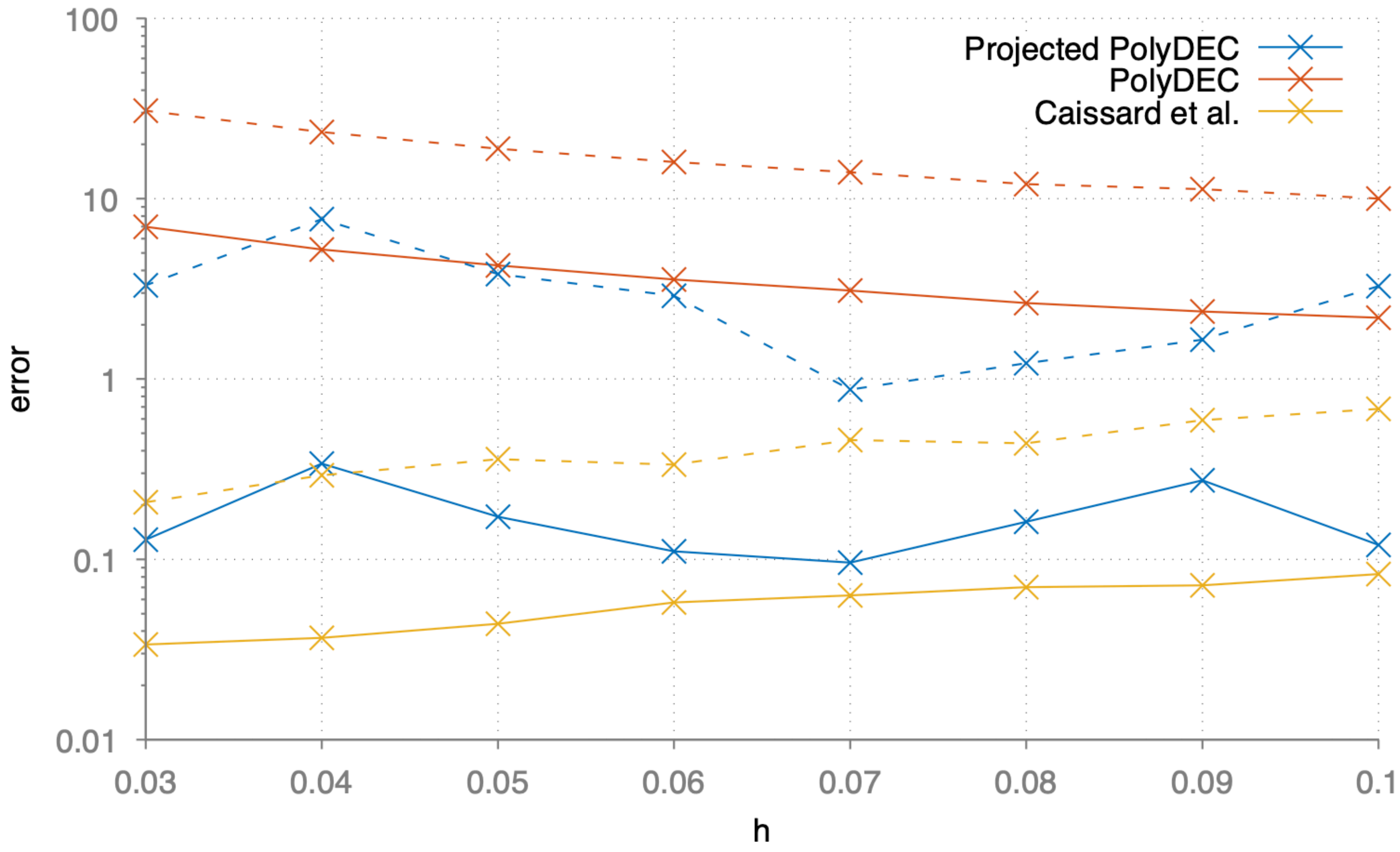


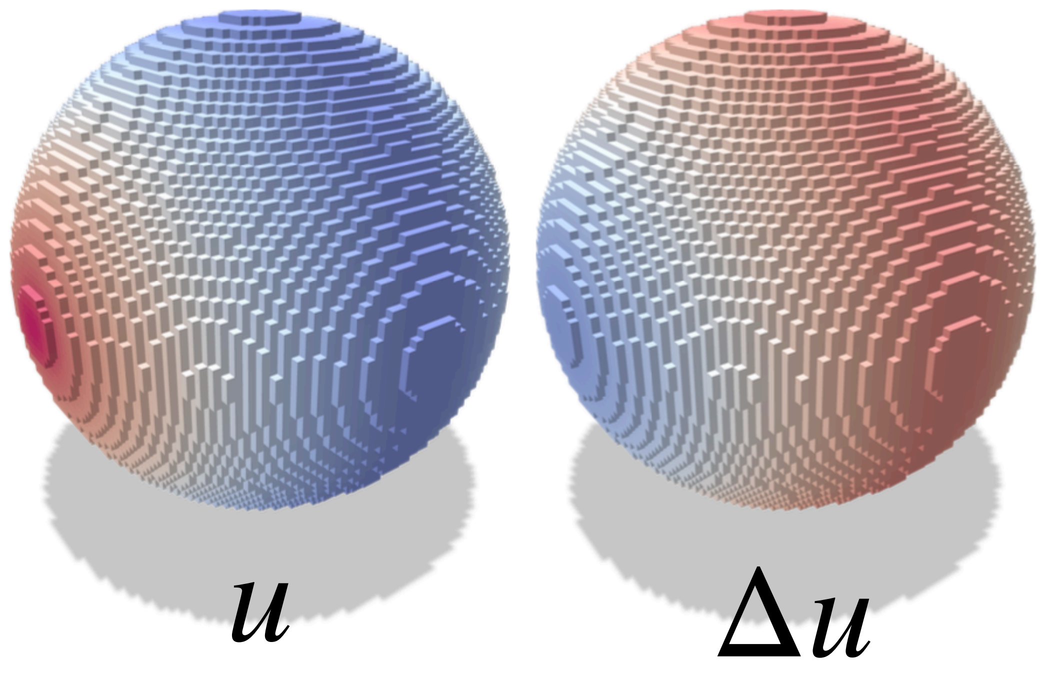
(a) (b) (c) (d)

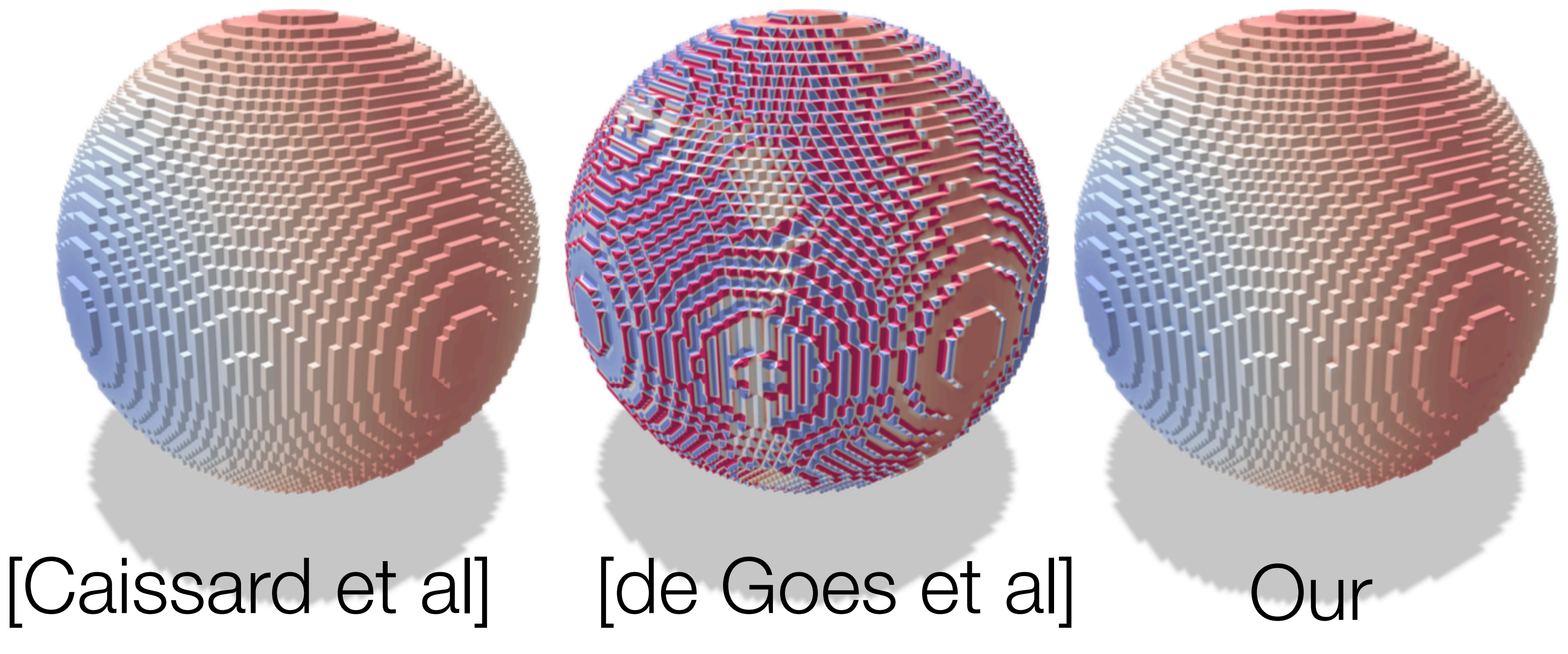


Experimental validation: Laplace-Beltrami operator

- **Setting**:
  - scalar function u on a sphere with closed form Δu
  - multigrid spheres and discrete operators
- Compared to [Caissard et al.] which is a **strong consistent** operator



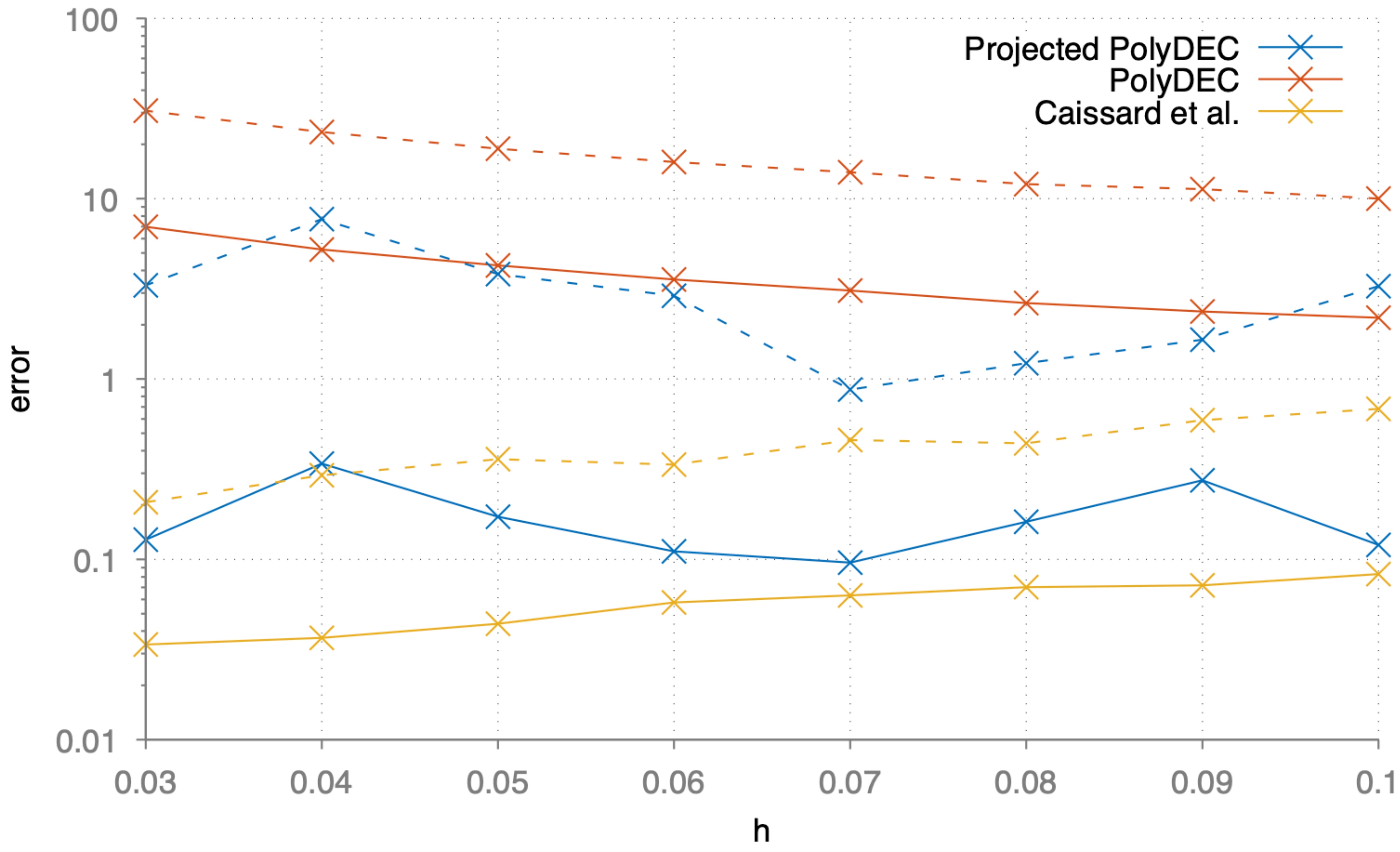




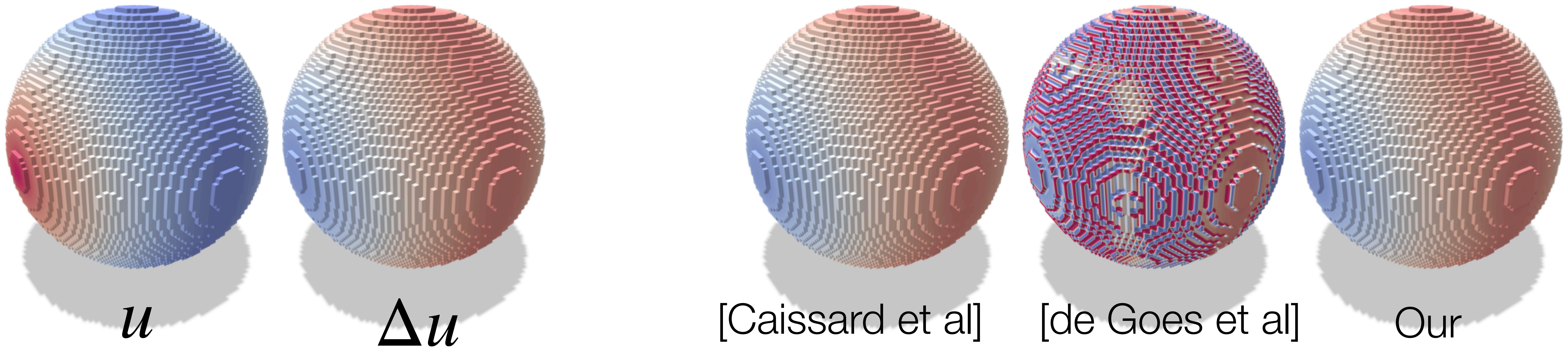
Experimental validation: Laplace-Beltrami operator

- **Setting**:
  - scalar function u on a sphere with closed form Δu
  - multigrid spheres and discrete operators

- Compared to [Caissard et al.] which is a **strong consistent** operator



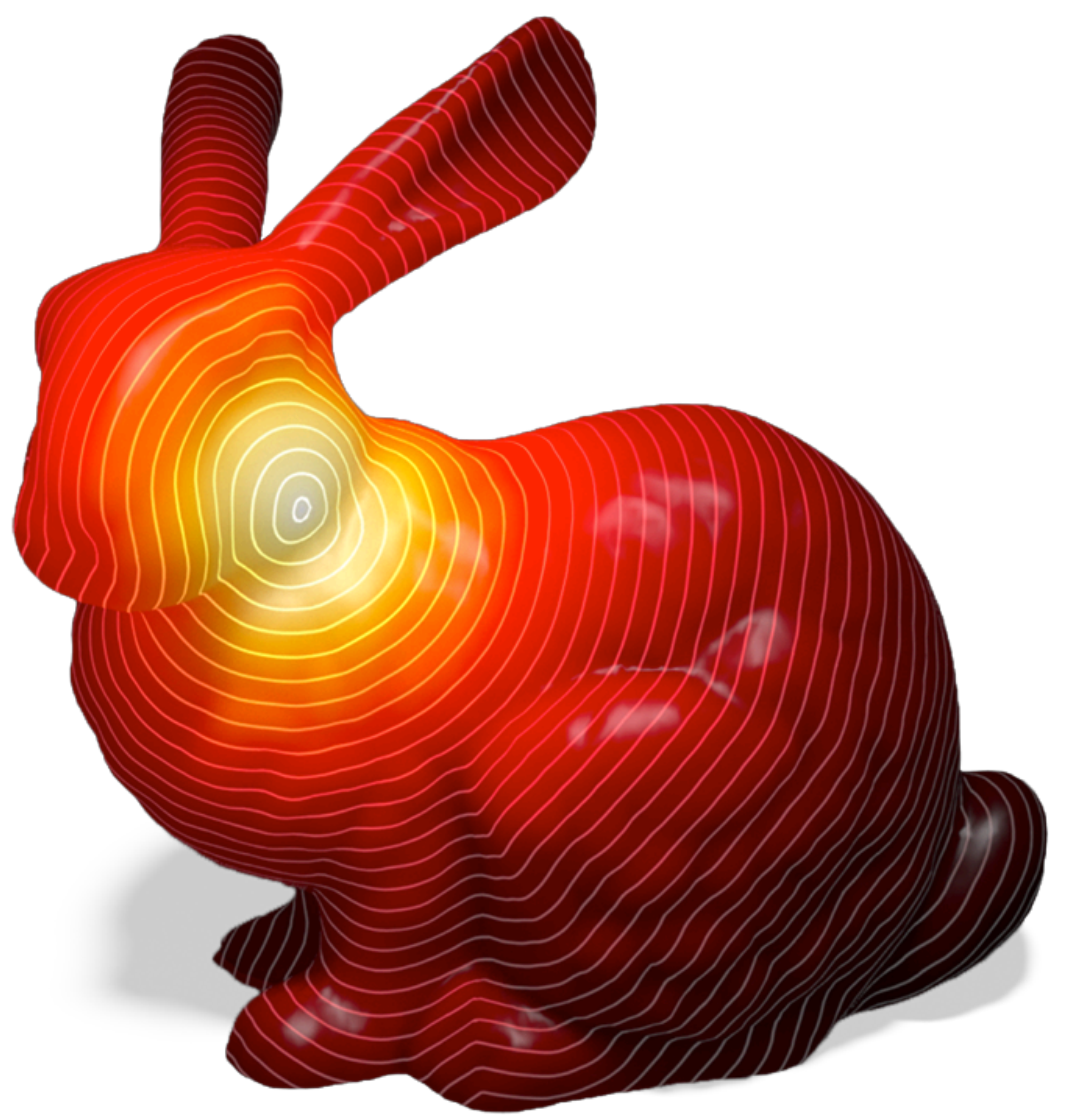
[Caissard et al] $O(n^2)$ construction time, not



Experimental validation: stability of Laplace-Beltrami eigenvectors

Algorithm 1 The Heat Method

I. Integrate the heat flow $\dot{u} = \Delta u$ for some fixed time t.
II. Evaluate the vector field $X = -\nabla u_t / |\nabla u_t|$.
III. Solve the Poisson equation $\Delta \phi = \nabla \cdot X$.

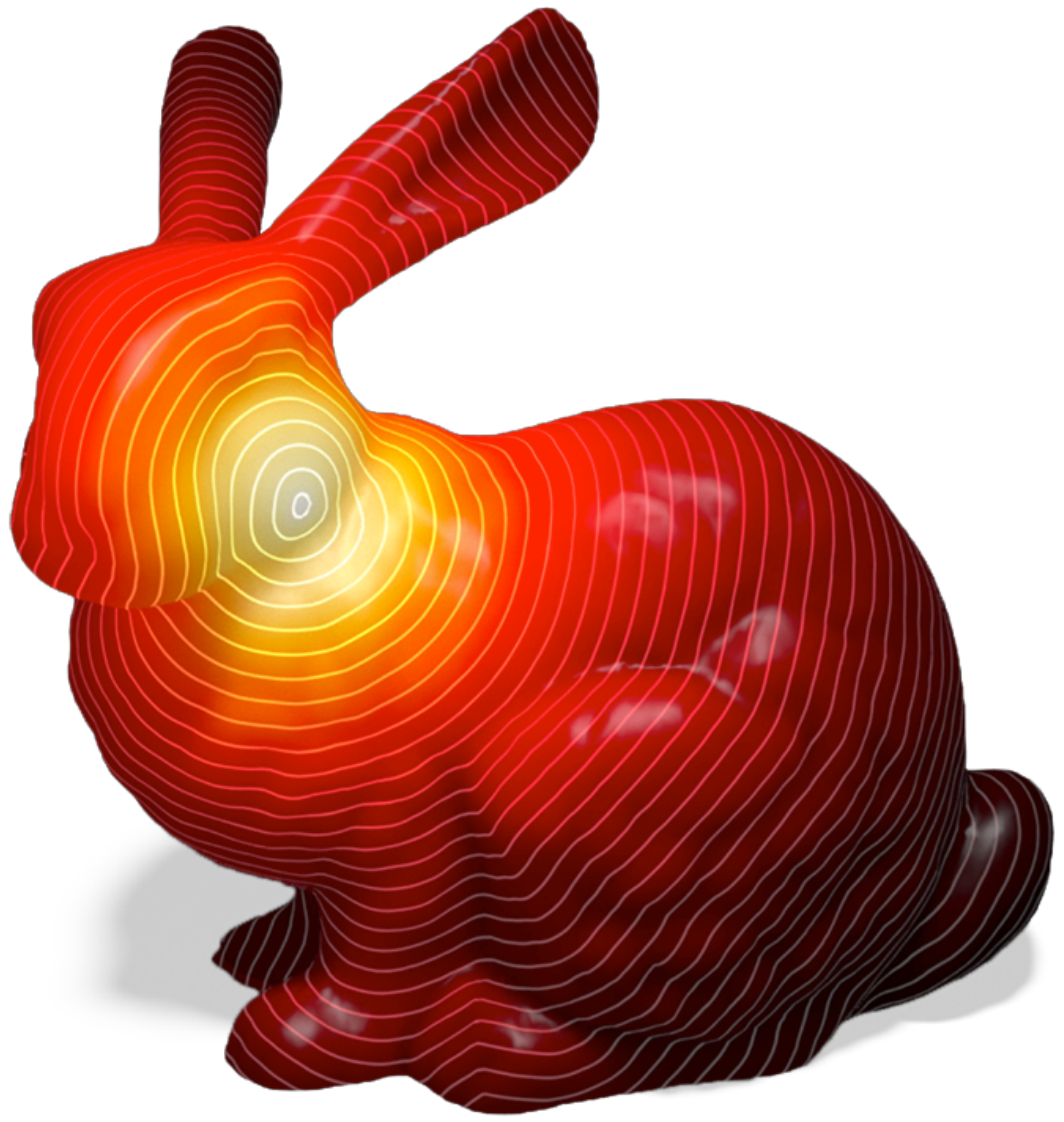


[Crane et al 13]

```
SparseMatrix<double> heatOpe = Mass + dt*lapGlobal;
PositiveDefiniteSolver<double> heatSolver(heatOpe);
PositiveDefiniteSolver<double> poissonSolver(lapGlobal);

// === Solve heat
Vector<double> heatVec = heatSolver.solve(U);

//  // === Normalize in each face and evaluate divergence
FaceData<Vector3> gradHeat(*mesh);
Vector<double> divergenceVec = Vector<double>::Zero(mesh->nVertices());
for(auto f: mesh->faces())
{
  //Construct div per vertex of the heatVec gradient
  Eigen::VectorXd Heatf( f.degree());
  cpt=0;
  for(auto v: f.adjacentVertices())
  {
    Heatf(cpt) = heatVec( v.getIndex() );
    ++cpt;
  }
  Eigen::Vector3d g = G(f) * Heatf;
  g.normalize();
  gradHeat[f] = toVector3(g);
  Eigen::MatrixXd oneForm = V(f)*g;
  Eigen::VectorXd divergence = D(f).transpose()*M(f)*oneForm;
  cpt=0;
  for(auto v: f.adjacentVertices())
  {
    divergenceVec(v.getIndex()) += divergence(cpt);
    ++cpt;
  }
}

// === Integrate divergence to get distance
Vector<double> distVec = Vector<double>::Ones(mesh->nVertices()) +
                         poissonSolver.solve(divergenceVec);
```

Algorithm 1 The Heat Method

I. Integrate the heat flow $\dot{u} = \Delta u$ for some fixed time t.
II. Evaluate the vector field $X = -\nabla u_t / |\nabla u_t|$.
III. Solve the Poisson equation $\Delta \phi = \nabla \cdot X$.

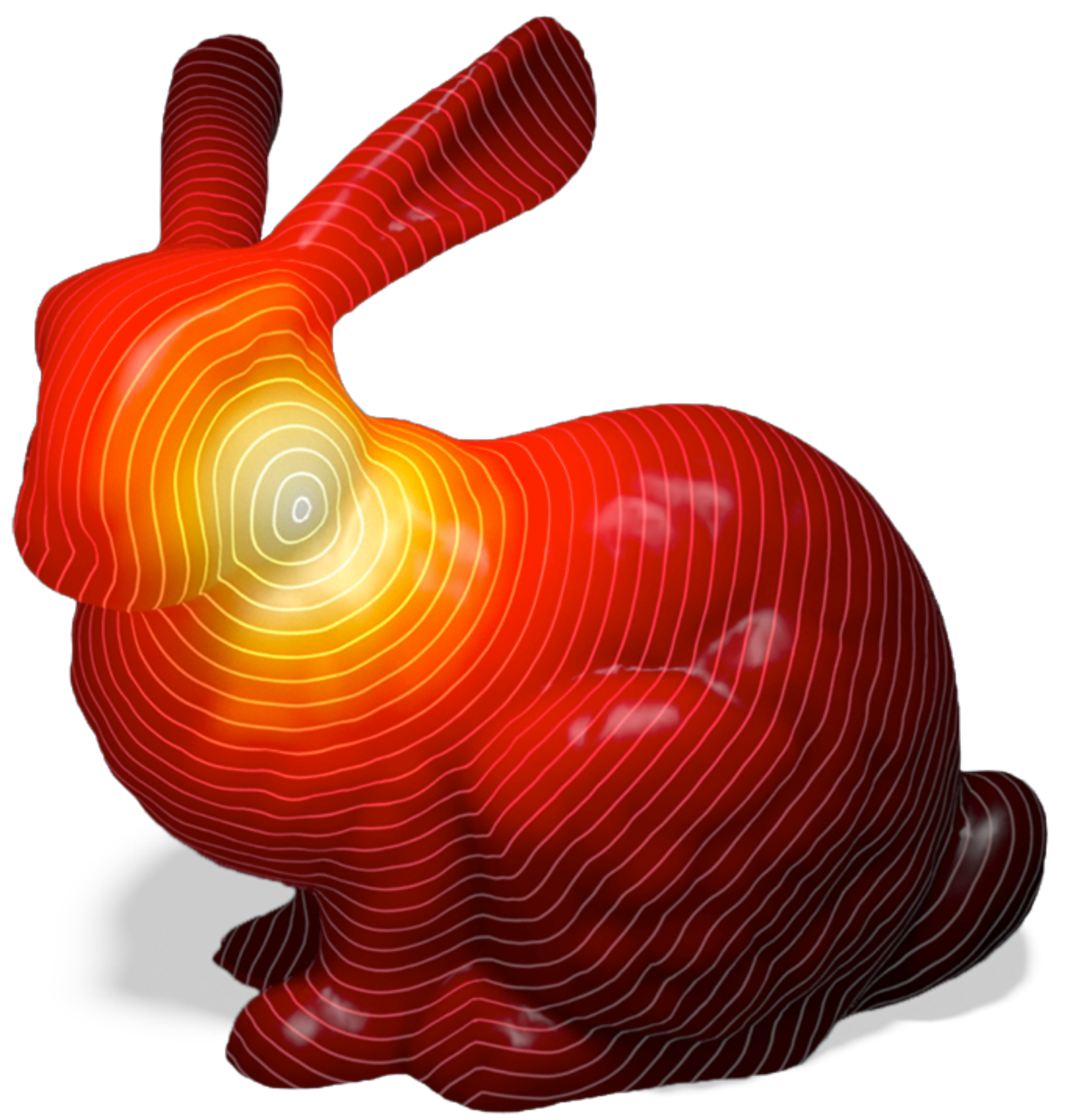


[Crane et al 13]

```
SparseMatrix<double> heatOpe = Mass + dt*lapGlobal;
PositiveDefiniteSolver<double> heatSolver(heatOpe);
PositiveDefiniteSolver<double> poissonSolver(lapGlobal);

// === Solve heat
Vector<double> heatVec = heatSolver.solve(U);

//  // === Normalize in each face and evaluate divergence
FaceData<Vector3> gradHeat(*mesh);
Vector<double> divergenceVec = Vector<double>::Zero(mesh->nVertices());
for(auto f: mesh->faces())
{
  //Construct div per vertex of the heatVec gradient
  Eigen::VectorXd Heatf( f.degree());
  cpt=0;
  for(auto v: f.adjacentVertices())
  {
    Heatf(cpt) = heatVec( v.getIndex() );
    ++cpt;
  }
  Eigen::Vector3d g = G(f) * Heatf;
  g.normalize();
  gradHeat[f] = toVector3(g);
  Eigen::MatrixXd oneForm = V(f)*g;
  Eigen::VectorXd divergence = D(f).transpose()*M(f)*oneForm;
  cpt=0;
  for(auto v: f.adjacentVertices())
  {
    divergenceVec(v.getIndex()) += divergence(cpt);
    ++cpt;
  }
}

// === Integrate divergence to get distance
Vector<double> distVec = Vector<double>::Ones(mesh->nVertices()) +
                         poissonSolver.solve(divergenceVec);
```

Algorithm 1 The Heat Method

I. Integrate the heat flow $\dot{u} = \Delta u$ for some fixed time t.
II. Evaluate the vector field $X = -\nabla u_t / |\nabla u_t|$.
III. Solve the Poisson equation $\Delta \phi = \nabla \cdot X$.

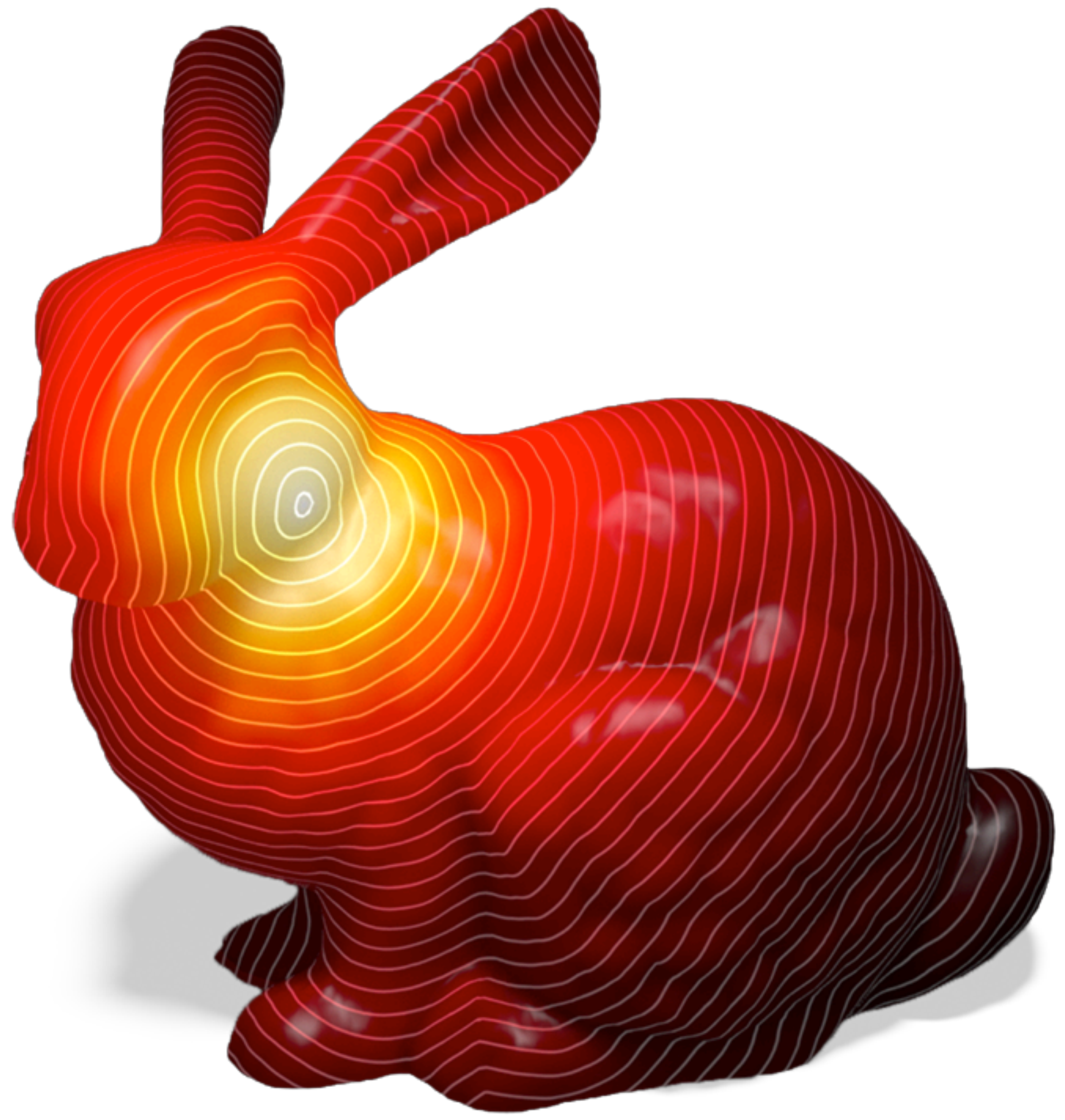


[Crane et al 13]

```
SparseMatrix<double> heatOpe = Mass + dt*lapGlobal;
PositiveDefiniteSolver<double> heatSolver(heatOpe);
PositiveDefiniteSolver<double> poissonSolver(lapGlobal);

// === Solve heat
Vector<double> heatVec = heatSolver.solve(U);

//  // === Normalize in each face and evaluate divergence
FaceData<Vector3> gradHeat(*mesh);
Vector<double> divergenceVec = Vector<double>::Zero(mesh->nVertices());
for(auto f: mesh->faces())
{
  //Construct div per vertex of the heatVec gradient
  Eigen::VectorXd Heatf( f.degree());
  cpt=0;
  for(auto v: f.adjacentVertices())
  {
    Heatf(cpt) = heatVec( v.getIndex() );
    ++cpt;
  }
  Eigen::Vector3d g = G(f) * Heatf;
  g.normalize();
  gradHeat[f] = toVector3(g);
  Eigen::MatrixXd oneForm = V(f)*g;
  Eigen::VectorXd divergence = D(f).transpose()*M(f)*oneForm;
  cpt=0;
  for(auto v: f.adjacentVertices())
  {
    divergenceVec(v.getIndex()) += divergence(cpt);
    ++cpt;
  }
}

// === Integrate divergence to get distance
Vector<double> distVec = Vector<double>::Ones(mesh->nVertices()) +
                        poissonSolver.solve(divergenceVec);
```

Algorithm 1 The Heat Method

I. Integrate the heat flow $\dot{u} = \Delta u$ for some fixed time t.
II. Evaluate the vector field $X = -\nabla u_t / |\nabla u_t|$.
III. Solve the Poisson equation $\Delta\phi = \nabla \cdot X$.



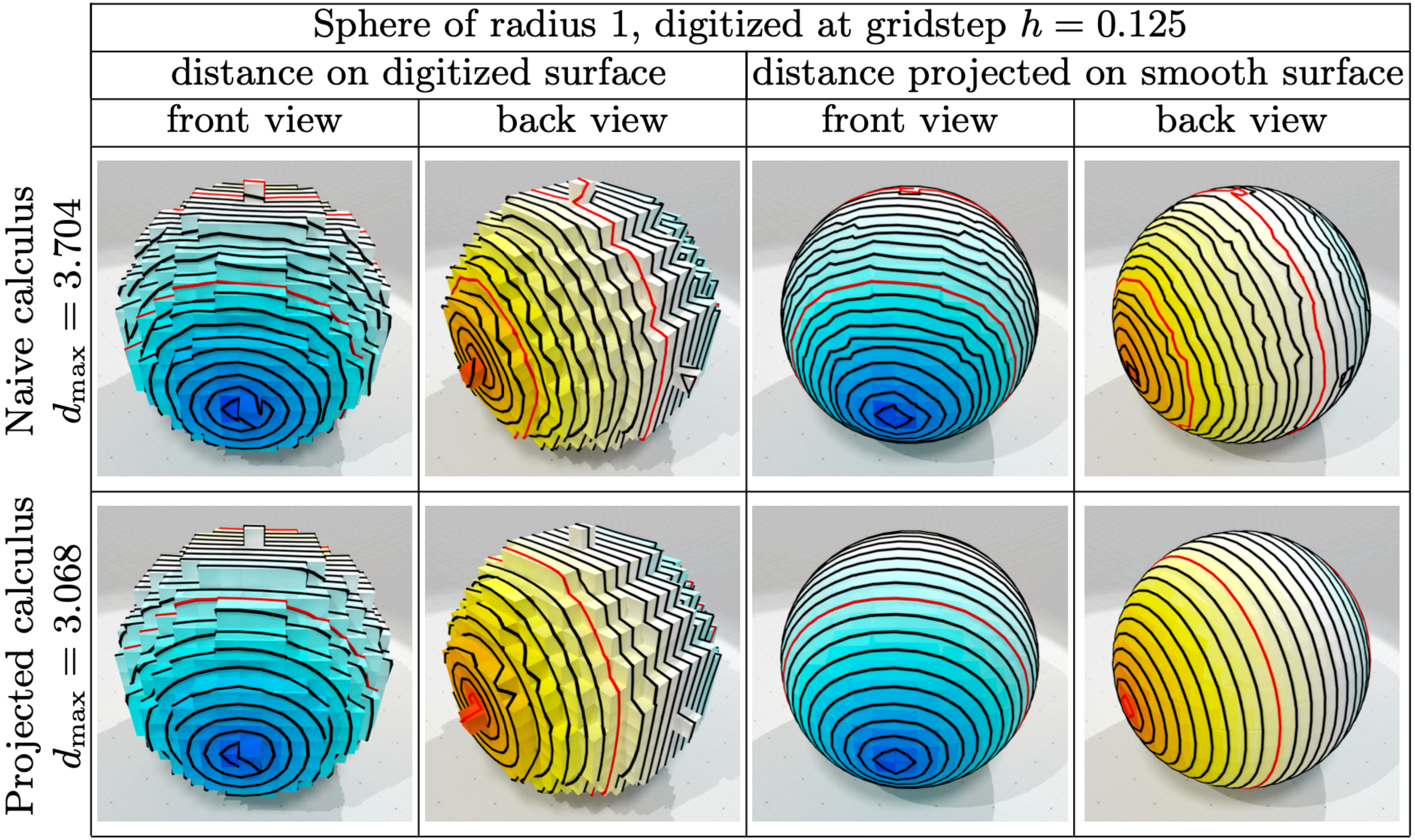
[Crane et al 13]

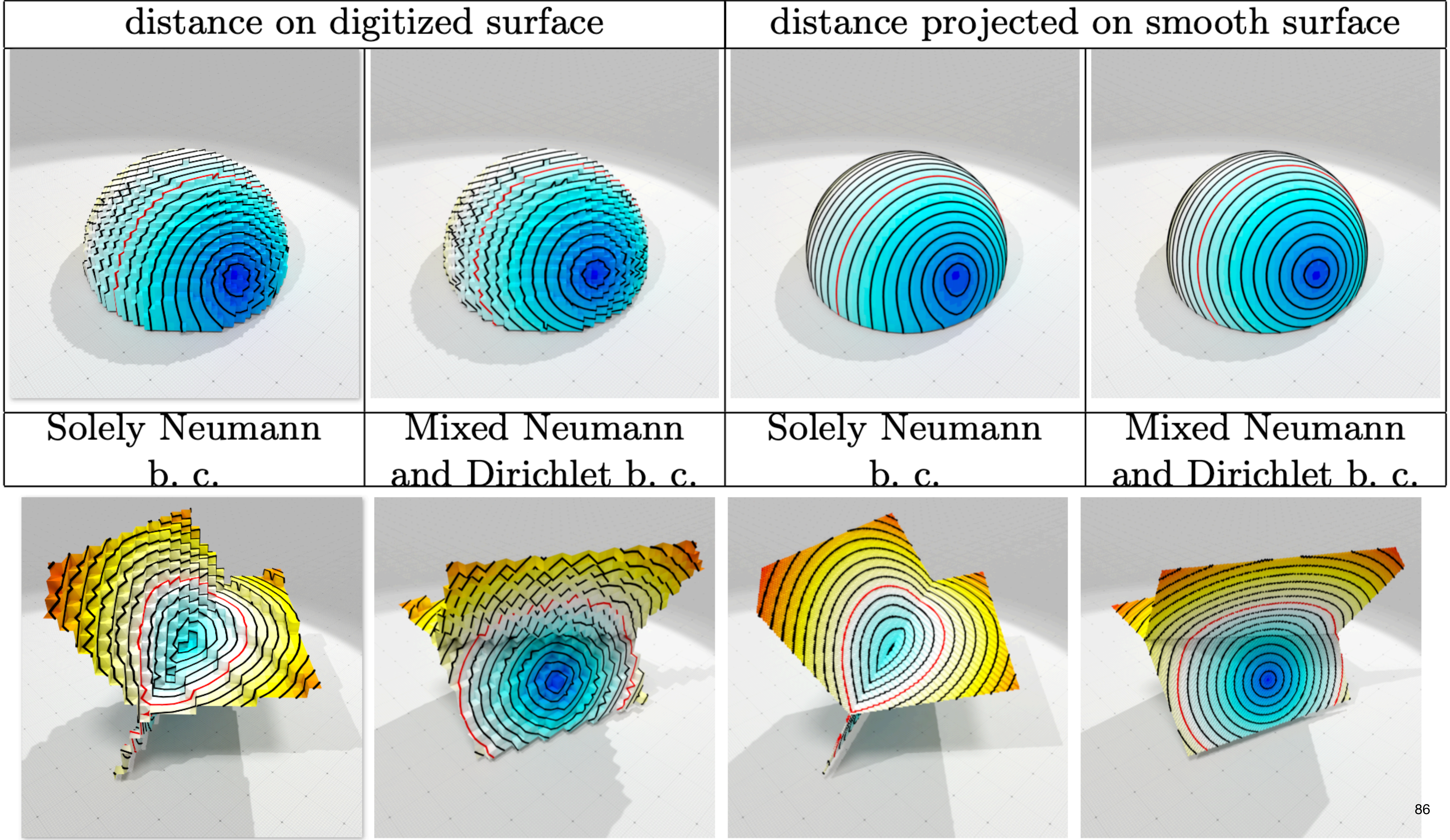
```
SparseMatrix<double> heatOpe = Mass + dt*lapGlobal;
PositiveDefiniteSolver<double> heatSolver(heatOpe);
PositiveDefiniteSolver<double> poissonSolver(lapGlobal);

// === Solve heat
Vector<double> heatVec = heatSolver.solve(U);
```

```
//  // === Normalize in each face and evaluate divergence
FaceData<Vector3> gradHeat(*mesh);
Vector<double> divergenceVec = Vector<double>::Zero(mesh->nVertices());
for(auto f: mesh->faces())
{
  //Construct div per vertex of the heatVec gradient
  Eigen::VectorXd Heatf( f.degree());
  cpt=0;
  for(auto v: f.adjacentVertices())
  {
    Heatf(cpt) = heatVec( v.getIndex() );
    ++cpt;
  }
  Eigen::Vector3d g = G(f) * Heatf;
  g.normalize();
  gradHeat[f] = toVector3(g);
```

```
  Eigen::MatrixXd oneForm = V(f)*g;
  Eigen::VectorXd divergence = D(f).transpose()*M(f)*oneForm;
  cpt=0;
  for(auto v: f.adjacentVertices())
  {
    divergenceVec(v.getIndex()) += divergence(cpt);
    ++cpt;
  }
}

// === Integrate divergence to get distance
Vector<double> distVec = Vector<double>::Ones(mesh->nVertices()) +
                         poissonSolver.solve(divergenceVec);
```

Experimental validation: Geodesics using the heat method




distance on digitized surface
distance projected on smooth surface
Solely Neumann b. c.
Mixed Neumann and Dirichlet b. c.
Solely Neumann b. c.
Mixed Neumann and Dirichlet b. c.

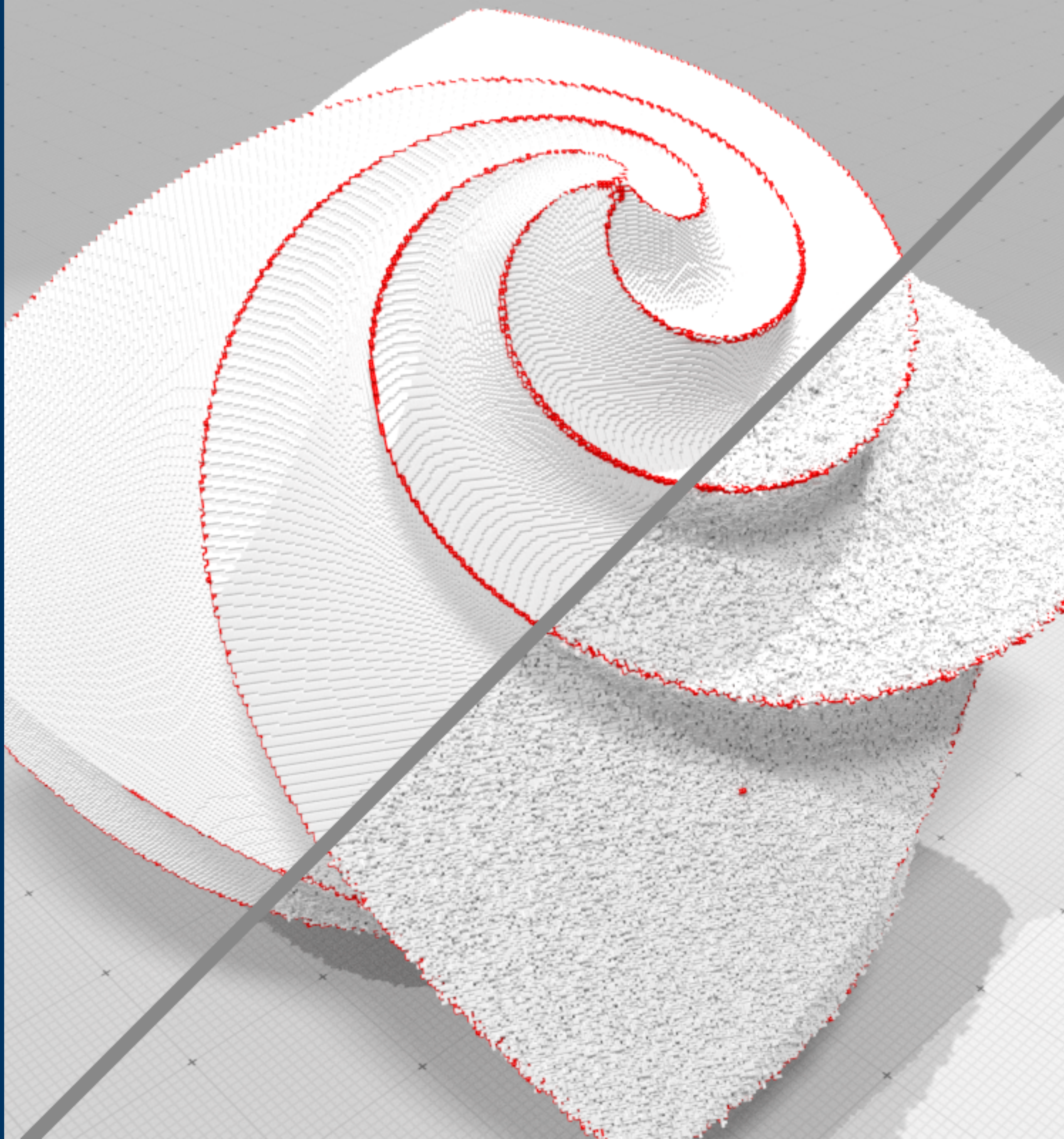
Additional operators

- **Intrinsic vector fields:** transport, connection, covariant derivratives, connection Laplacian…

- **Extrinsic operator:** Shape operator

<demo>

quick wrap-up example

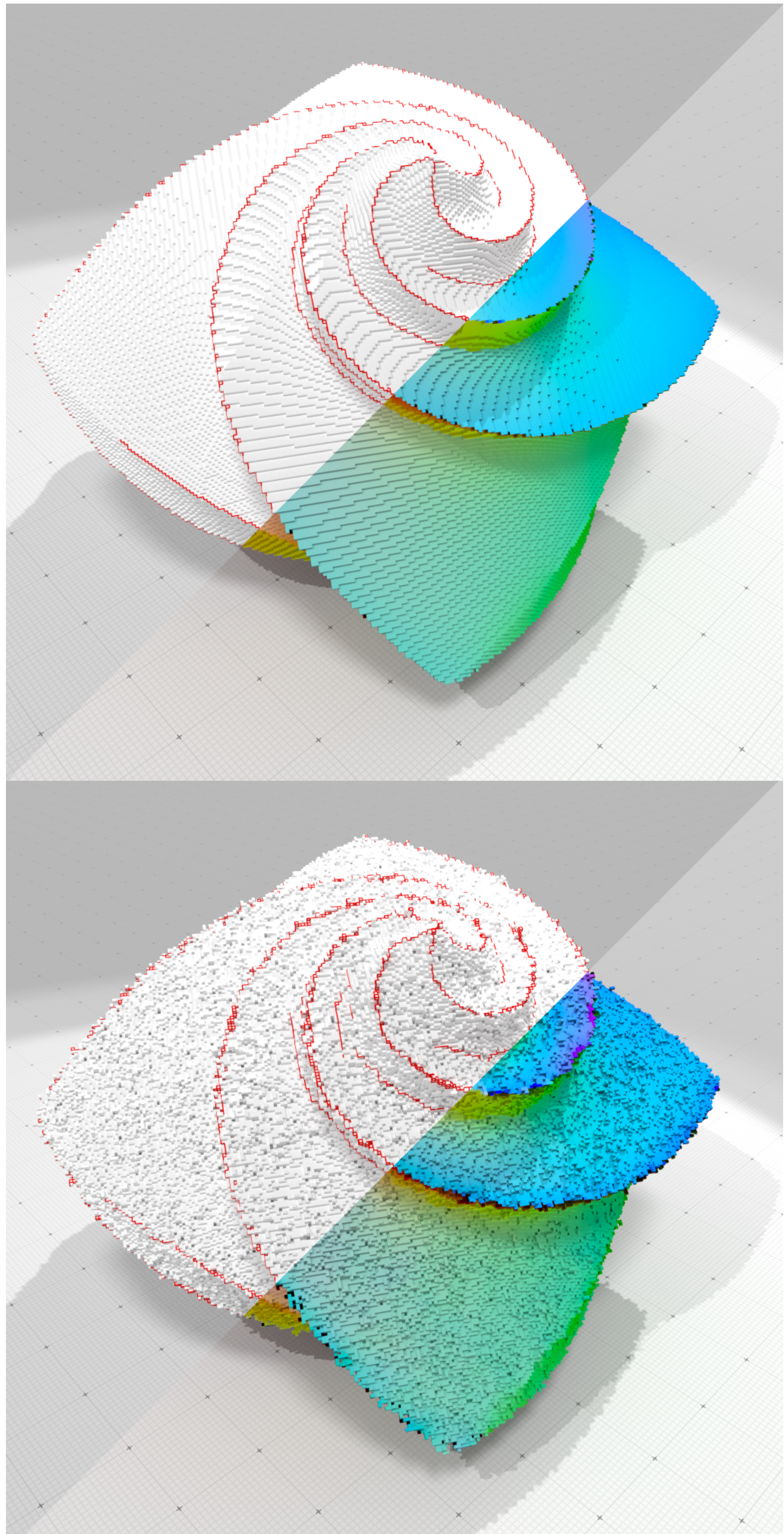


Piecewise smooth reconstruction

***Step1**: normal vector field reconstruction*

Ambrosio-Tortorelli functional: solve u,v s.t.

$$AT_\epsilon(u,v) = \alpha \int_M |u-g|^2 dx + \int_M |v\nabla u|^2 + \lambda\epsilon|\nabla v|^2 + \frac{1}{4\epsilon}|1-v|^2 dx$$



[C. et al 16]

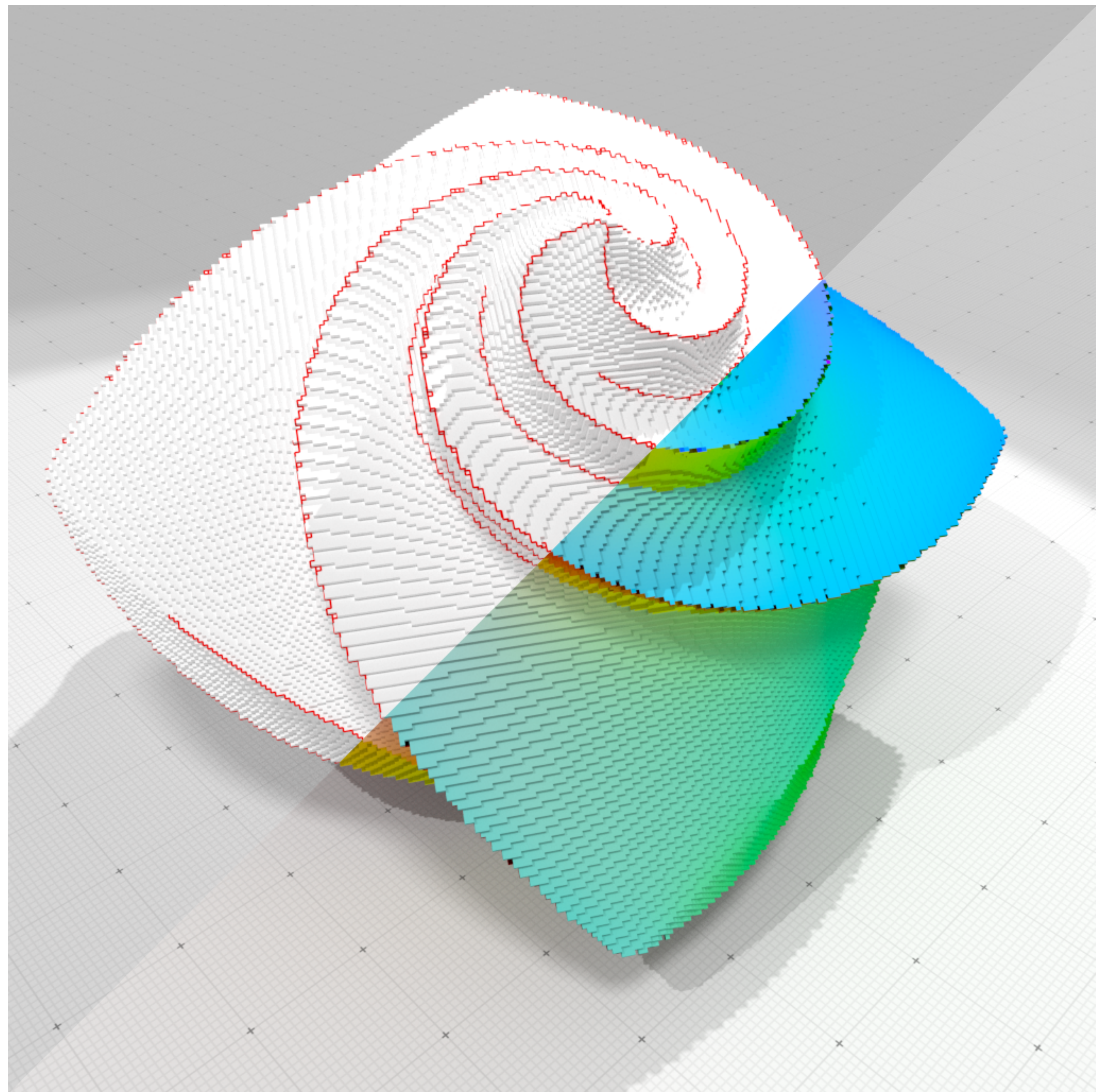
Piecewise smooth reconstruction

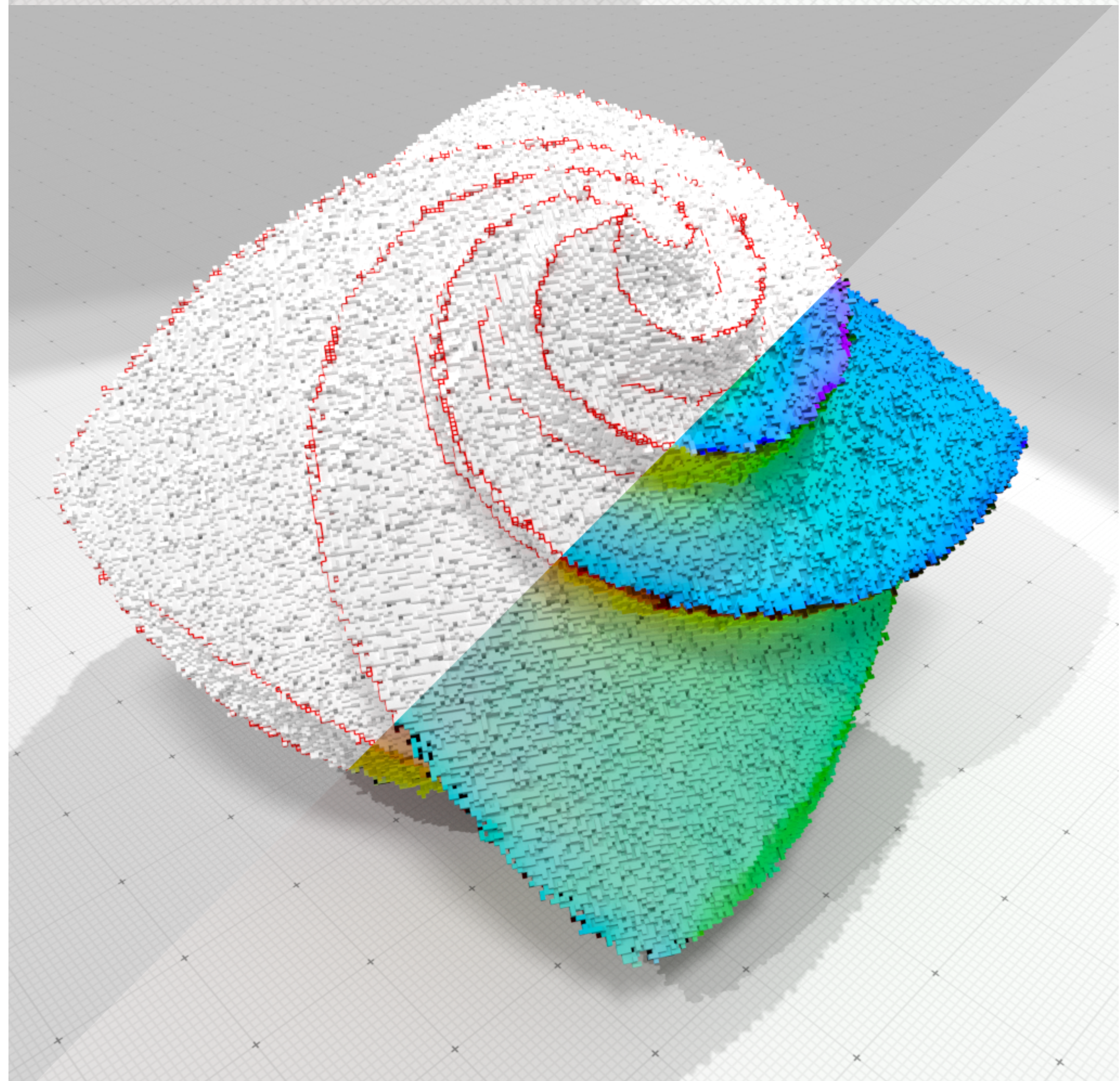
***Step1**: normal vector field reconstruction*

Ambrosio-Tortorelli functional: solve u,v s.t.

$$AT_\epsilon(u,v) = \alpha \int_M |u - g|^2 dx + \int_M |v \nabla u|^2 + \lambda\epsilon |\nabla v|^2 + \frac{1}{4\epsilon} |1 - v|^2 dx$$

Reconstructed normals are close to the input ones





[C. et al 16]

Piecewise smooth reconstruction

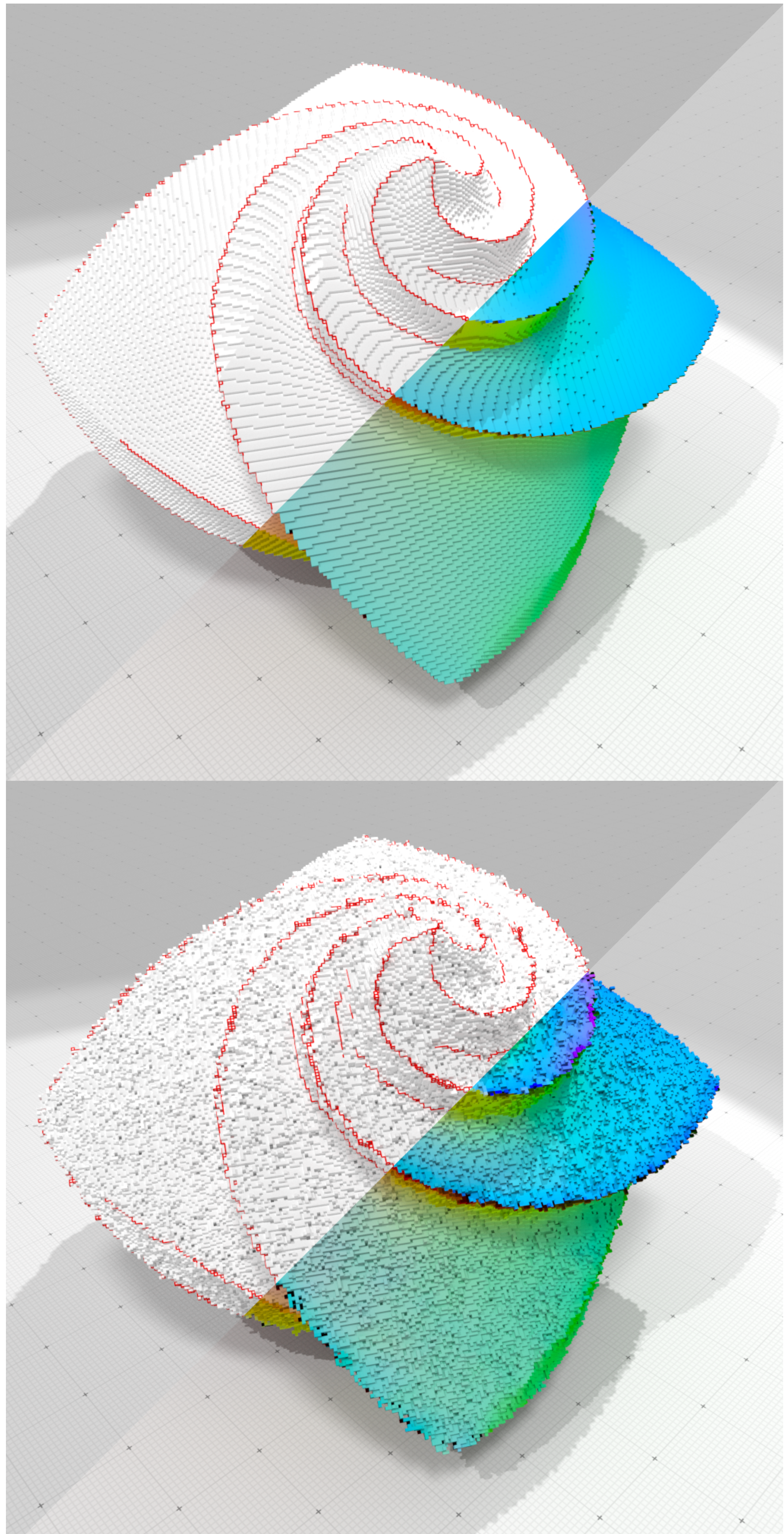
***Step1**: normal vector field reconstruction*

Ambrosio-Tortorelli functional: solve u,v s.t.

$$AT_\epsilon(u,v) = \alpha \int_M |u-g|^2 dx + \int_M |v \nabla u|^2 + \lambda\epsilon |\nabla v|^2 + \frac{1}{4\epsilon}|1-v|^2 dx$$

Reconstructed normals are close to the input ones

Normal field must be smooth except at singularities v



[C. et al 16]

Piecewise smooth reconstruction

***Step1**: normal vector field reconstruction*

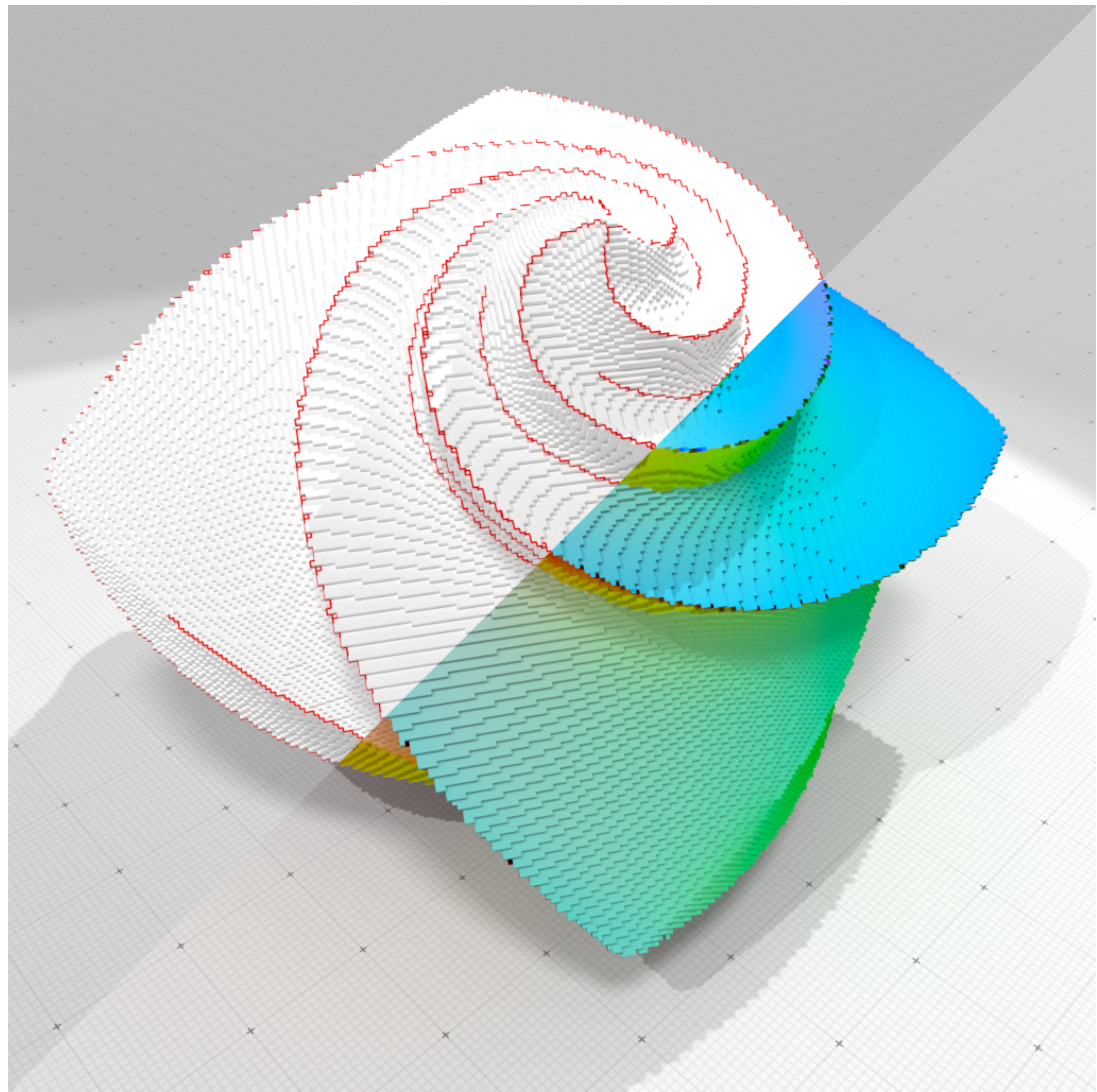
Ambrosio-Tortorelli functional: solve u,v s.t.

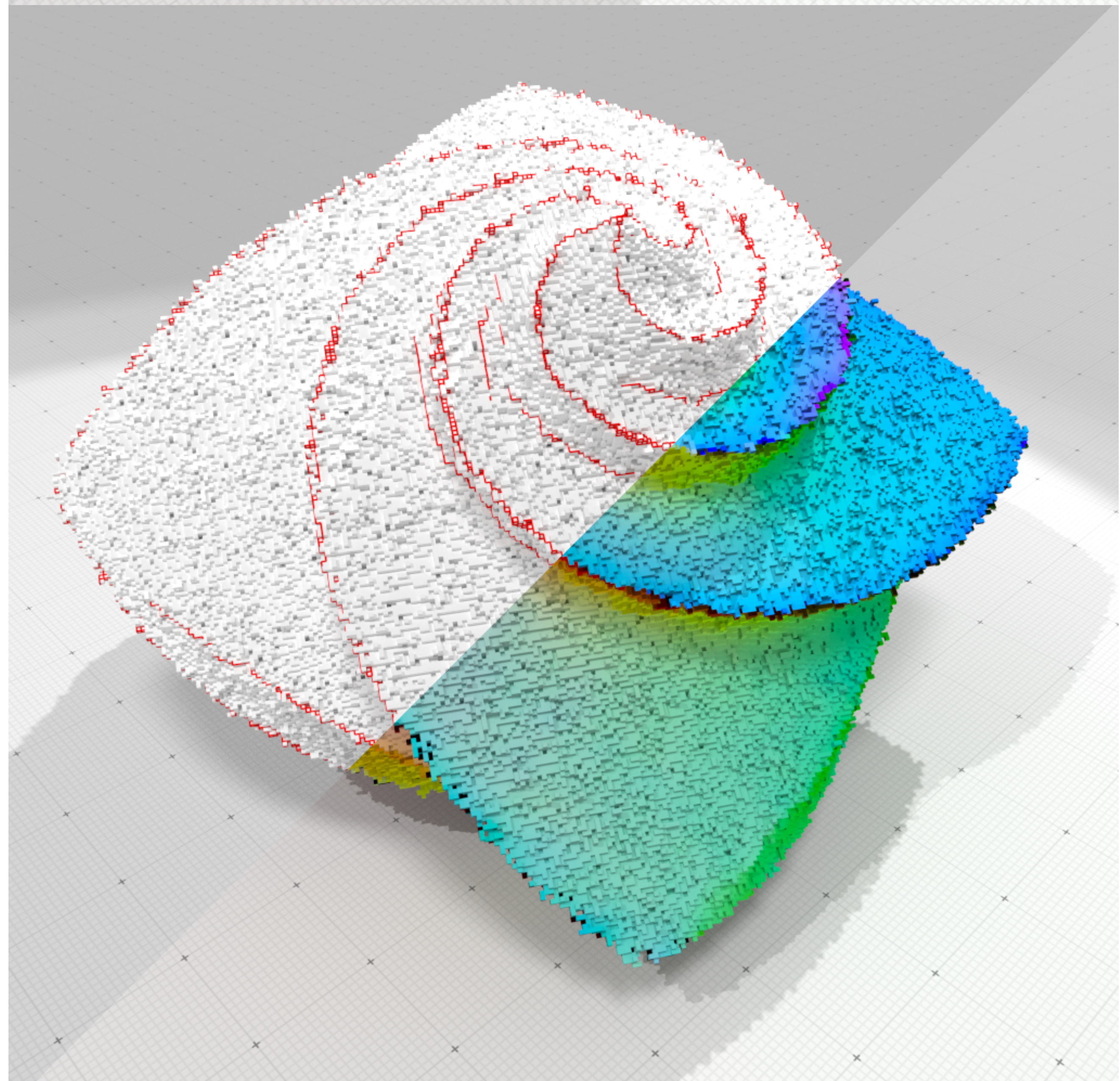
$$AT_\epsilon(u,v) = \alpha \int_M |u-g|^2 dx + \int_M |v\nabla u|^2 + \lambda\epsilon|\nabla v|^2 + \frac{1}{4\epsilon}|1-v|^2 dx$$

Reconstructed normals are close to the input ones

Normal field must be smooth except at singularities v

Penalizes the *length* of singularities





[C. et al 16]

Piecewise smooth reconstruction

***Step1**: normal vector field reconstruction*

Ambrosio-Tortorelli functional: solve u,v s.t.

$$AT_\epsilon(u,v) = \alpha \int_M |u-g|^2 dx + \int_M |v \nabla u|^2 + \lambda\epsilon |\nabla v|^2 + \frac{1}{4\epsilon}|1-v|^2 dx$$

Reconstructed normals are close to the input ones

Normal field must be smooth except at singularities v

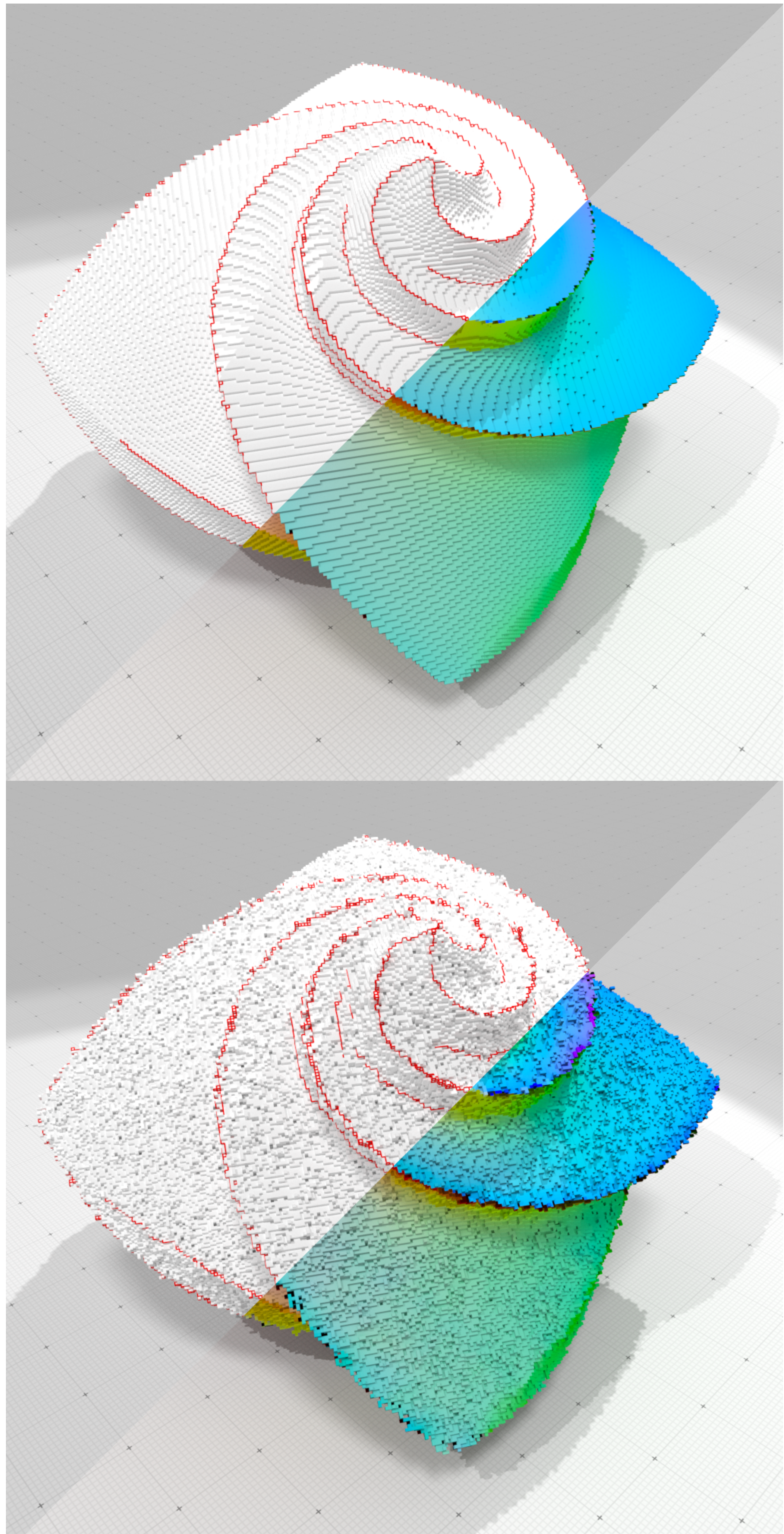
Penalizes the *length* of singularities

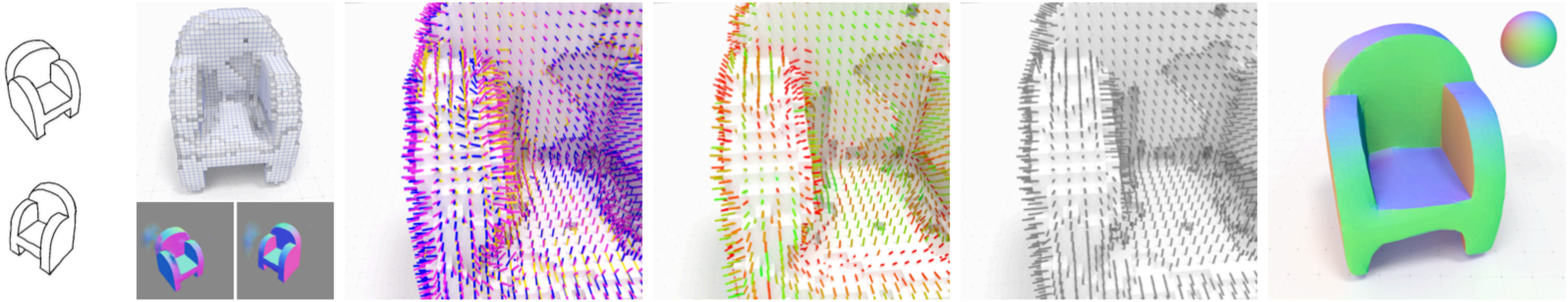
digital DEC:

$$AT_\epsilon(u,v) = \alpha \sum_{i=1}^{3} \langle u_i - g_i, u_i - g_i \rangle_{\bar{0}} + \sum_{i=1}^{3} \langle v \wedge d_{\bar{0}} u_i, v \wedge d_{\bar{0}} u_{i\bar{1}} \rangle_{\bar{1}} + \lambda\epsilon \langle d_0 v, d_0 v \rangle_1 + \frac{\lambda}{4\epsilon} \langle 1-v, 1-v \rangle_0$$

+ energy is convex for fixed u or v $\Rightarrow$ alternate minimization

[C. et al 16]





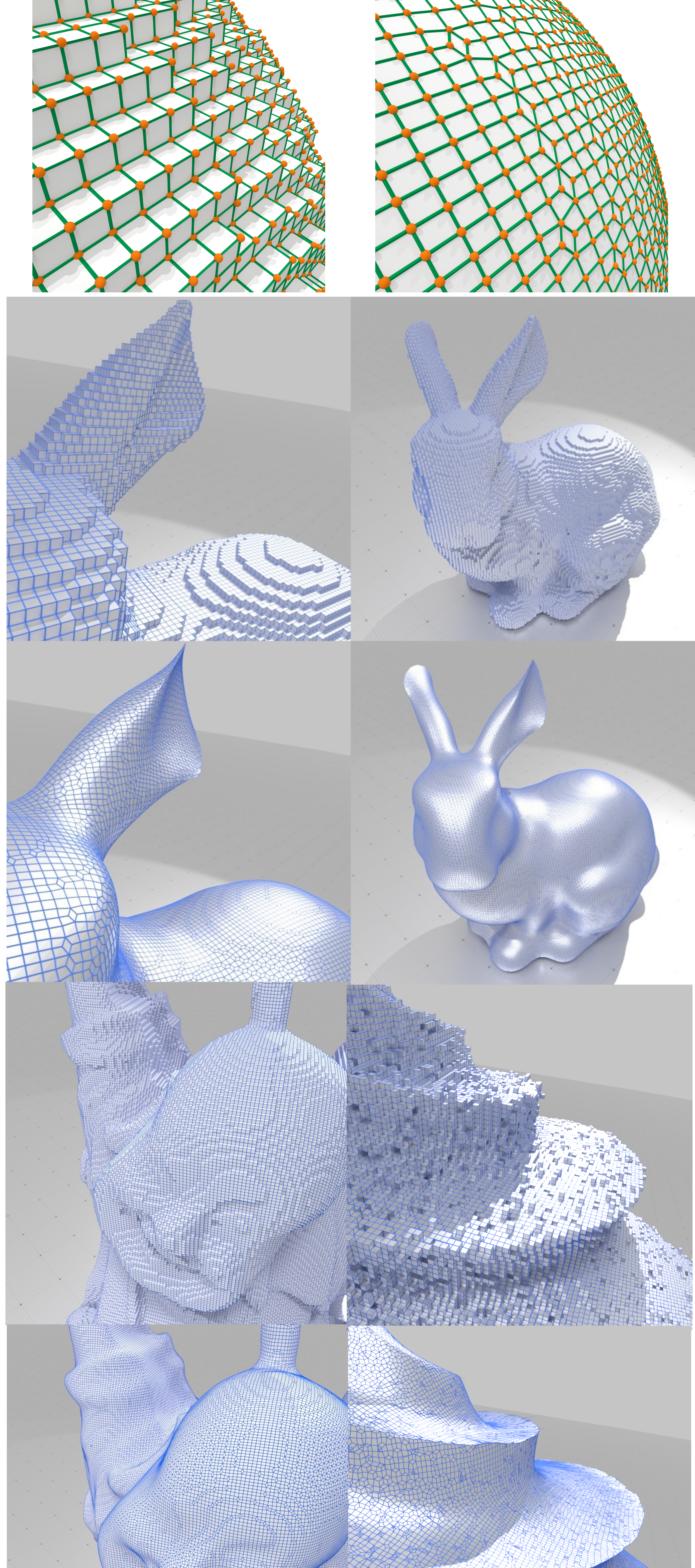
(a) Input (b) CNNs predictions (c) Candidate normals (d) Aggregated normals (e) Piecewise-smooth normals (f) Final surface

[Delanoy et al 19]

Piecewise smooth reconstruction

Step 2: surface reconstruction

$$\mathscr{E}(\hat{P}) := \alpha \sum_{i=1}^{n} \|\mathbf{p}_i - \hat{\mathbf{p}}_i\|^2 + \beta \sum_{f \in F} \sum_{\hat{\mathbf{e}}_j \in \partial f} (\hat{\mathbf{e}}_j \cdot \mathbf{n}_f)^2 + \gamma \sum_{i=1}^{n} \|\hat{\mathbf{p}}_i - \hat{\mathbf{b}}_i\|^2$$

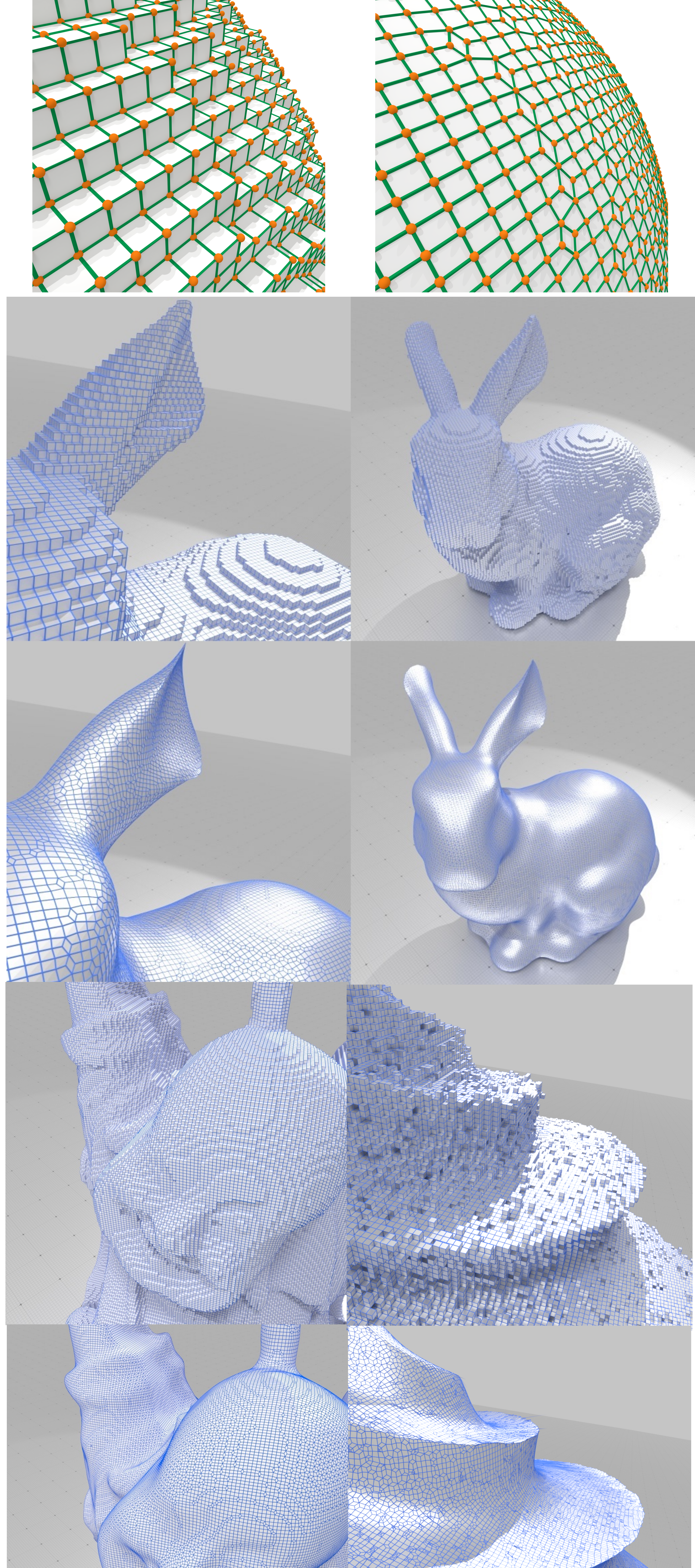


Piecewise smooth reconstruction

Step 2: *surface reconstruction*

$$\mathcal{E}(\hat{P}) := \alpha \sum_{i=1}^{n} \|\mathbf{p}_i - \hat{\mathbf{p}}_i\|^2 + \beta \sum_{f\in F} \sum_{\hat{\mathbf{e}}_j\in\partial f} (\hat{\mathbf{e}}_j \cdot \mathbf{n}_f)^2 + \gamma \sum_{i=1}^{n} \|\hat{\mathbf{p}}_i - \hat{\mathbf{b}}_i\|^2$$

optimized vertices are not too far from original ones



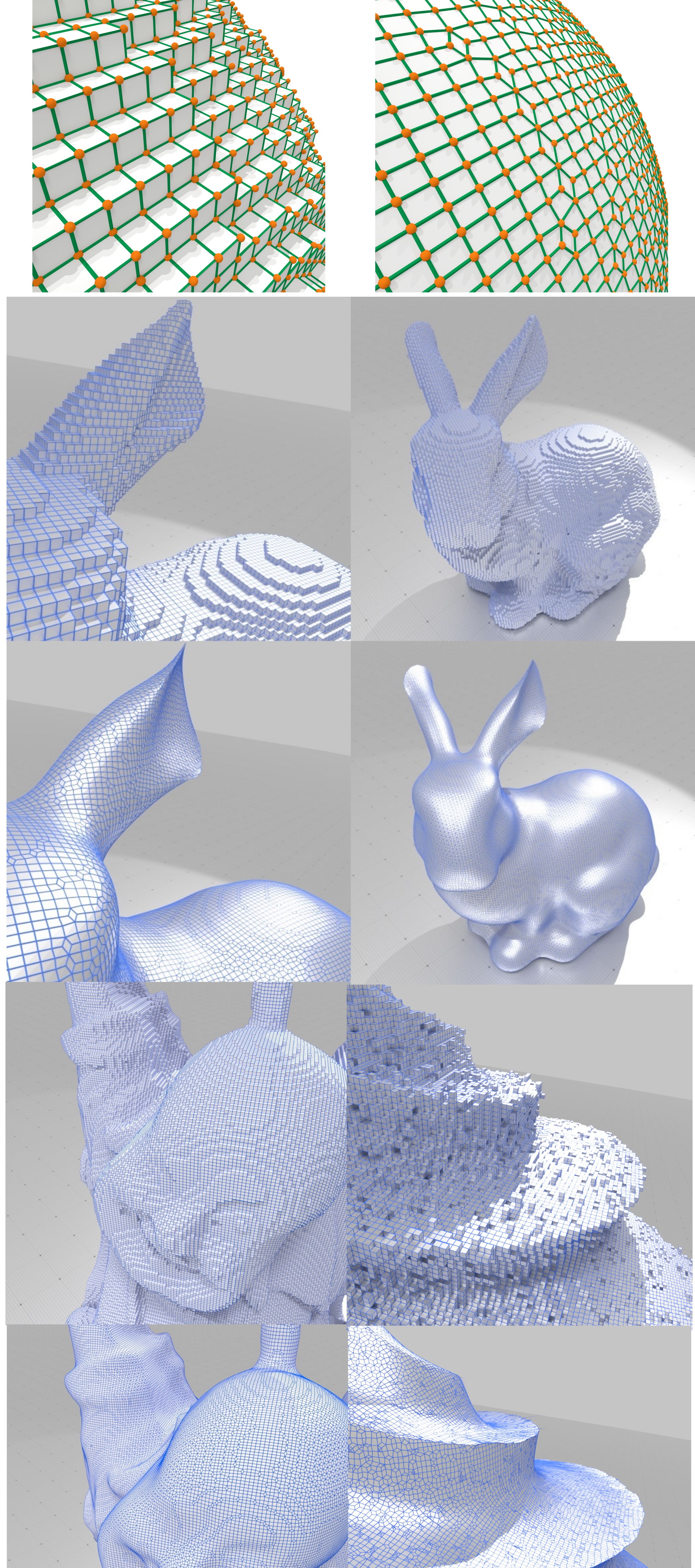
Piecewise smooth reconstruction

Step 2: *surface reconstruction*

$$\mathcal{E}(\hat{P}) := \alpha \sum_{i=1}^{n} \|\mathbf{p}_i - \hat{\mathbf{p}}_i\|^2 + \beta \sum_{f\in F} \sum_{\hat{\mathbf{e}}_j \in \partial f} (\hat{\mathbf{e}}_j \cdot \mathbf{n}_f)^2 + \gamma \sum_{i=1}^{n} \|\hat{\mathbf{p}}_i - \hat{\mathbf{b}}_i\|^2$$

optimized vertices are not too far from original ones

Edges must be as orthogonal as possible to the given normal vectors



Piecewise smooth reconstruction

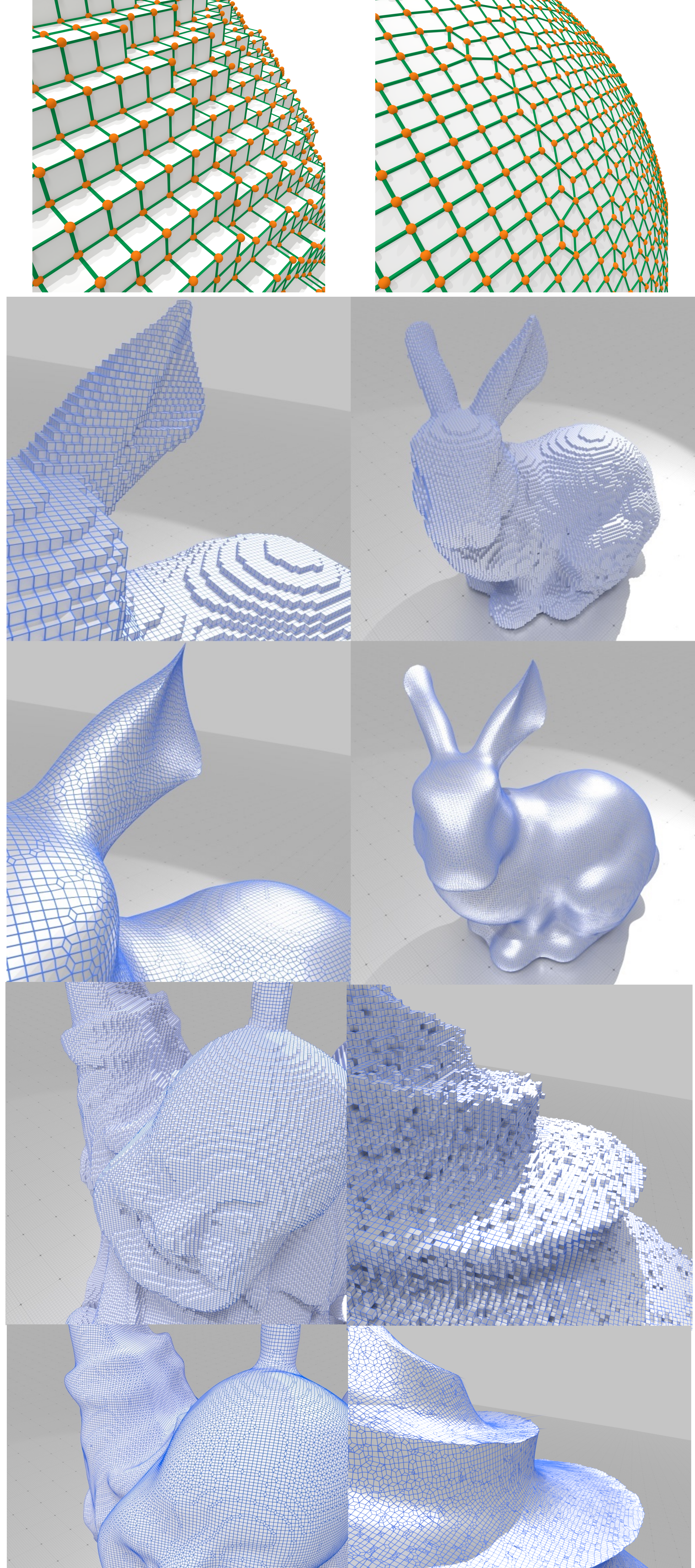
***Step 2:** surface reconstruction*

$$\mathscr{E}(\hat{P}) := \alpha \sum_{i=1}^{n} \|\mathbf{p}_i - \hat{\mathbf{p}}_i\|^2 + \beta \sum_{f \in F} \sum_{\hat{\mathbf{e}}_j \in \partial f} (\hat{\mathbf{e}}_j \cdot \mathbf{n}_f)^2 + \gamma \sum_{i=1}^{n} \|\hat{\mathbf{p}}_i - \hat{\mathbf{b}}_i\|^2$$

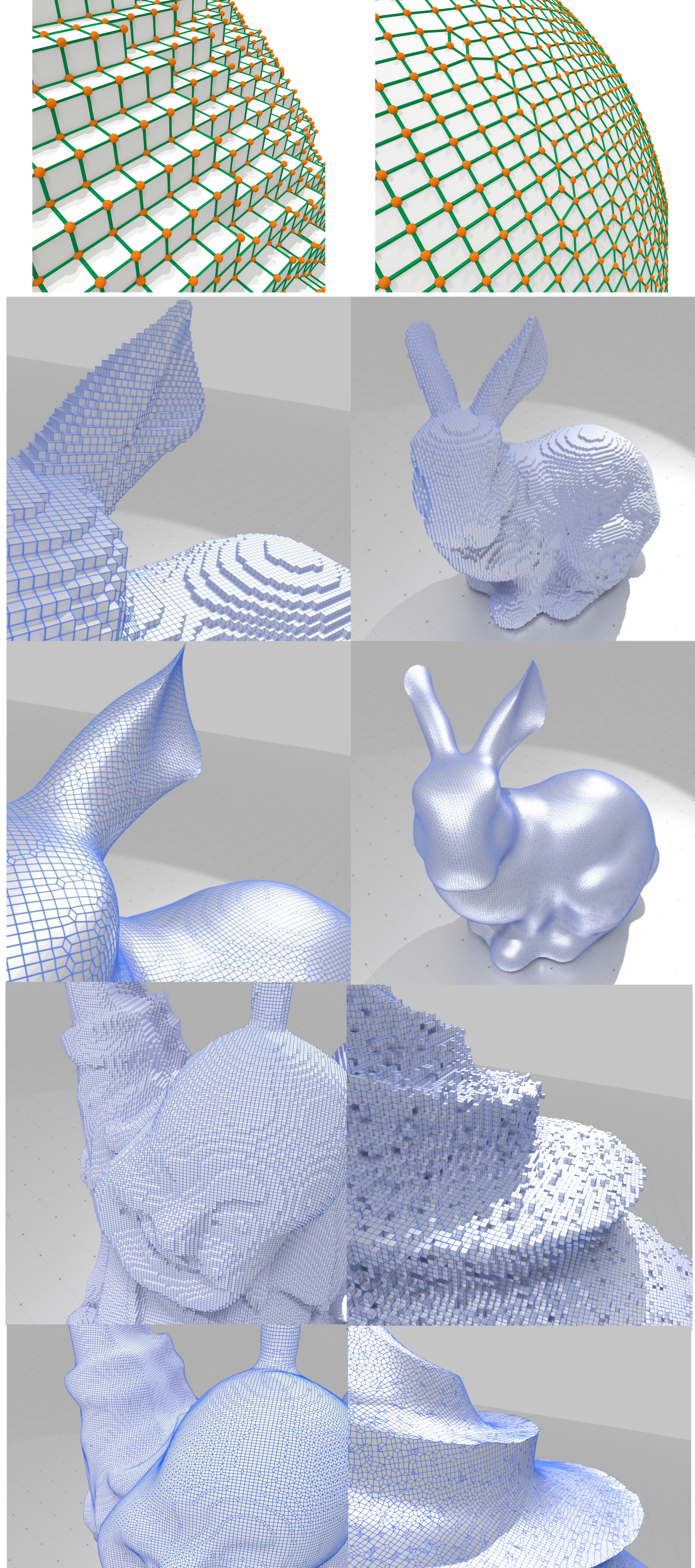
optimized vertices are not too far from original ones

Edges must be as orthogonal as possible to the given normal vectors

Fairness term



Piecewise smooth reconstruction



Step 2: *surface reconstruction*

$$\mathcal{E}(\hat{P}) := \alpha \sum_{i=1}^{n} \|\mathbf{p}_i - \hat{\mathbf{p}}_i\|^2 + \beta \sum_{f \in F} \sum_{\hat{\mathbf{e}}_j \in \partial f} (\hat{\mathbf{e}}_j \cdot \mathbf{n}_f)^2 + \gamma \sum_{i=1}^{n} \|\hat{\mathbf{p}}_i - \hat{\mathbf{b}}_i\|^2$$

optimized vertices are not too far from original ones

Edges must be as orthogonal as possible to the given normal vectors

Fairness term

Using multigrid convergent normal vector field or its piecewise smooth regularization:

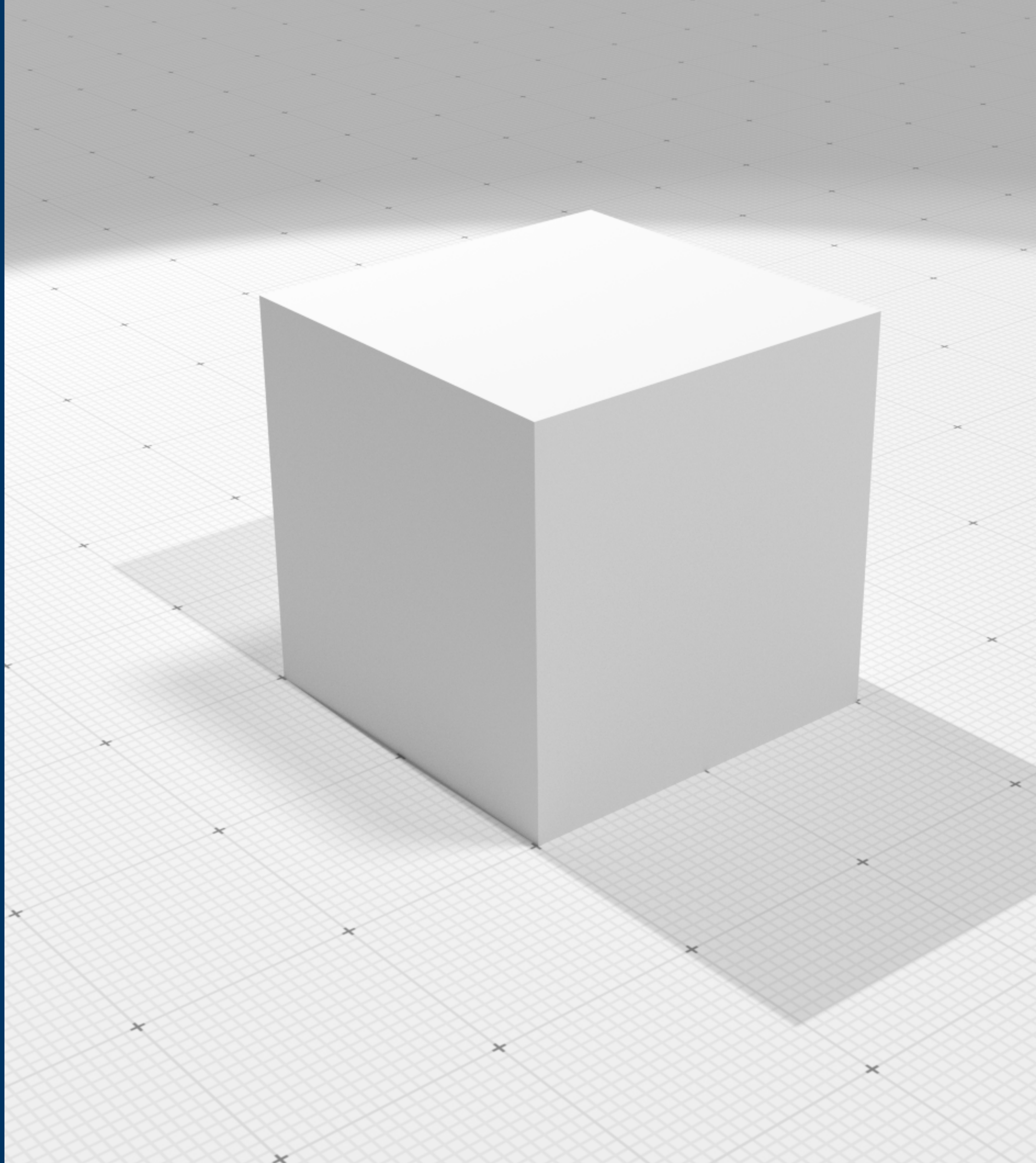
$$\frac{1}{n} \sum_{i=1}^{n} \|\mathbf{p}_i^* - \mathbf{p}_i\| \leq C \cdot h$$

$$\frac{1}{n} \sum_{i=1}^{n} d(\mathbf{p}_i^*, \partial M) \leq C' \cdot h$$

+ topological guarantee
+ multi-label case
+ fast GPU based minimization
+ …

[C. et al 21]

conclusion



Conclusion

Topology and geometry processing on regular data:

- fast algorithms thanks to the regularity of the data
- simple topological structure
- integer based computations
- advanced surface based geometry processing

… in $\mathbb{Z}^d$

DGtal

dgtal.org

https://github.com/dcoeurjo/SGP-GraduateSchool-digitalgeometry

(slides + code)

Challenges

- Corrected digital calculus, what kind of guarantee can we get?
- DEC operators targeting the limit surface (à-la *Subdivision Exterior Calculus*)
- Localized geometry processing operators on DAG Sparse Voxel Octrees

dgtal.org

https://github.com/dcoeurjo/SGP-GraduateSchool-digitalgeometry

(slides + code)

References

[Villanueva et al 17] Alberto Jaspe Villanueva, Fabio Marton, and Enrico Gobbetti, Symmetry-aware Sparse Voxel DAGs (SSVDAGs) for compression-domain tracing of high-resolution geometric scenes, Journal of Computer Graphics Techniques (JCGT), vol. 6, no. 2, 1-30, 2017

[Chen et al 2020] Half-Space Power Diagrams and Discrete Surface Offsets, Zhen Chen, Daniele Panozzo, Jérémie Dumas. In TVCG, 2019.

[C. et al 07] Optimal Separable Algorithms to Compute the Reverse Euclidean Distance Transformation and Discrete Medial Axis in Arbitrary Dimension, David Coeurjolly Annick Montanvert, IEEE Transactions on Pattern Analysis and Machine Intelligence, March 2007

[Martinez et al 20] Orthotropic k-nearest Foams for Additive Manufacturing, Jonàs Martínez, Haichuan Song, Jérémie Dumas, Sylvain Lefebvre, ACM TOG 2017

[Liu et al 18] Narrow-band topology optimization on a sparsely populated grid. Liu, H., Hu, Y., Zhu, B., Matusik, W., & Sifakis, E. (2018). ACM Transactions on Graphics (TOG), 37(6), 1-14.

[de Goes et al 20] Discrete Differential Operators on Polygonal Meshes, de Goes, Butts, Desbrun SIGGRAPH / ACM Transactions on Graphics (2020)

[C. et al 21] Digital surface regularization with guarantees, David Coeurjolly, Jacques-Olivier Lachaud, Pierre Gueth, IEEE Transactions on Visualization and Computer Graphics, January 2021

[C. et al 16] Piecewise smooth reconstruction of normal vector field on digital data, David Coeurjolly, Marion Foare, Pierre Gueth, Computer Graphics Forum (Proceedin Pacific Graphics), September 2016

[Caissard et al 19] Laplace–Beltrami Operator on Digital Surfaces, Thomas Caissard, David Coeurjolly, Jacques-Olivier Lachaud, Tristan Roussillon, Journal of Mathema Imaging and Vision, January 2019

[Delanoy et al 19] Combining voxel and normal predictions for multi-view 3D sketching, Johanna Delanoy, David Coeurjolly, Jacques-Olivier Lachaud, Adrien Bousseau Computers and Graphics, June 2019

[Belkin et al 08] Belkin, M., Sun, J., Wang, Y.: Discrete laplace operator on meshed surfaces. In: M. Teillaud (ed.) Proceedings of the 24th ACM Symposium on Computational Geometry, College Park, MD, USA, June 9-11, 2008, pp. 278–287. ACM (2008)

References

[Bertrand94] Bertrand, Gilles. "Simple points, topological numbers and geodesic neighborhoods in cubic grids." *Pattern recognition letters* 15.10 (1994): 1003-1011.

[BC94] Bertrand, Gilles, and Michel Couprie. "On parallel thinning algorithms: minimal non-simple sets, P-simple points and critical kernels." *Journal of Mathematical Imaging and Vision* 35.1 (2009): 23-35.

[YLJ18] Yan, Yajie, David Letscher, and Tao Ju. "Voxel cores: Efficient, robust, and provably good approximation of 3d medial axes." *ACM Transactions on Graphics (TO* 37.4 (2018): 1-13.

[LT16] Lachaud, Jacques-Olivier, and Boris Thibert. "Properties of gauss digitized shapes and digital surface integration." *Journal of Mathematical Imaging and Vision* 54 (2016): 162-180.

[LTC17] Lachaud, Jacques-Olivier, David Coeurjolly, and Jérémy Levallois. "Robust and convergent curvature and normal estimators with digital integral invariants." *Moc Approaches to Discrete Curvature*. Springer, Cham, 2017. 293-348.

[LRTC20] Lachaud, Jacques-Olivier, Pascal Romon, Boris Thibert, and David Coeurjolly. "Interpolated corrected curvature measures for polygonal surfaces." *Computer Graphics Forum*. Vol. 39. No. 5. 2020.