

# Black-Box Analysis

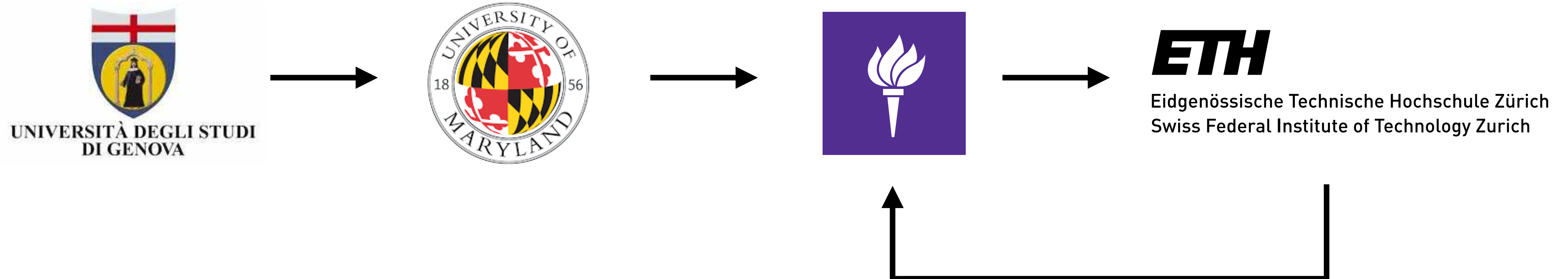
Daniele Panozzo

<https://cims.nyu.edu/gcl/>



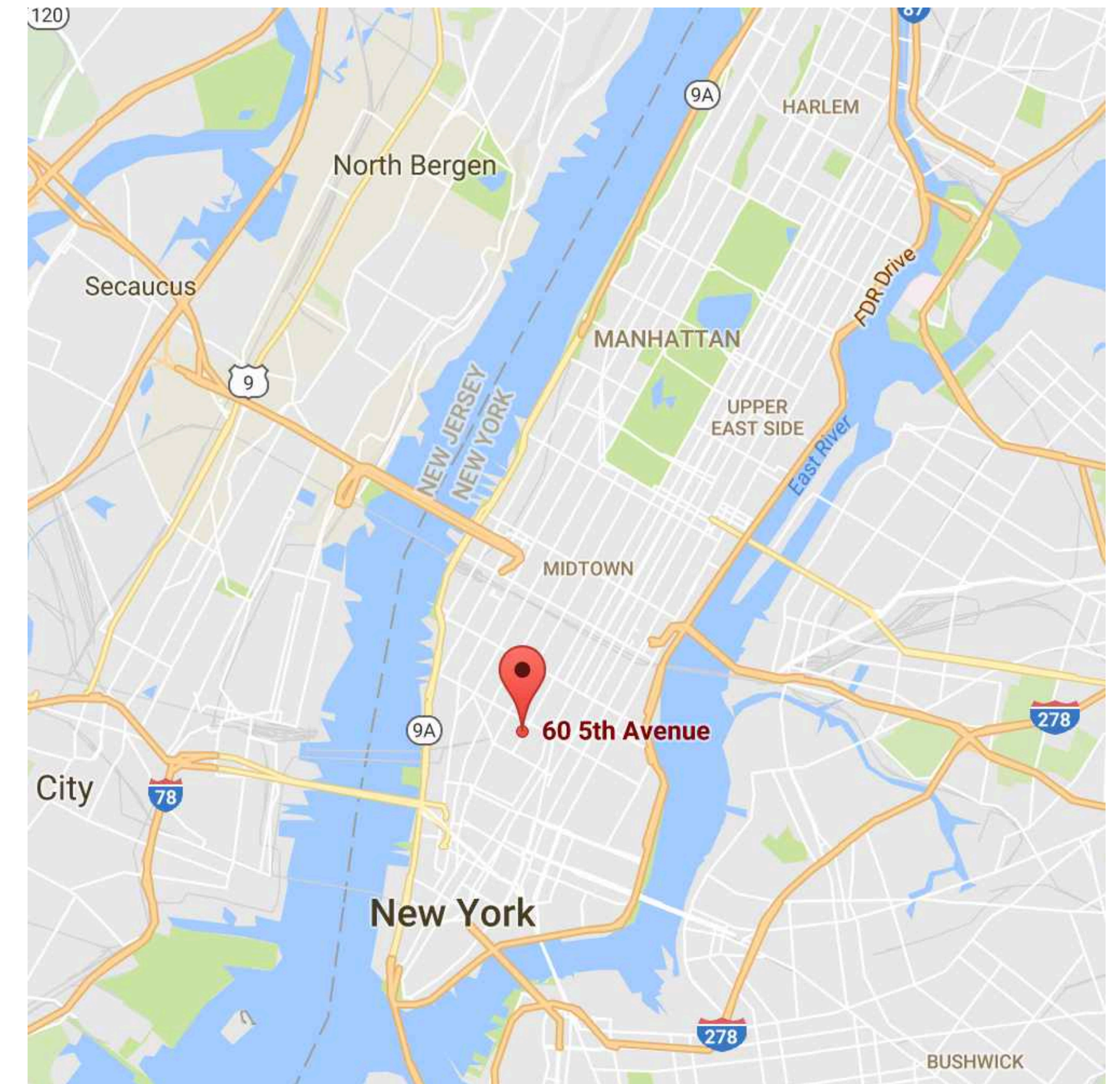
# Who Am I?

- Assistant Professor of Computer Science at New York University





# Courant Institute Of Mathematical Sciences





# Geometric Computing Lab @ NYU

## Faculty

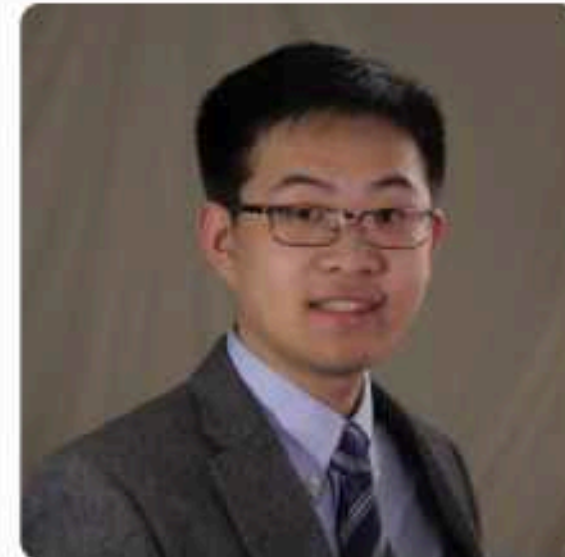


Daniele Panozzo



Denis Zorin

## PhD Students



Zhongshi Jiang



Yixin Hu



Hanxiao Shen

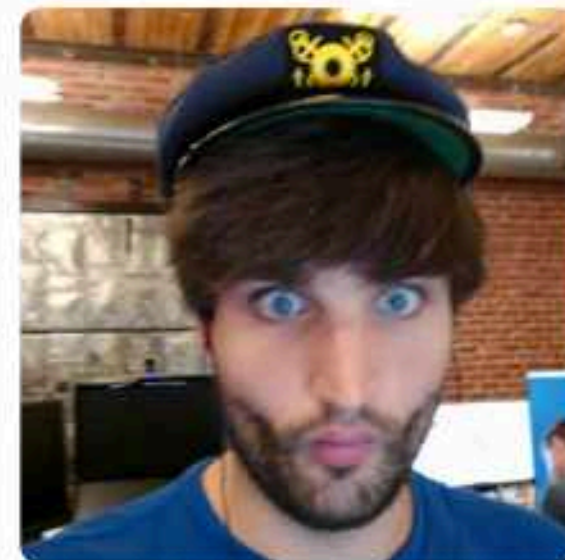


Libin Liu

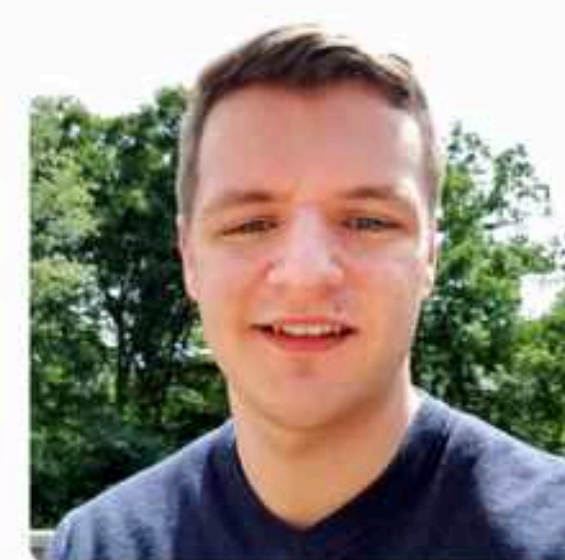
## Postdoctoral Researchers



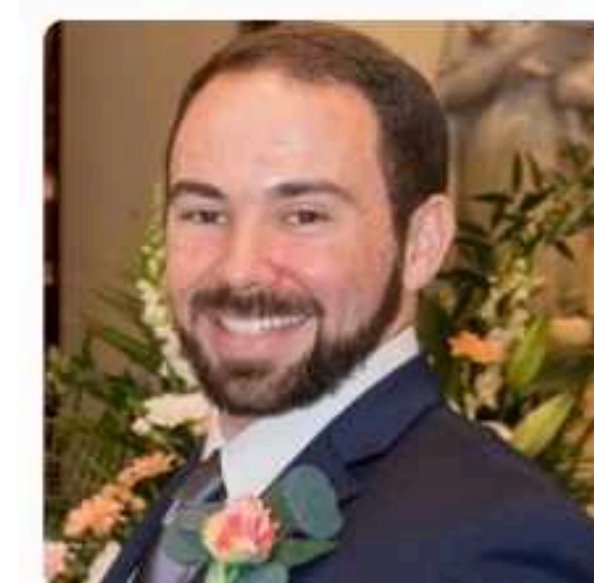
Teseo Schneider



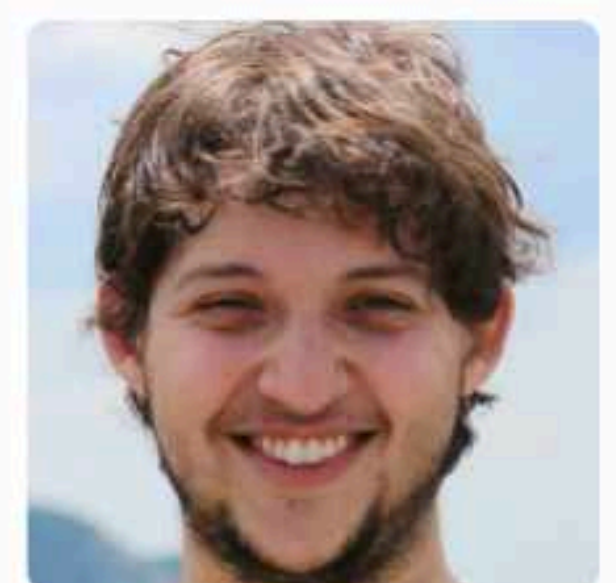
Francis Williams



Zachary Ferguson



Matt Morse



Davi Colli Tozoni



Francisca Gil Ureta



Chelsea Tymms

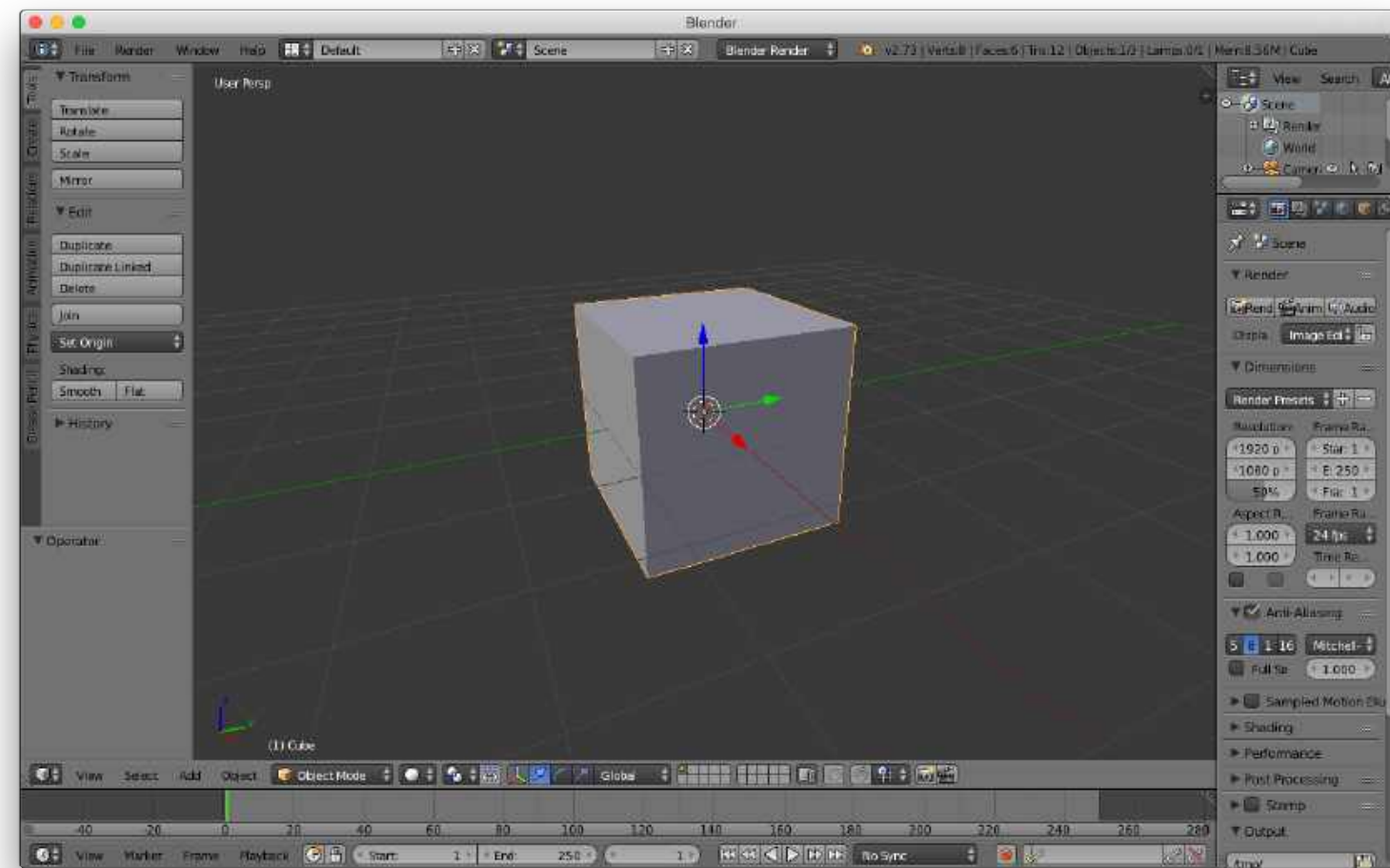
<https://cims.nyu.edu/gcl/>



# Research Overview



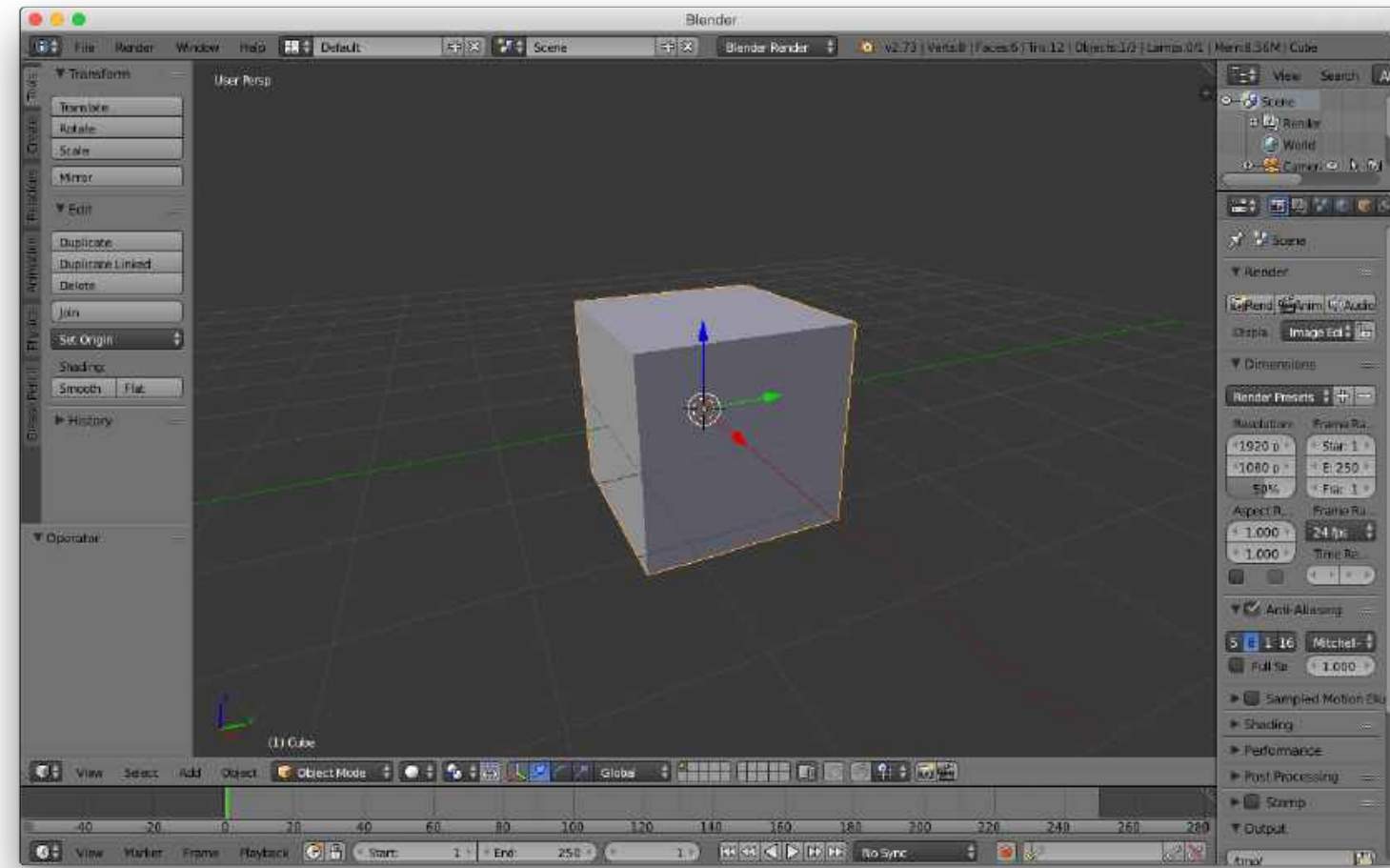
# Research Overview



Blender



# Research Overview



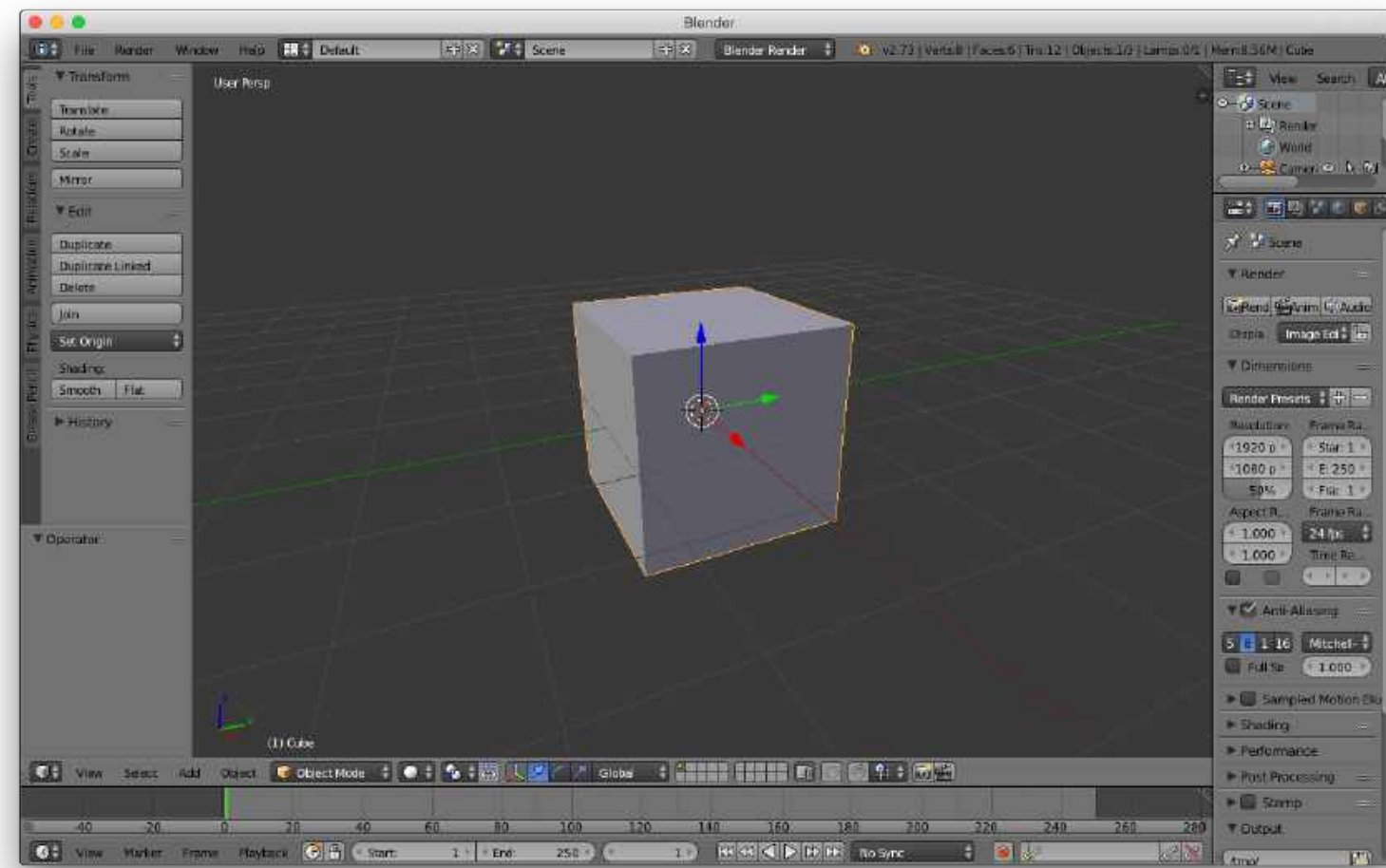
Blender



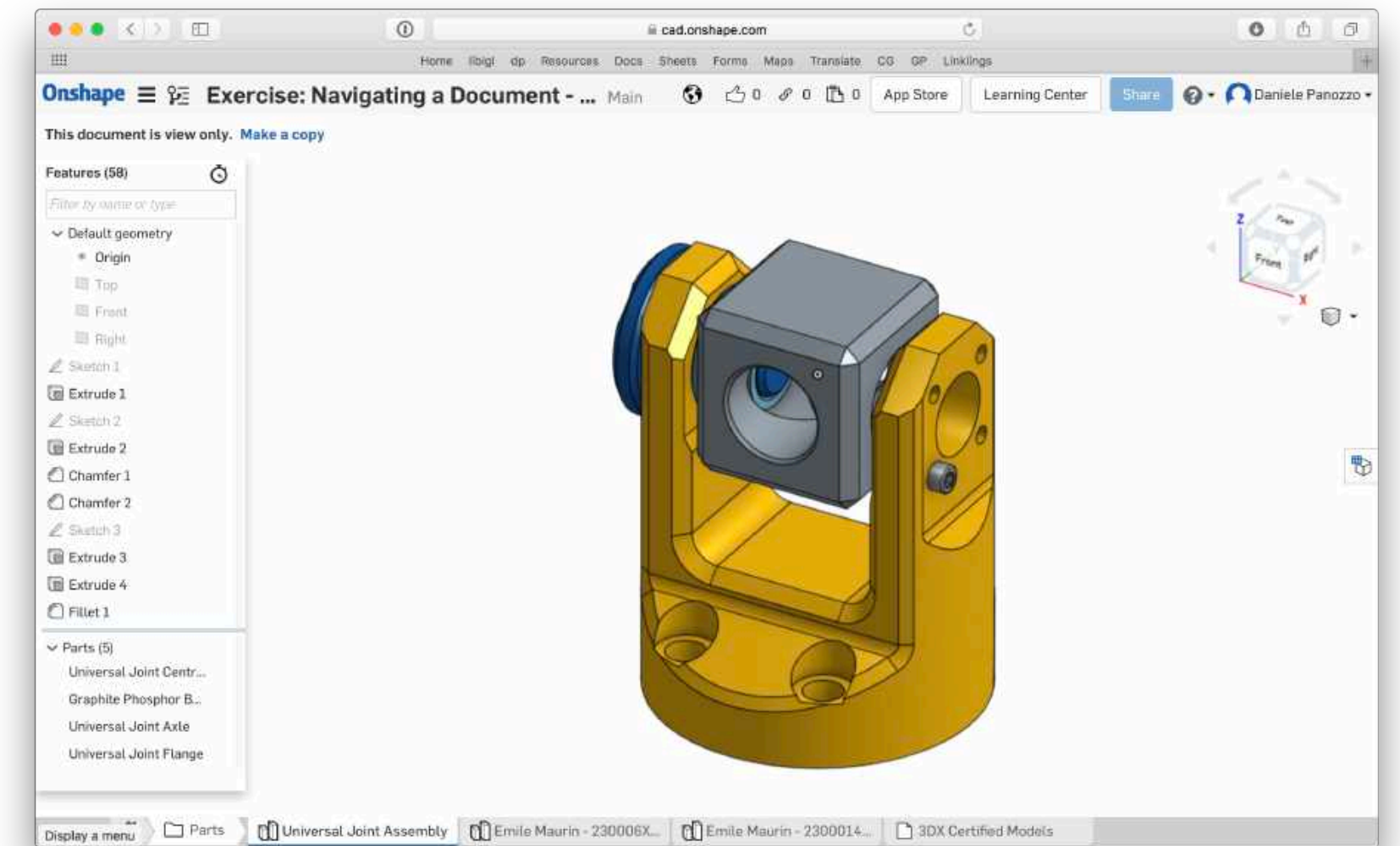
Maya



# Research Overview



Blender



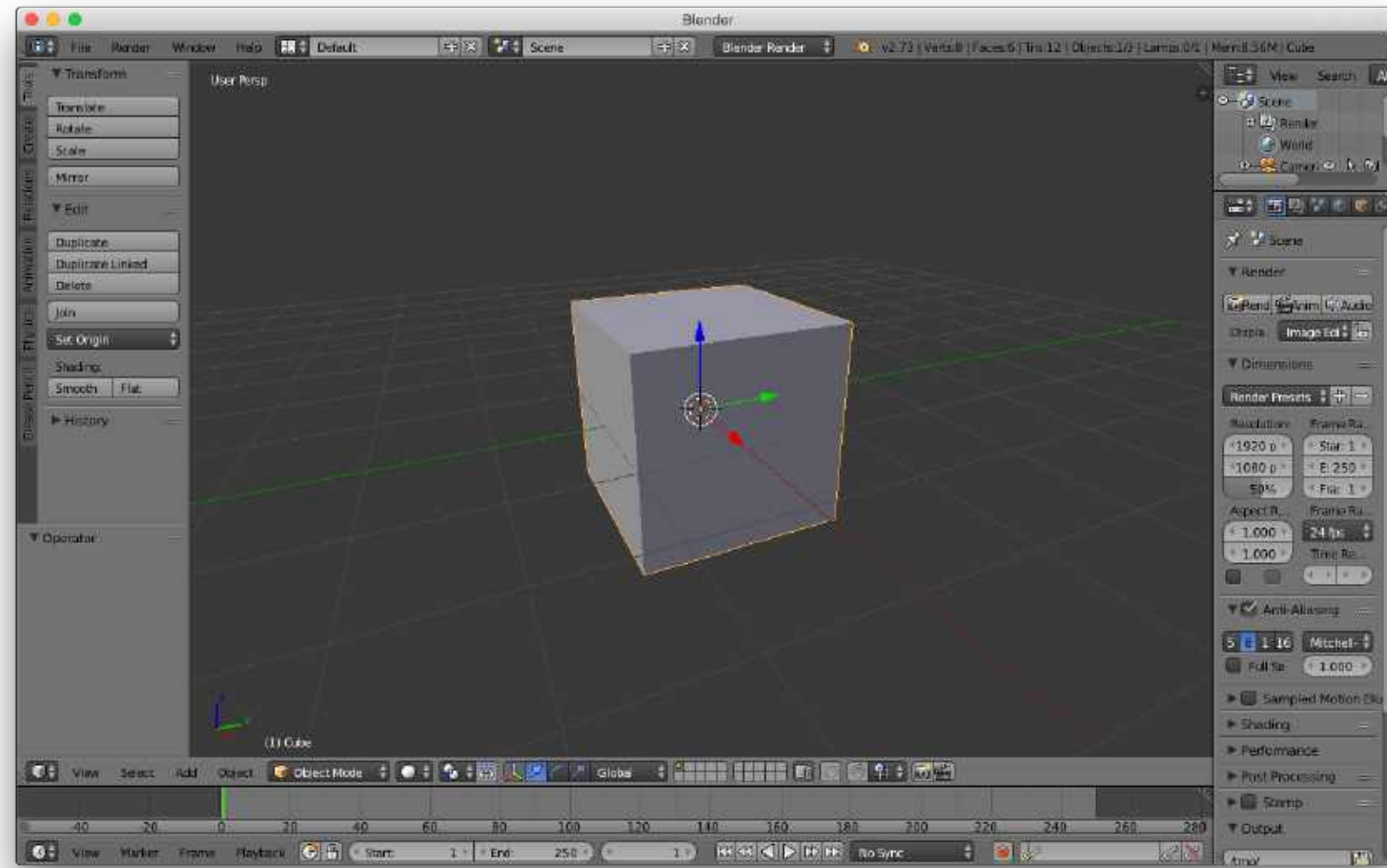
OnShape



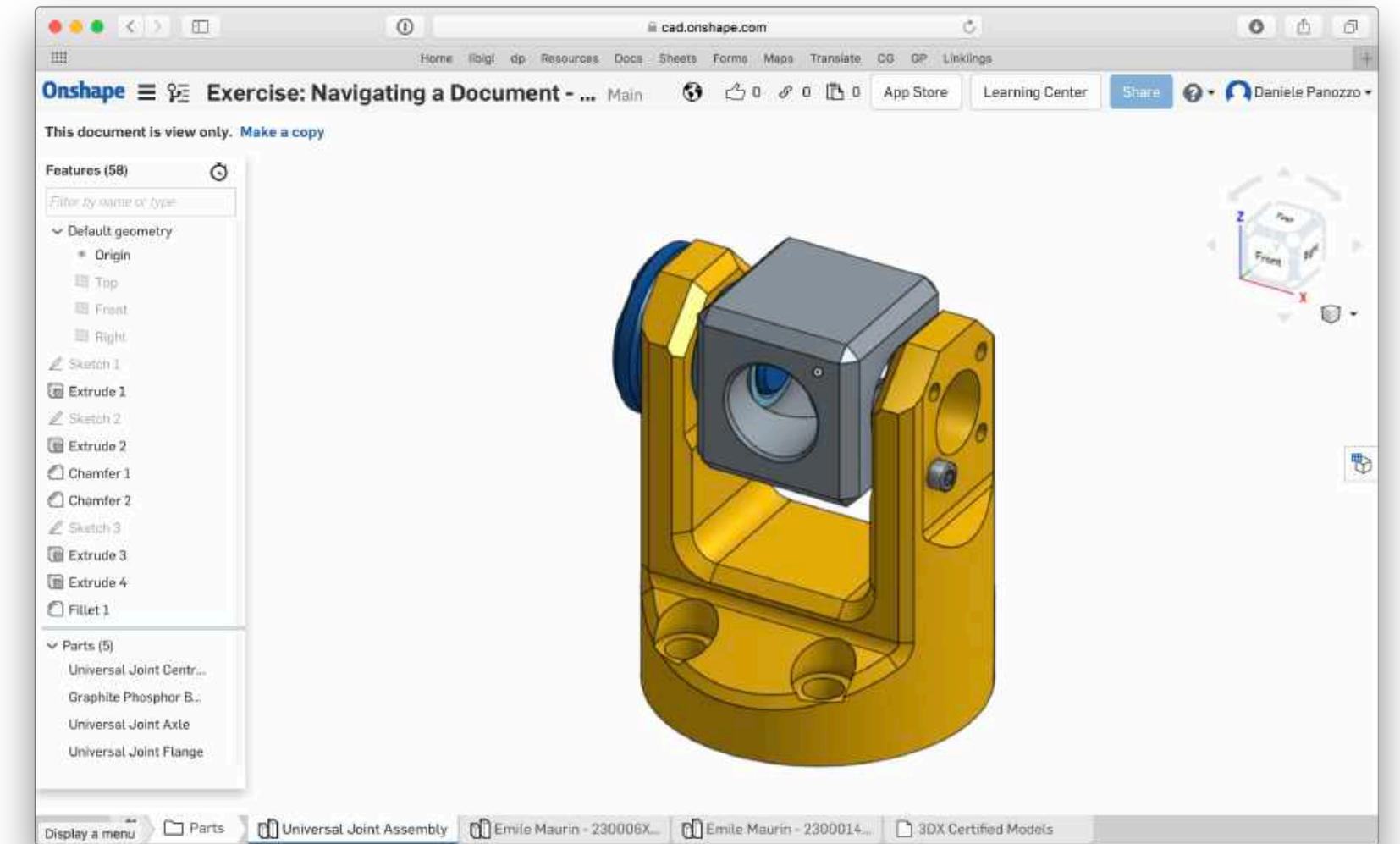
Maya



# Research Overview



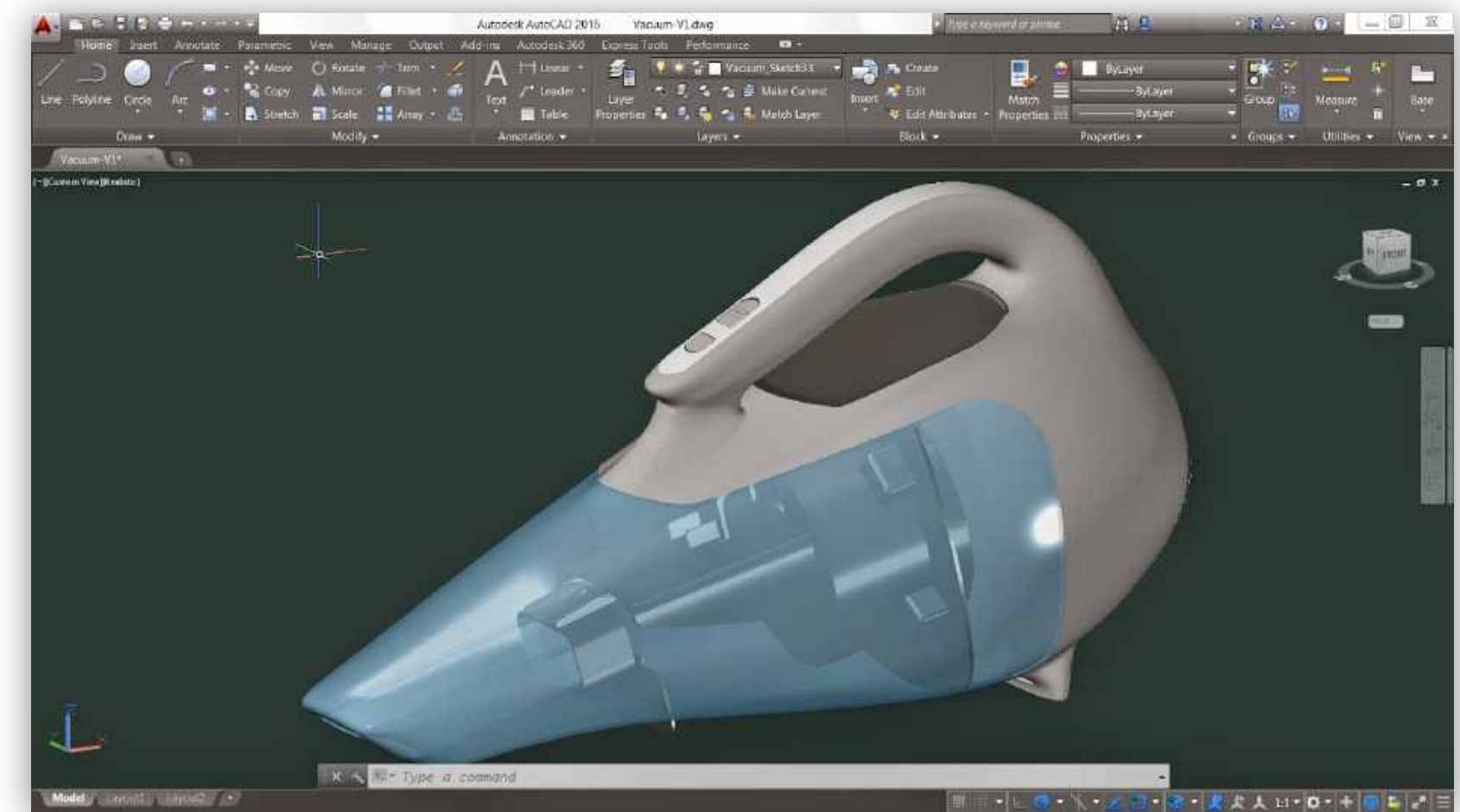
Blender



OnShape



Maya

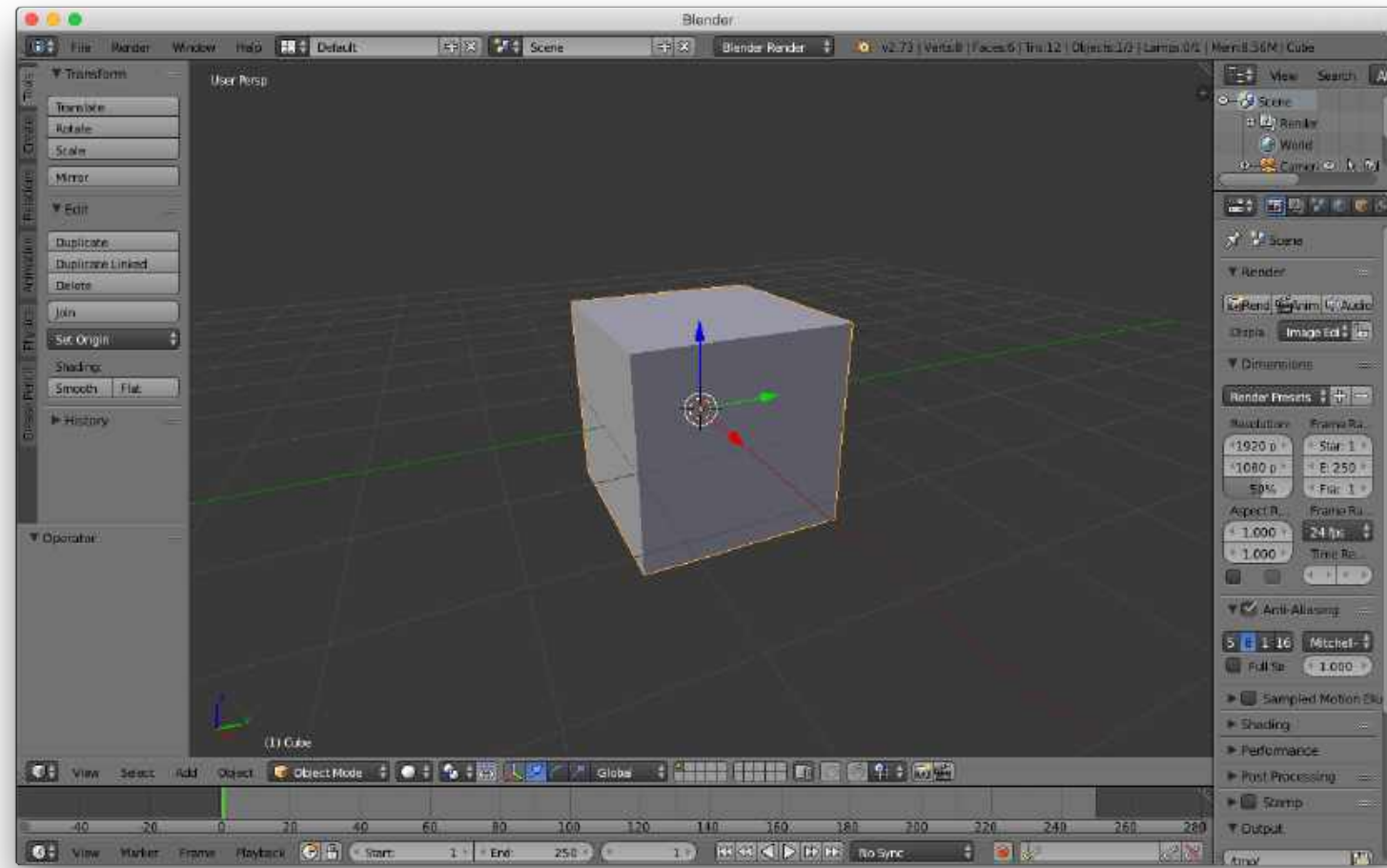


AutoCAD

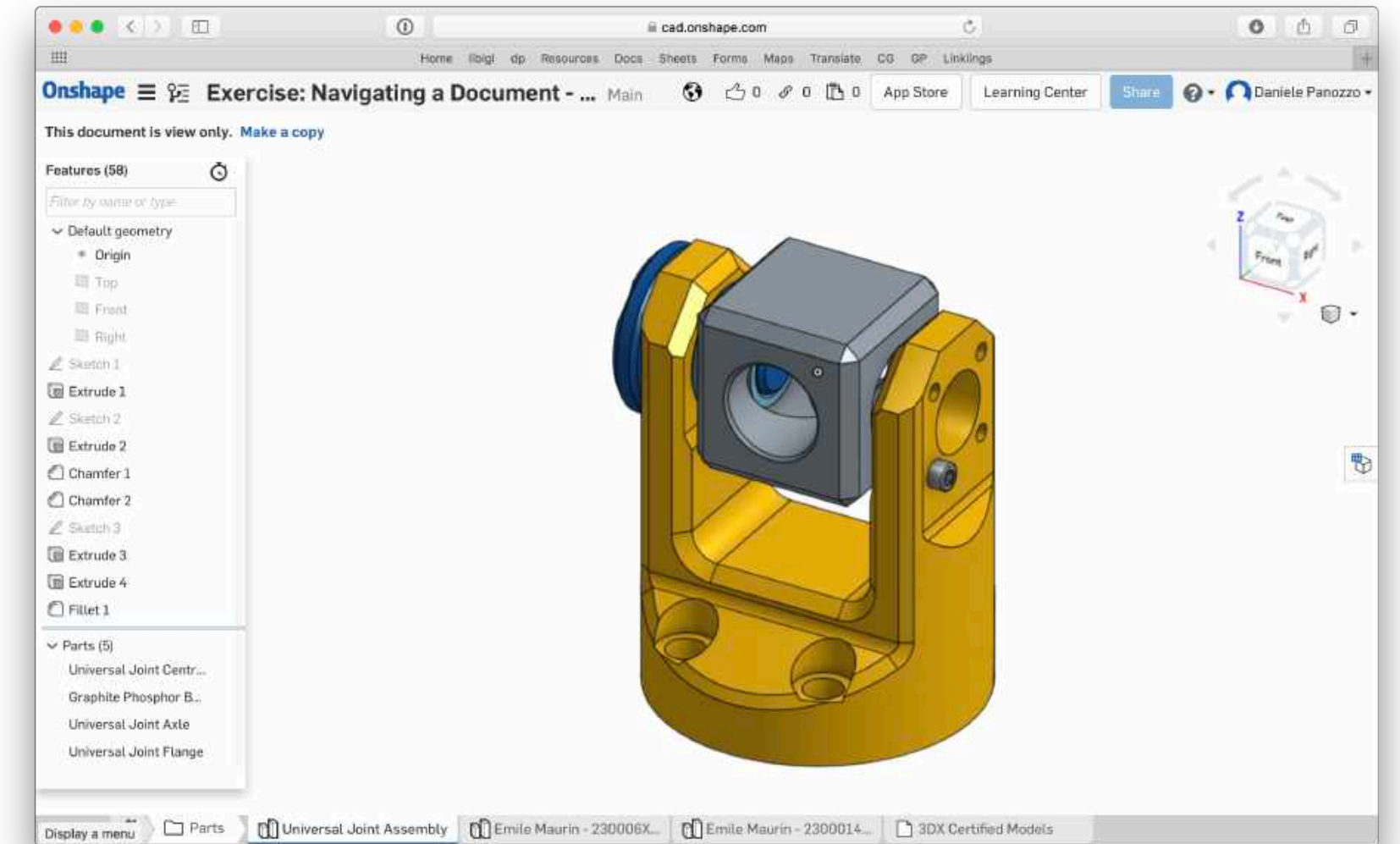


# Research Overview

## Geometry



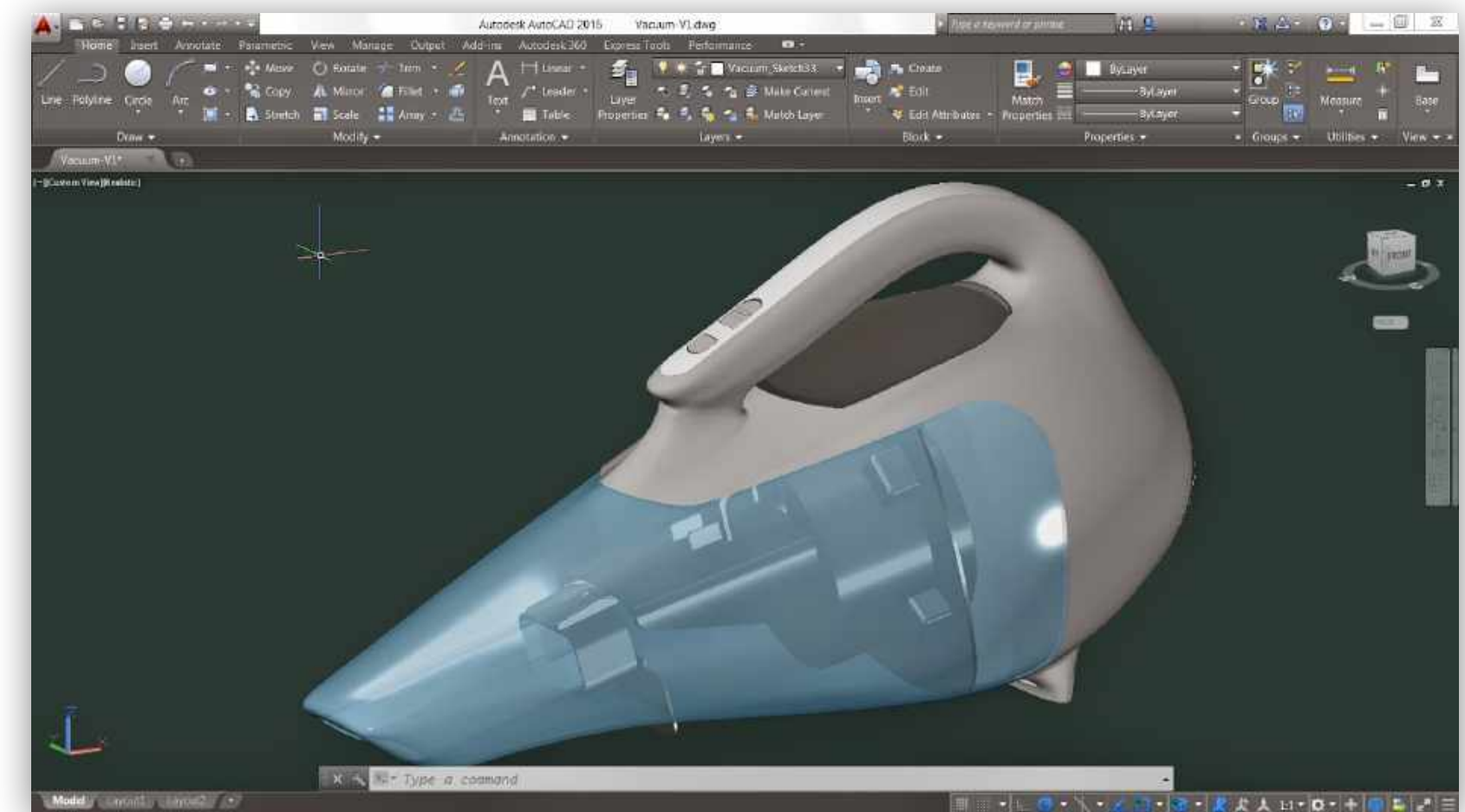
Blender



OnShape



Maya

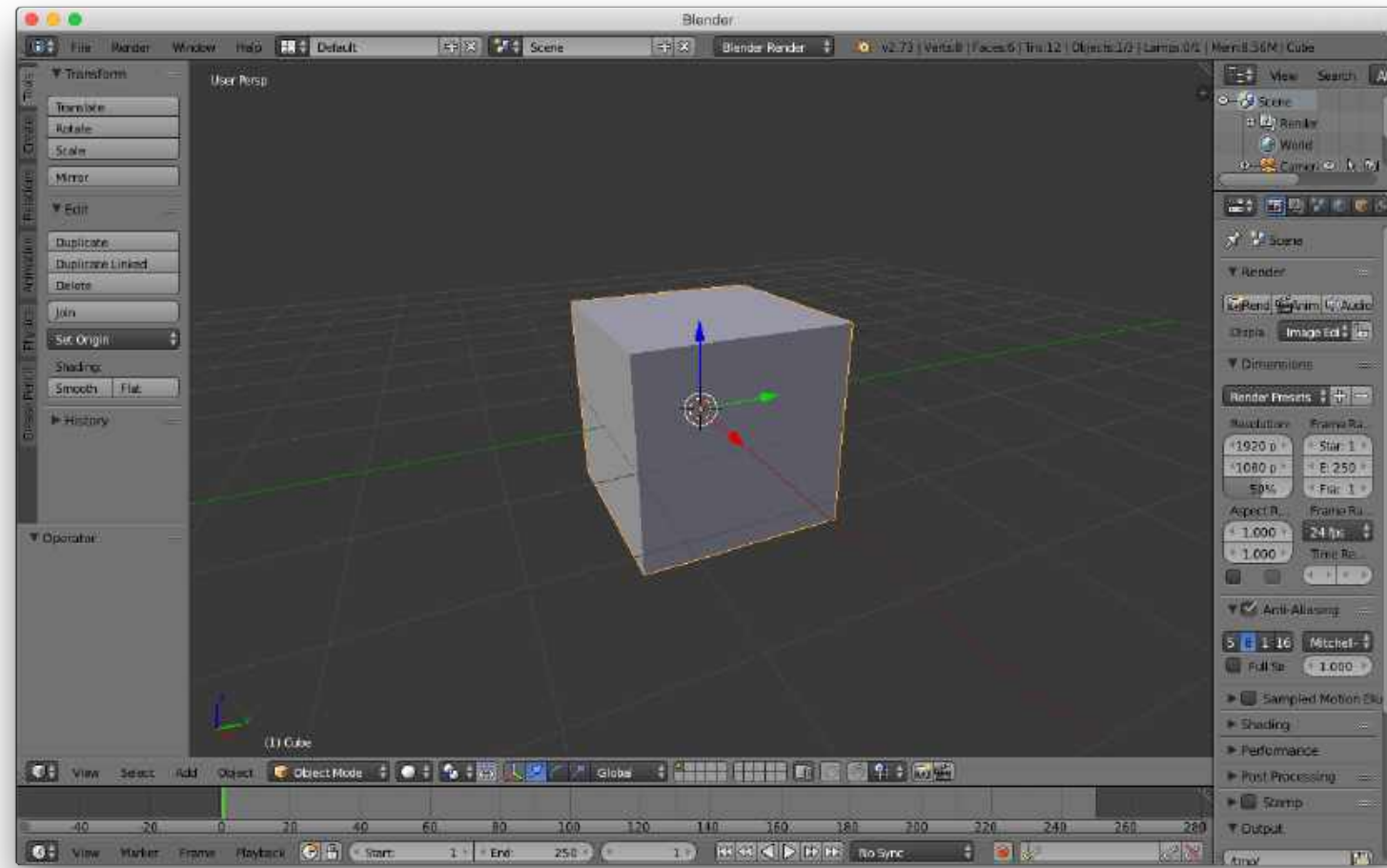


AutoCAD

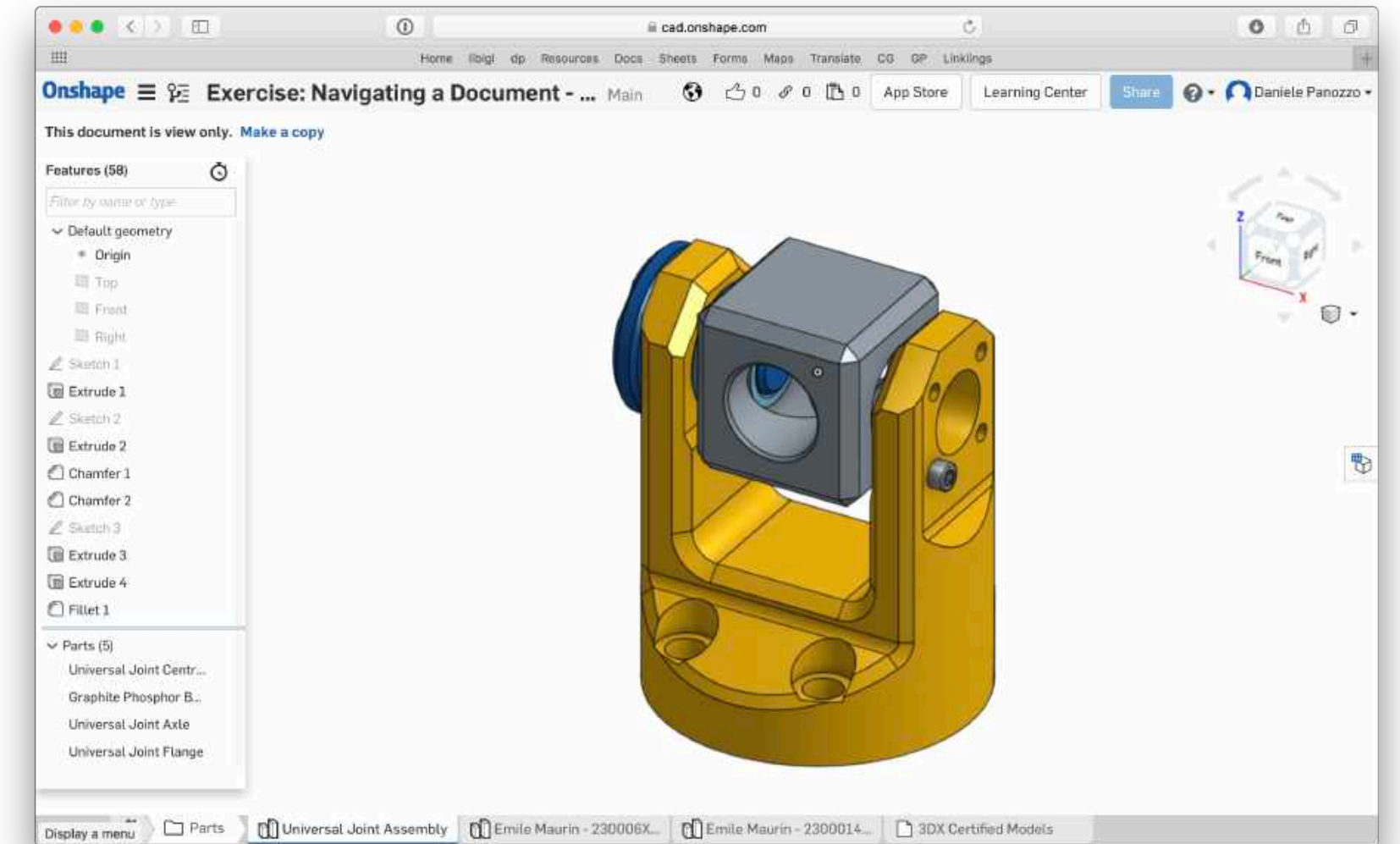


# Research Overview

## Geometry Appearance



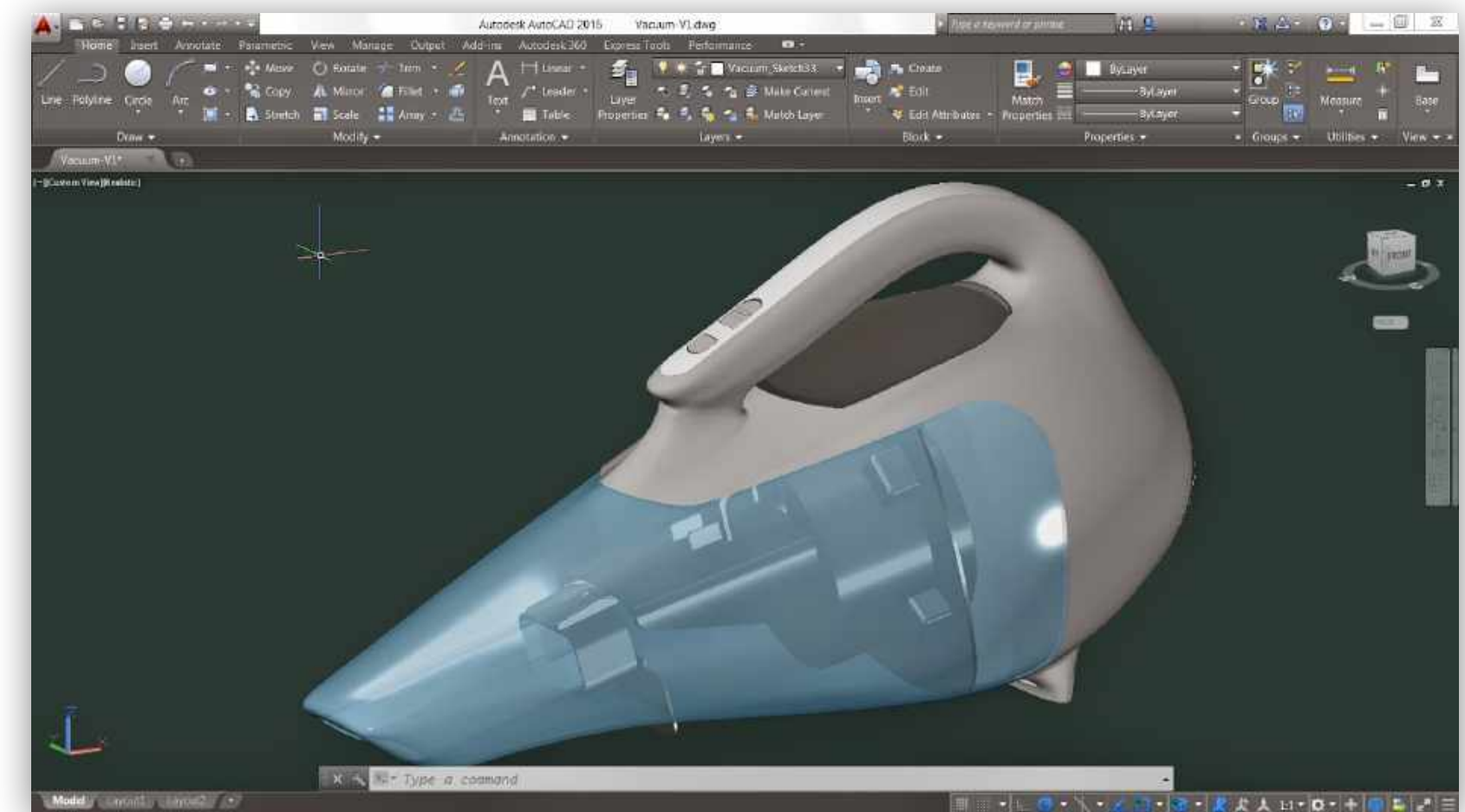
Blender



OnShape



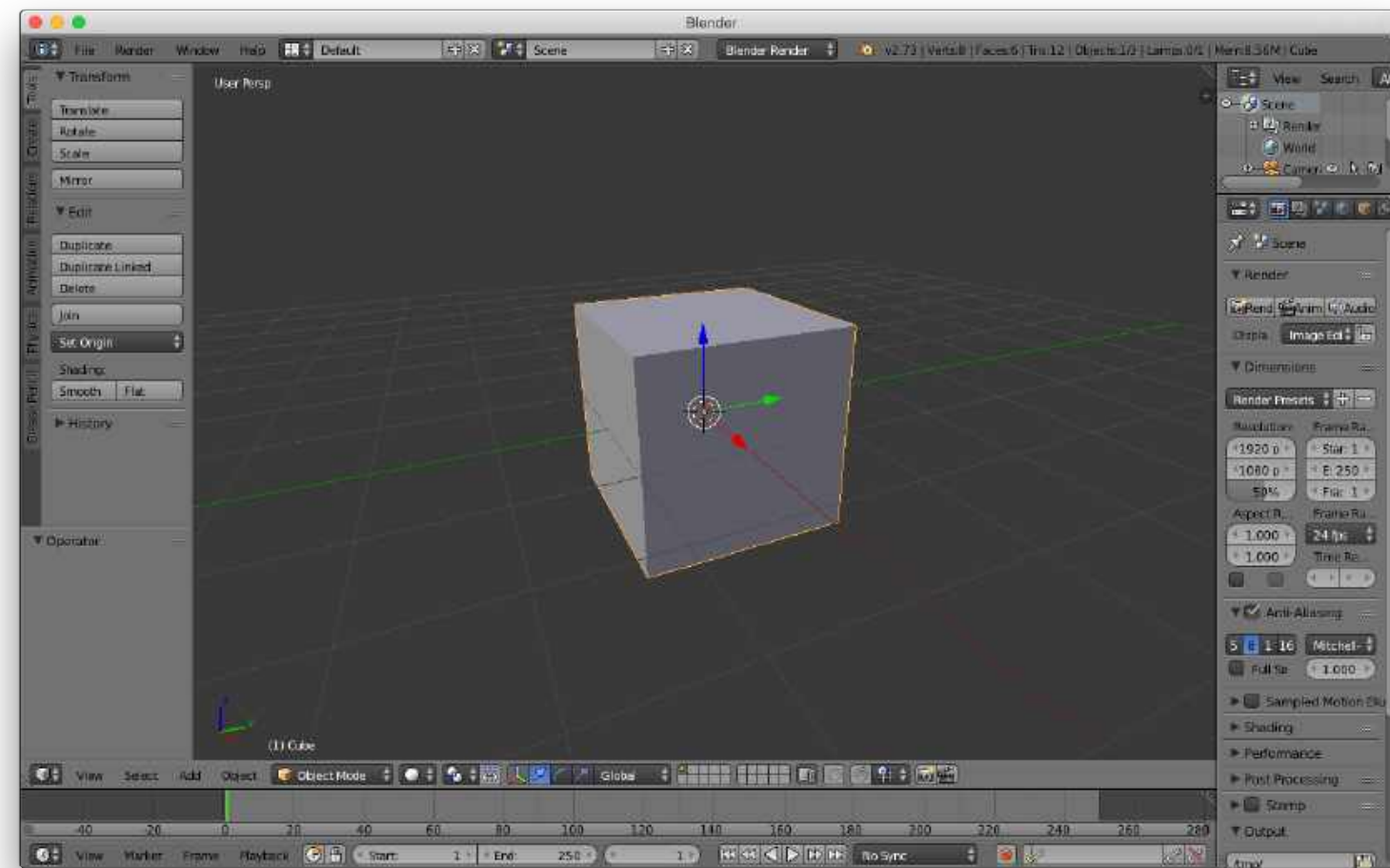
Maya




AutoCAD

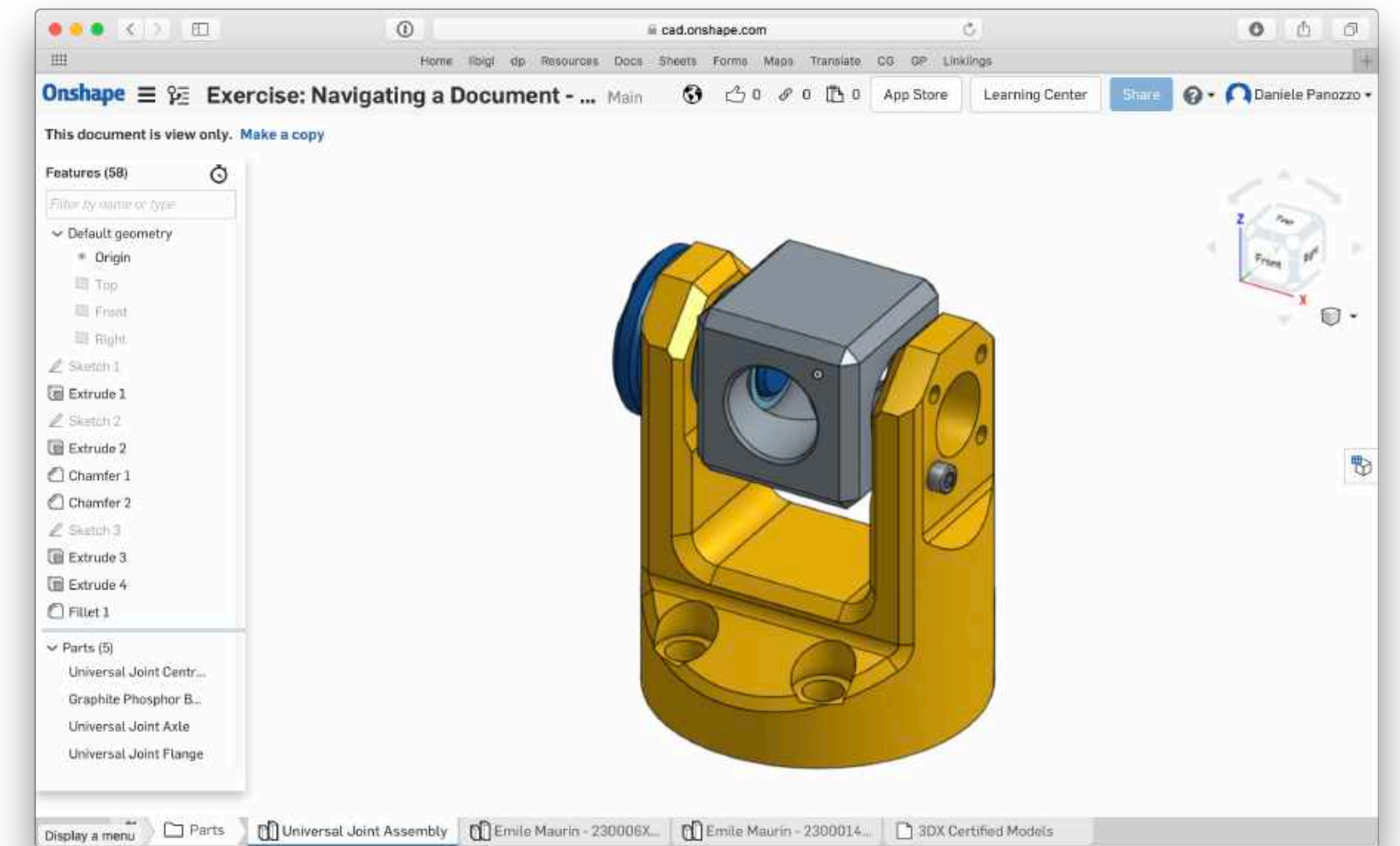


# Research Overview



Blender

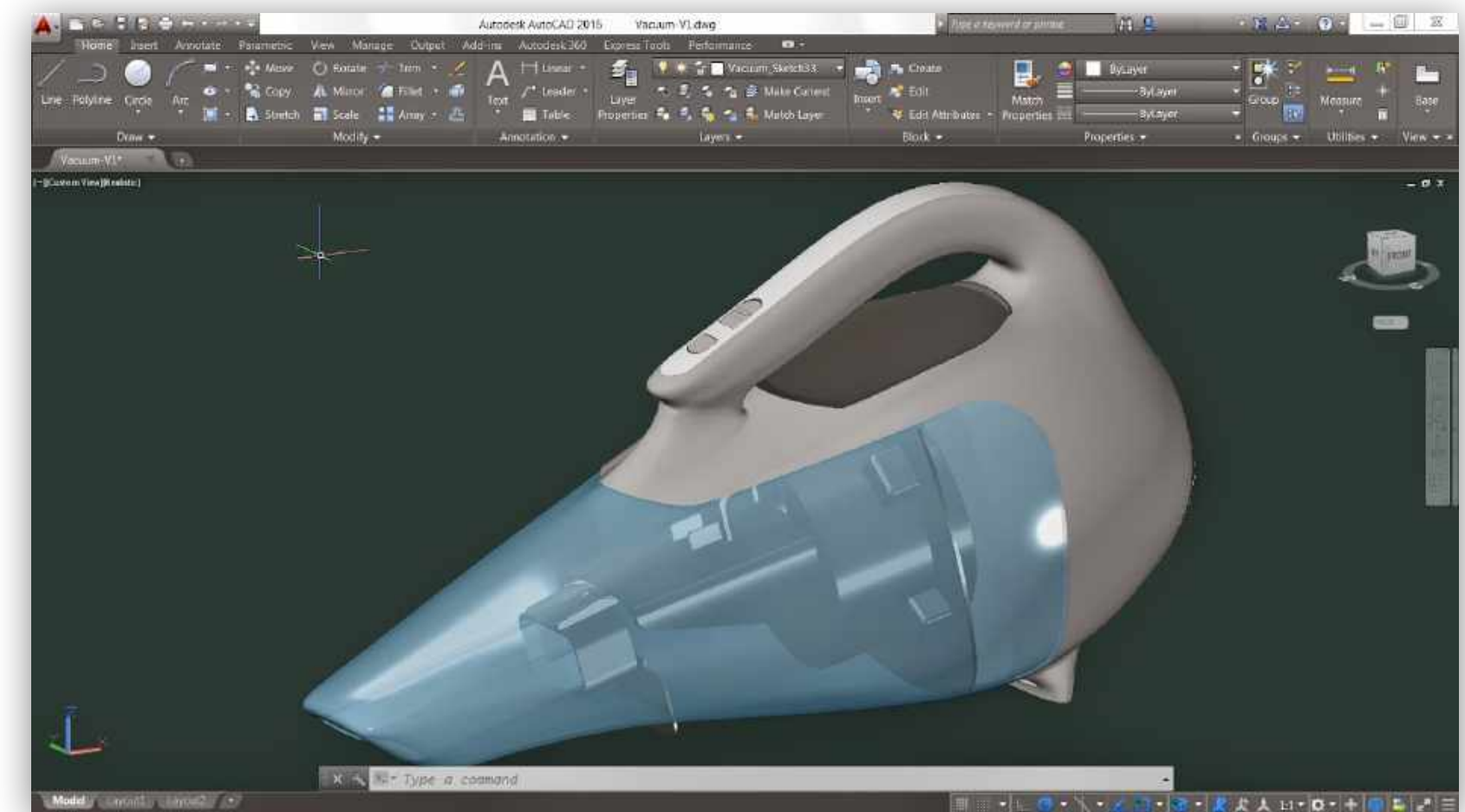
Geometry   
Appearance



OnShape



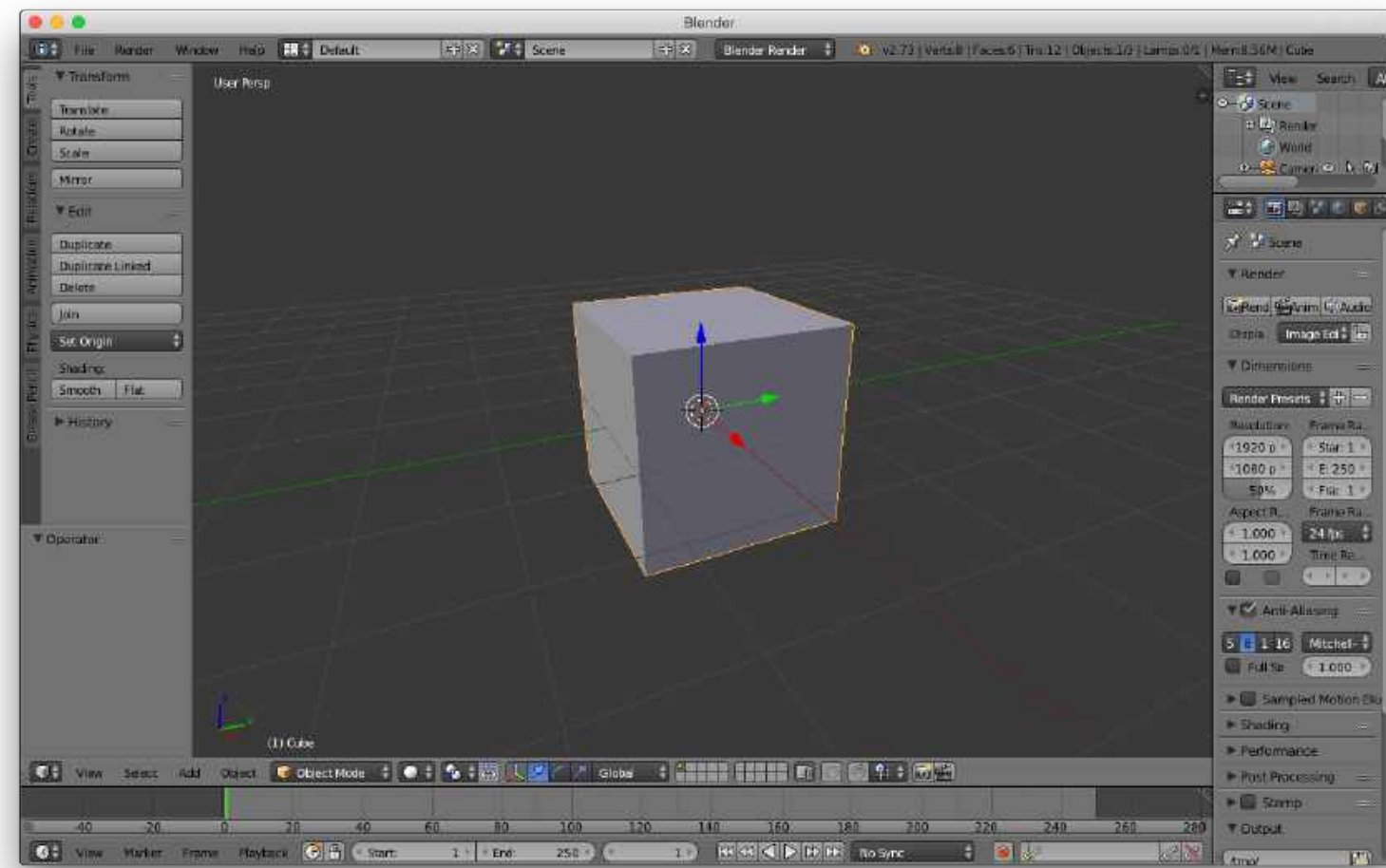
Maya



AutoCAD

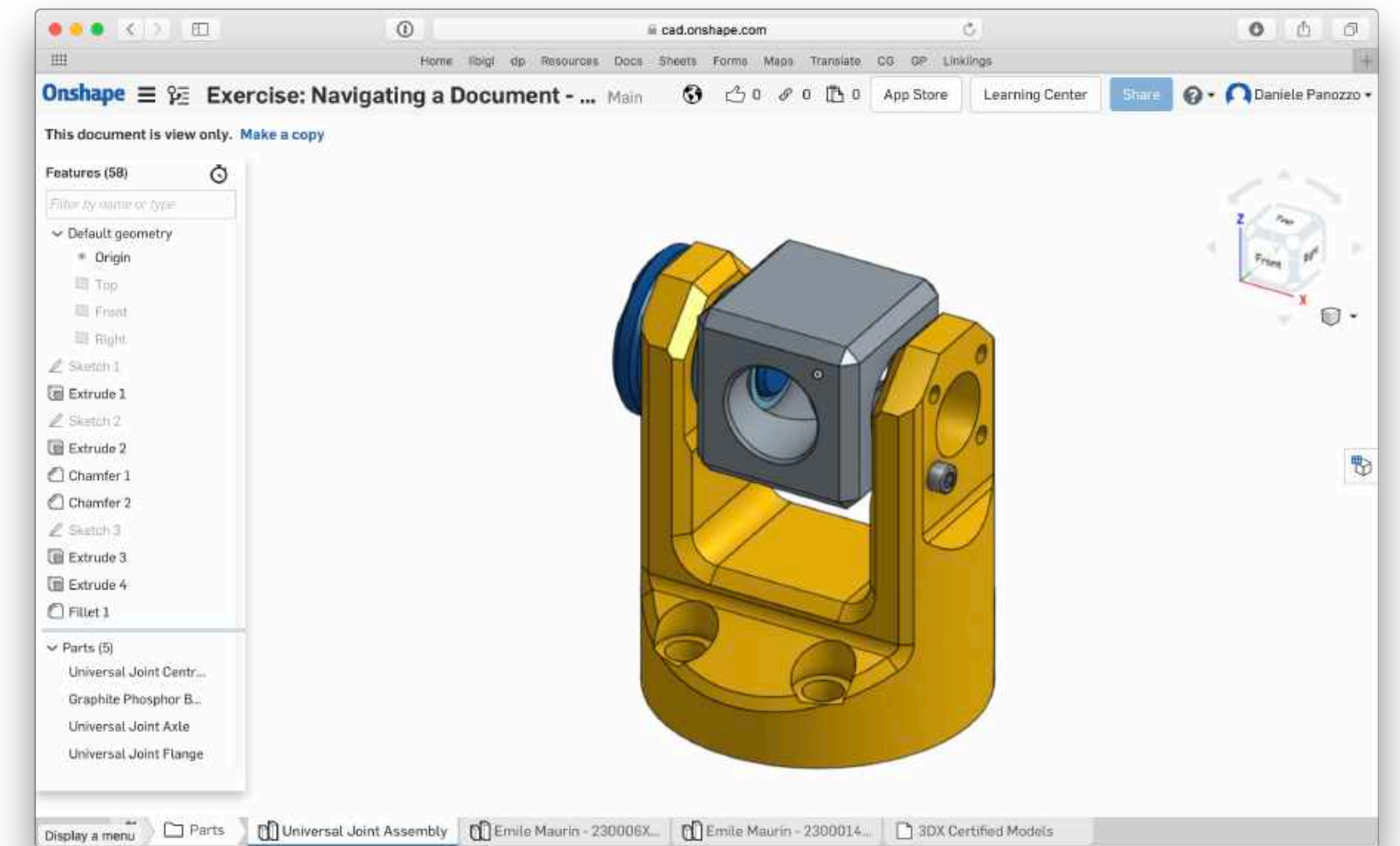


# Research Overview



Blender

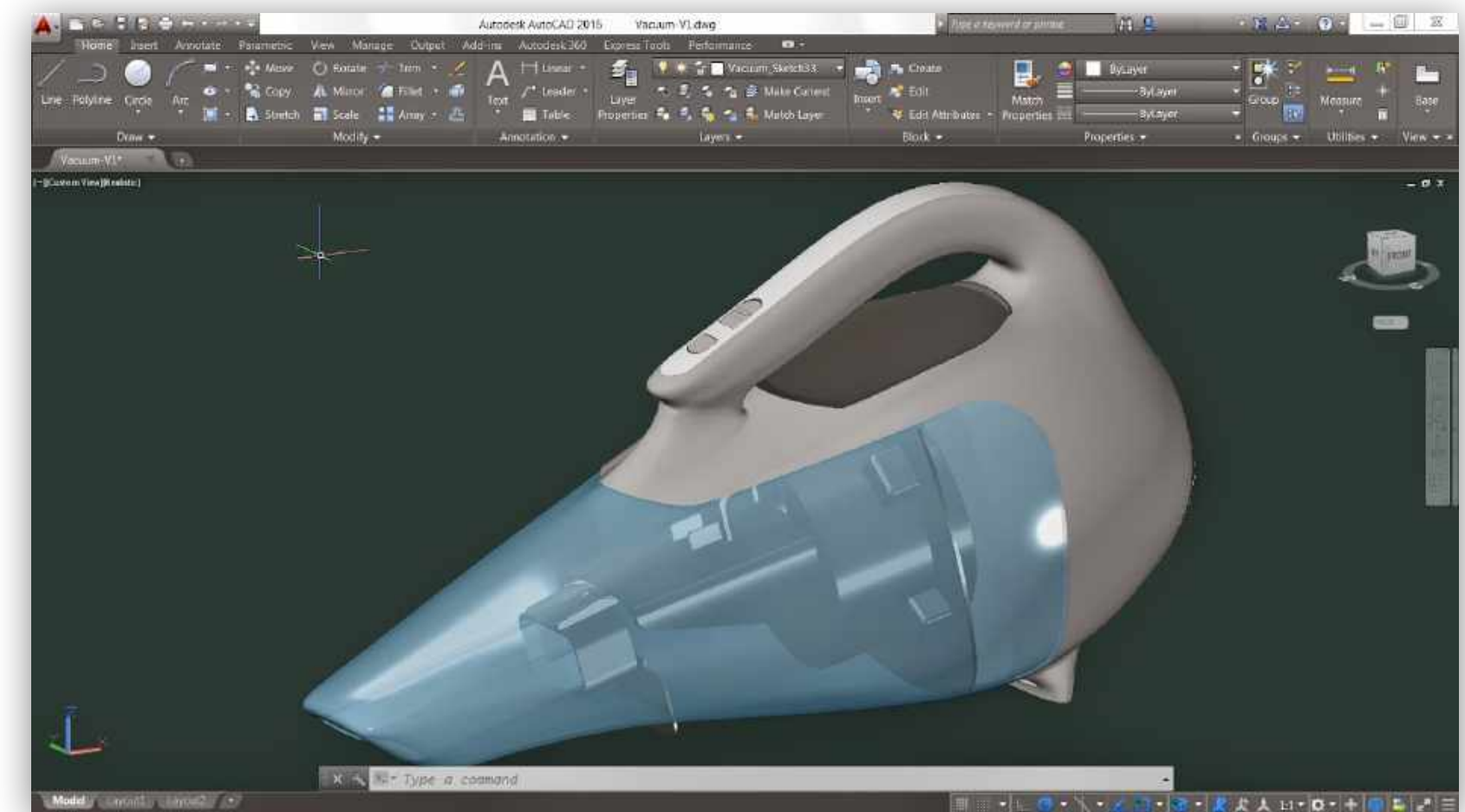
Geometry ✓  
Appearance ?



OnShape



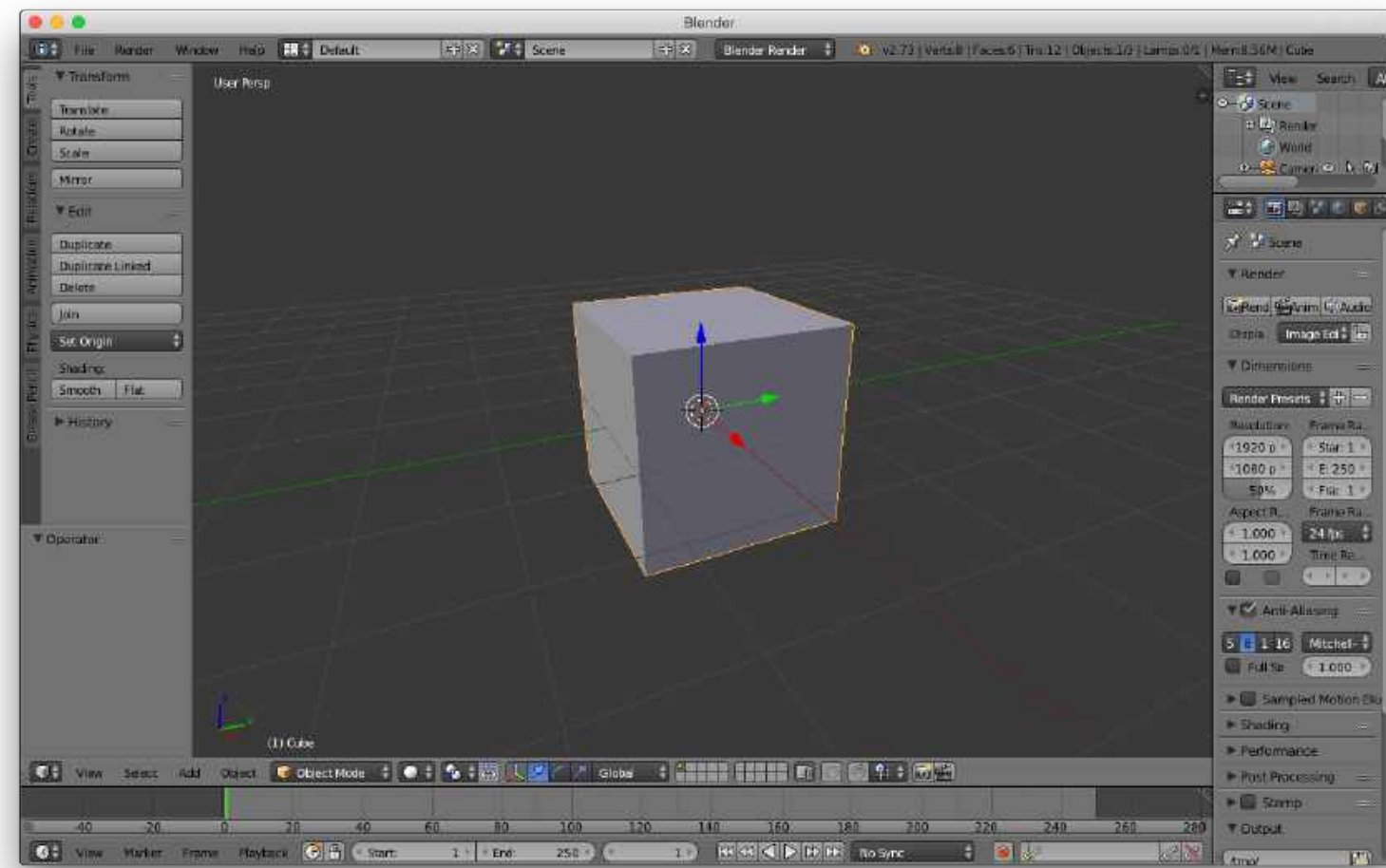
Maya



AutoCAD

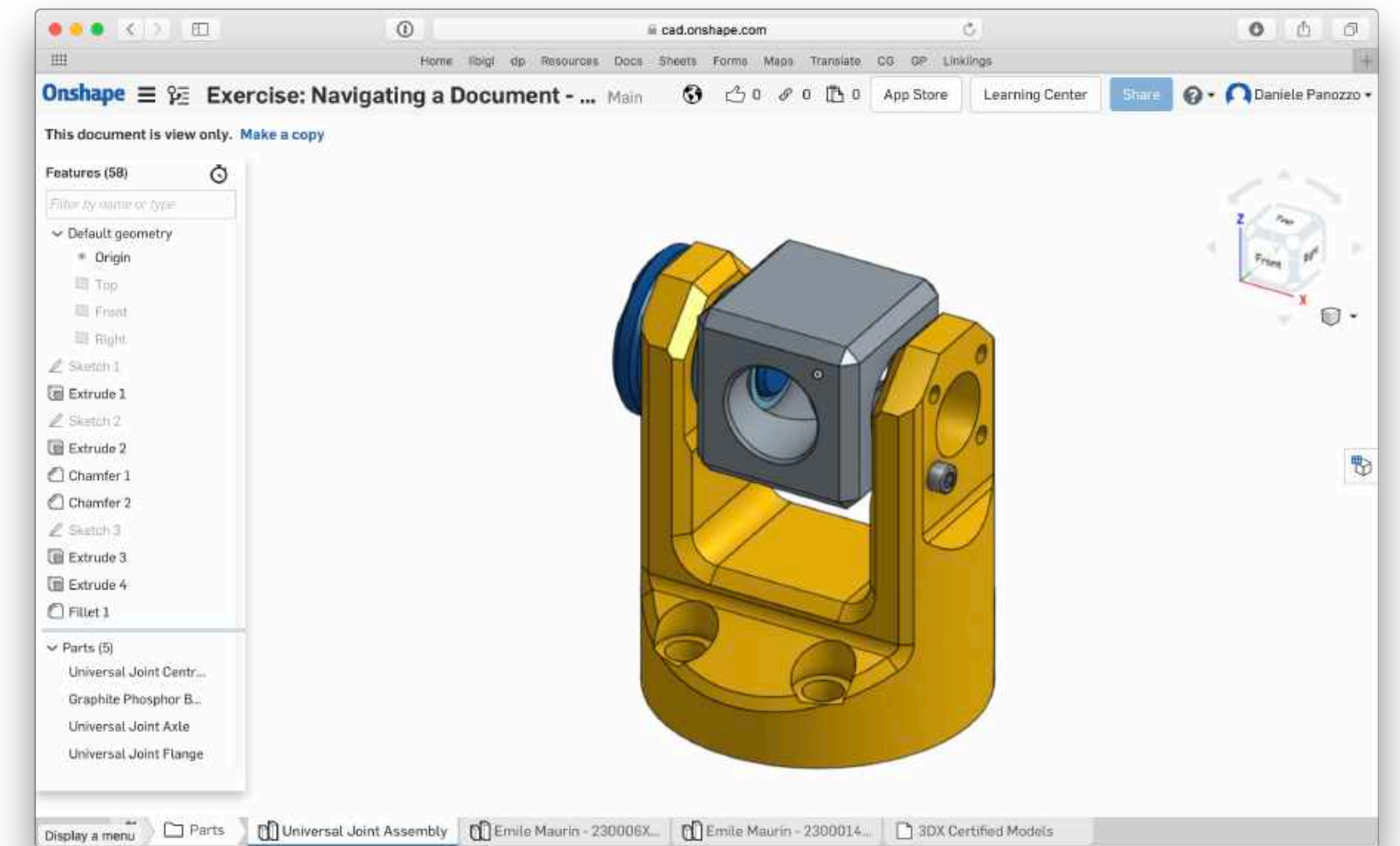


# Research Overview



Blender

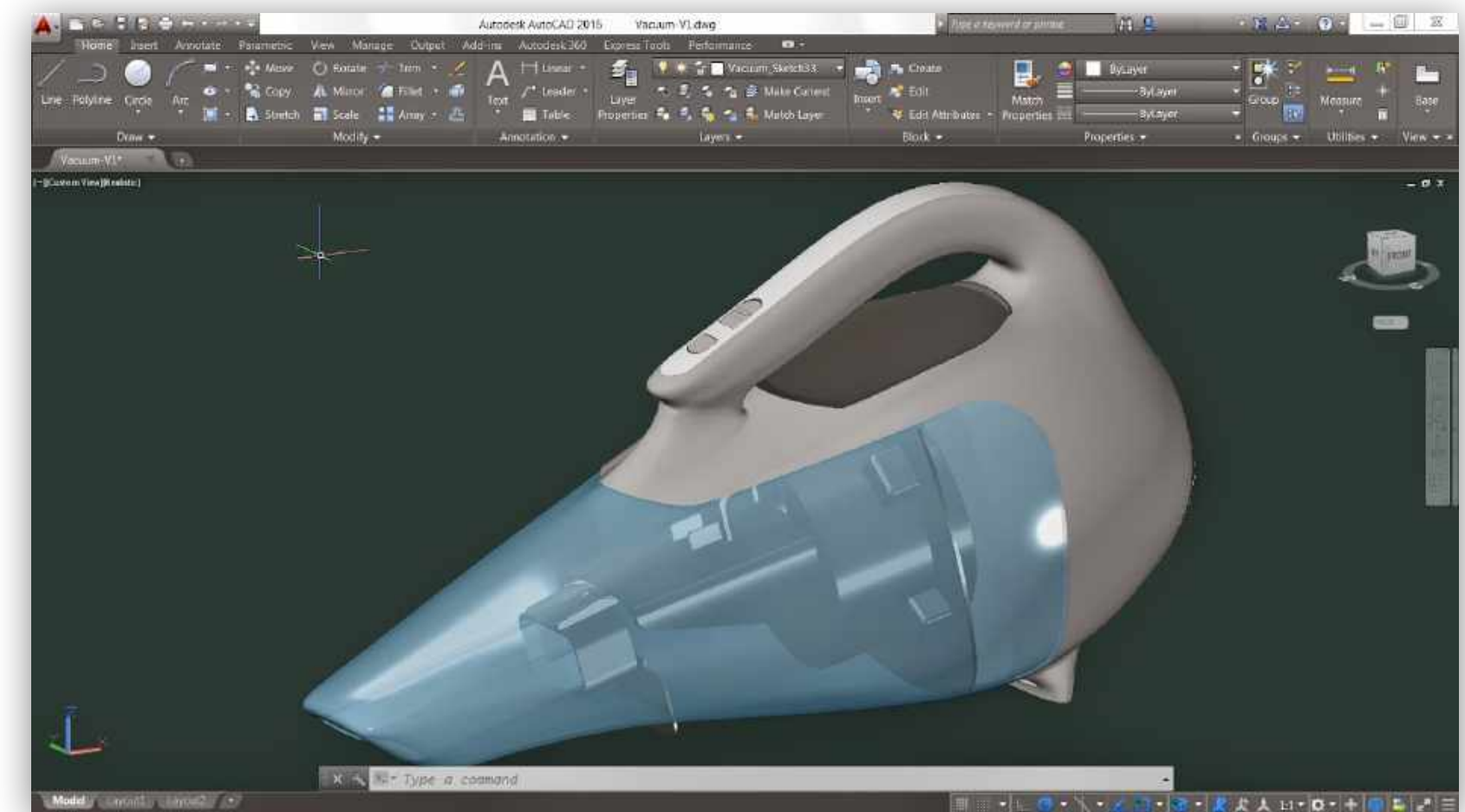
Geometry ✓  
Appearance  
Fabricability ?



OnShape



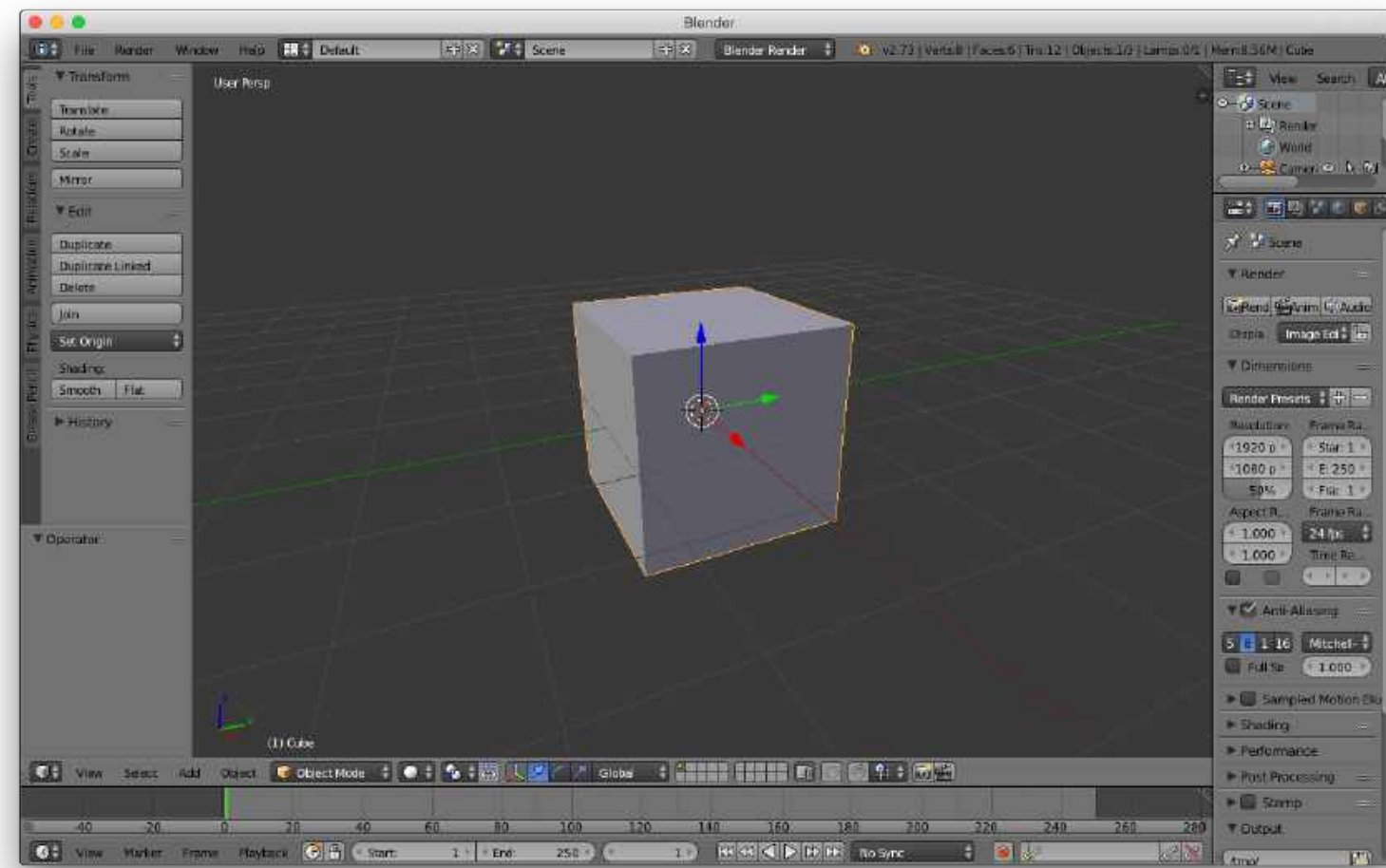
Maya



AutoCAD



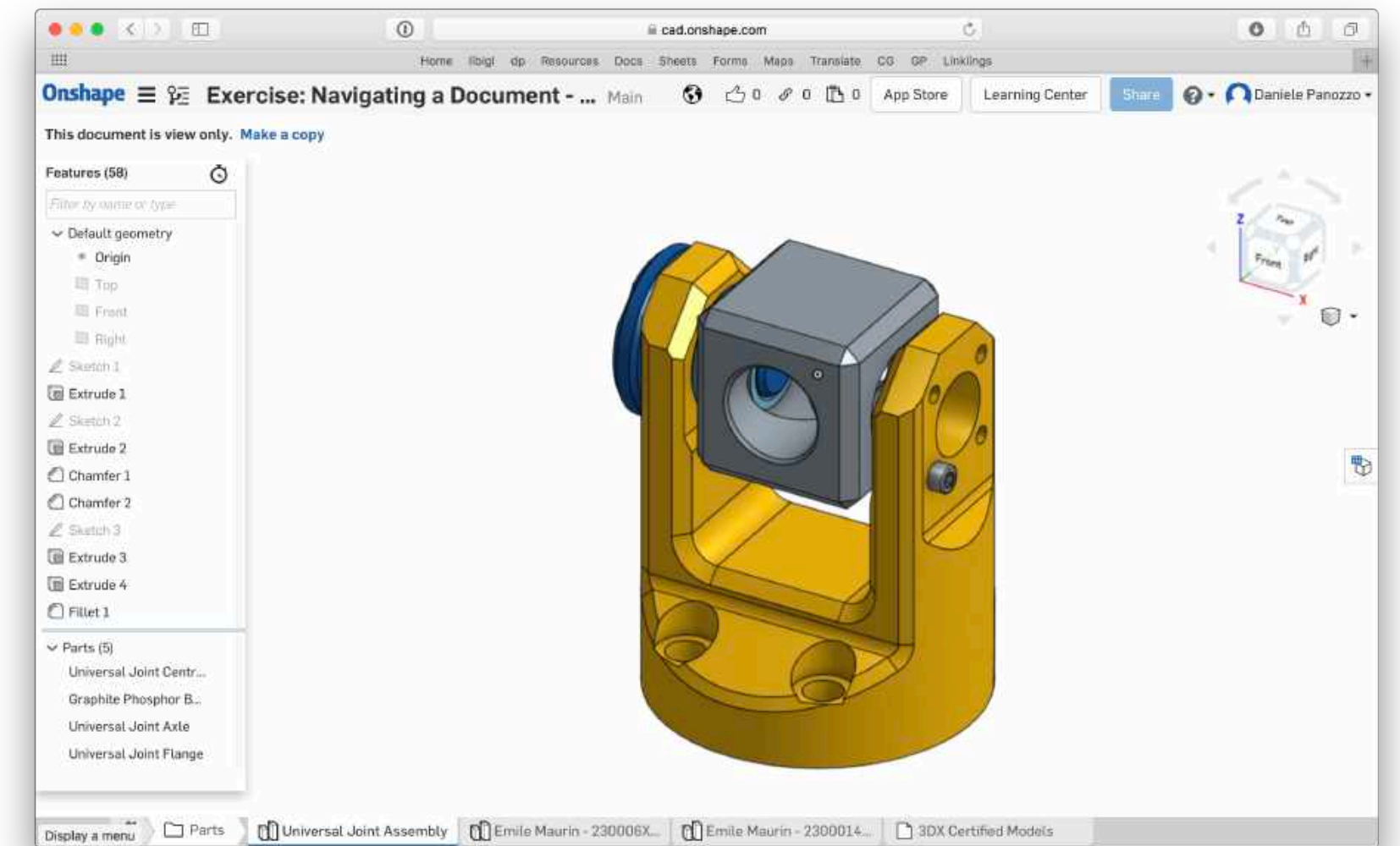
# Research Overview



Blender

Geometry ✓  
Appearance

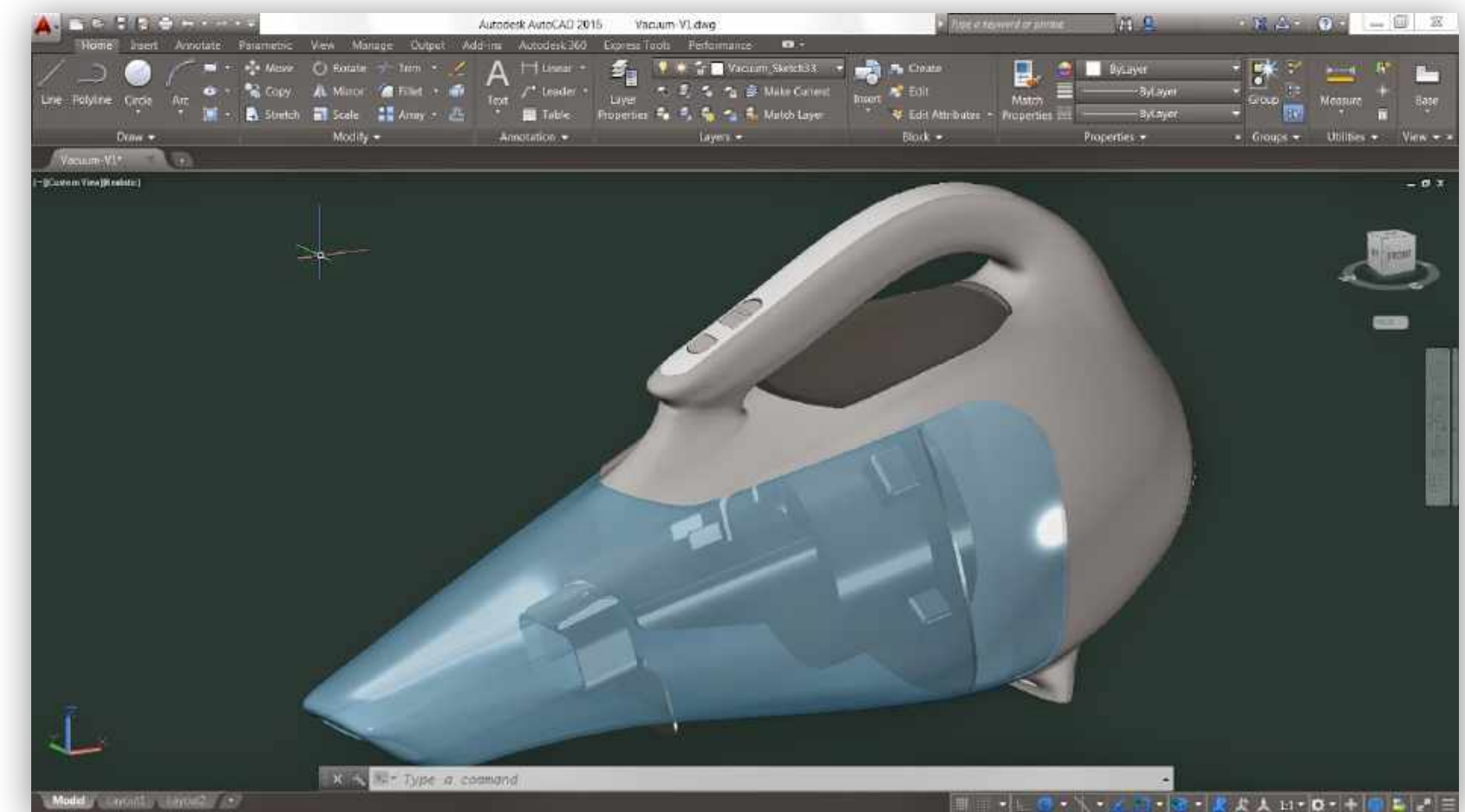
Fabricability ?  
Stability



OnShape



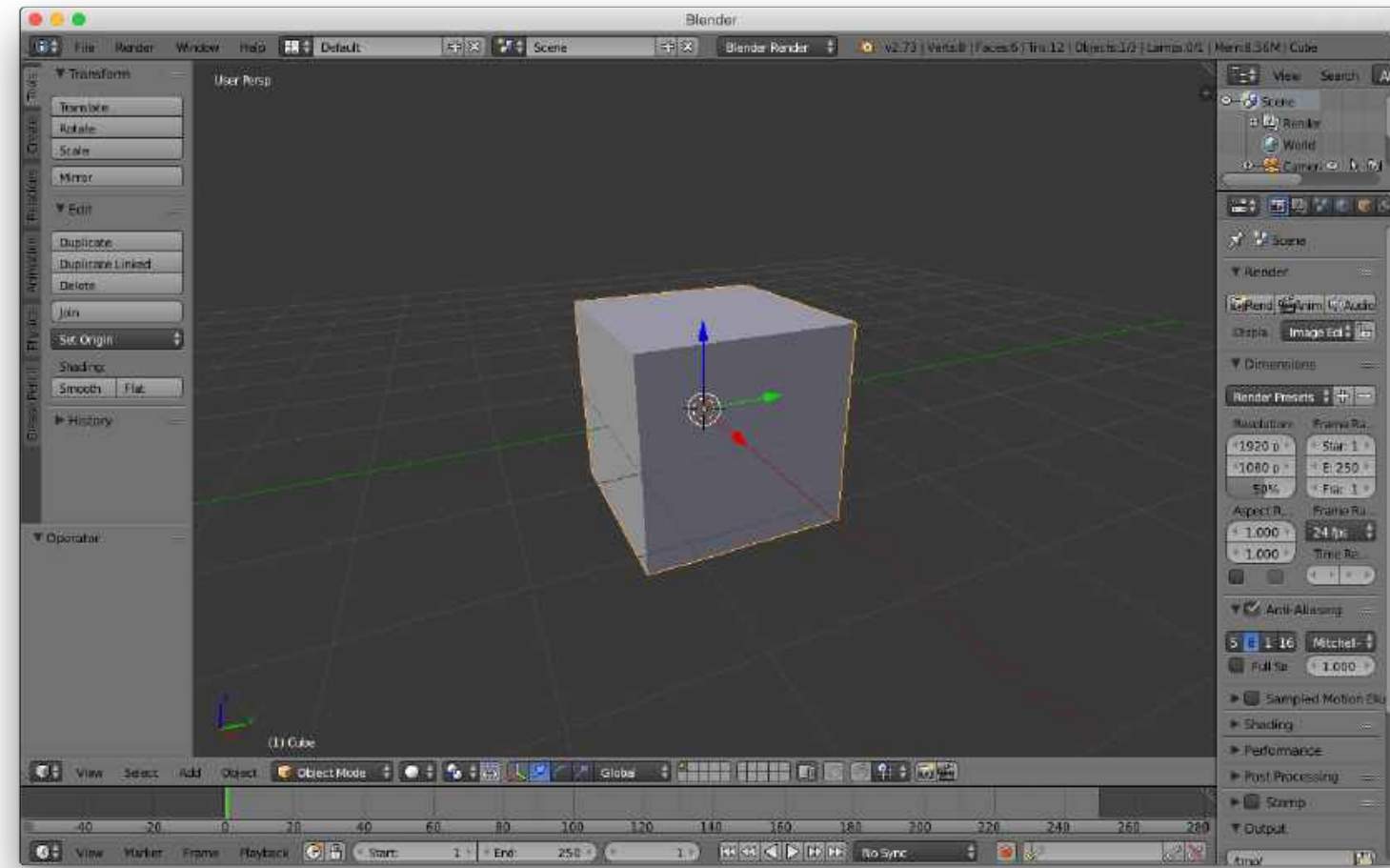
Maya



AutoCAD



# Research Overview



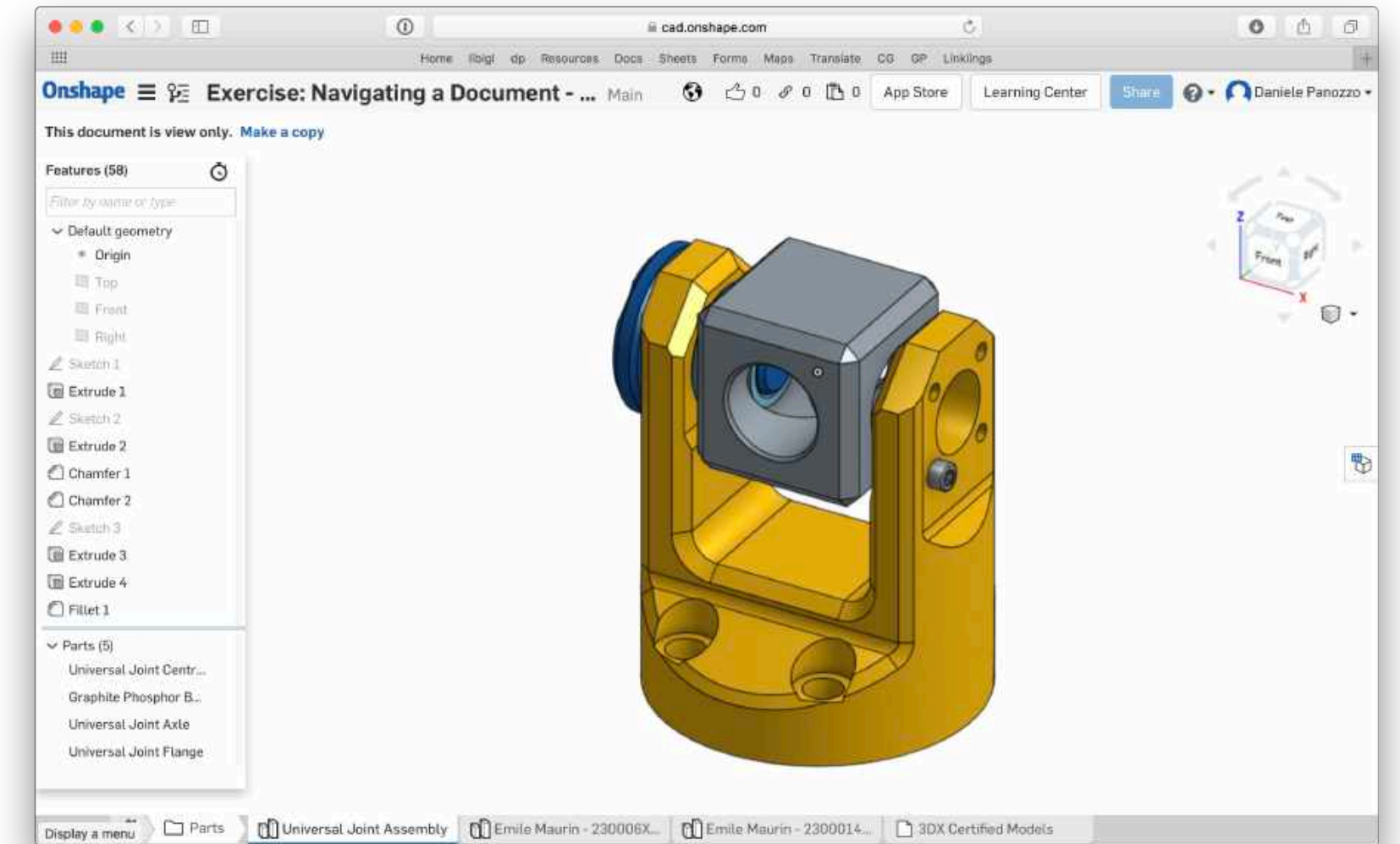
Blender



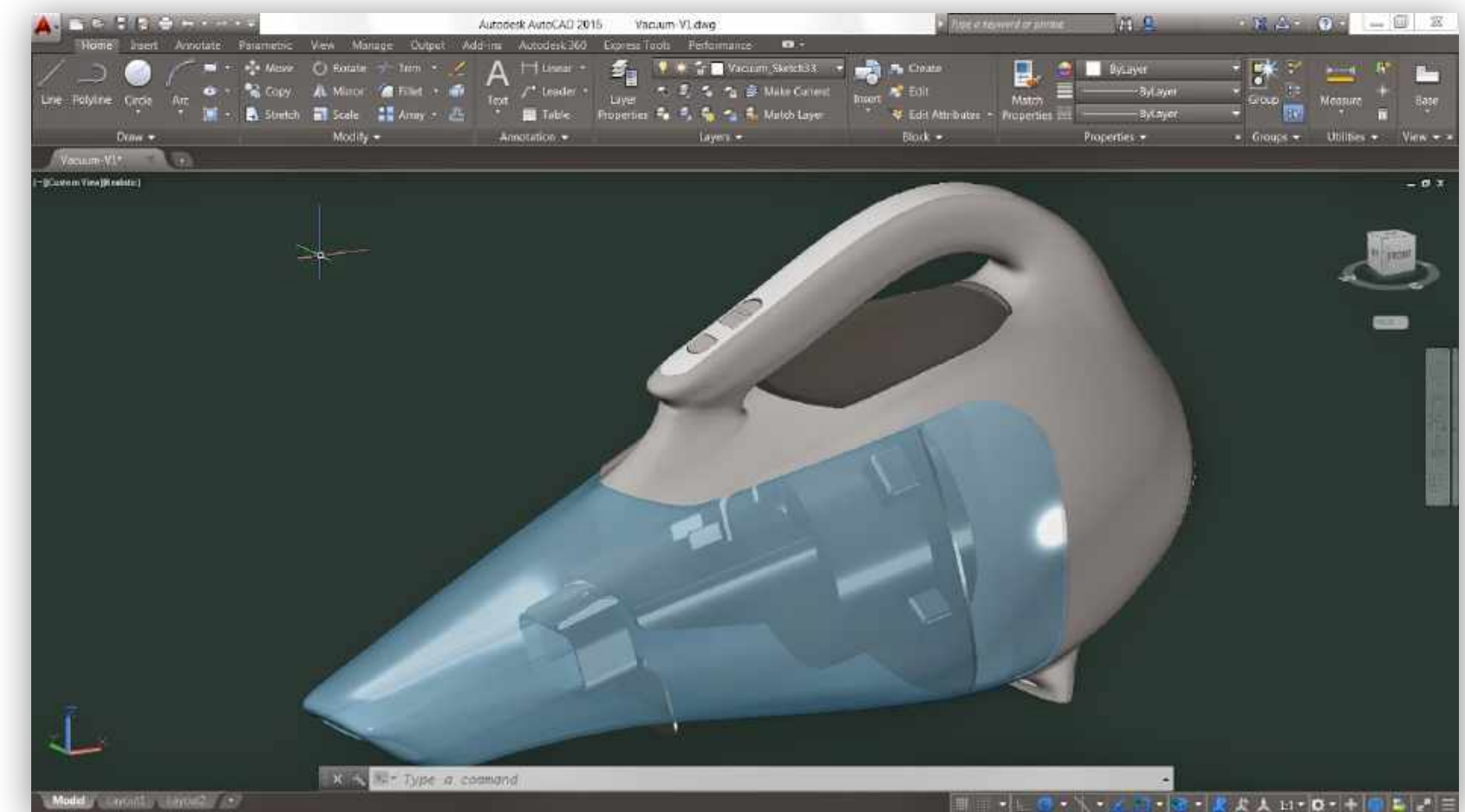
Maya

Geometry ✓  
Appearance  
Fabricability ?

Stability  
Robustness



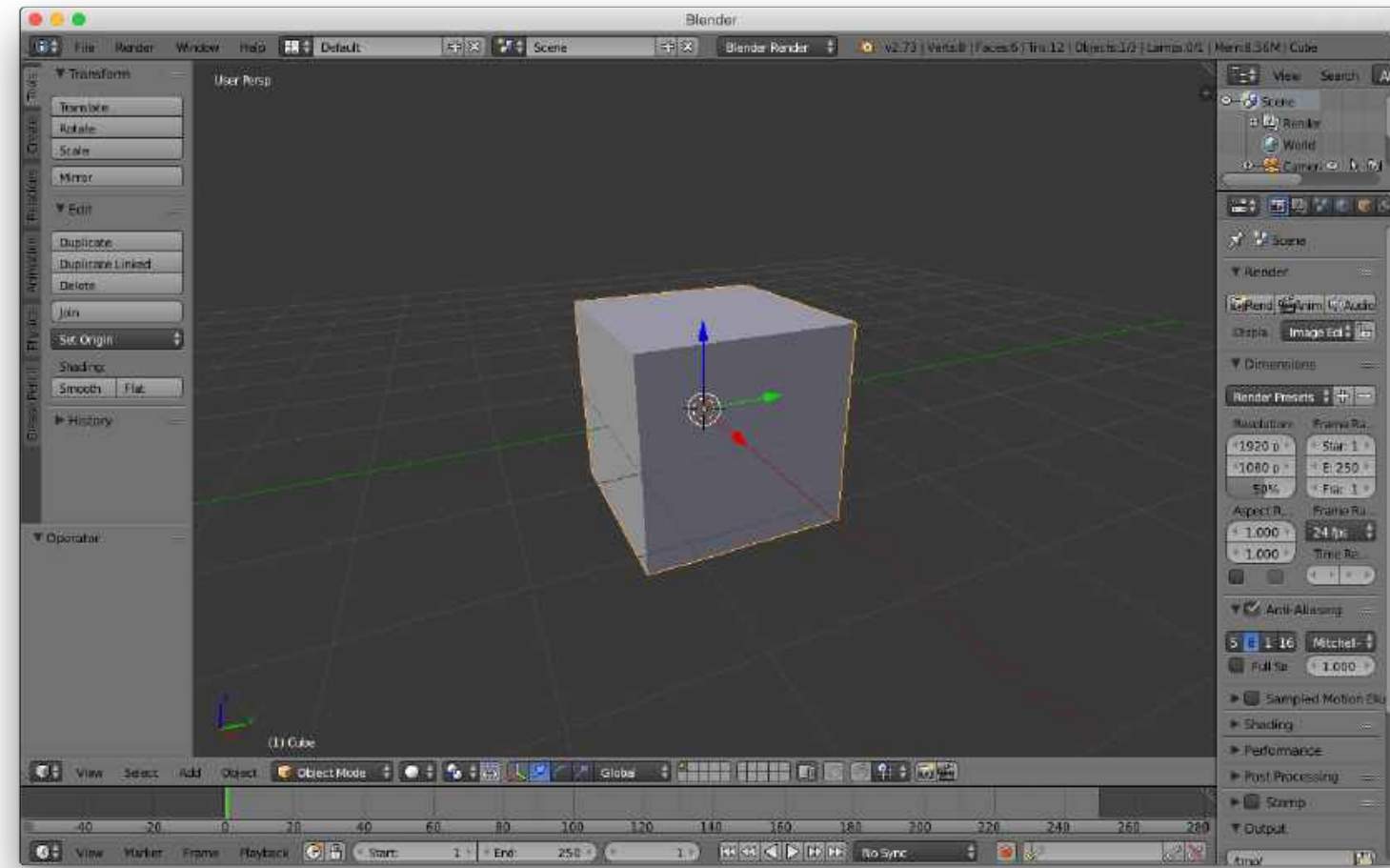
OnShape



AutoCAD



# Research Overview



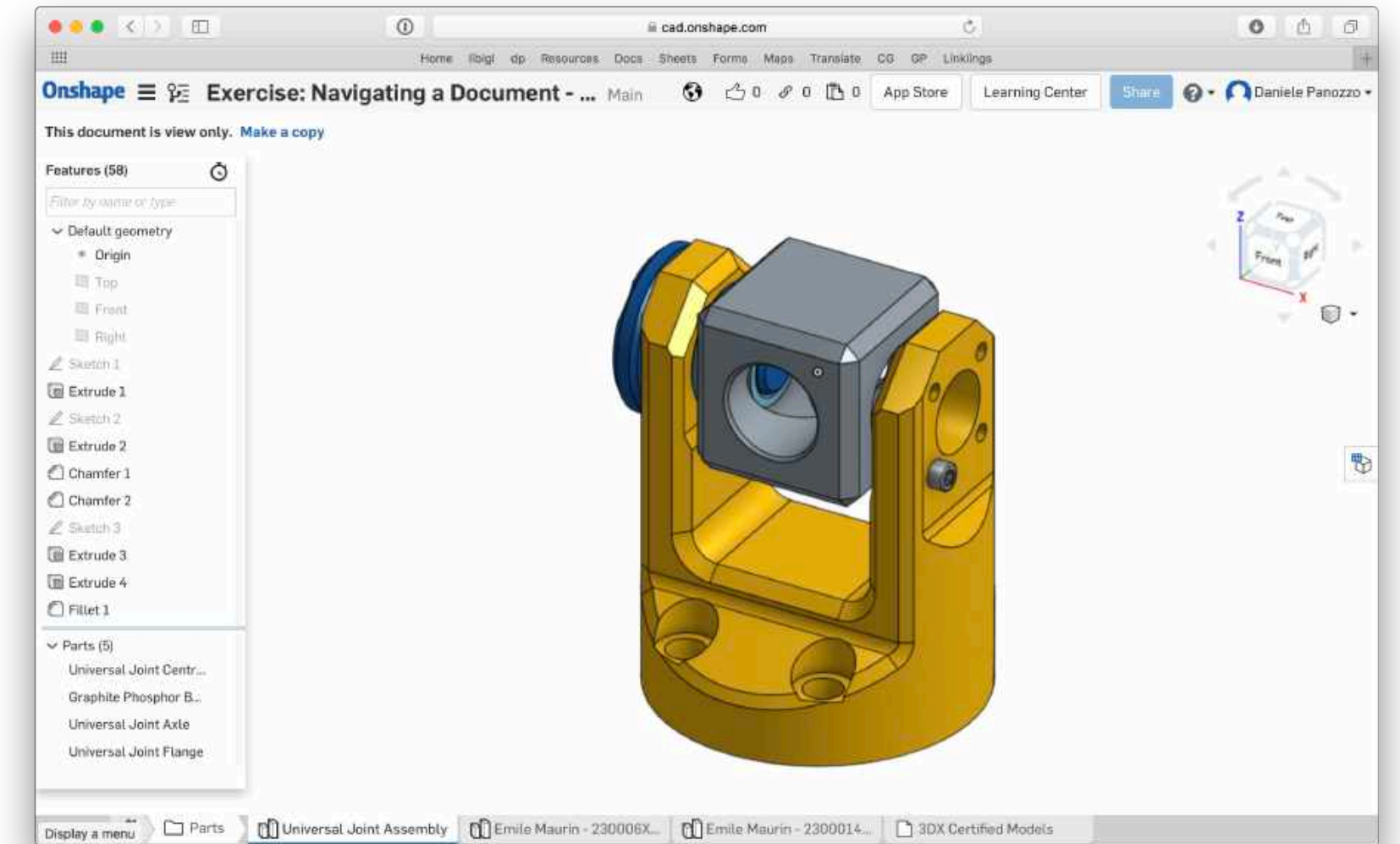
Blender

Geometry ✓  
Appearance  
Fabricability ?

Stability

Robustness

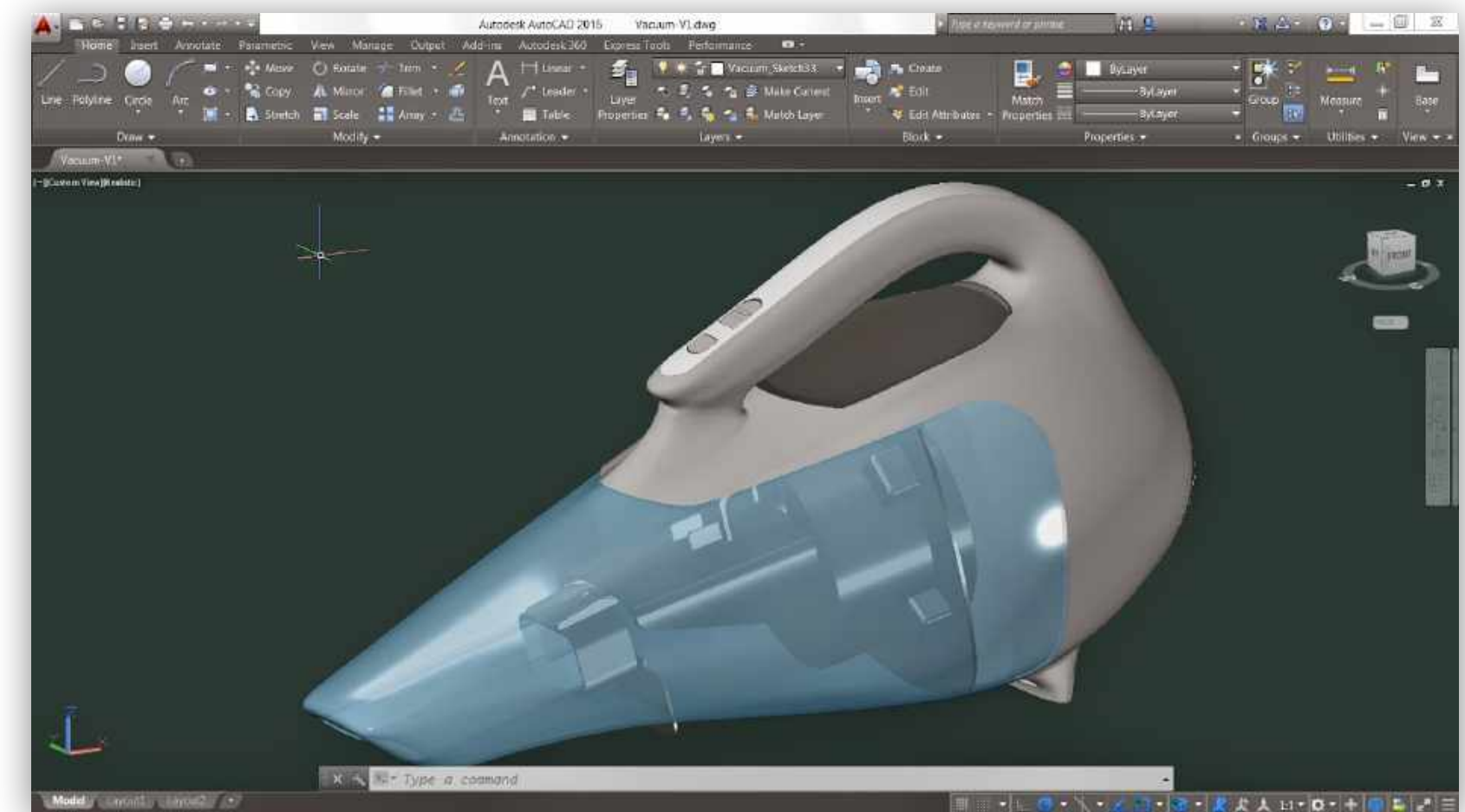
Cost



OnShape



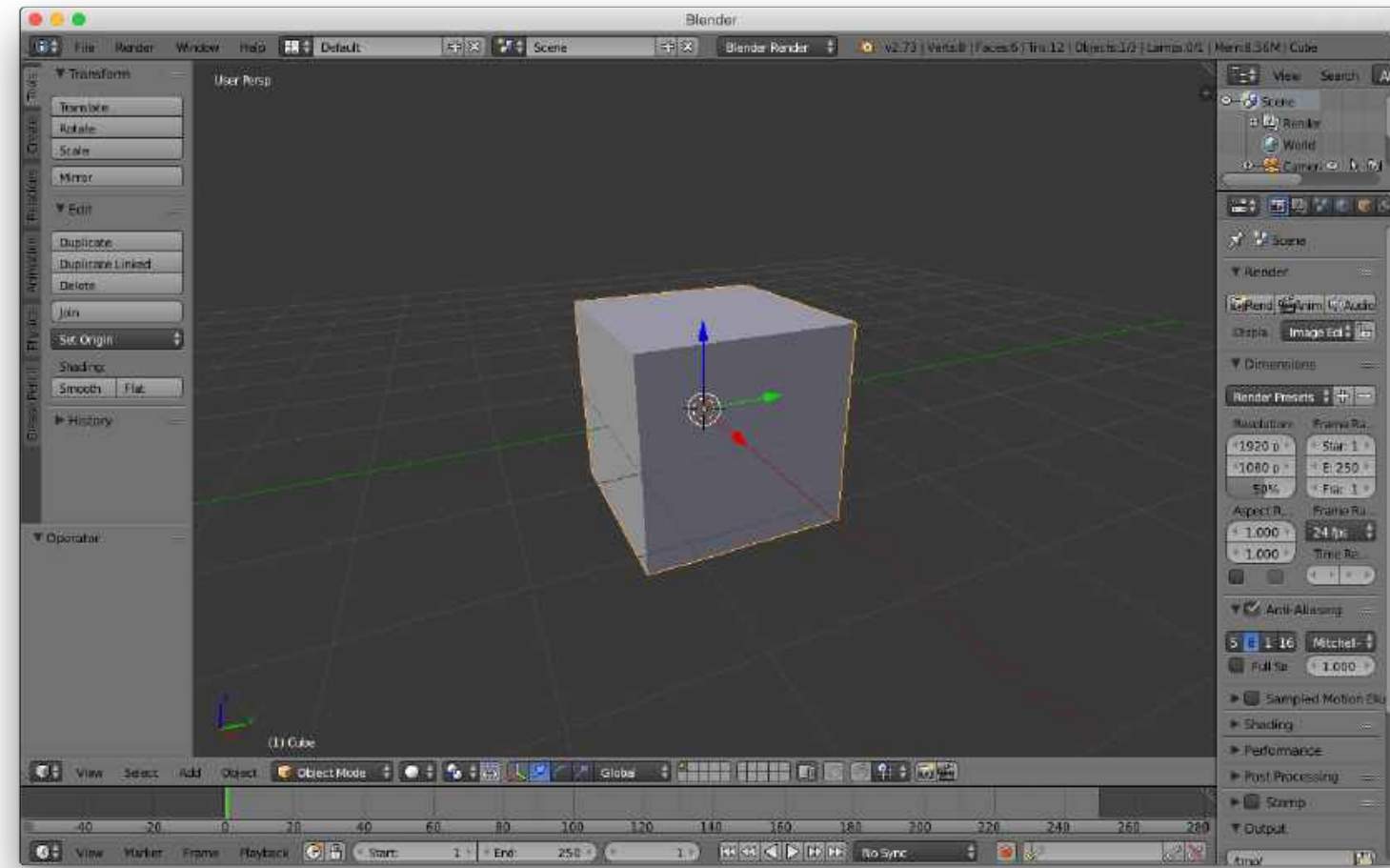
Maya



AutoCAD



# Research Overview



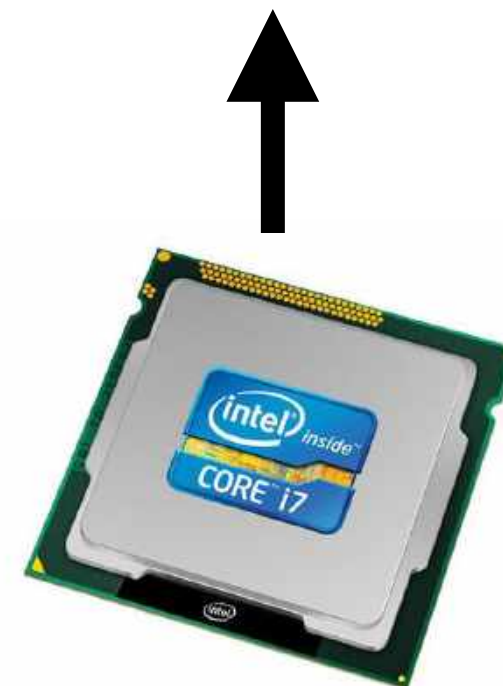
Blender

Geometry ✓  
Appearance  
Fabricability ?

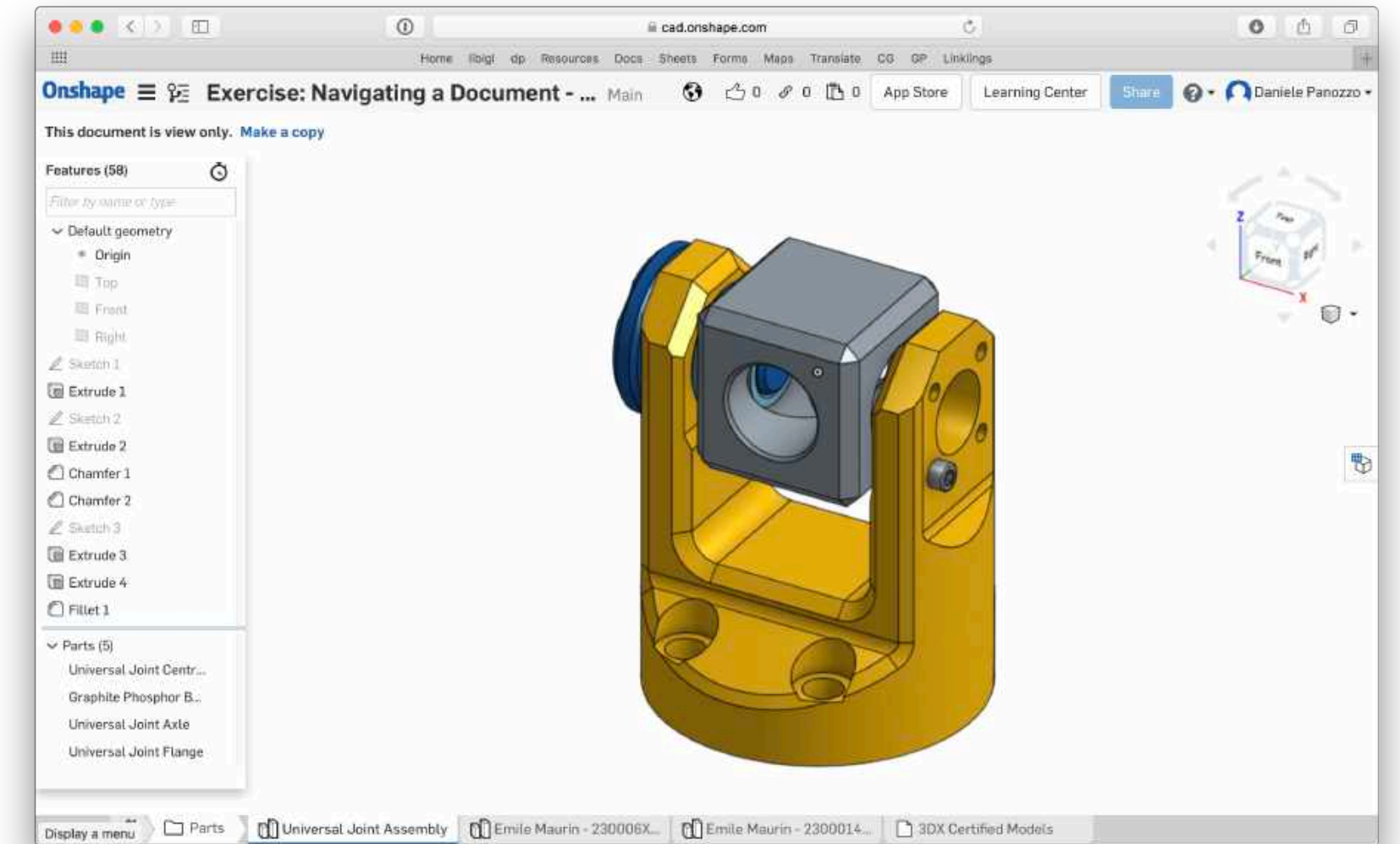
Stability

Robustness

Cost



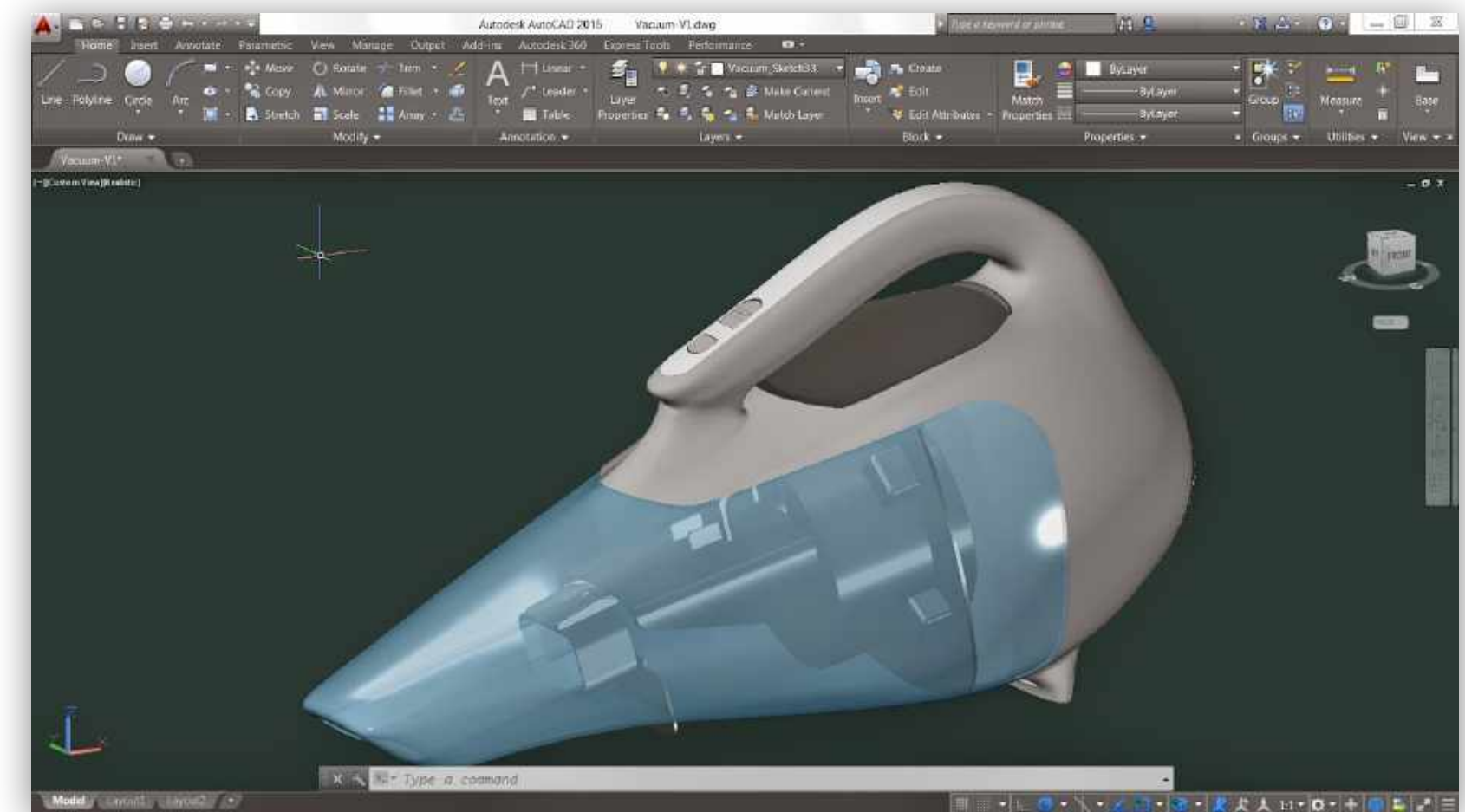
5



OnShape



Maya



AutoCAD



# Examples



# Examples





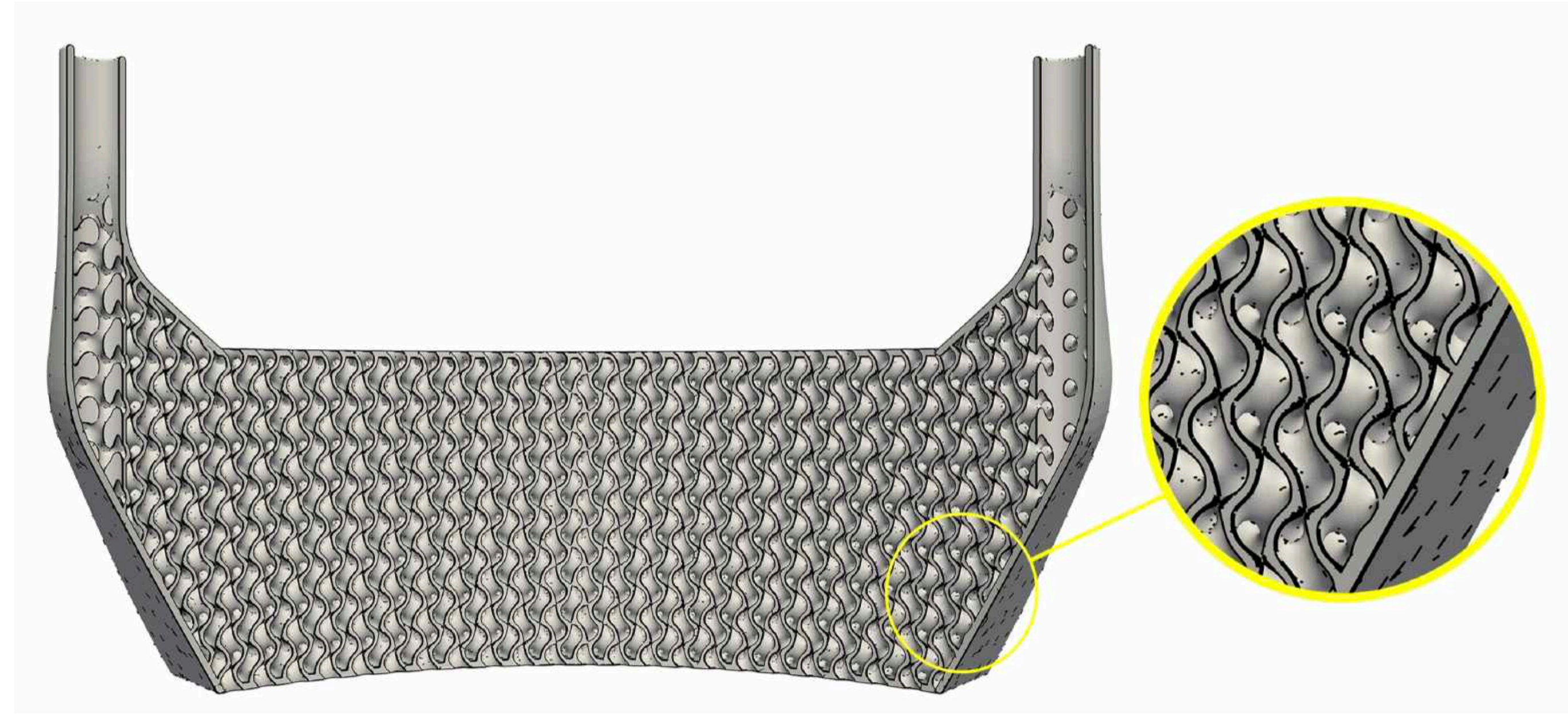
# Examples



Volume Reduction



# Examples



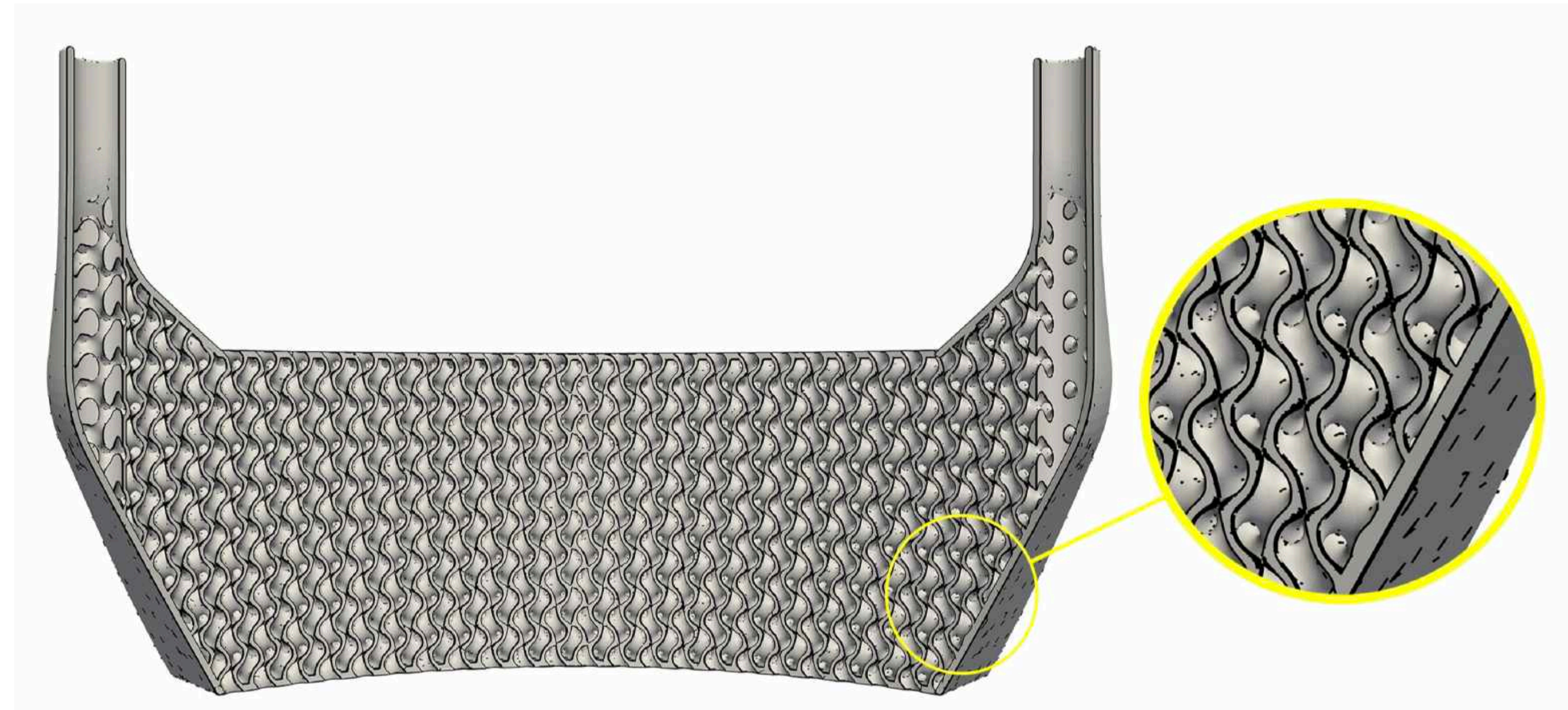
Volume Reduction



# Examples



Volume Reduction



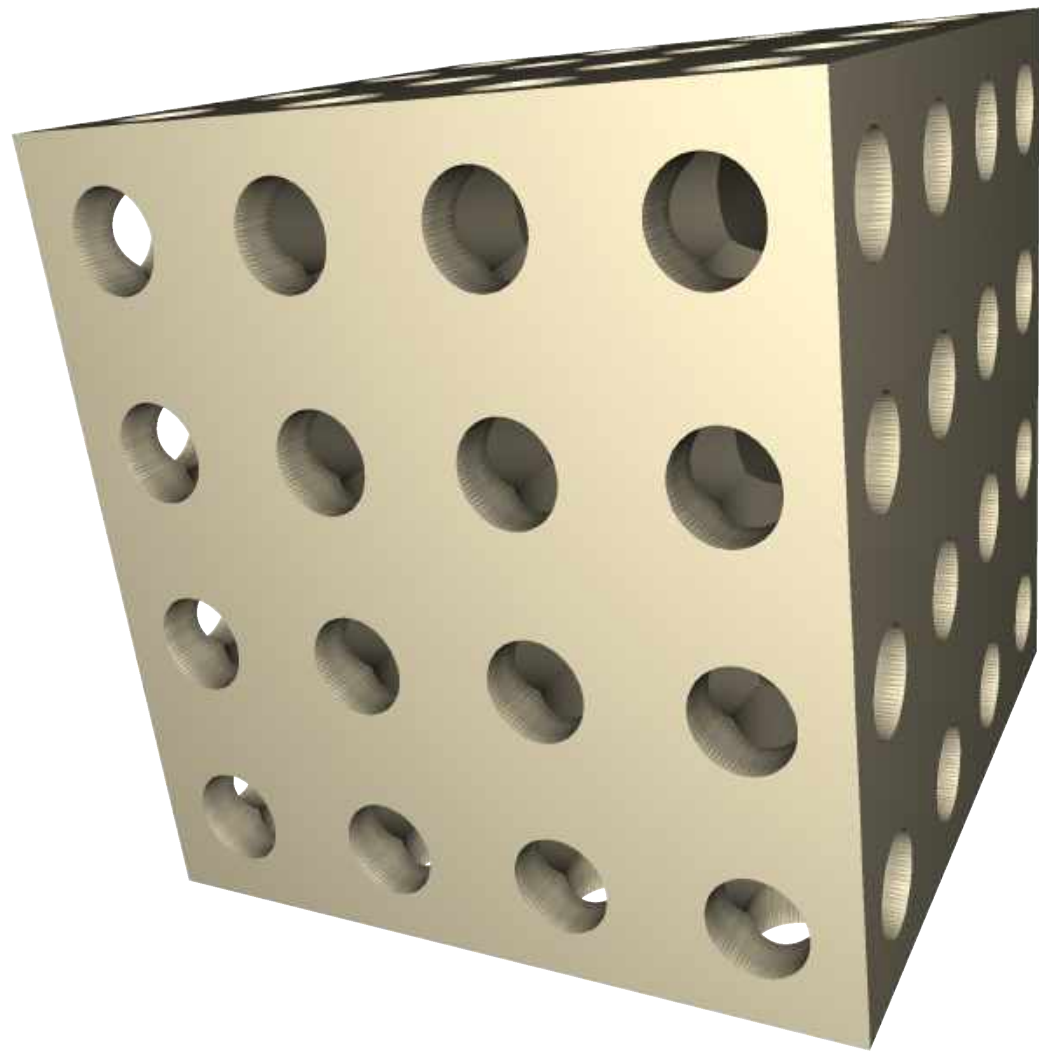
Heat Flux Increase



# Design Pipeline

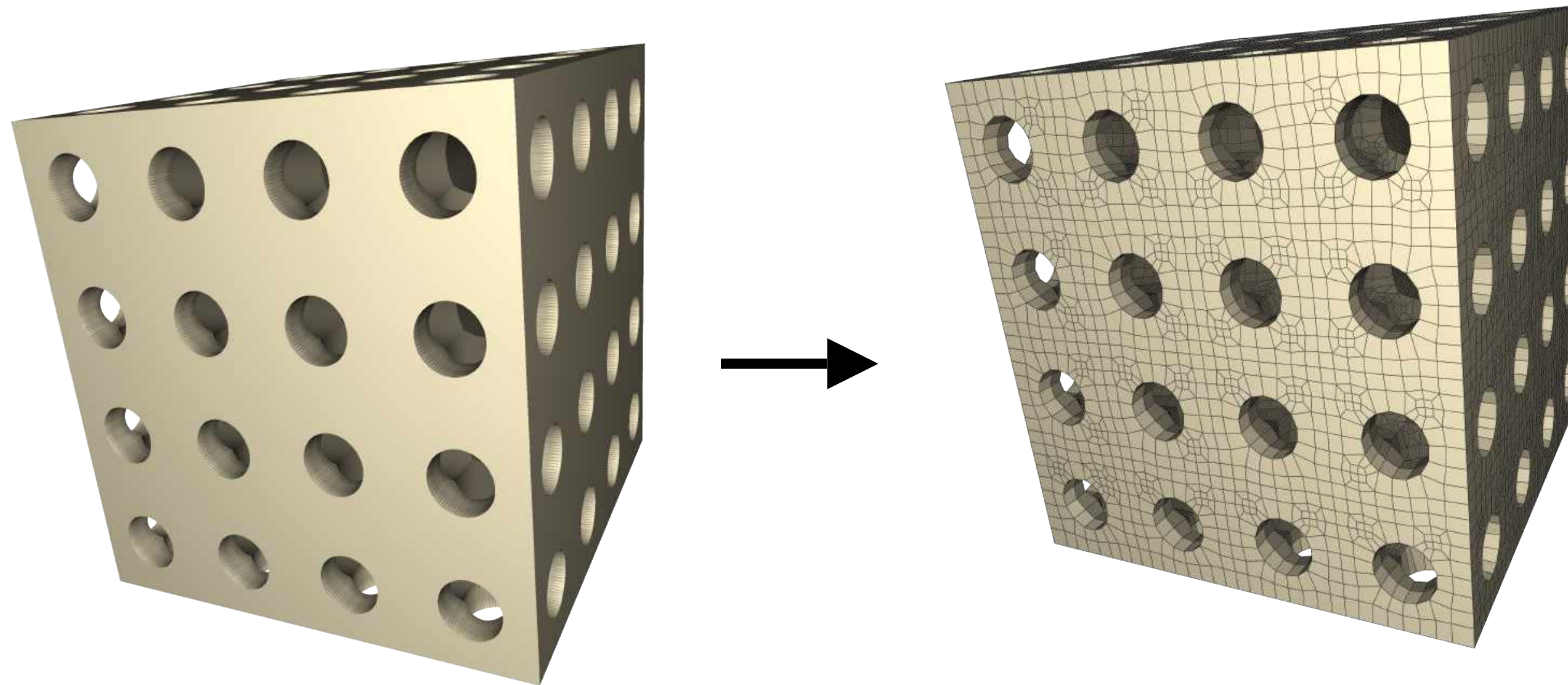


# Design Pipeline



Design

# Design Pipeline

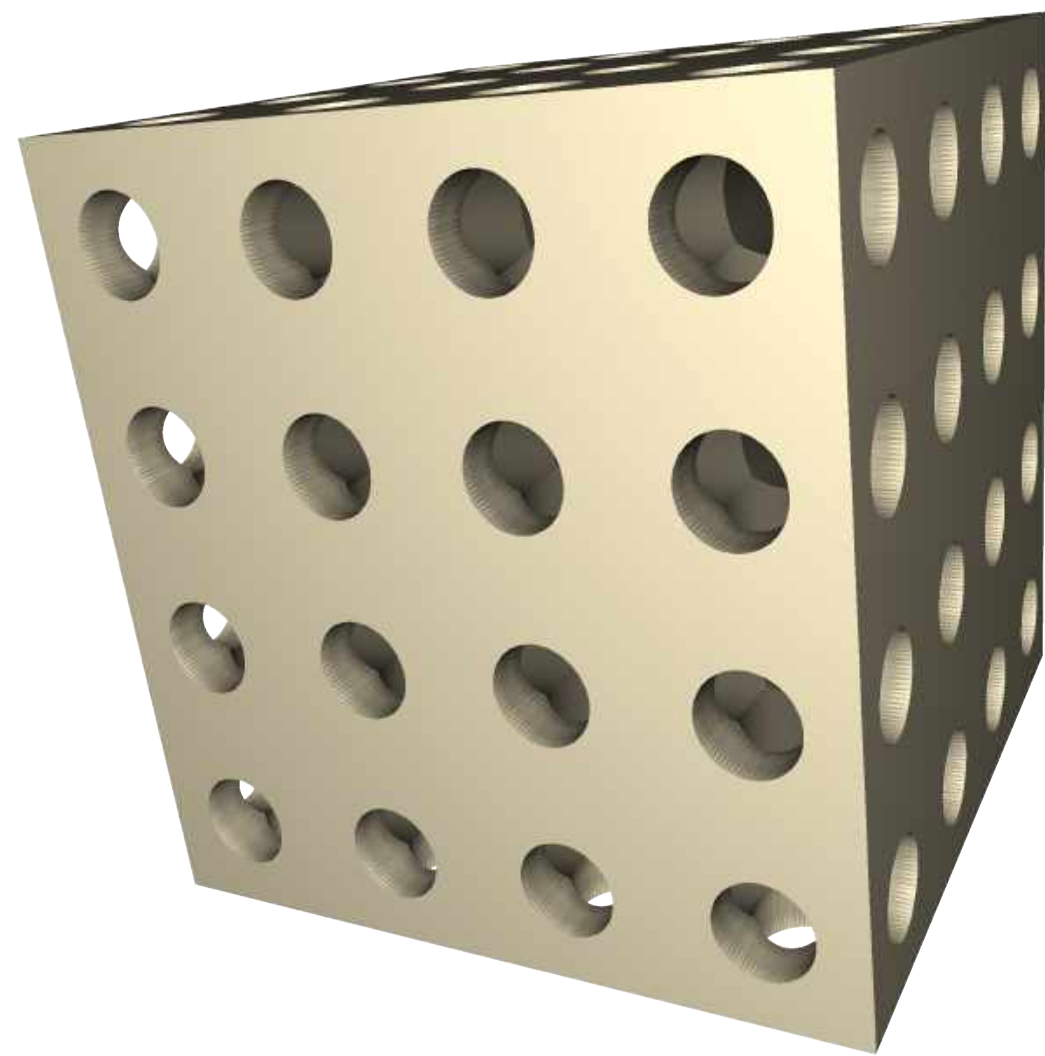


Design

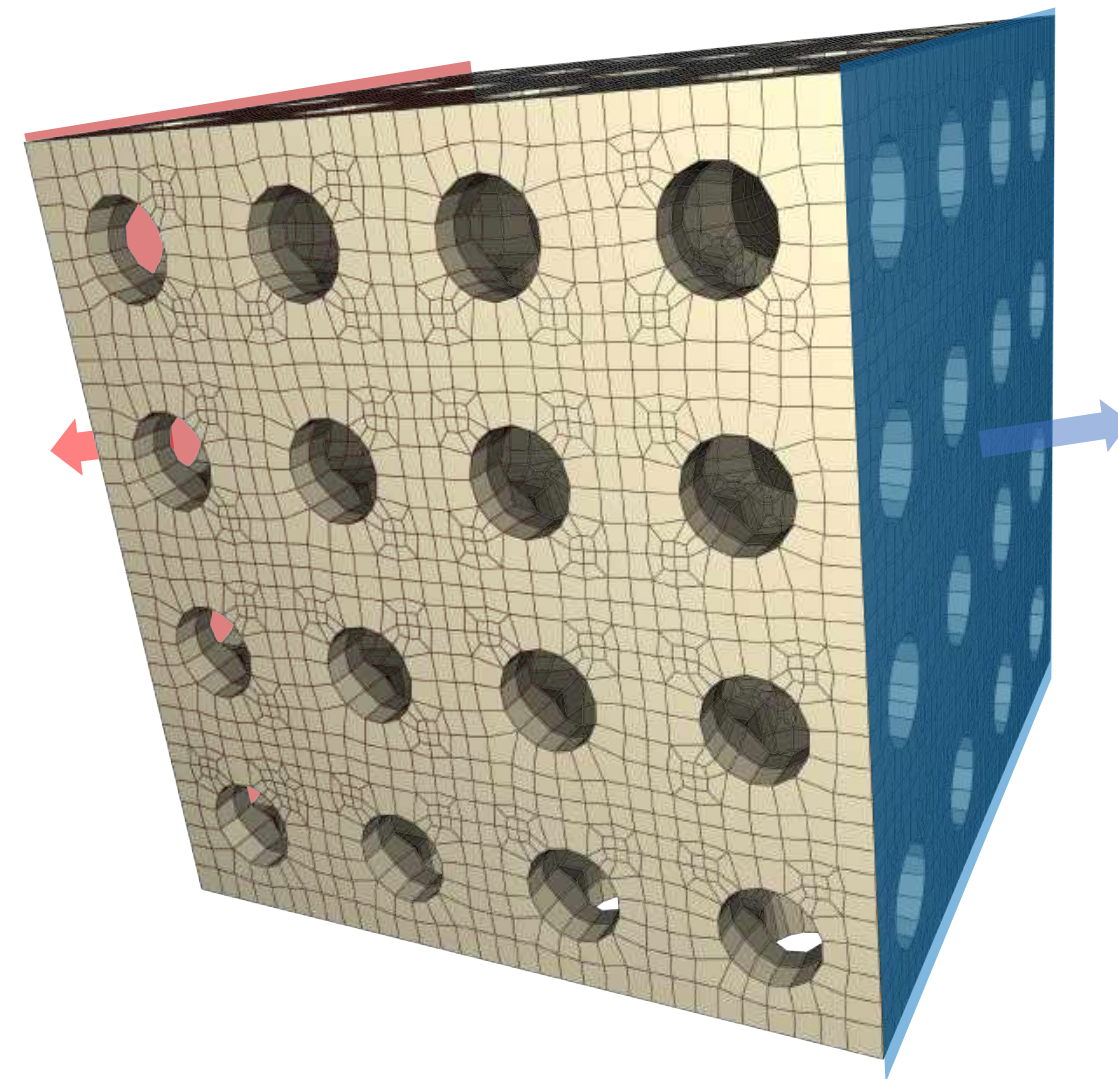
Discretization



# Design Pipeline



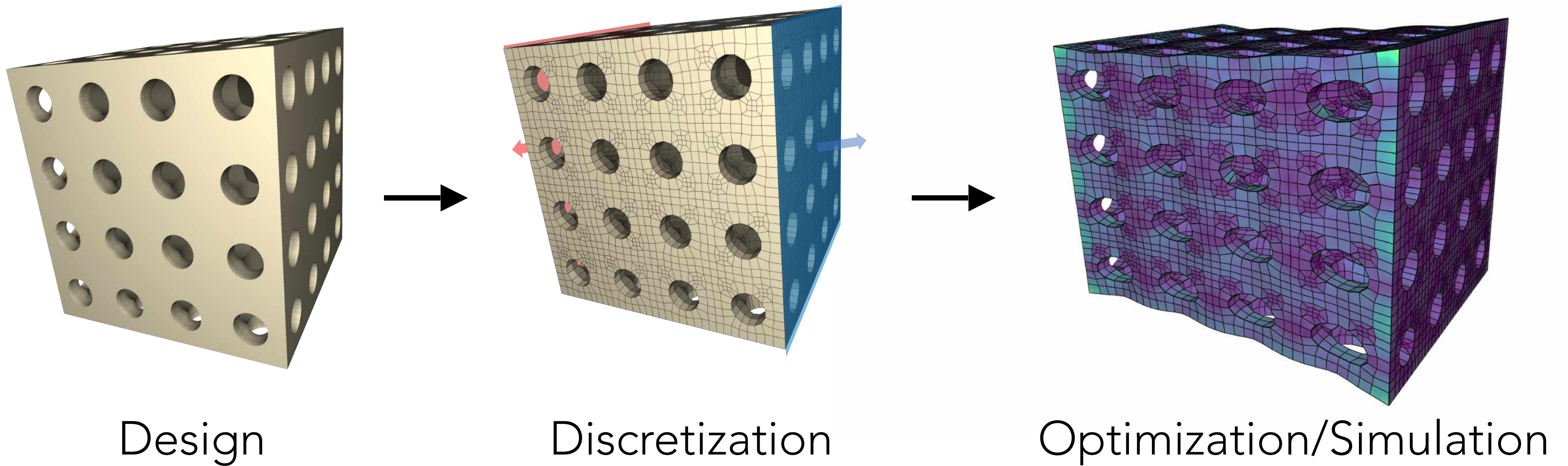
Design



Discretization

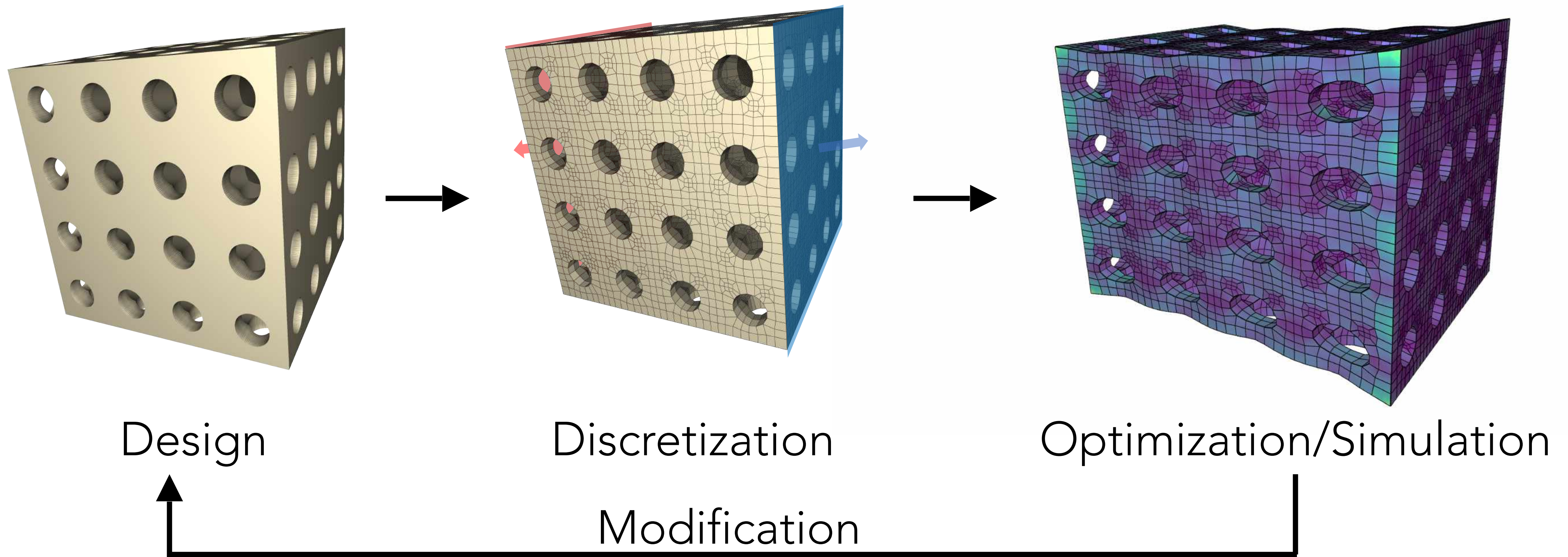


# Design Pipeline



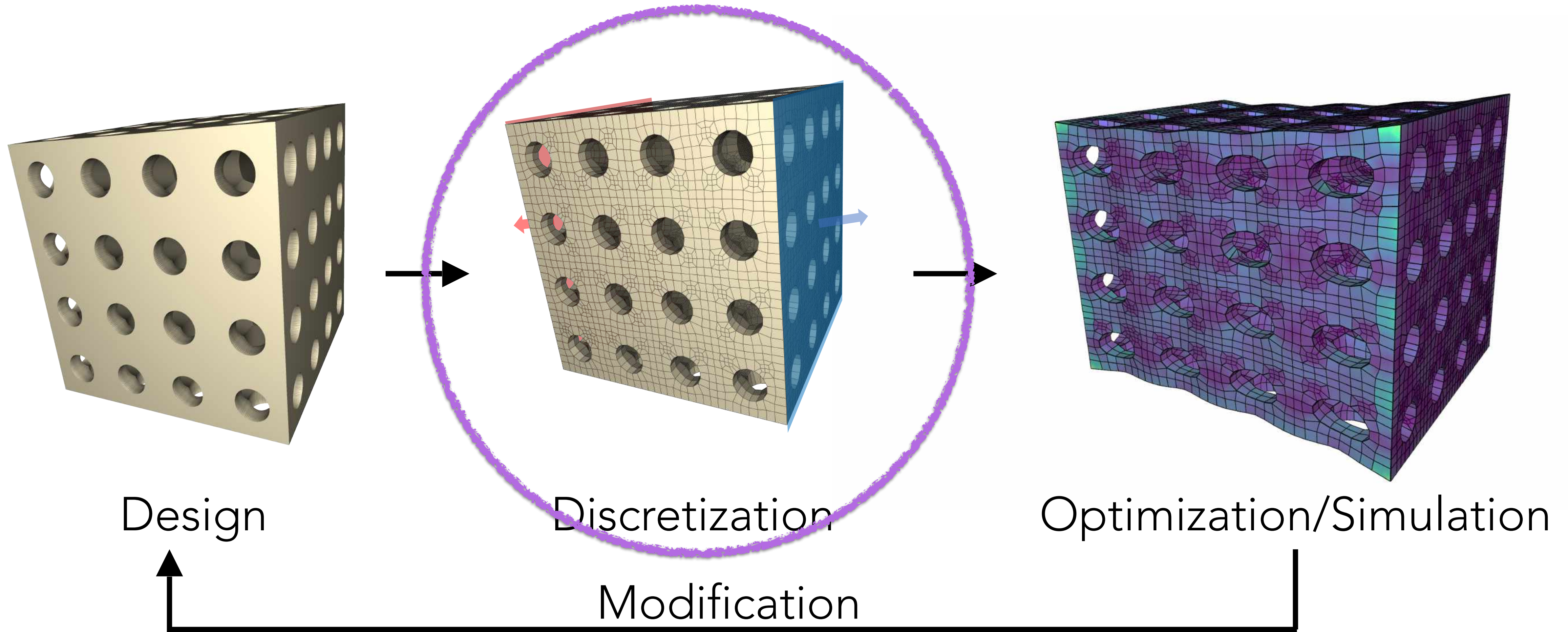


# Design Pipeline



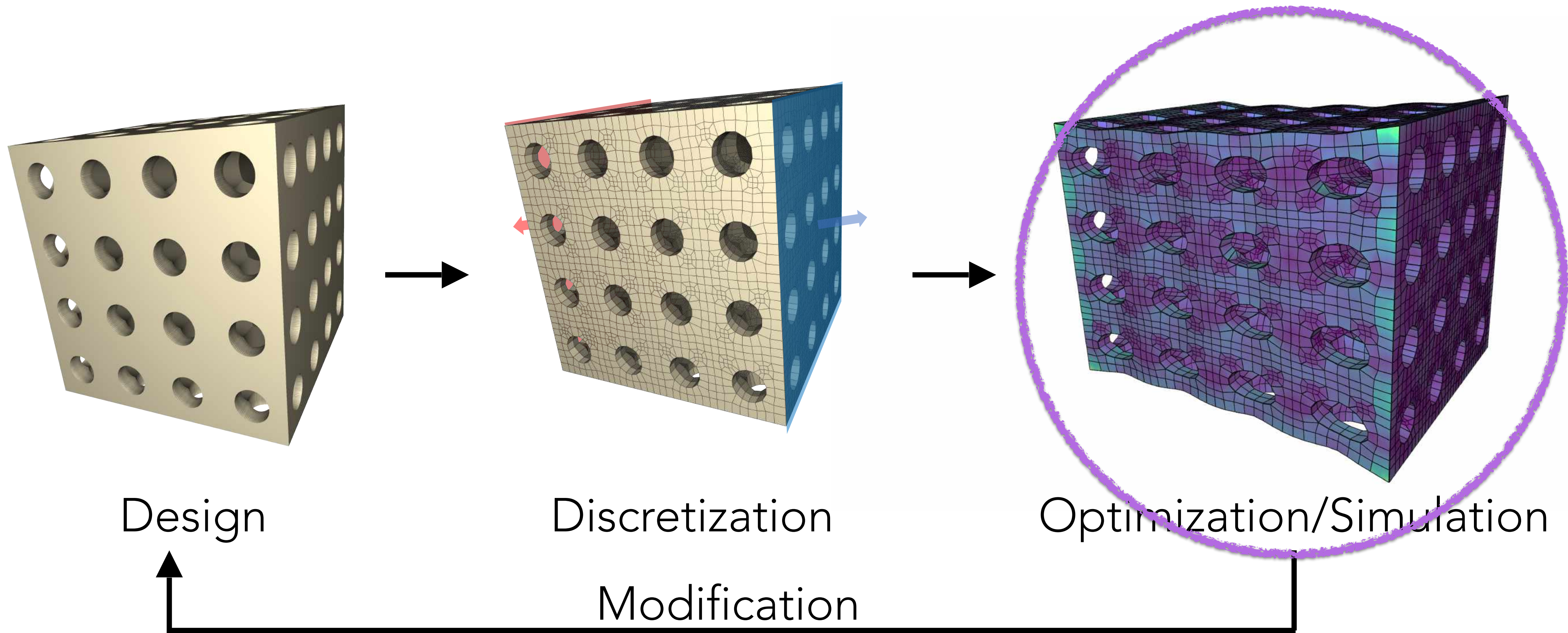


# Design Pipeline



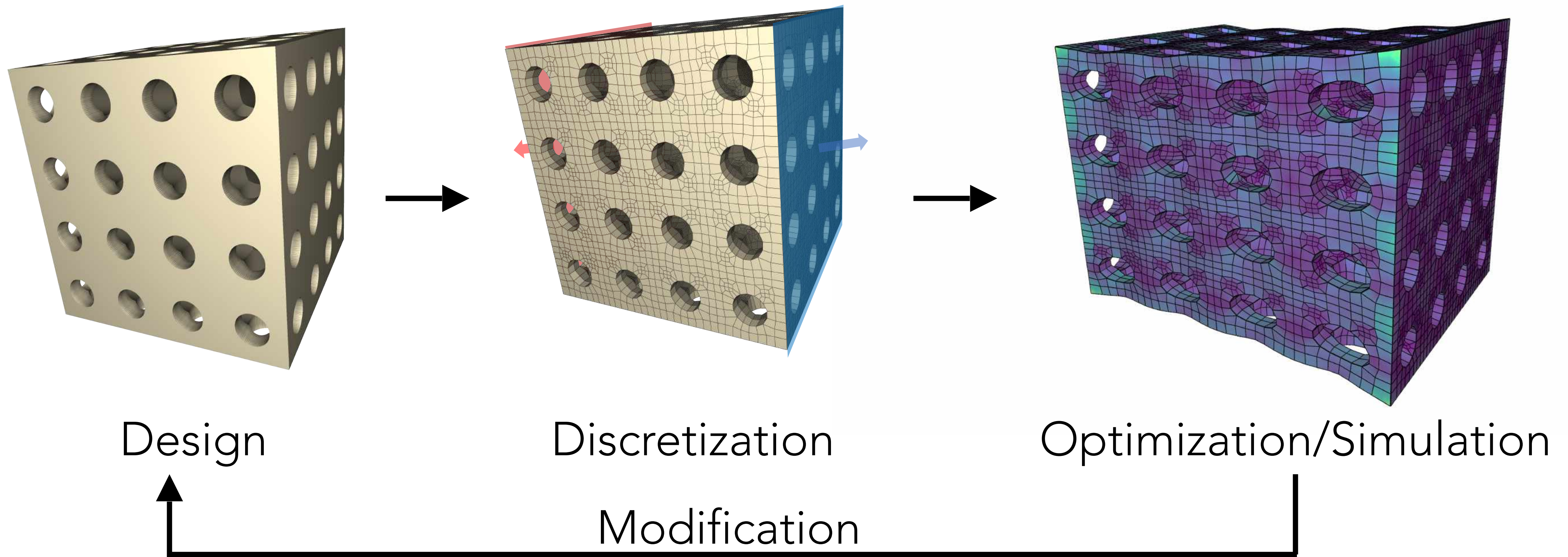


# Design Pipeline





# Design Pipeline











# Spreadsheets





Spreadsheets



Image Editing





Spreadsheets



Image Editing



Video Editing





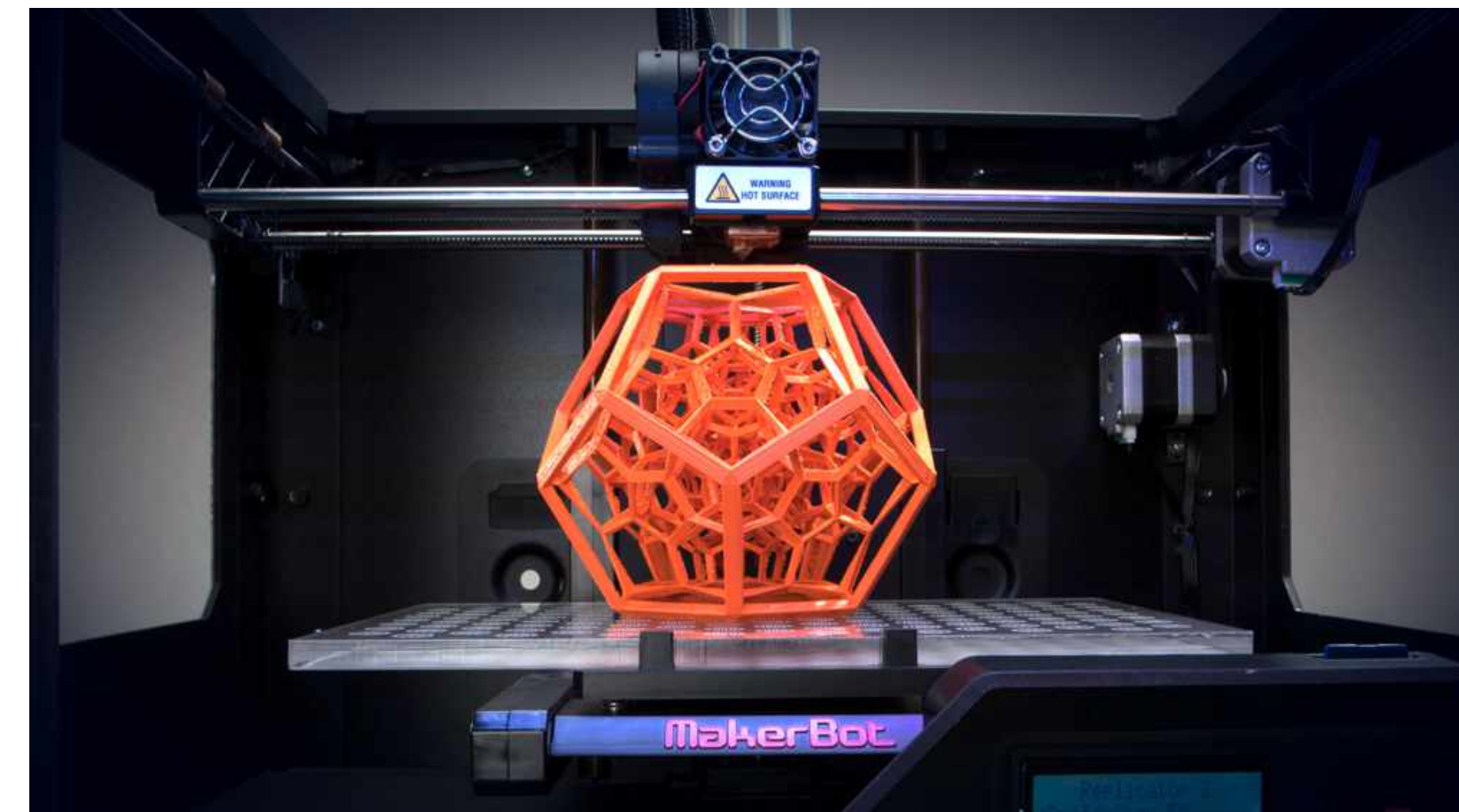
Spreadsheets



Image Editing



Video Editing



Geometry Editing



# 3D Geometry Is Challenging



# 3D Geometry Is Challenging

- A canonical representation does not exist



# 3D Geometry Is Challenging

- A canonical representation does not exist
- Most operations are not closed under the floating point representation:



# 3D Geometry Is Challenging

- A canonical representation does not exist
- Most operations are not closed under the floating point representation:
- Not handling this results in lack of robustness



# 3D Geometry Is Challenging

- A canonical representation does not exist
- Most operations are not closed under the floating point representation:
  - Not handling this results in lack of robustness
  - Handling it increases dramatically the algorithmic complexity, increasing the chances of implementation errors (which are a nightmare to debug)

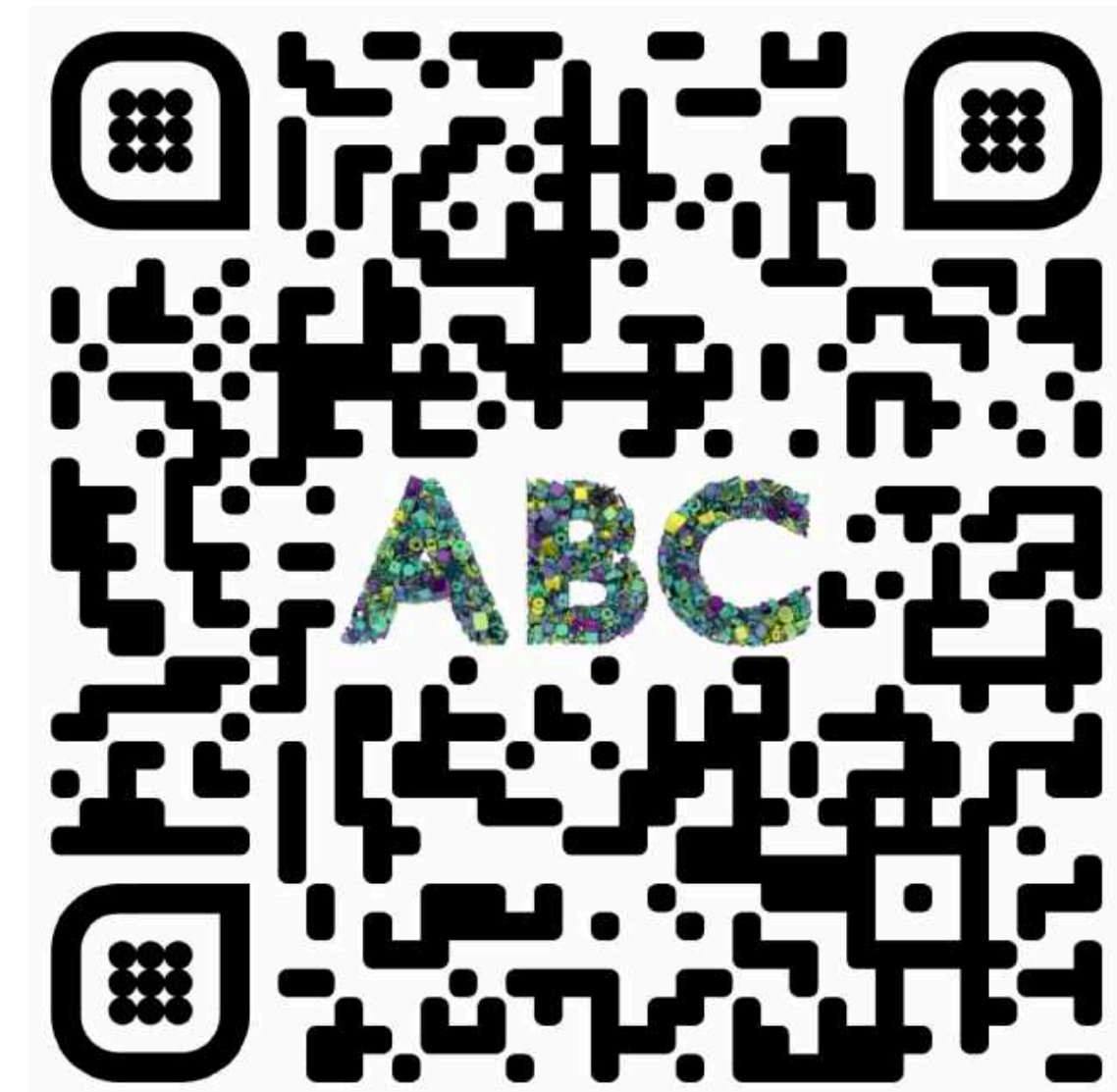


# Normal Computation



## ABC: A Big CAD Model Dataset For Geometric Deep Learning

We introduce ABC-Dataset, a collection of one million Computer-Aided Design (CAD) models for research of geometric deep learning methods and applications. Each model is a collection of explicitly parametrized curves and surfaces, providing ground truth for differential quantities, patch segmentation, geometric feature detection, and shape reconstruction. Sampling the parametric descriptions of surfaces and curves allows generating data in different formats and resolutions, enabling fair comparisons for a wide range of geometric learning algorithms. As a use case for our dataset, we perform a large-scale benchmark for estimation of surface normals, comparing existing data driven methods and evaluating their performance against both the ground truth and traditional normal estimation methods.



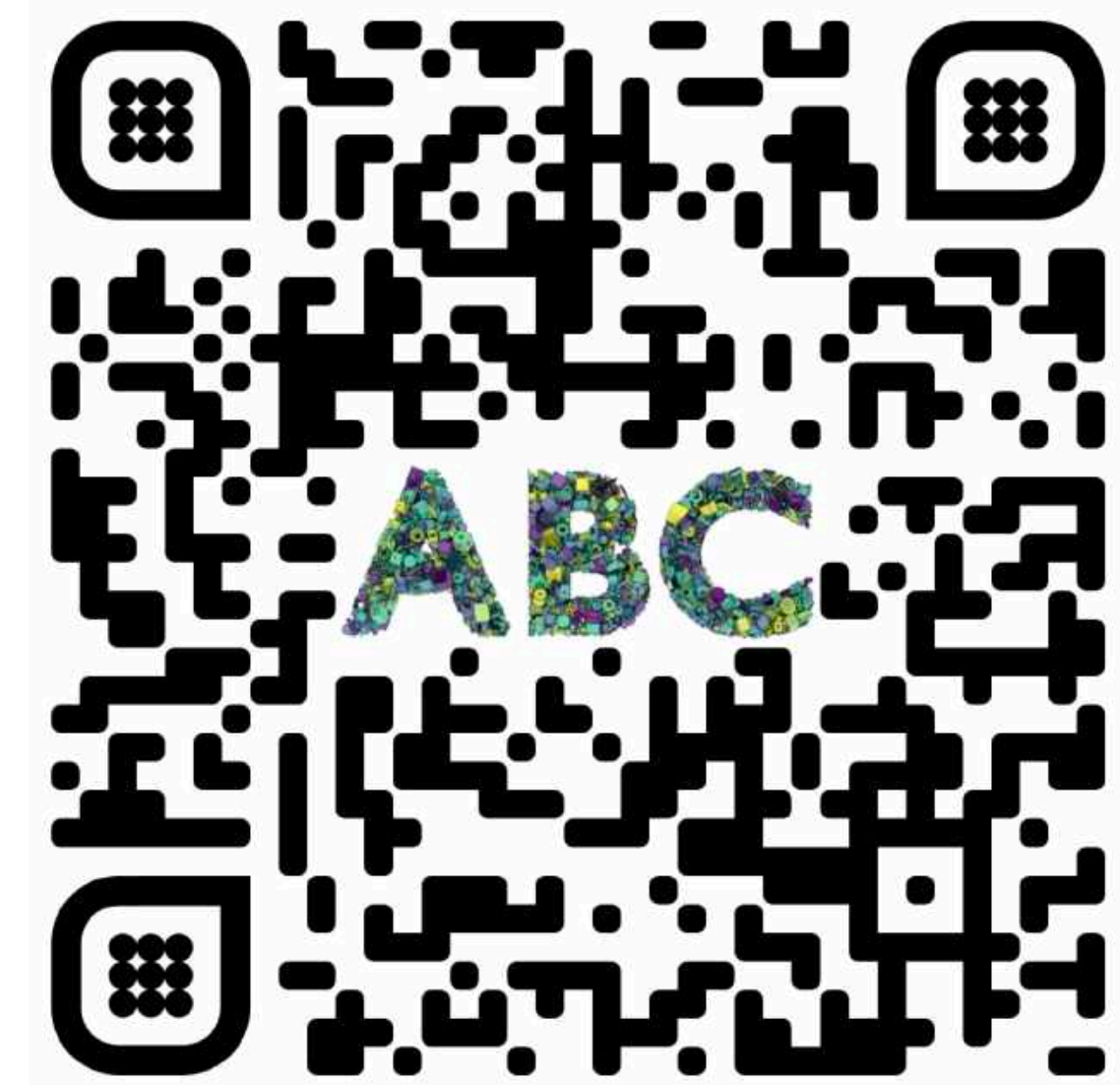


# Normal Computation



## ABC: A Big CAD Model Dataset For Geometric Deep Learning

We introduce ABC-Dataset, a collection of one million Computer-Aided Design (CAD) models for research of geometric deep learning methods and applications. Each model is a collection of explicitly parametrized curves and surfaces, providing ground truth for differential quantities, patch segmentation, geometric feature detection, and shape reconstruction. Sampling the parametric descriptions of surfaces and curves allows generating data in different formats and resolutions, enabling fair comparisons for a wide range of geometric learning algorithms. As a use case for our dataset, we perform a large-scale benchmark for estimation of surface normals, comparing existing data driven methods and evaluating their performance against both the ground truth and traditional normal estimation methods.



*Loop over faces*

```
N(:, :) = 0;
```

```
for face f
```

```
    for corner vertex i
```

```
        N(i) += f's normal
```

```
for vertex i
```

```
    N(i).normalize
```

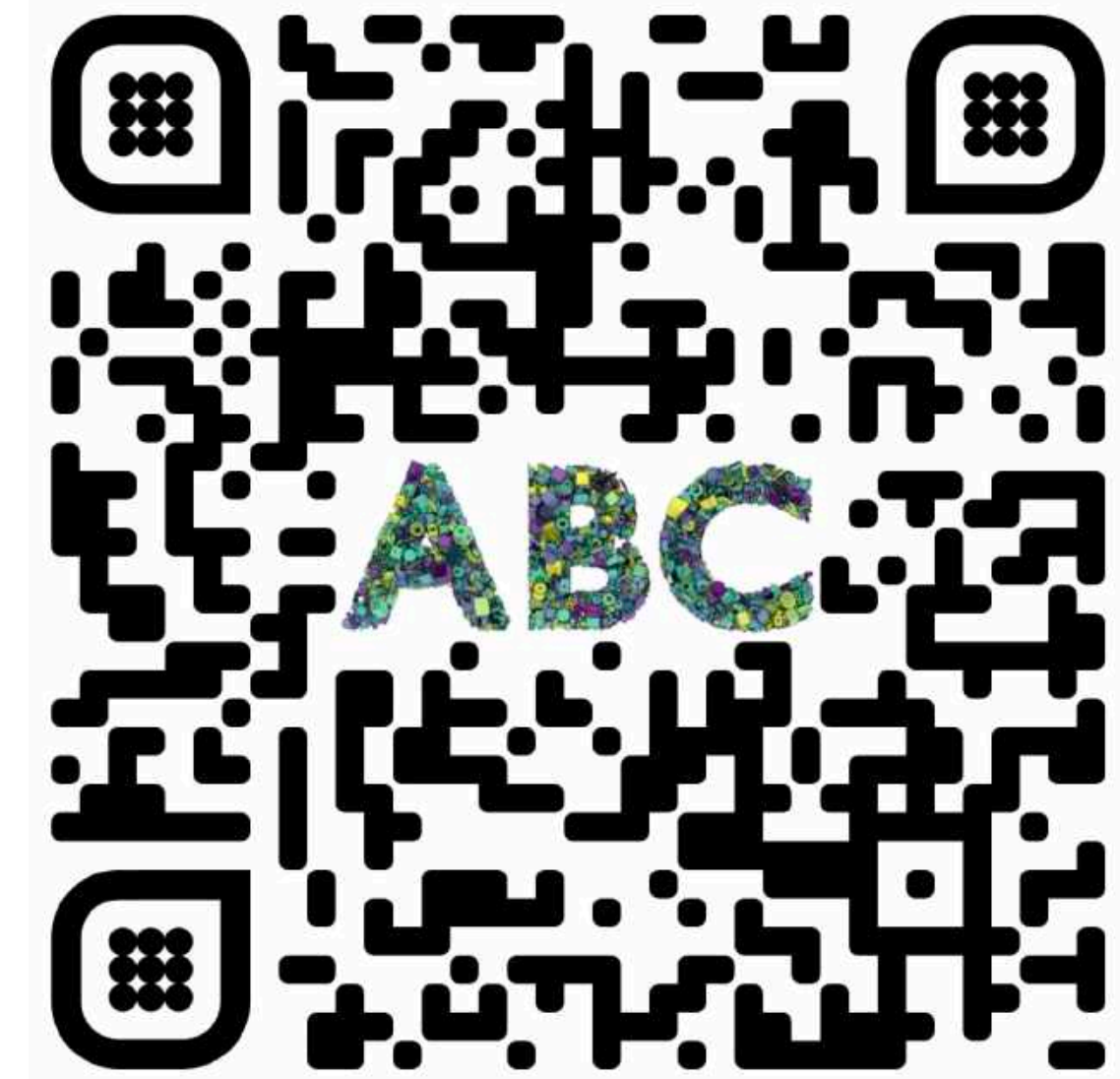


# Normal Computation



## ABC: A Big CAD Model Dataset For Geometric Deep Learning

We introduce ABC-Dataset, a collection of one million Computer-Aided Design (CAD) models for research of geometric deep learning methods and applications. Each model is a collection of explicitly parametrized curves and surfaces, providing ground truth for differential quantities, patch segmentation, geometric feature detection, and shape reconstruction. Sampling the parametric descriptions of surfaces and curves allows generating data in different formats and resolutions, enabling fair comparisons for a wide range of geometric learning algorithms. As a use case for our dataset, we perform a large-scale benchmark for estimation of surface normals, comparing existing data driven methods and evaluating their performance against both the ground truth and traditional normal estimation methods.



*Loop over faces*

$N(:, :) = 0;$

for face  $f$

    for corner vertex  $i$

$N(i) += f.s \text{ normal}$

for vertex  $i$

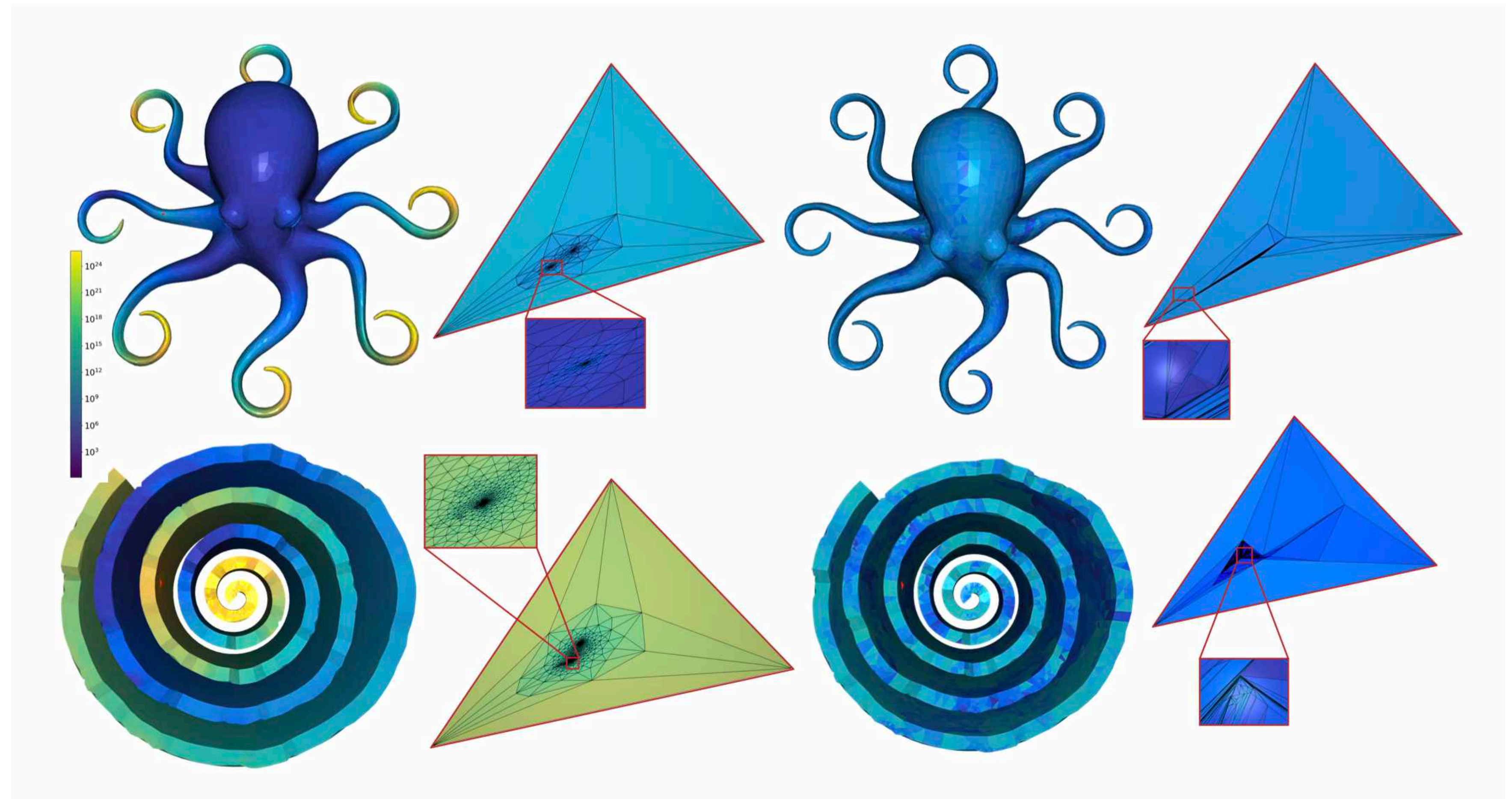
$N(i).normalize$



# Tutte Embedding



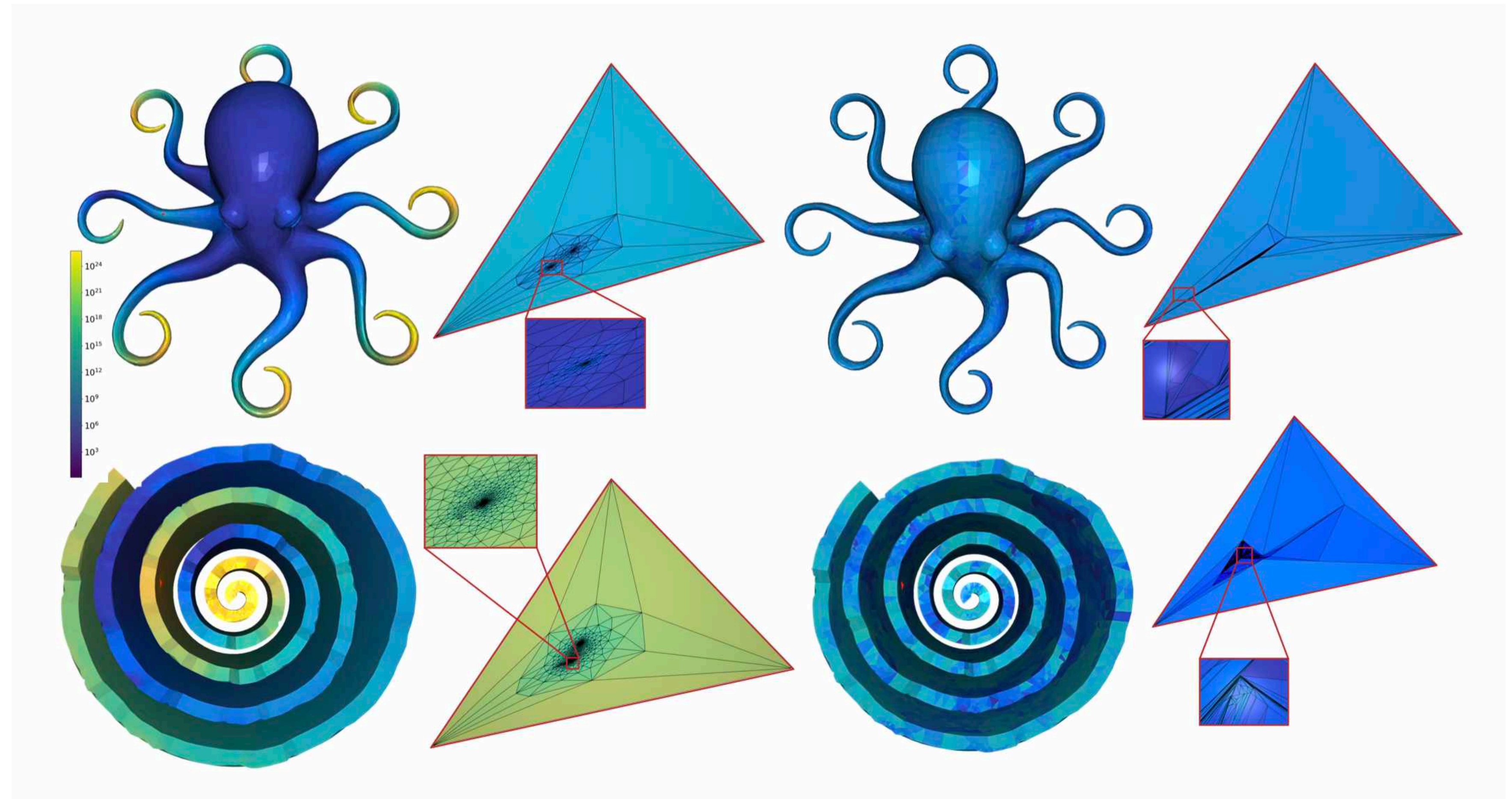
# Tutte Embedding





# Tutte Embedding

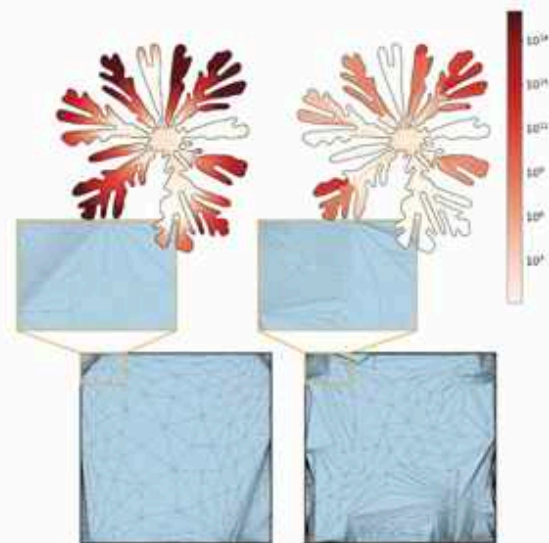
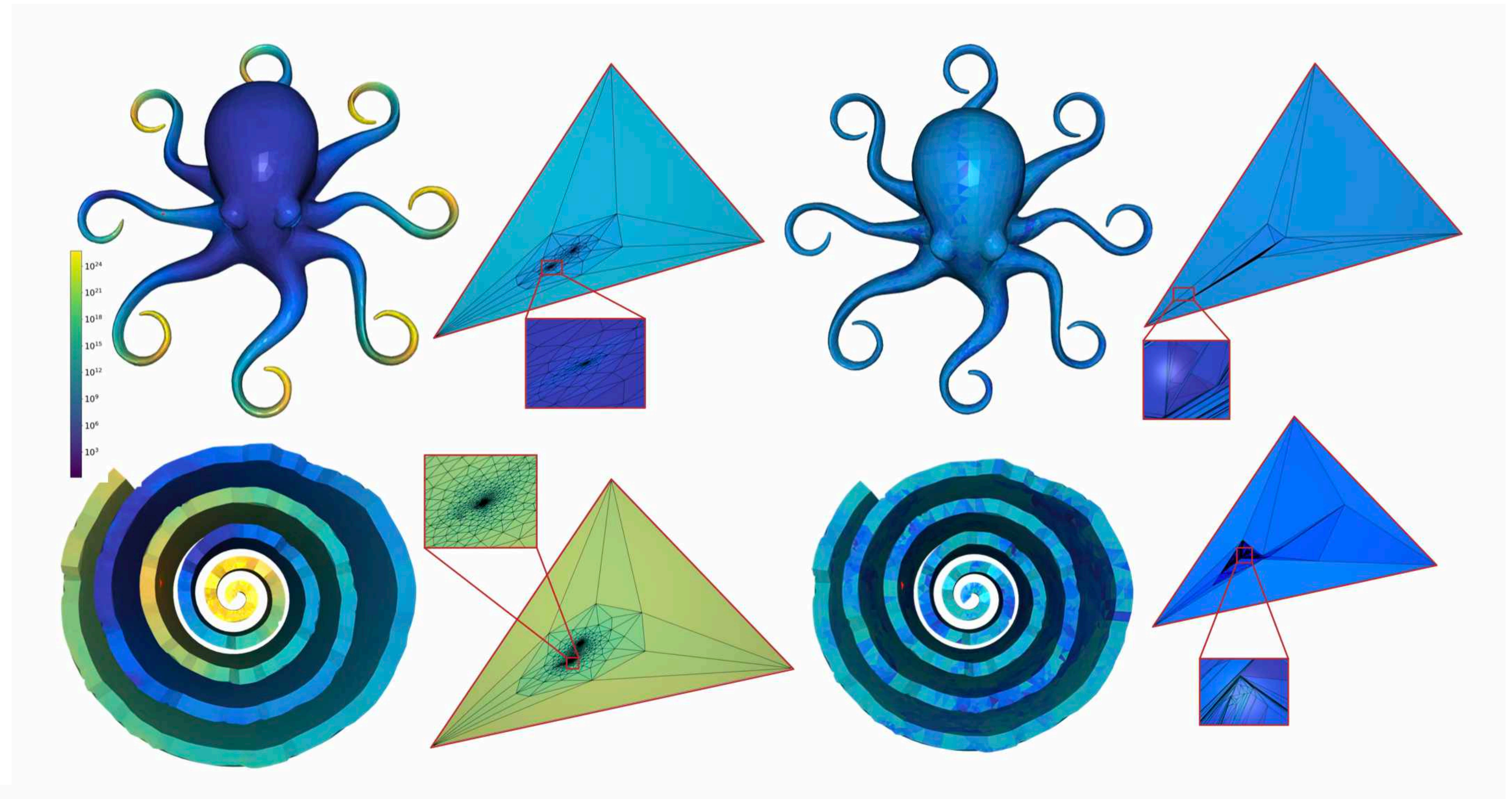
- 80/2718 failures on Genus 0 models in Thingy 10k





# Tutte Embedding

- 80/2718 failures on Genus 0 models in Thingy 10k



## Progressive Embedding

Hanxiao Shen, Zhongshi Jiang, Denis Zorin, Daniele Panozzo,

ACM Transaction on Graphics (SIGGRAPH), 2019

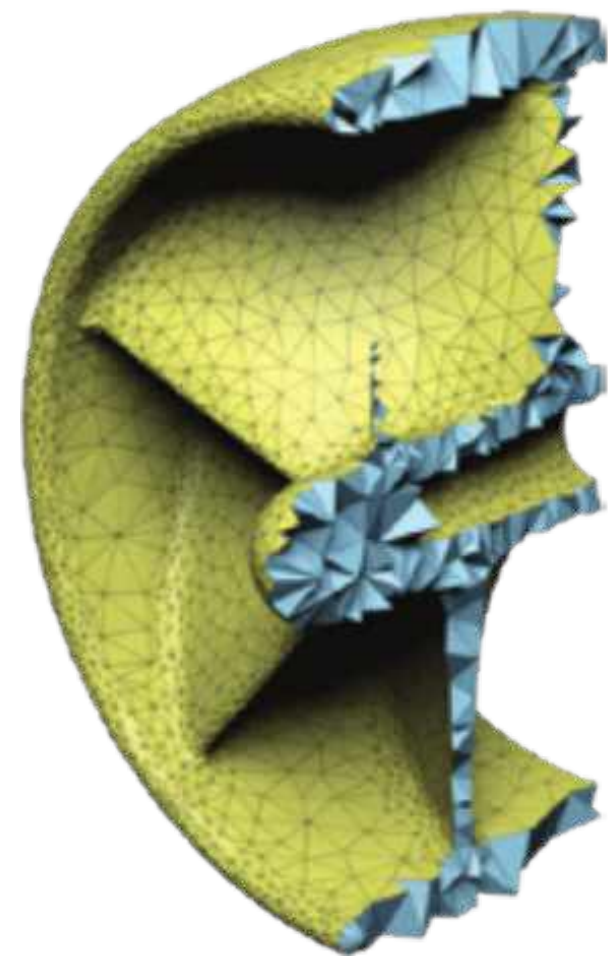
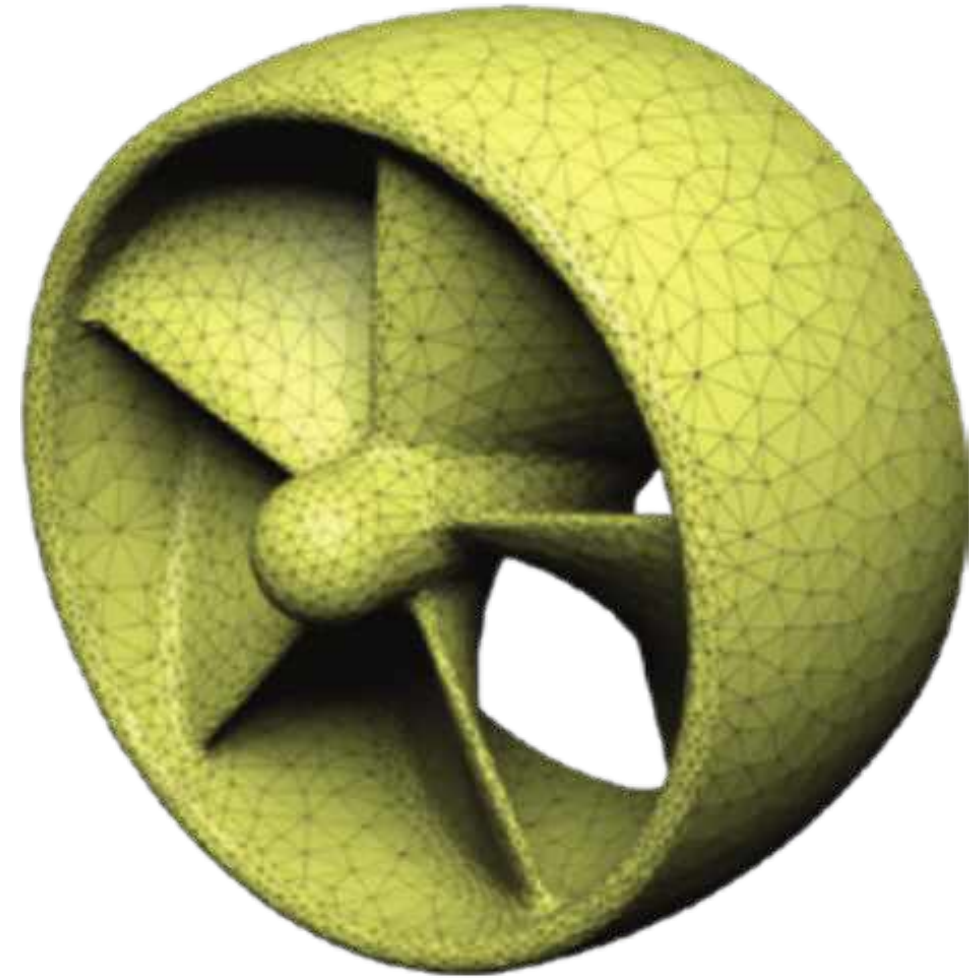
[\[Paper\]](#) [\[Code\]](#) [\[Data\]](#)



# Tetrahedral Meshing



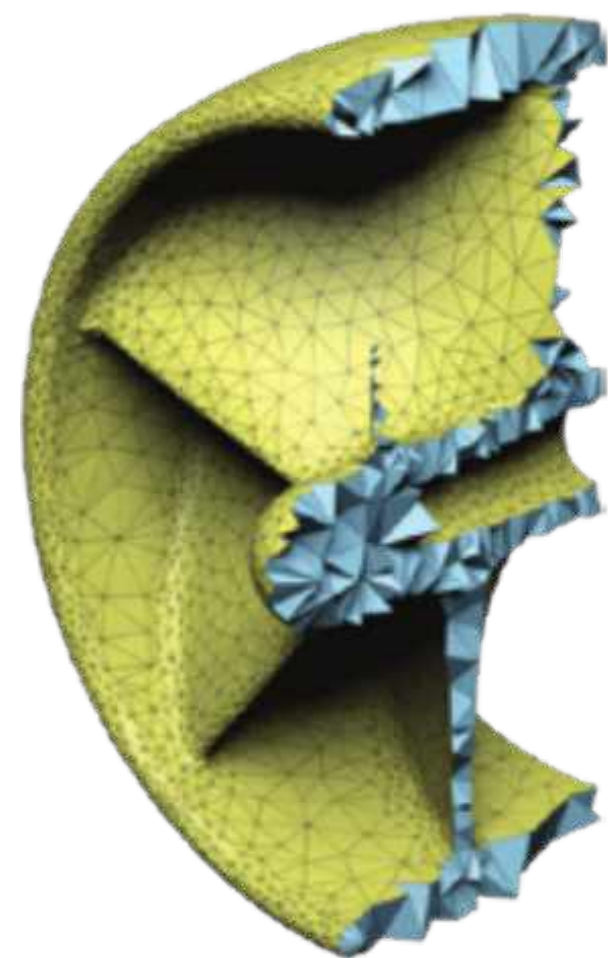
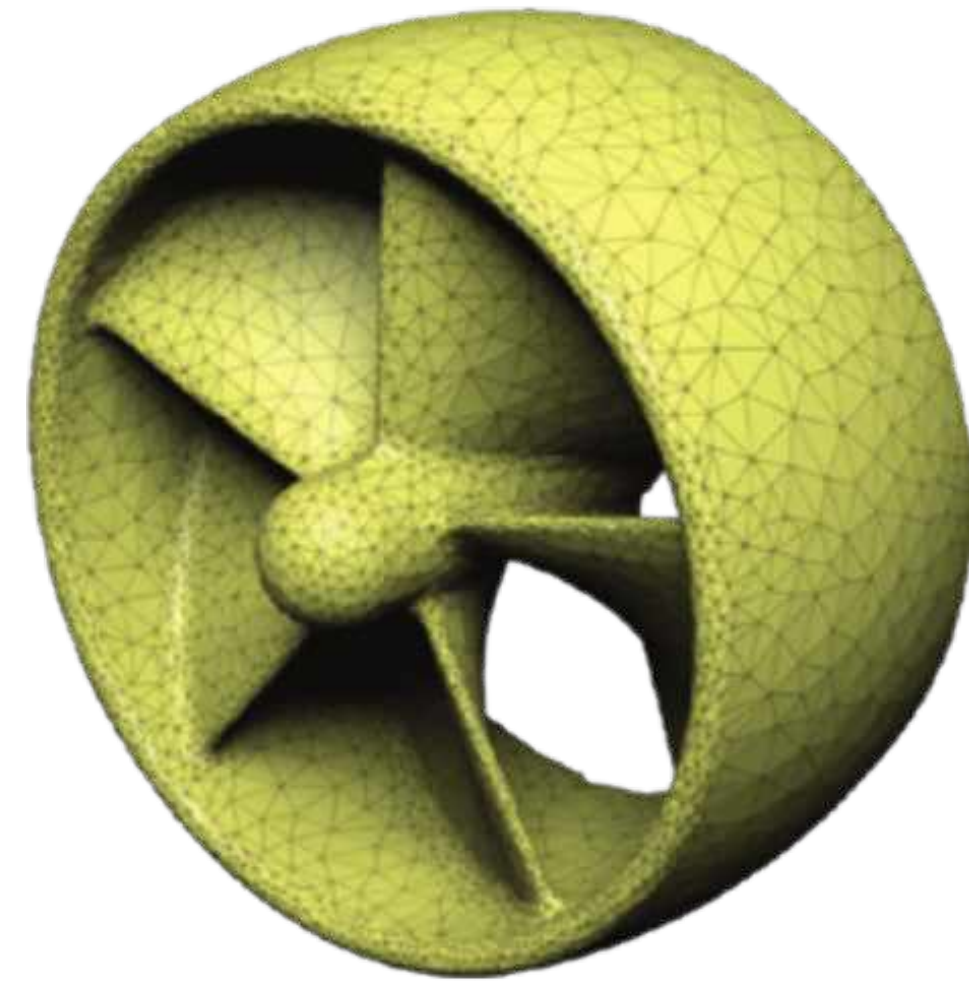
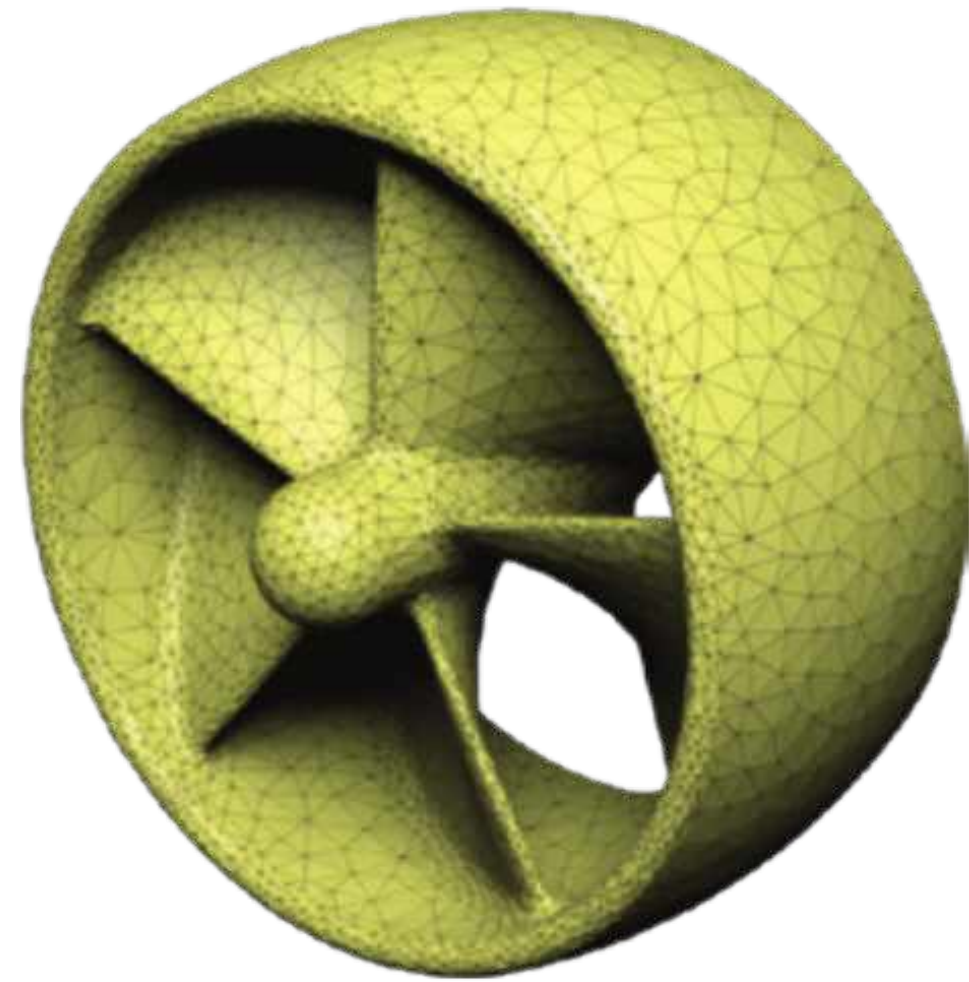
# Tetrahedral Meshing



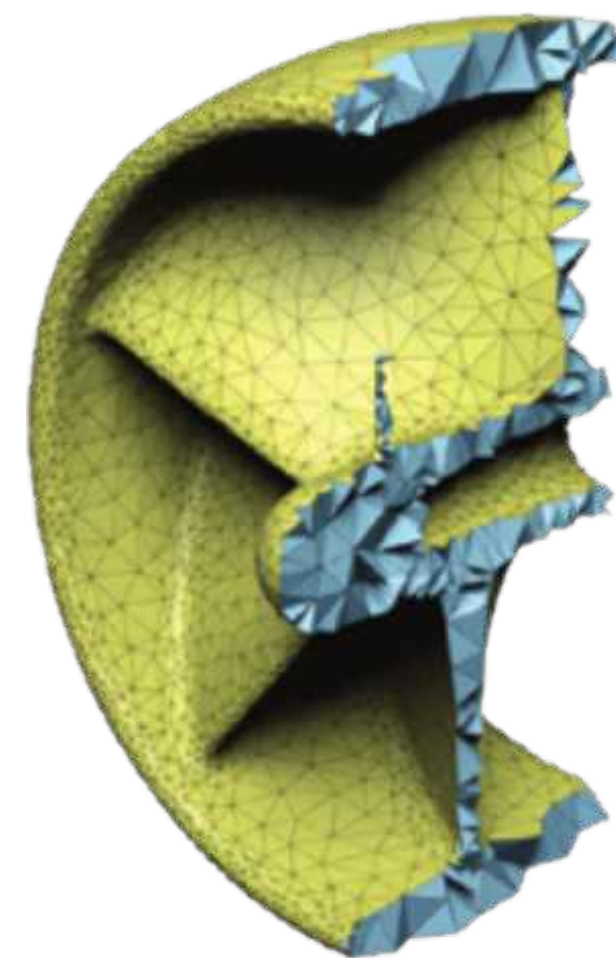
CGAL



# Tetrahedral Meshing



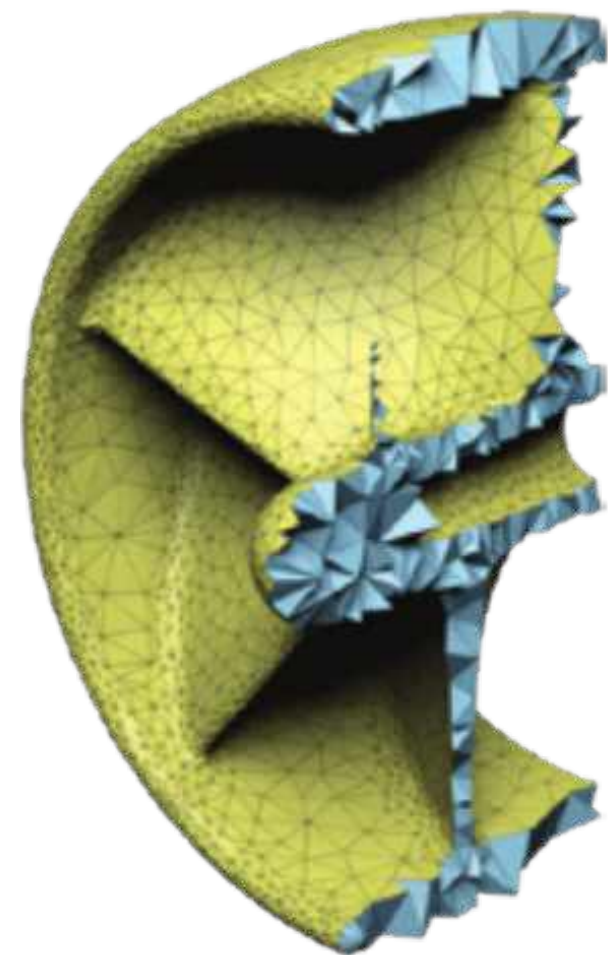
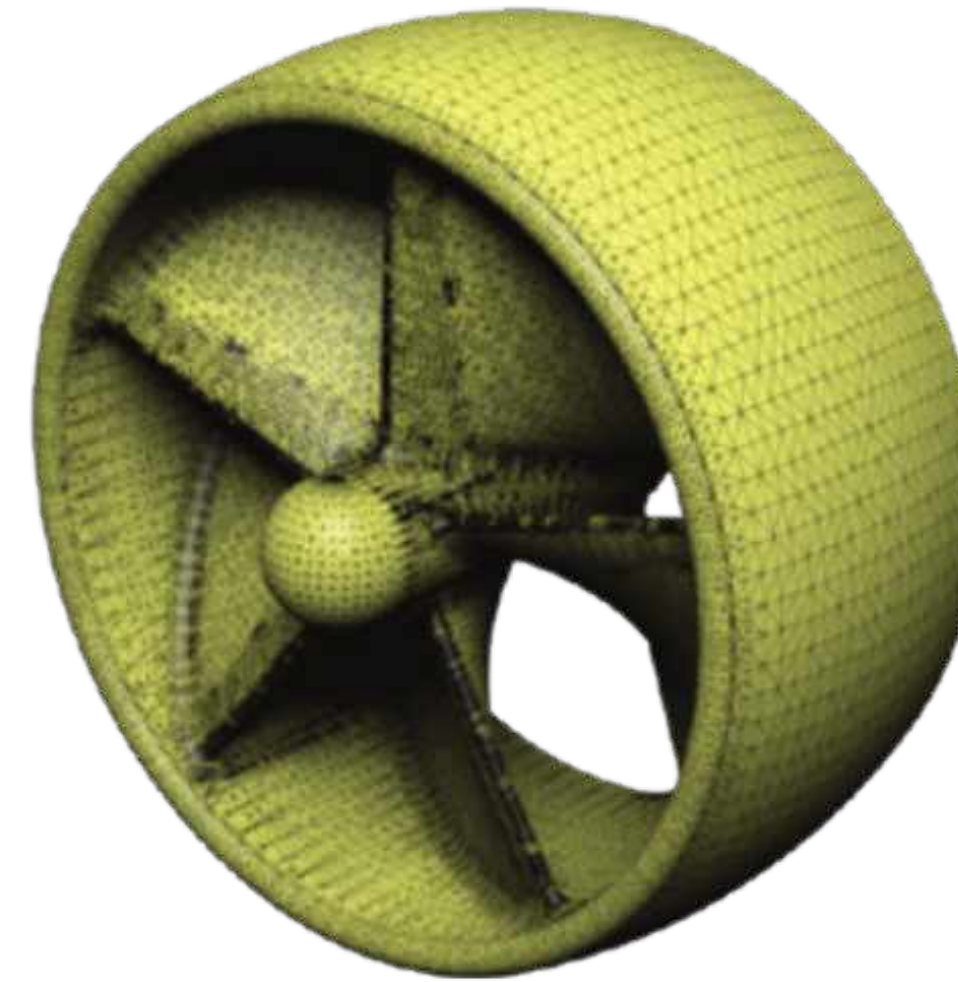
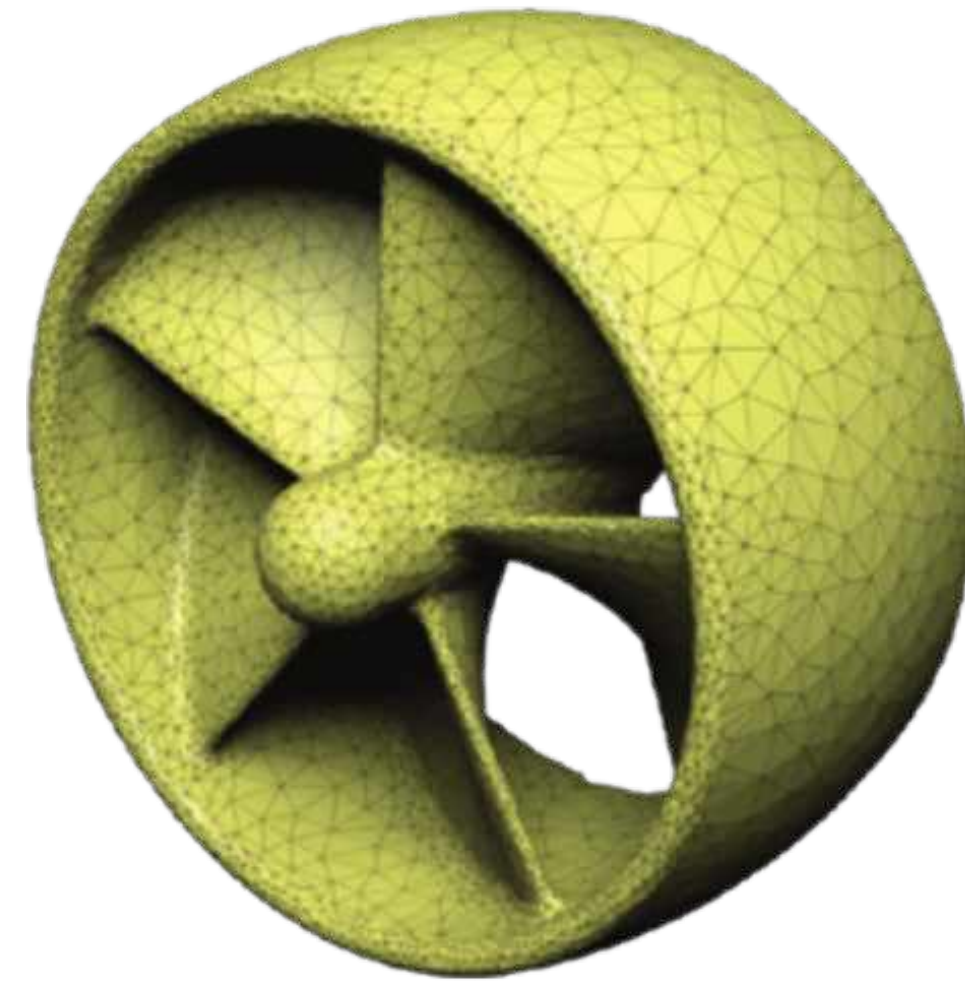
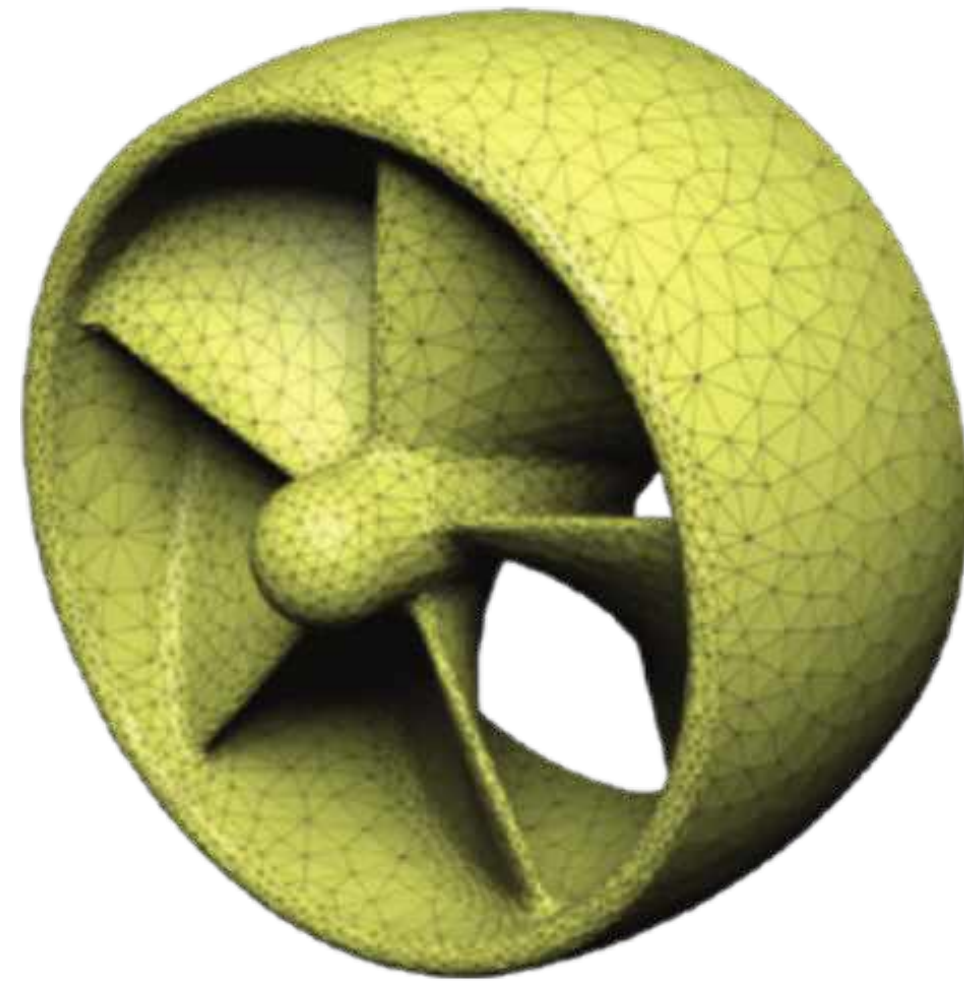
CGAL



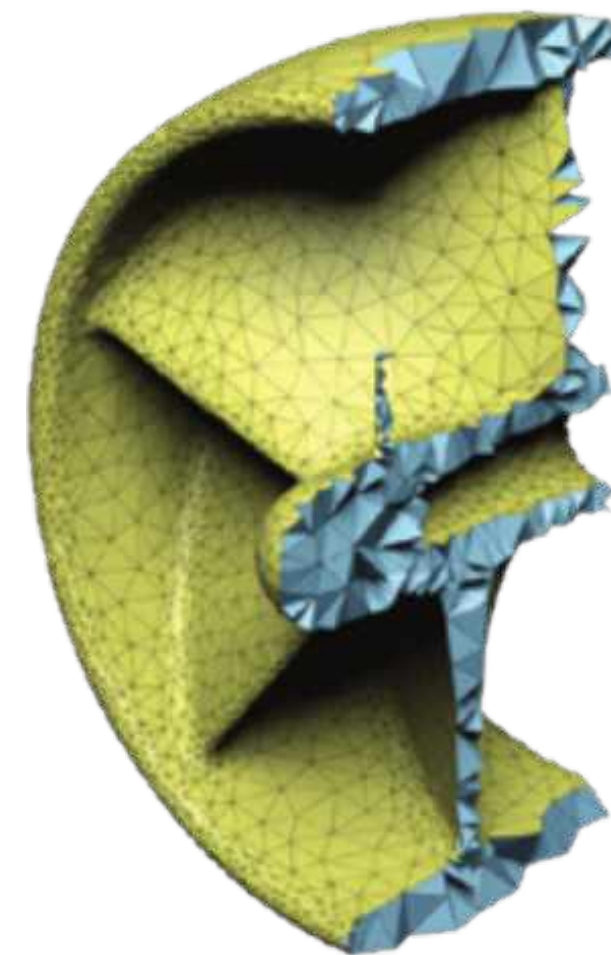
CGAL  
(without feature)



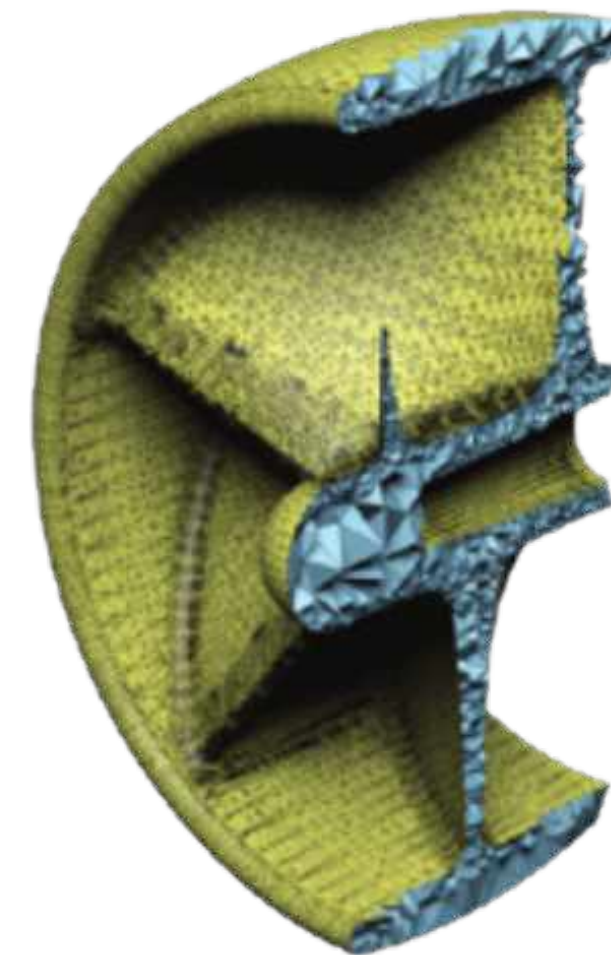
# Tetrahedral Meshing



CGAL



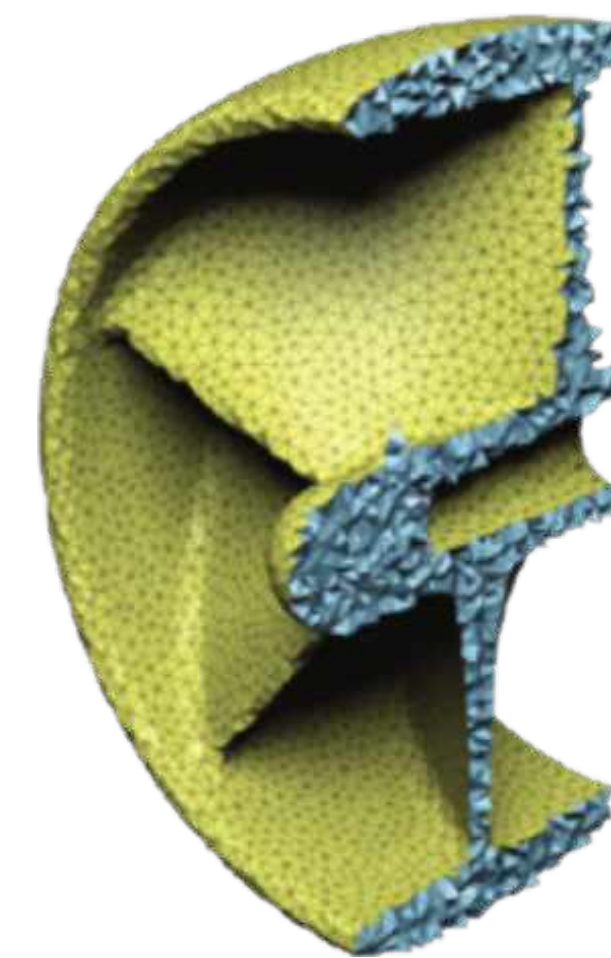
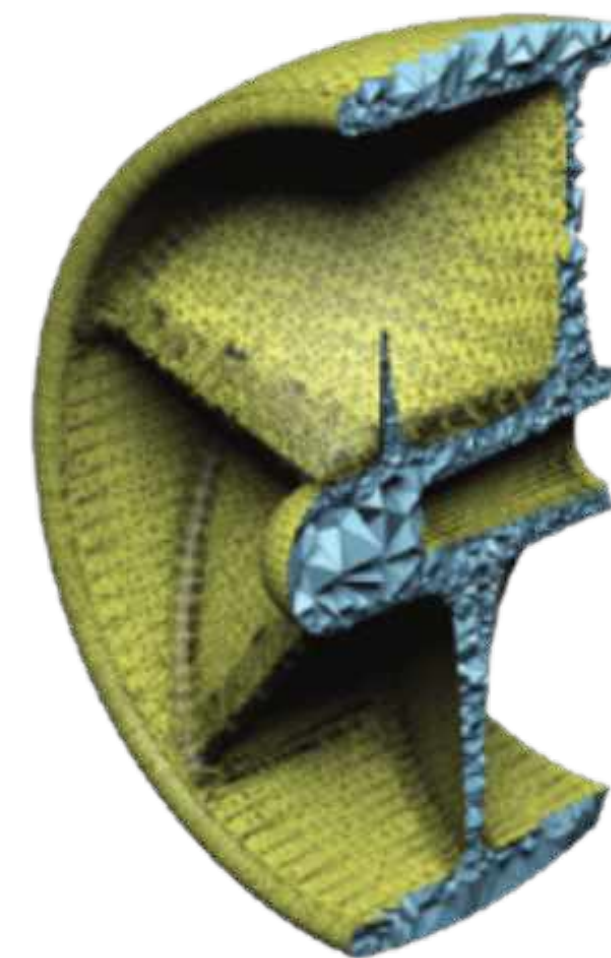
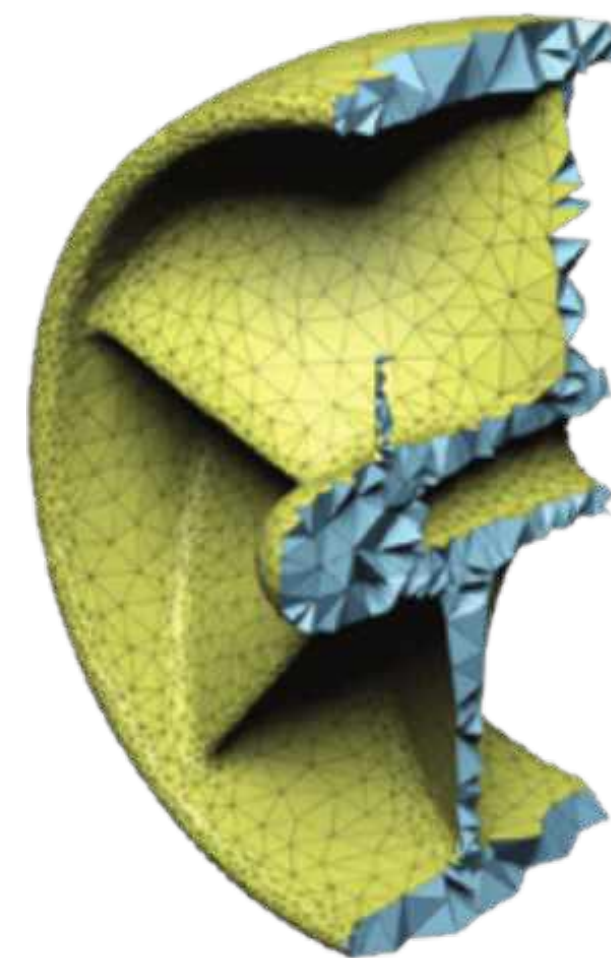
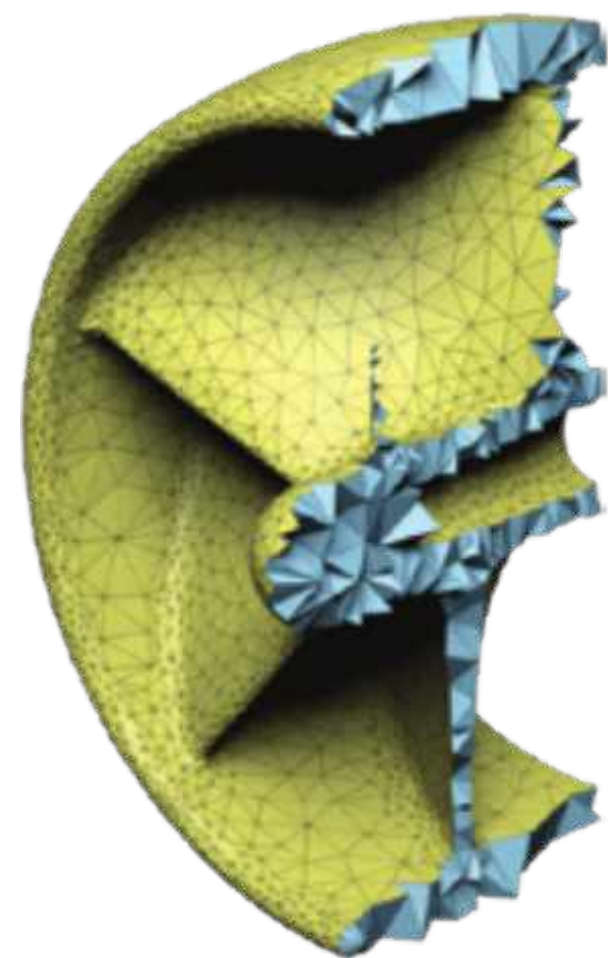
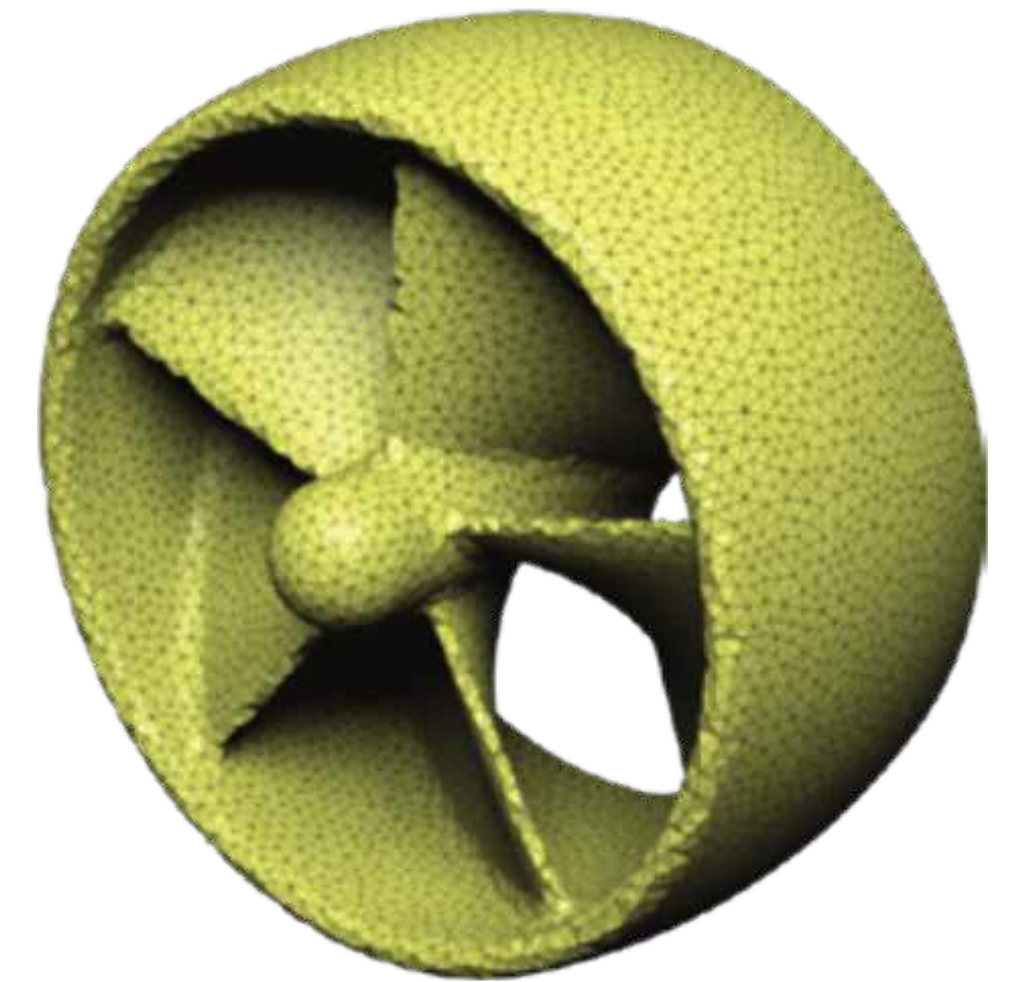
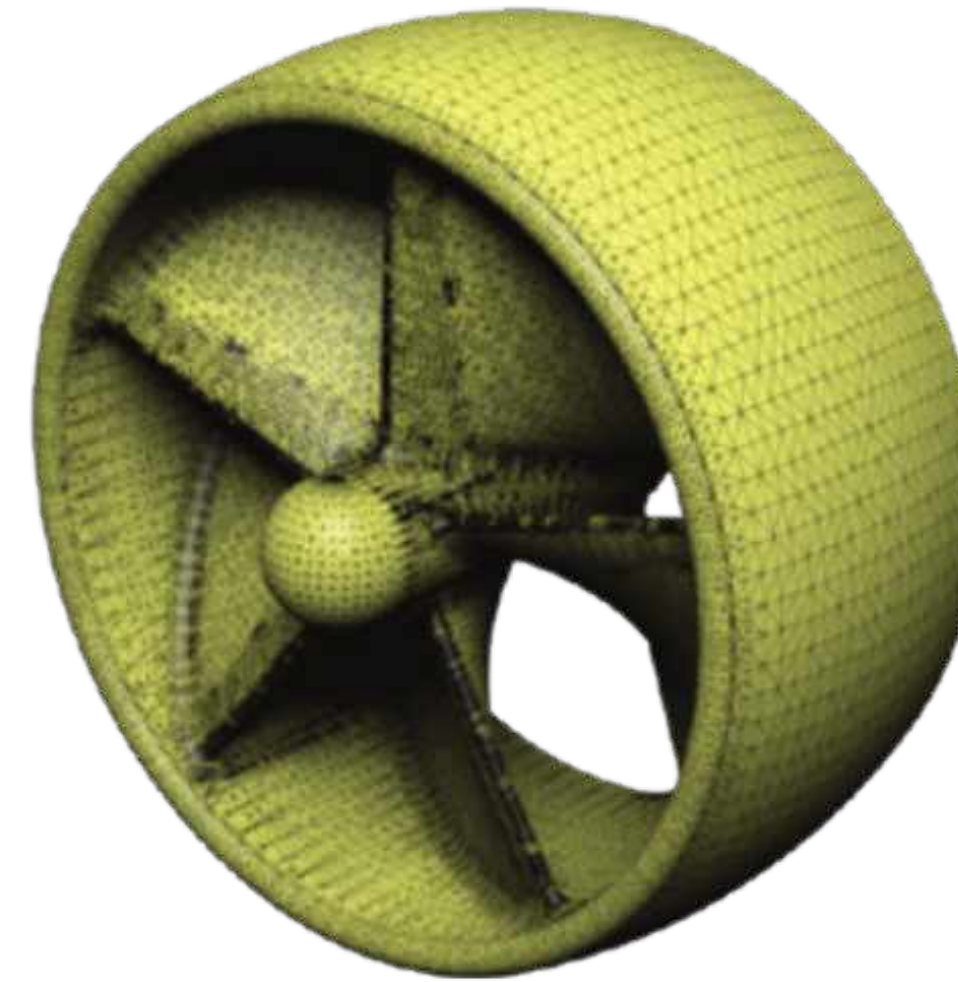
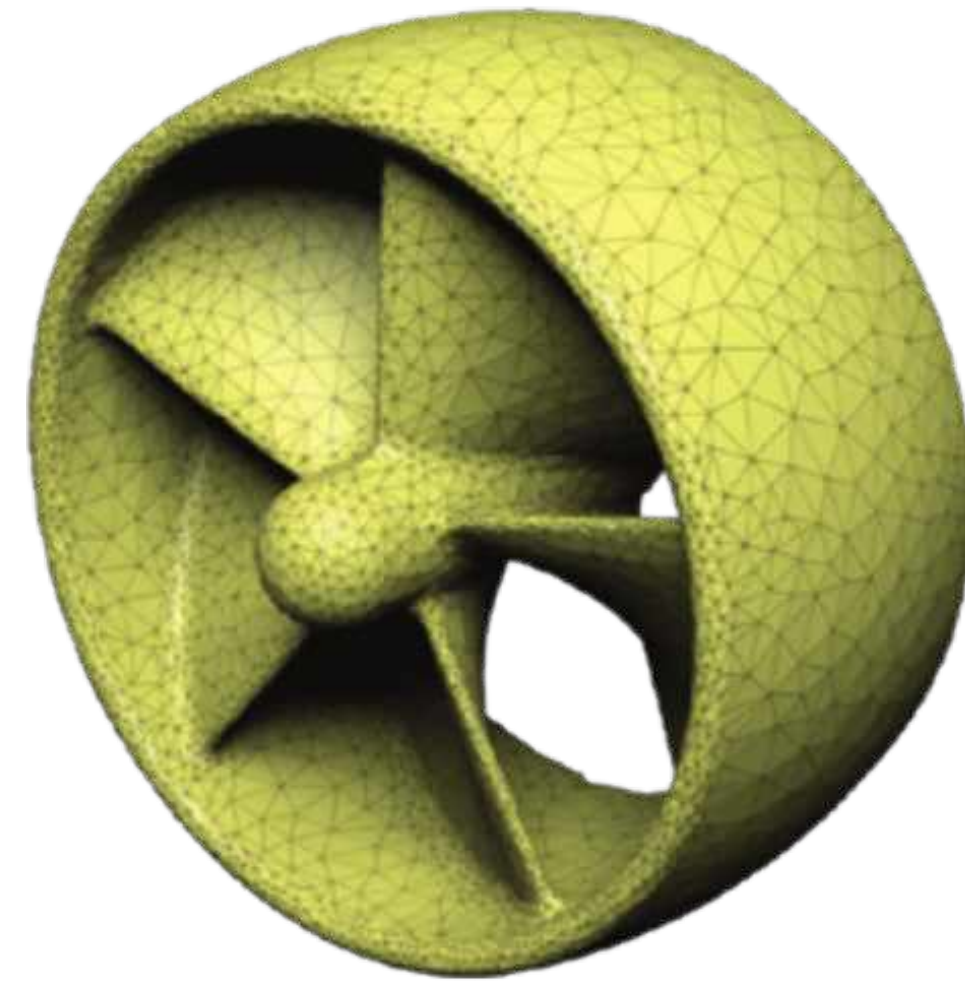
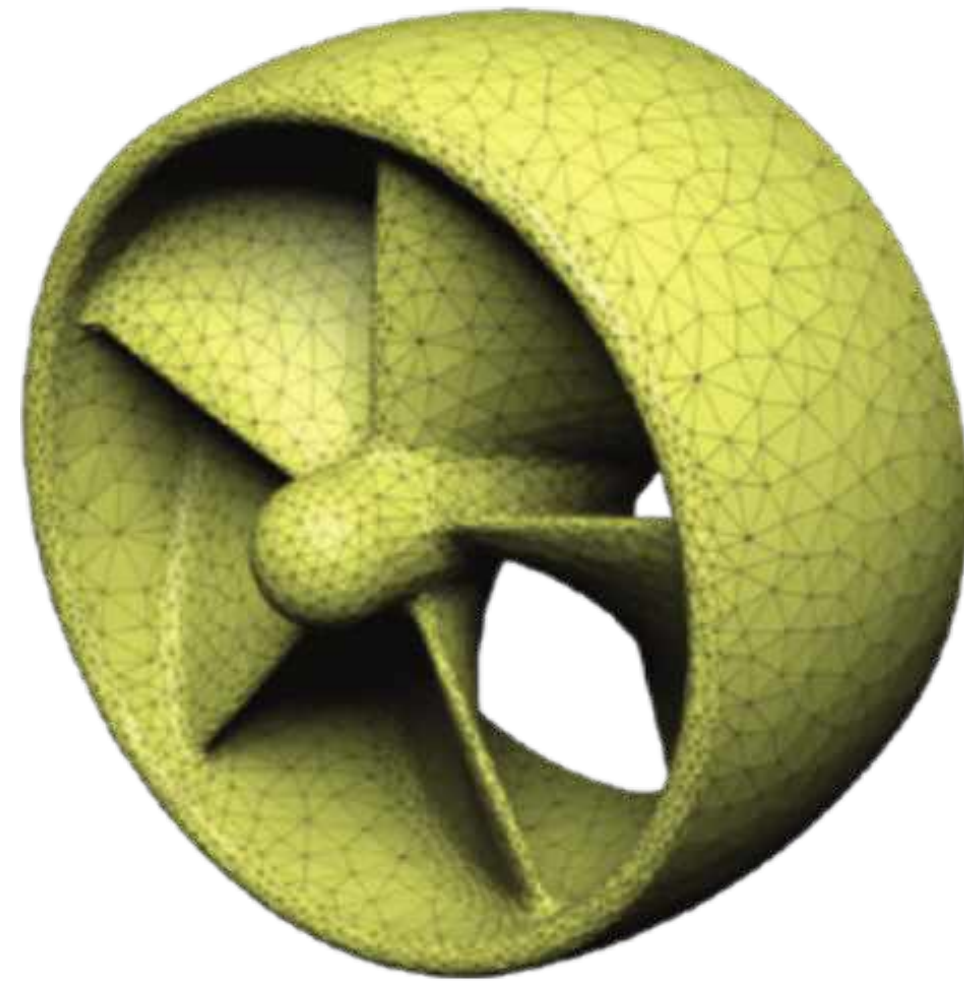
CGAL  
(without feature)



TetGen



# Tetrahedral Meshing



CGAL

CGAL  
(without feature)

TetGen

DelPSC







# Success Rate

CGAL  
57.2%

CGAL  
(no features)  
79.0%

TetGen  
49.5%

DeIPSC  
37.1%



# Why?



# Why?

- Problem statement imposes **strong assumptions on the input**, which are rare in real-data



# Why?

- Problem statement imposes **strong assumptions on the input**, which are rare in real-data
- Implementation of a complex algorithm in **floating point** is a major challenge, even if the algorithm is provably correct in arbitrary precision



# Why?

- Problem statement imposes **strong assumptions on the input**, which are rare in real-data
- Implementation of a complex algorithm in **floating point** is a major challenge, even if the algorithm is provably correct in arbitrary precision
- Modeling tools use **operations not closed under the representation** (for example trimming for NURBS), introducing a plethora of degenerate configurations



# Why?

- Problem statement imposes **strong assumptions on the input**, which are rare in real-data
- Implementation of a complex algorithm in **floating point** is a major challenge, even if the algorithm is provably correct in arbitrary precision
- Modeling tools use **operations not closed under the representation** (for example trimming for NURBS), introducing a plethora of degenerate configurations
- Large collections of data was not available during the development of these methods



Let's do it again



# Let's do it again

- High running times are *preferable* than a failure, since they enable **automation**



# Let's do it again

- High running times are *preferable* than a failure, since they enable **automation**
- Robust floating-point computation is difficult to get right, **exact** computation leads to simpler, but slower, algorithms



# Let's do it again

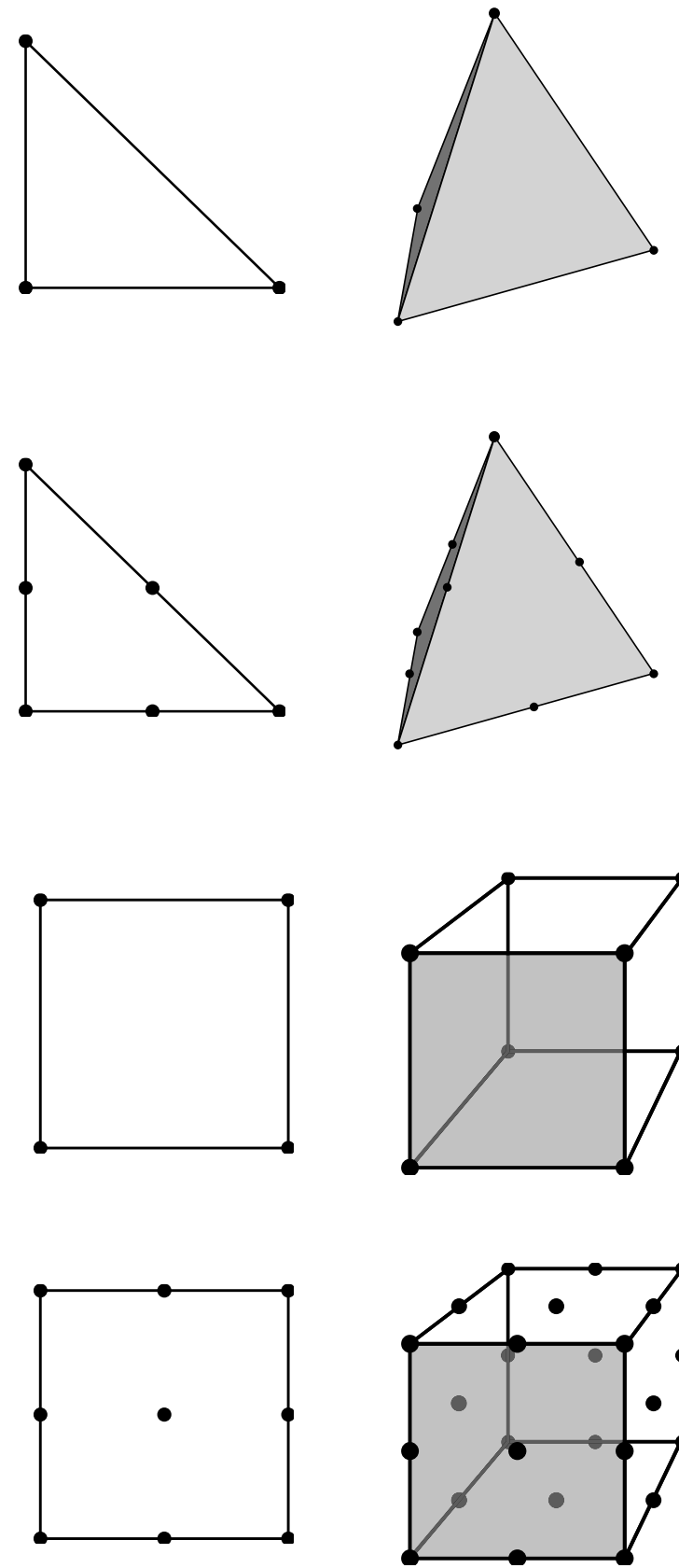
- High running times are *preferable* than a failure, since they enable **automation**
- Robust floating-point computation is difficult to get right, **exact** computation leads to simpler, but slower, algorithms
- Exact geometry is often **not required** (and sometimes not desired)



# Overview



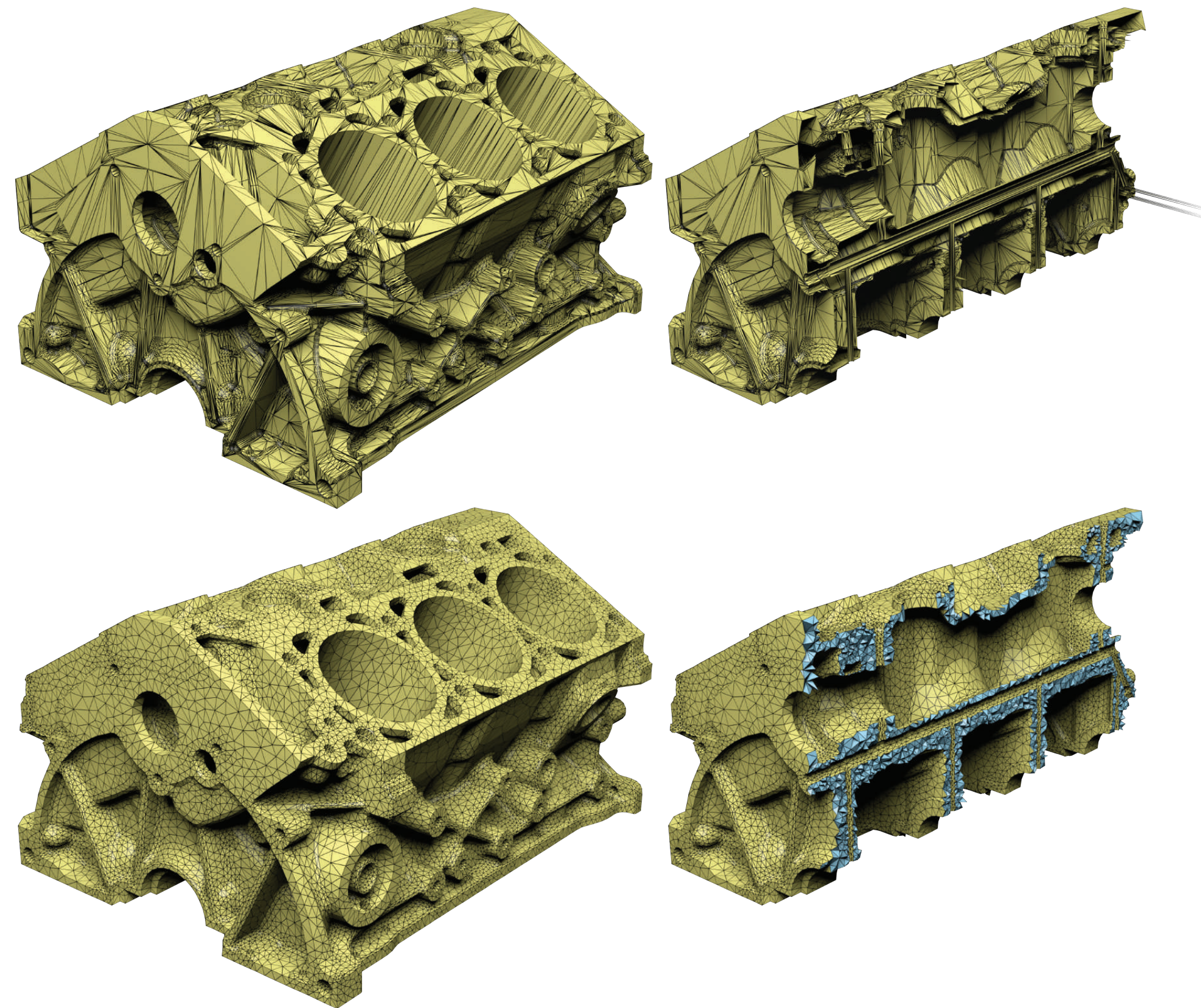
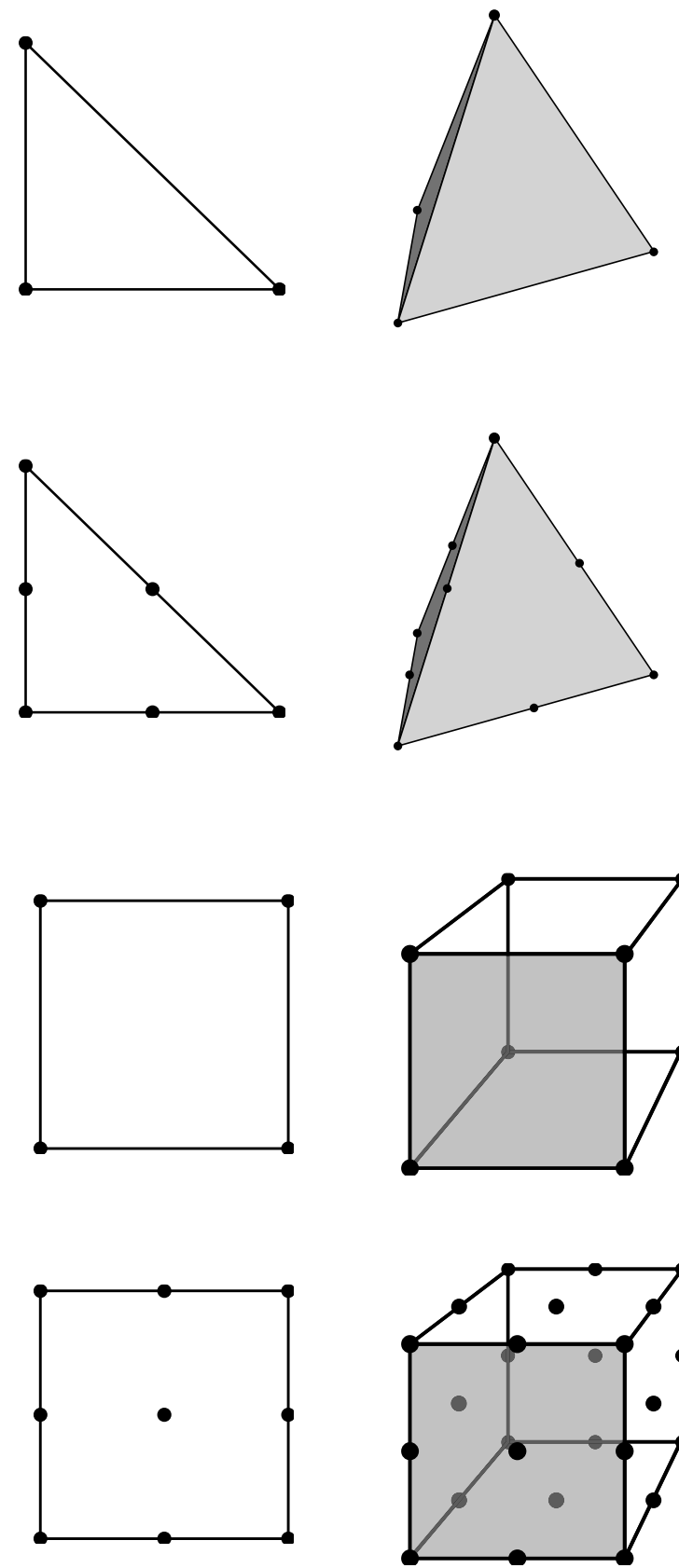
# Overview



Which discretization provides lower running time for a fixed accuracy?



# Overview

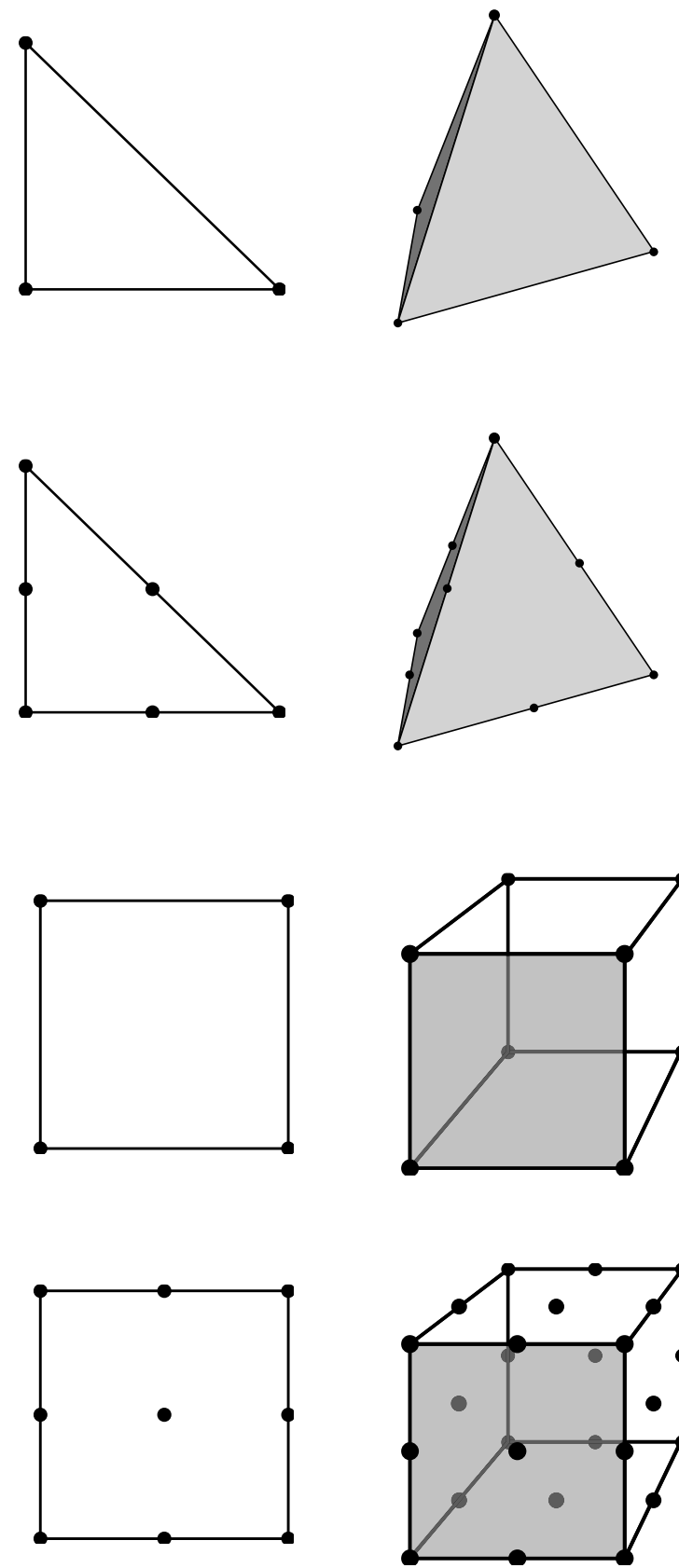


Which discretization provides lower running time for a fixed accuracy?

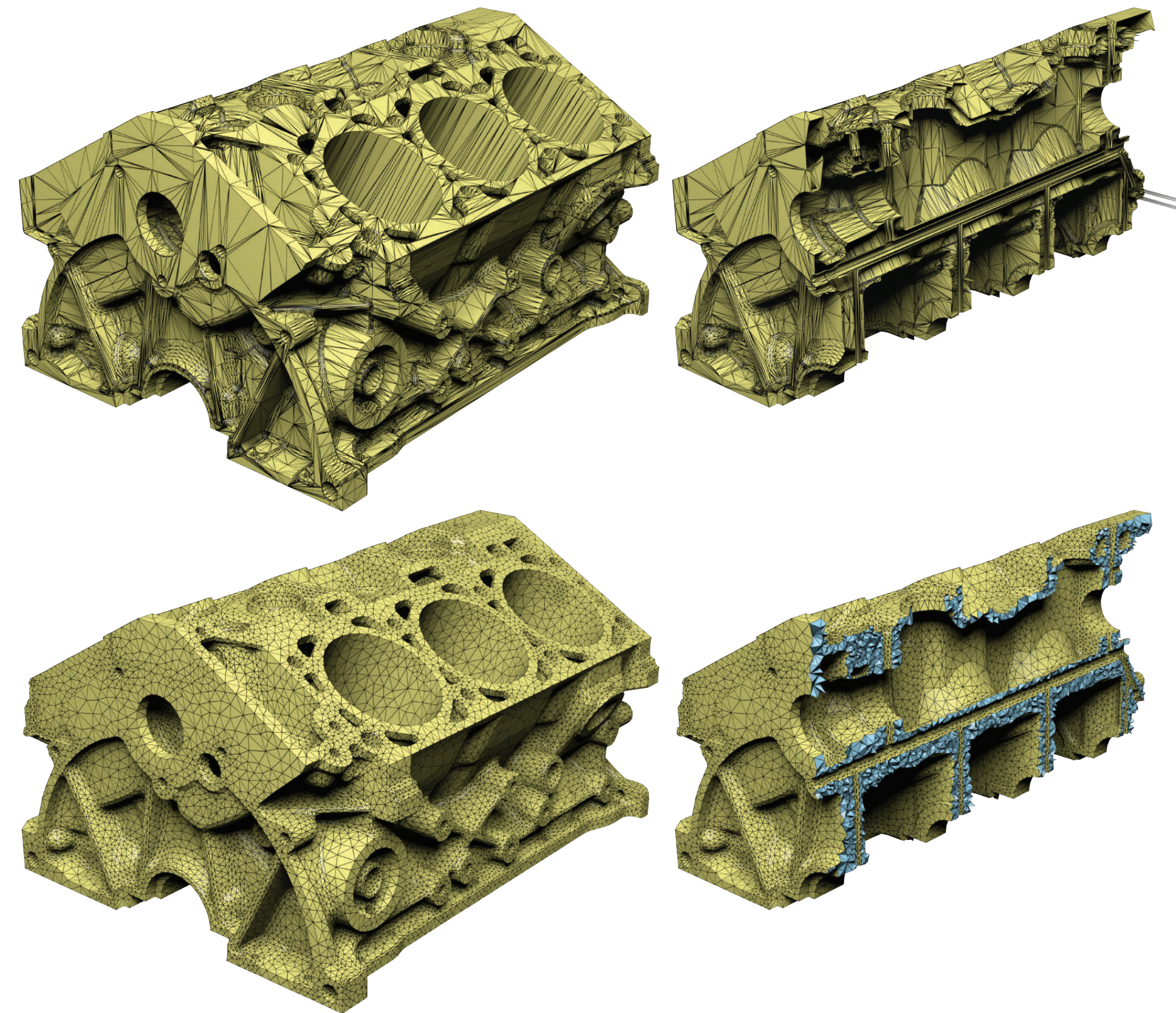
Can you mesh robustly without any assumption on the input?



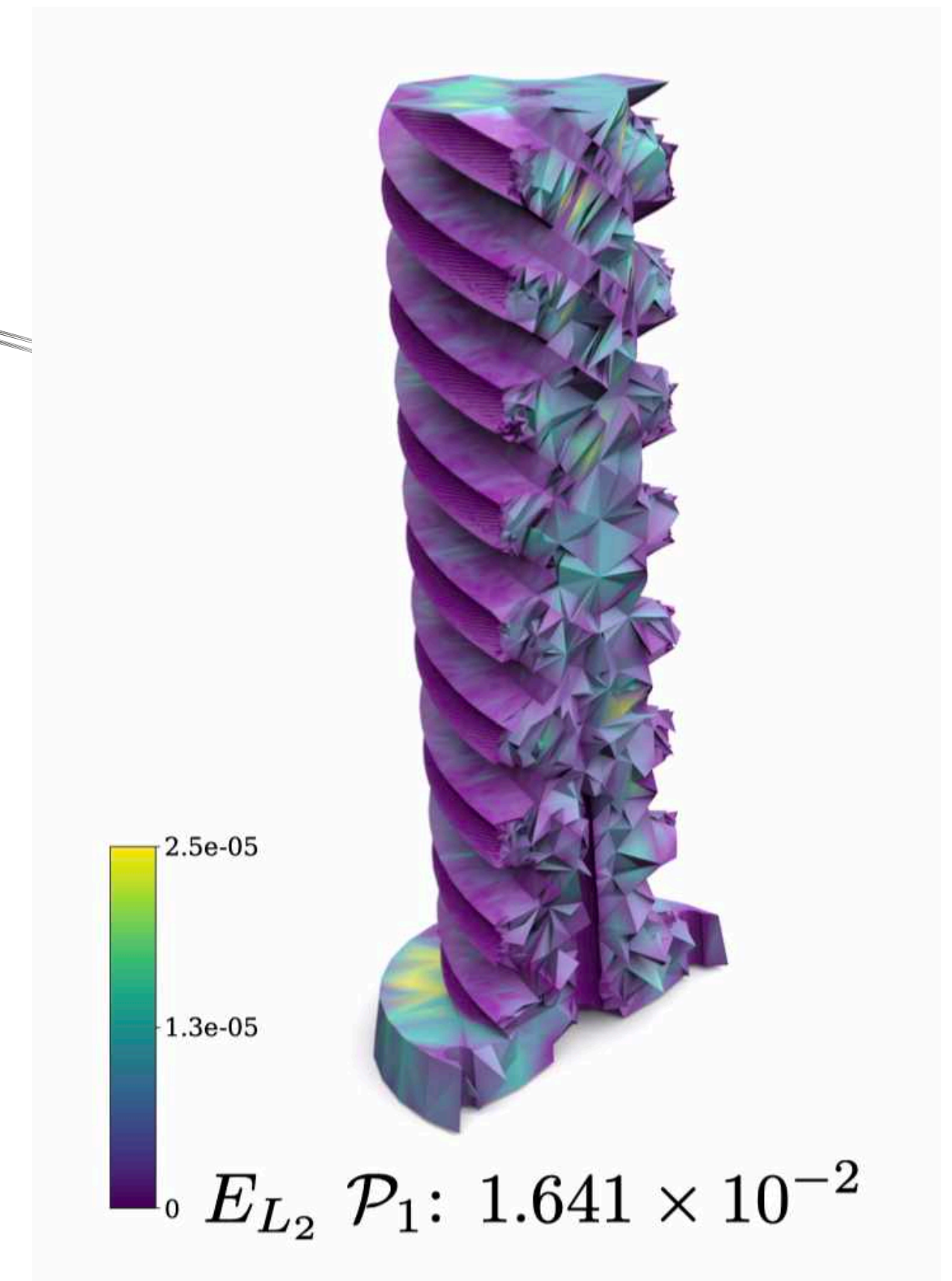
# Overview



Which discretization provides lower running time for a fixed accuracy?



Can you mesh robustly without any assumption on the input?

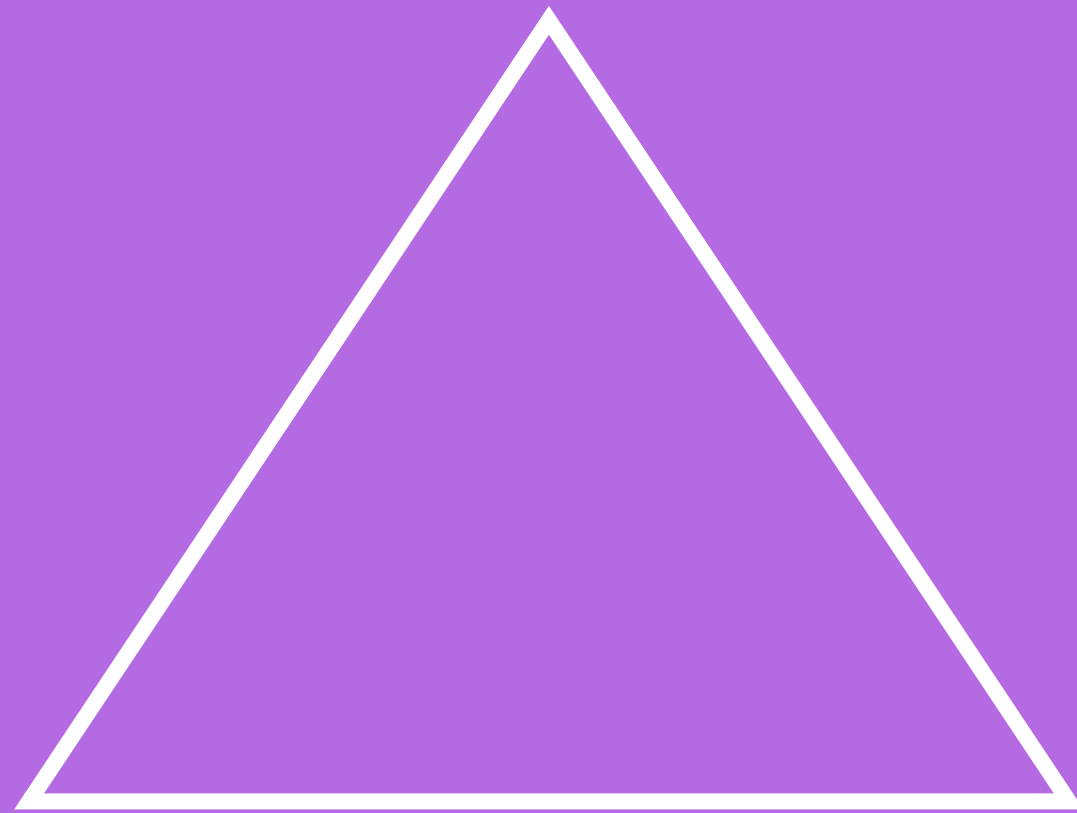


Does mesh quality affect the accuracy of the FEM solution?



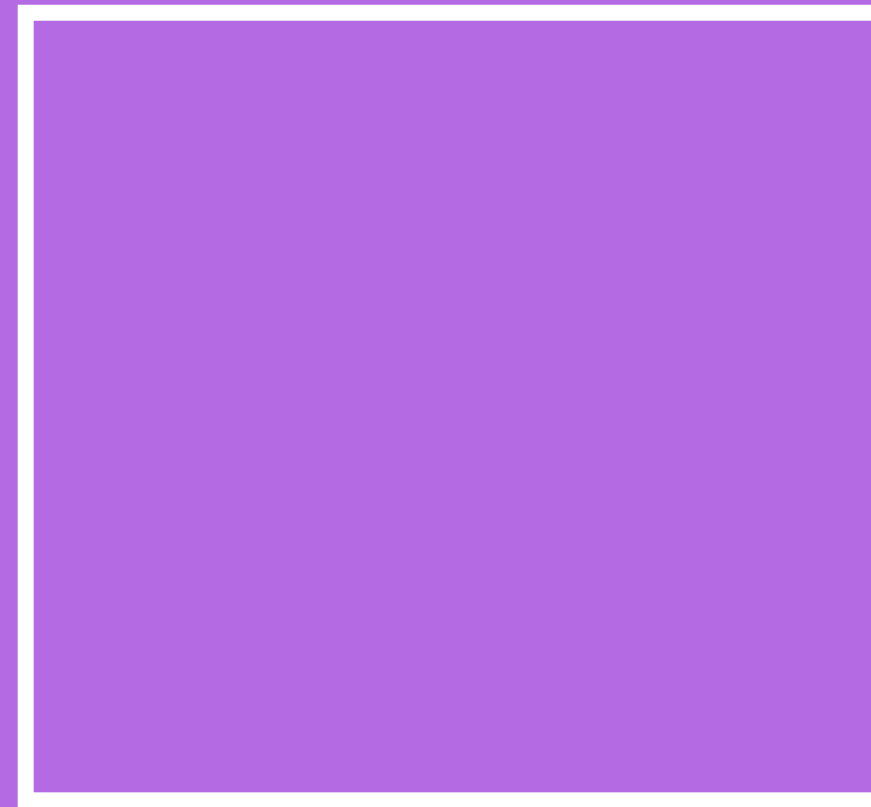
Which element is more accurate for a non-linear elasticity problem  
given a fixed wall clock time budget?

1



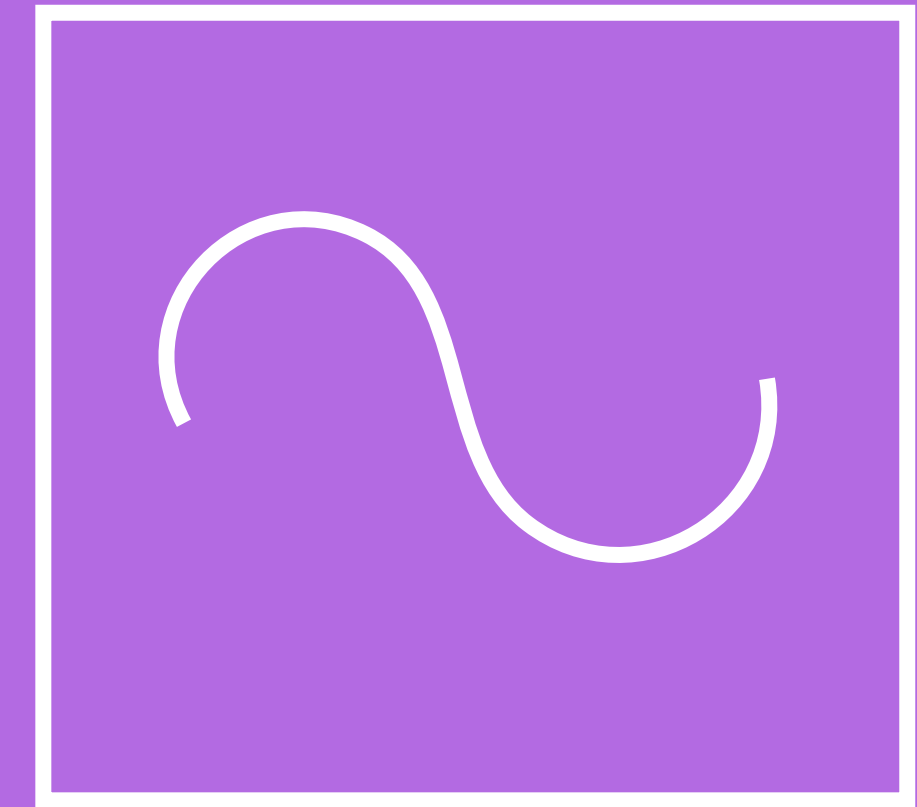
Quadratic  
Lagrangian  
Tetrahedra

2



Quadratic  
Lagrangian/Serendipity  
Hexahedra

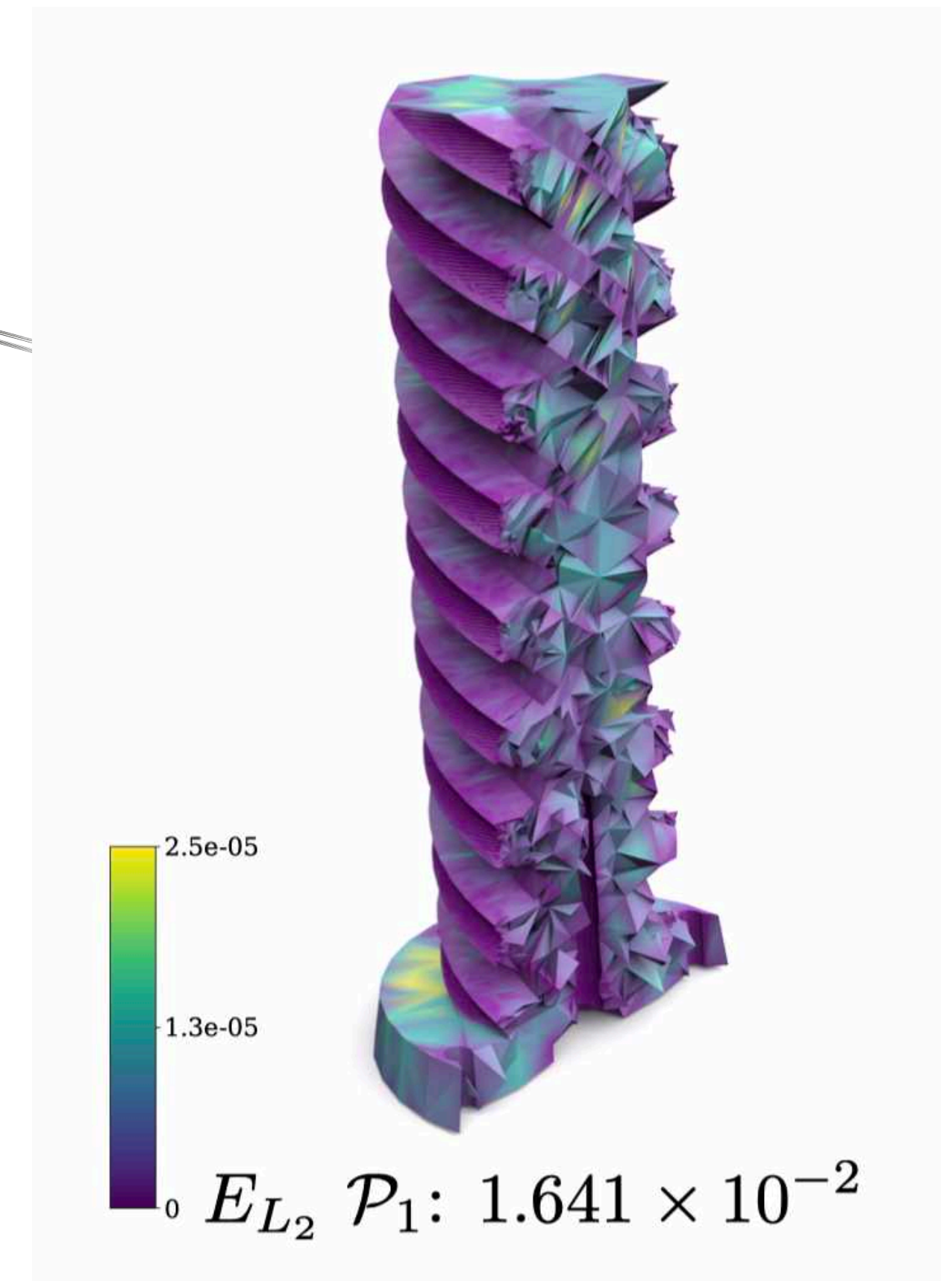
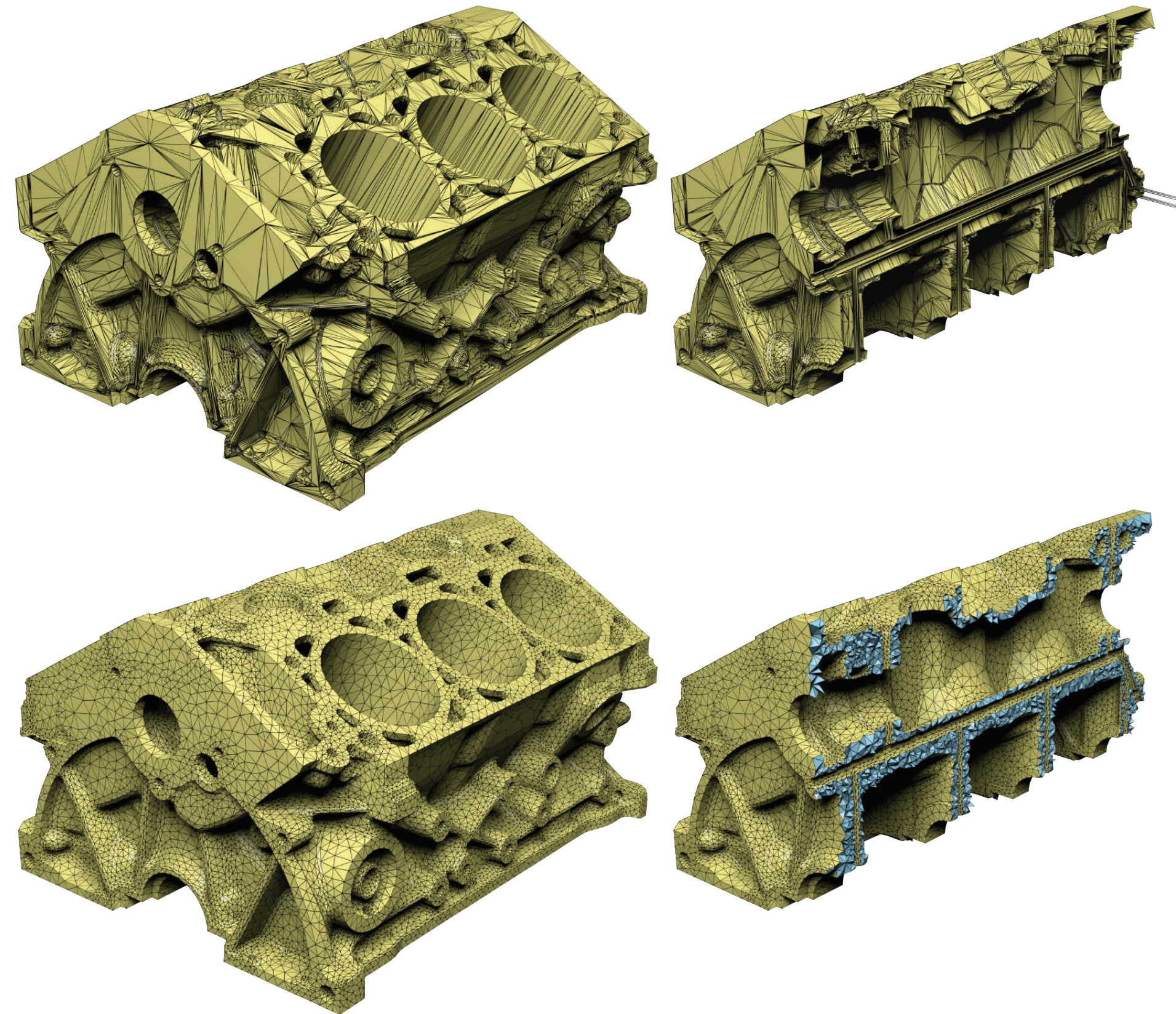
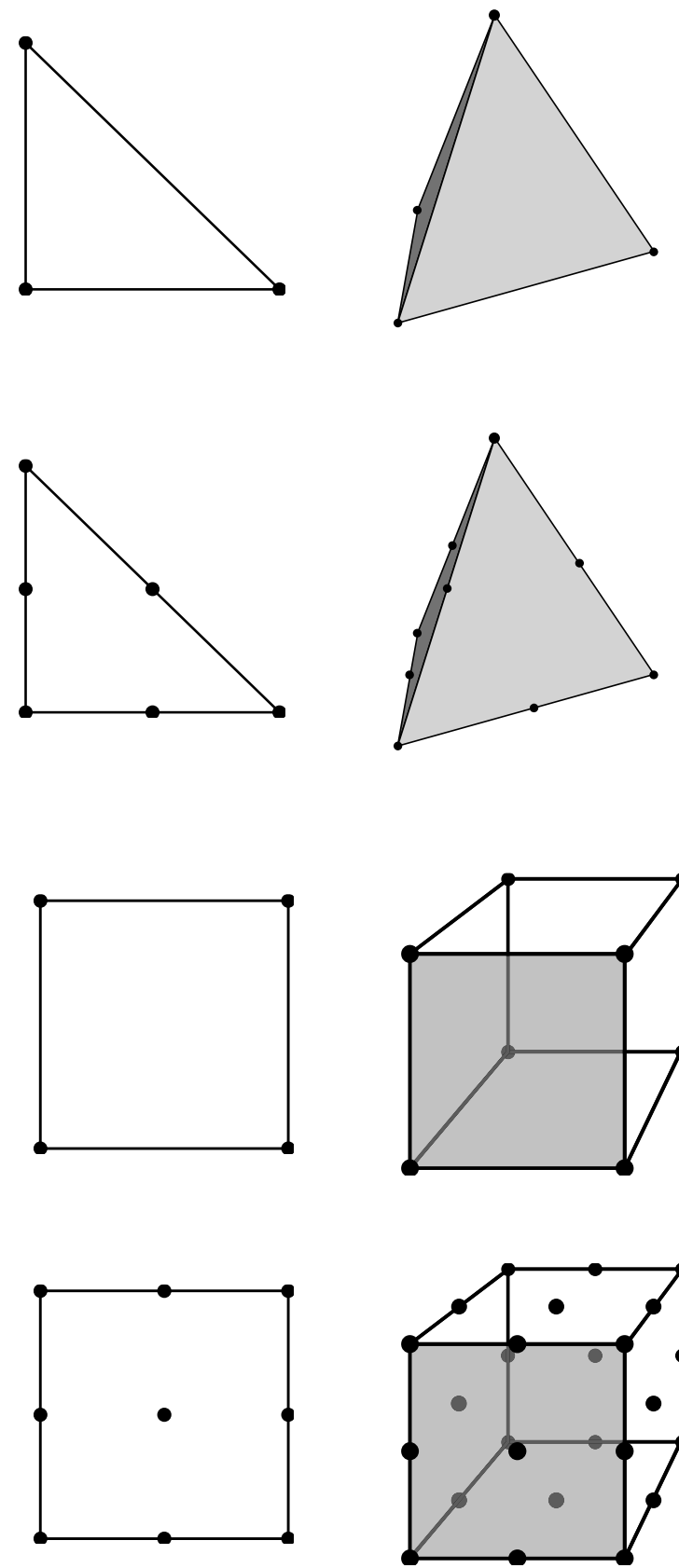
3



Quadratic  
Splines on  
Hexahedra (IGA)



# Overview



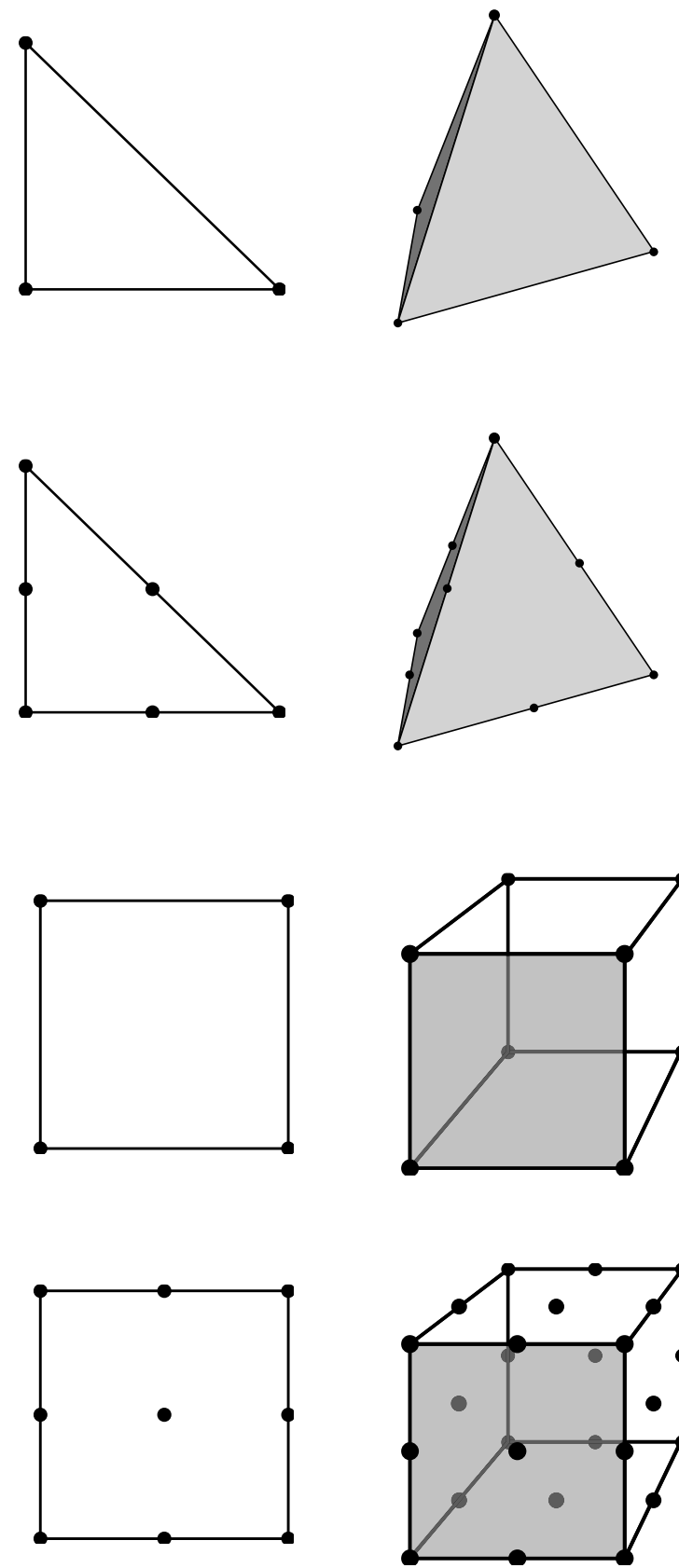
Which discretization provides lower running time for a fixed accuracy?

Can you mesh robustly without any assumption on the input?

Does mesh quality affect the accuracy of the FEM solution?

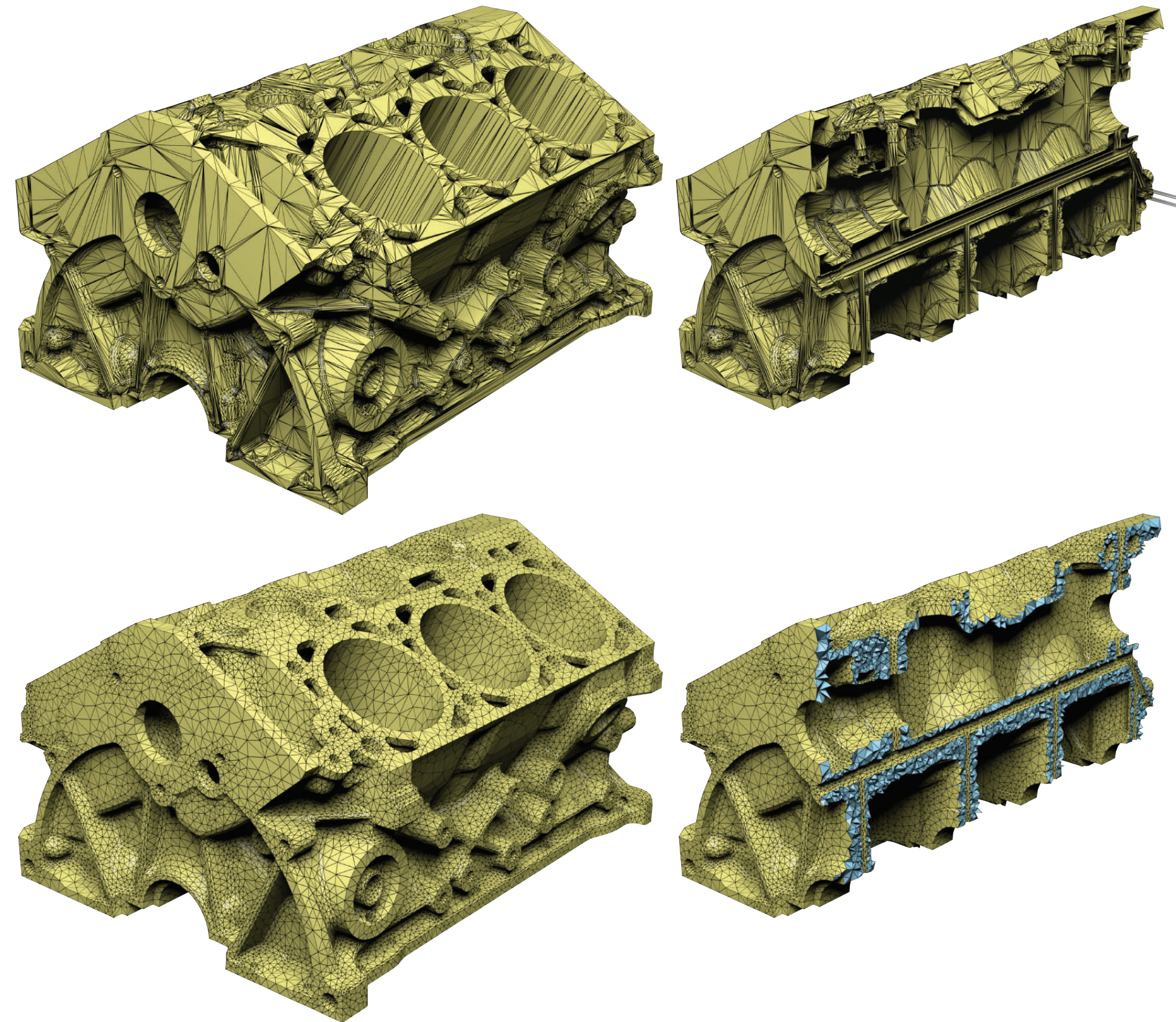


# Overview

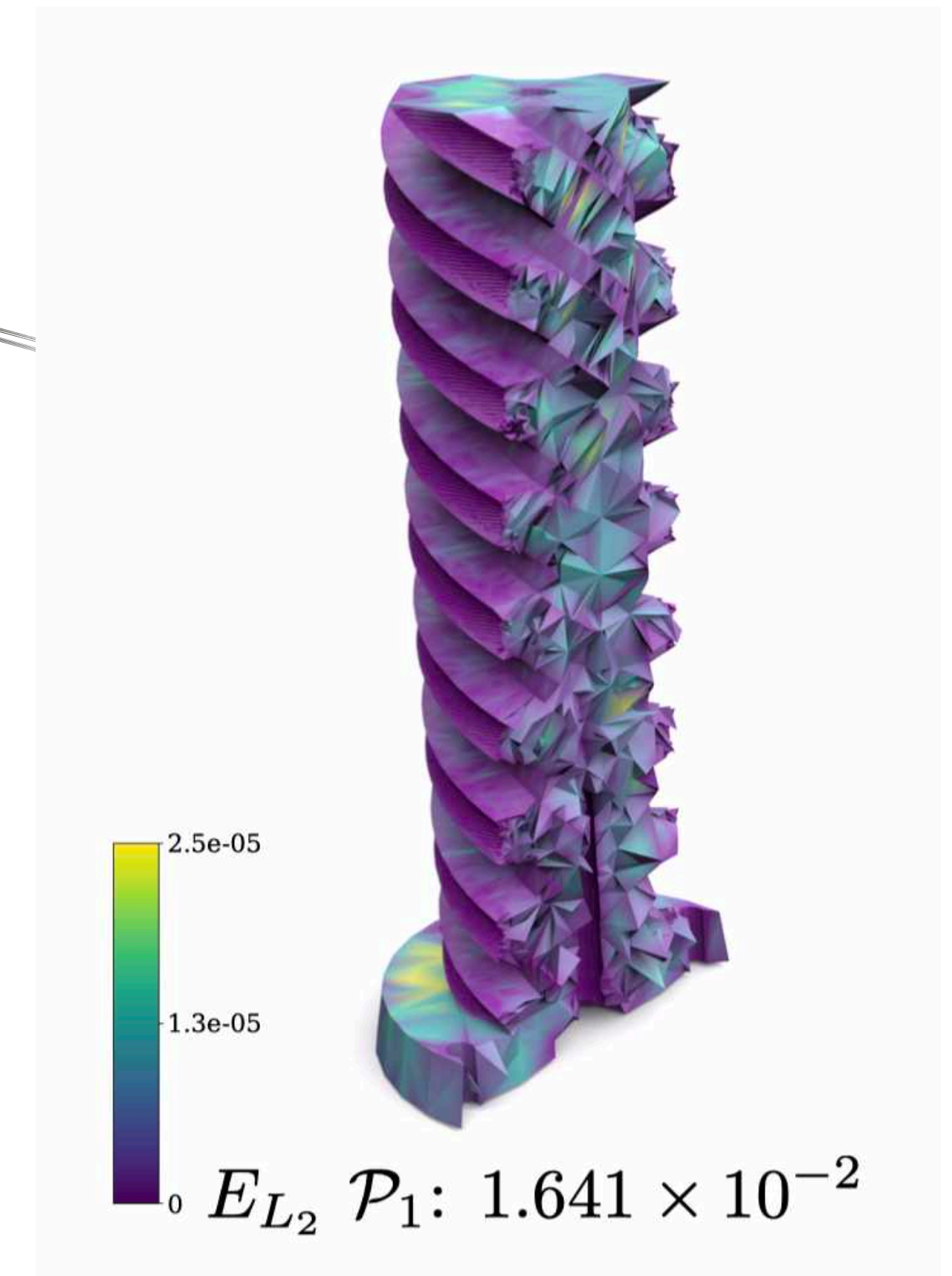


Which discretization provides lower running time for a fixed accuracy?

**Tetrahedra :)**



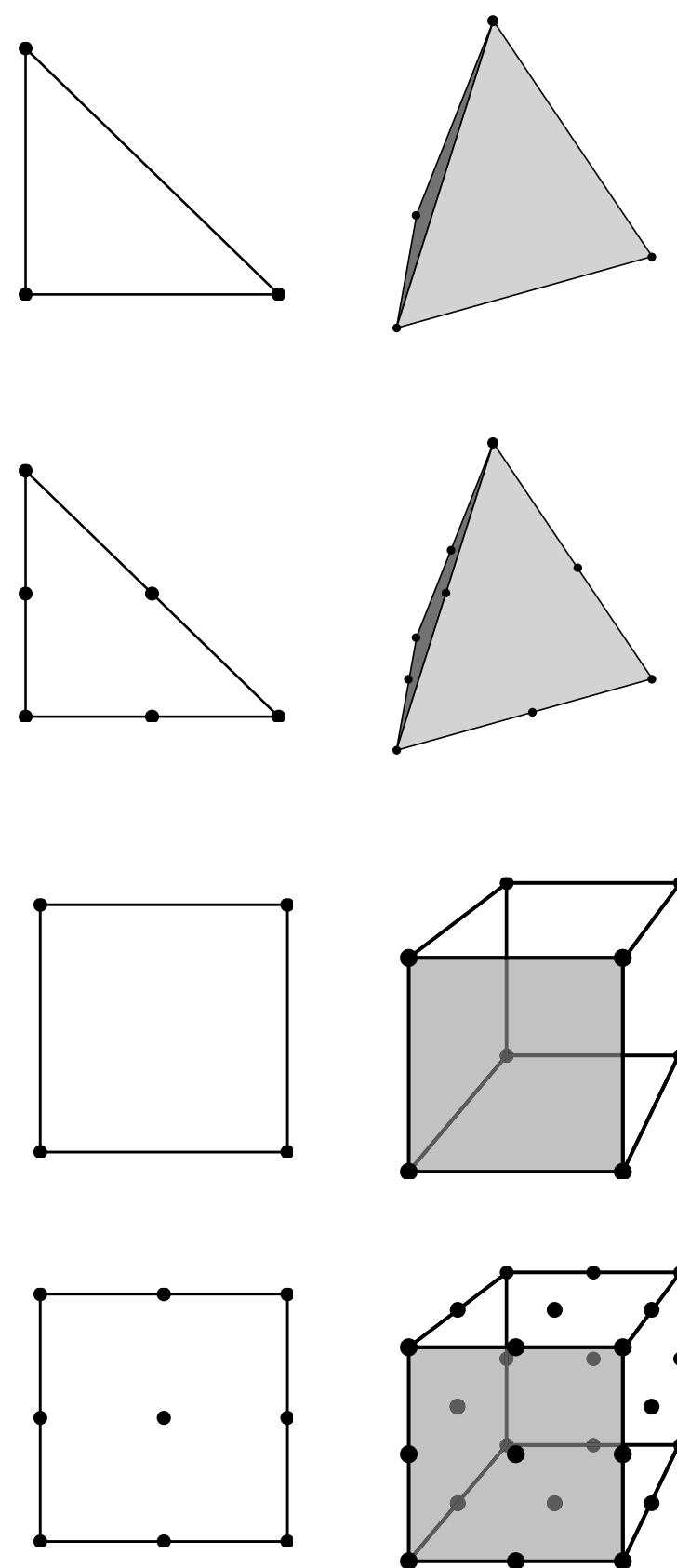
Can you mesh robustly without any assumption on the input?



Does mesh quality affect the accuracy of the FEM solution?

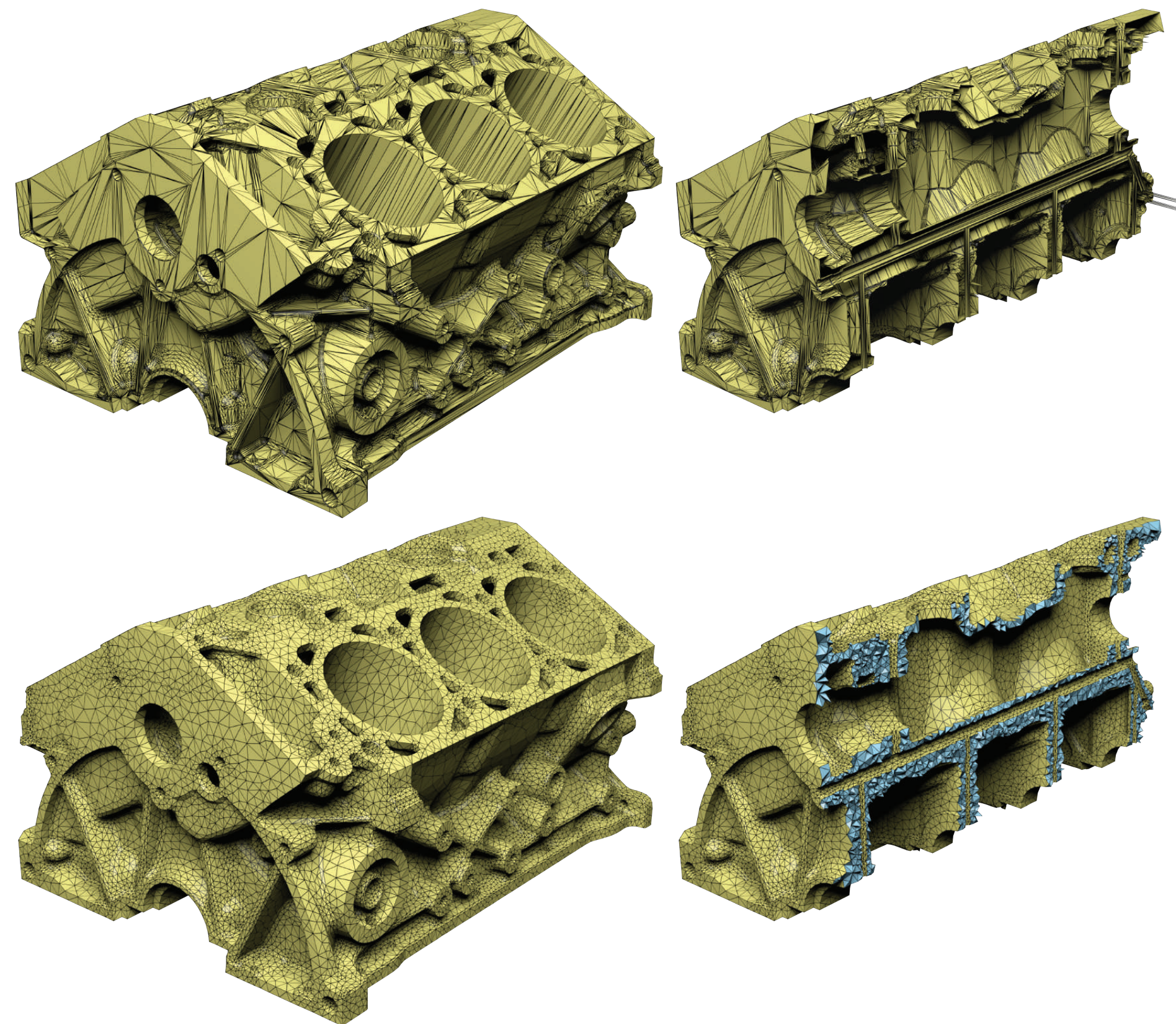


# Overview



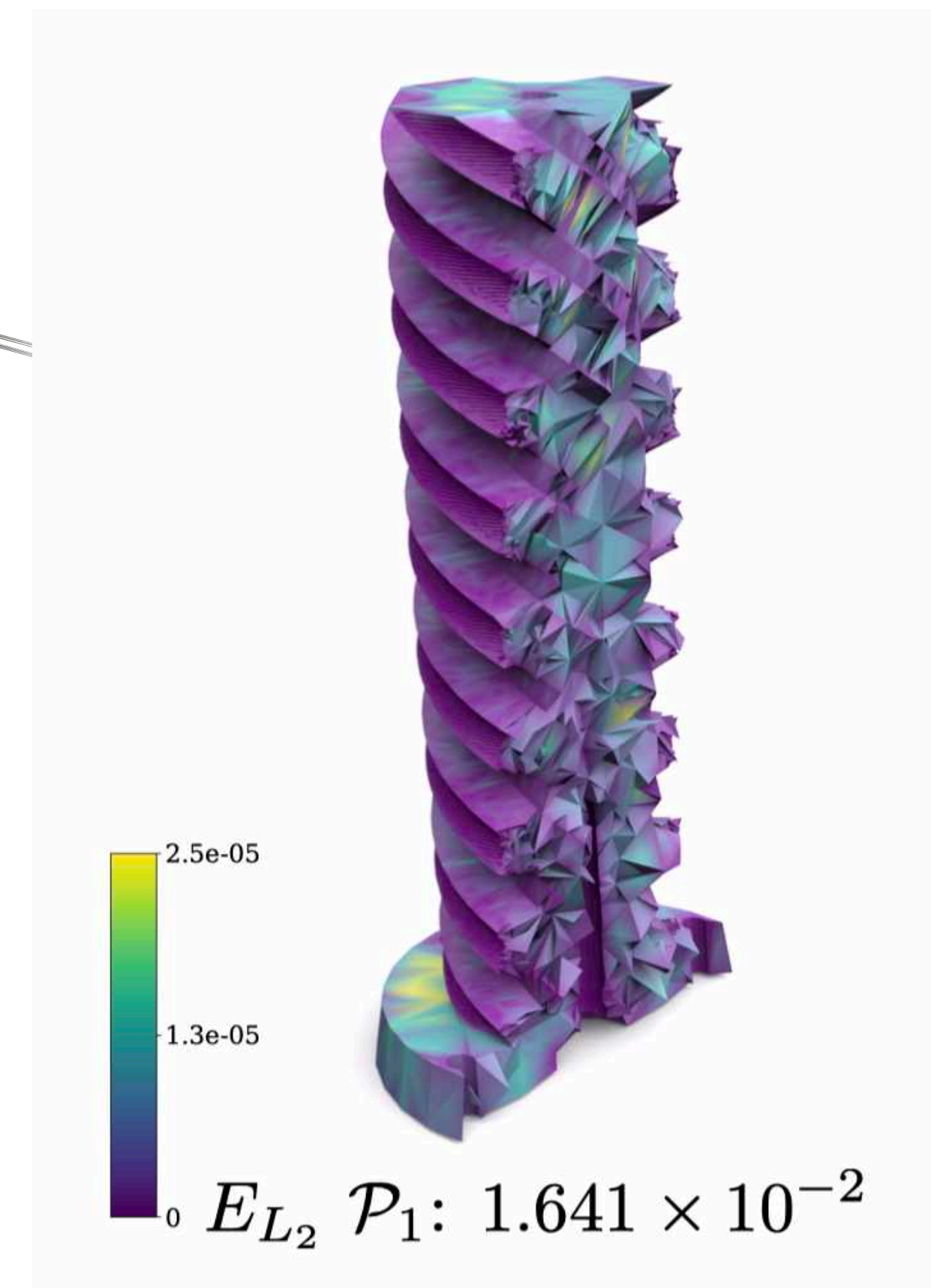
Which discretization provides lower running time for a fixed accuracy?

**Tetrahedra :)**



Can you mesh robustly without any assumption on the input?

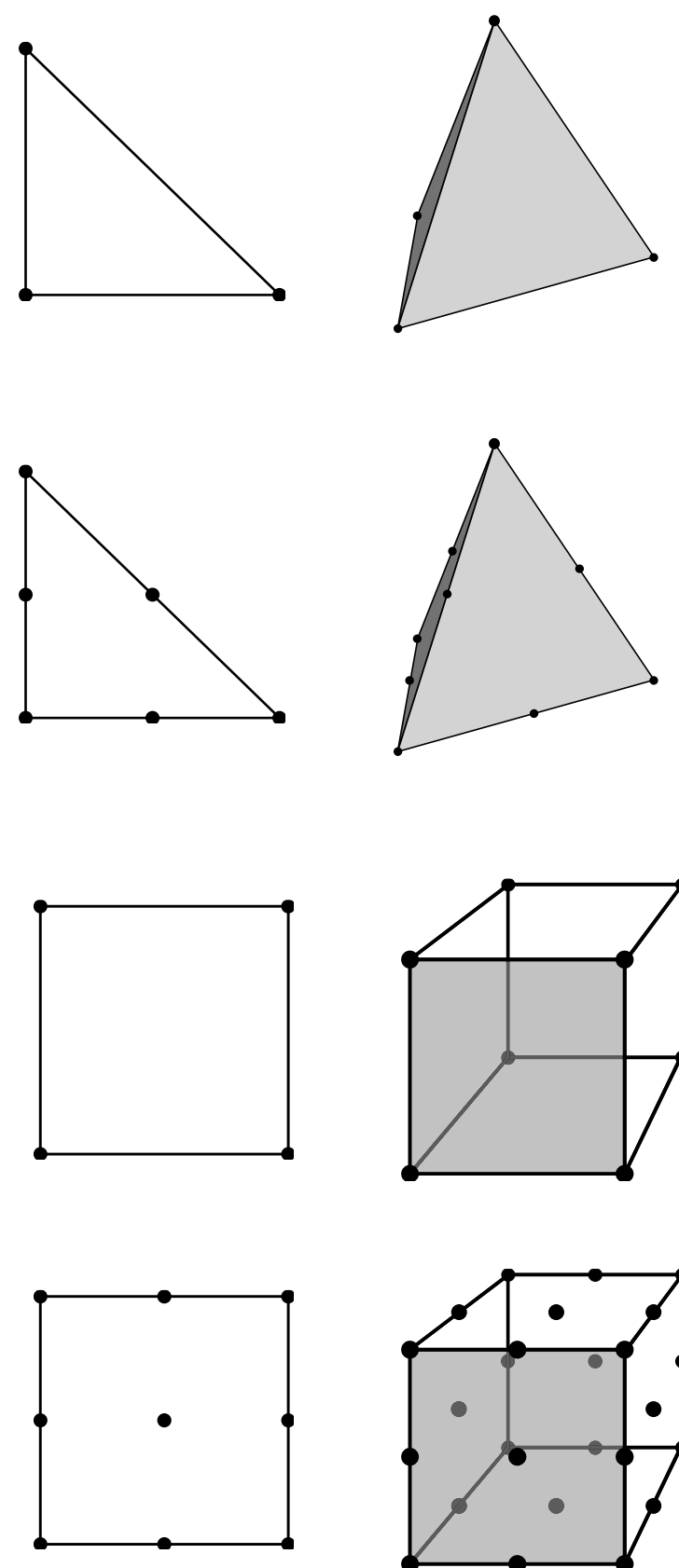
**Yes!**



Does mesh quality affect the accuracy of the FEM solution?

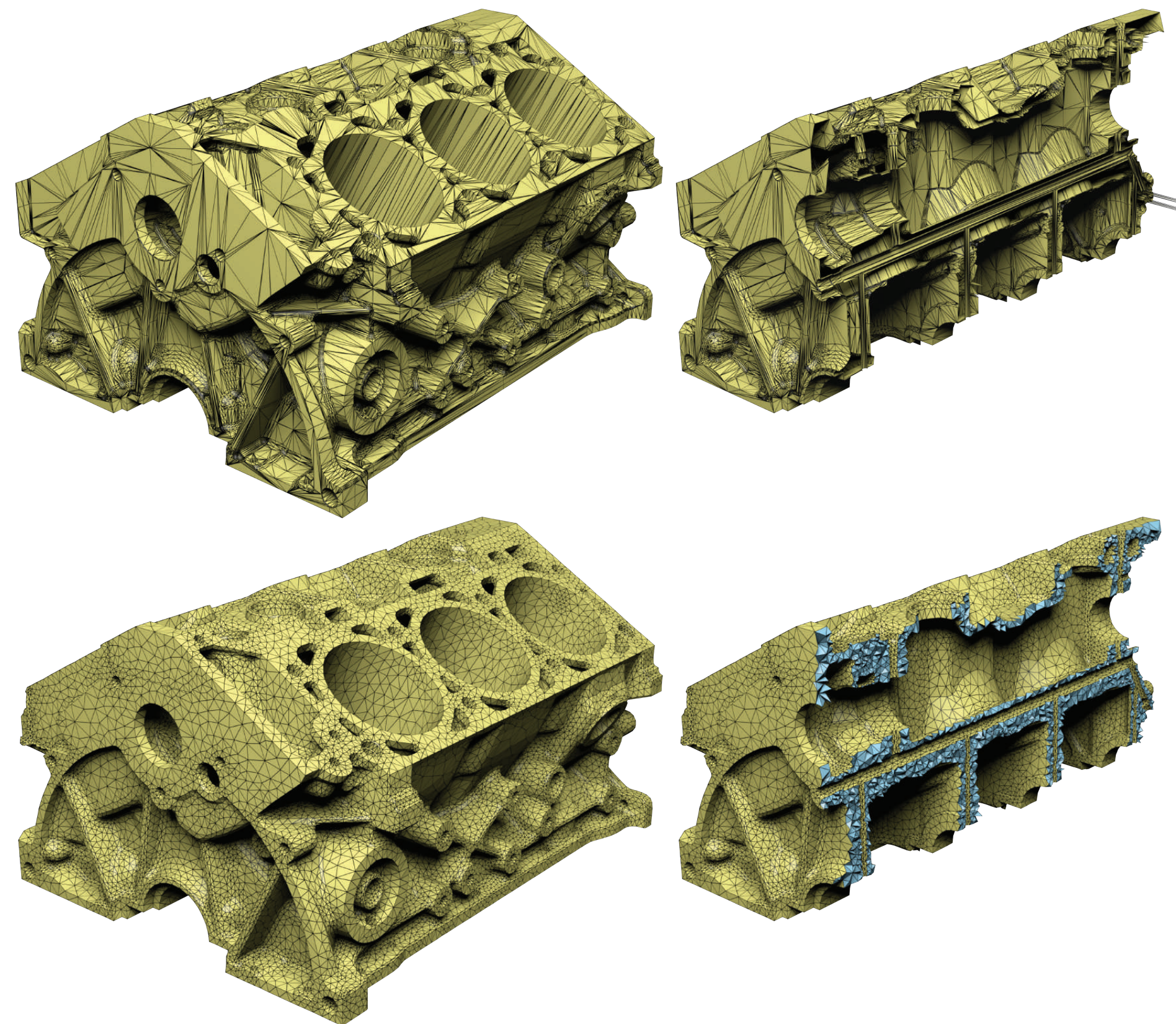


# Overview



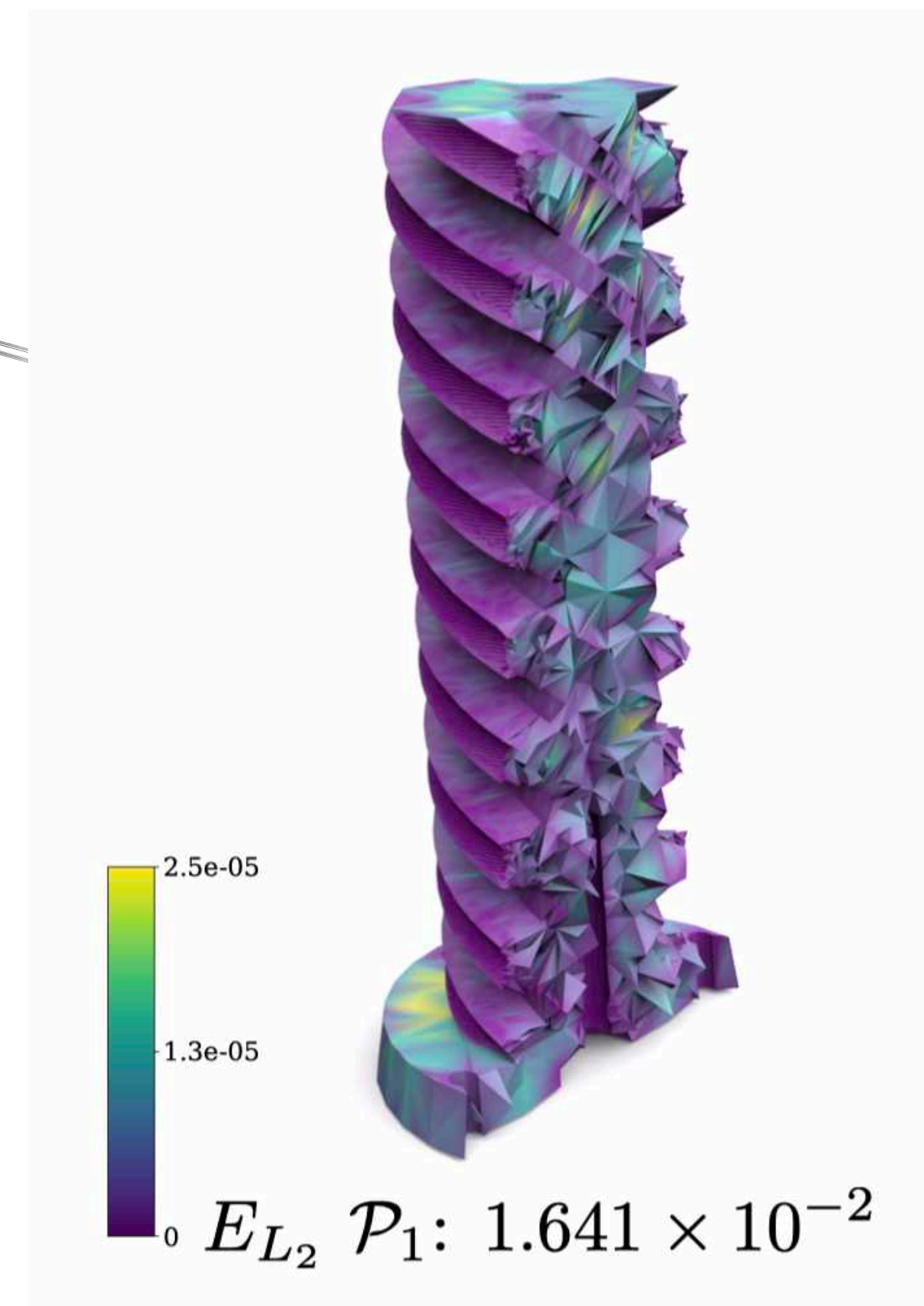
Which discretization provides lower running time for a fixed accuracy?

**Tetrahedra :)**



Can you mesh robustly without any assumption on the input?

**Yes!**



Does mesh quality affect the accuracy of the FEM solution?

**No!\***

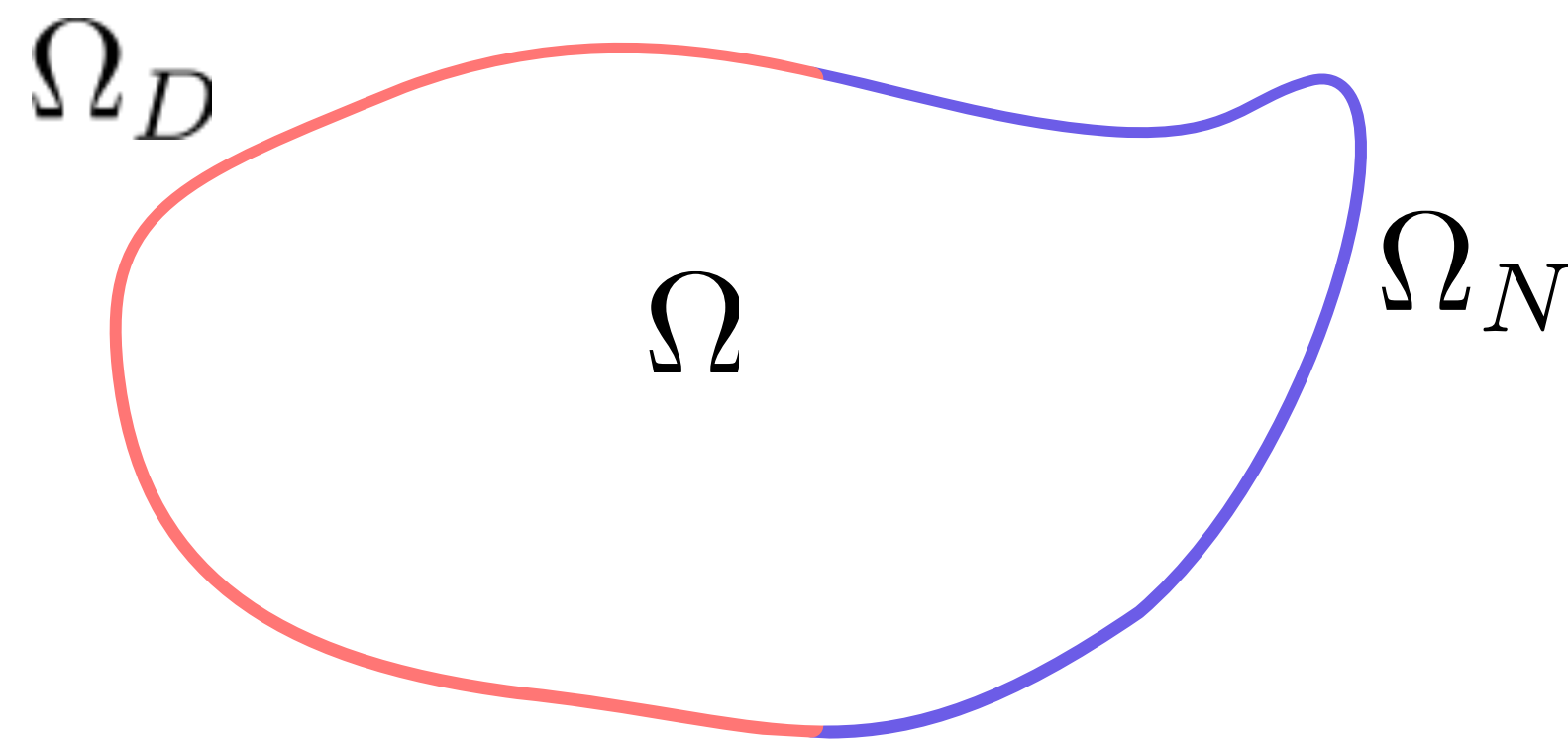


# Problem

- Solve elliptic PDE  $\mathcal{F}(x, u, \nabla u, D^2 u) = b$

subject to  $u = d$  on  $\partial\Omega_D$  and  $\nabla u \cdot n = f$  on  $\partial\Omega_N$

- For common elliptic PDEs
  - Elasticity (Linear and Non-Linear)
  - Stokes
  - Helmholtz
  - Poisson





# Dataset



# Dataset

- Hexalab <https://www.hexalab.net/>
  - 16 state-of-the-art hex-meshing algorithms
  - 237 meshes
  - 8 flips 3.4%



# Dataset

- Hexalab <https://www.hexalab.net/>
  - 16 state-of-the-art hex-meshing algorithms
  - 237 meshes
  - 8 flips 3.4%
- Thingi10k
  - 3200 meshes with MeshGems
  - 577 flips 18.0%



# Dataset

- Hexalab <https://www.hexalab.net/>
  - 16 state-of-the-art hex-meshing algorithms
  - 237 meshes
  - 8 flips 3.4%
- Thingi10k
  - 3200 meshes with MeshGems
  - 577 flips 18.0%
- For a given hex mesh, we generate a tetrahedral mesh with the same number of vertices



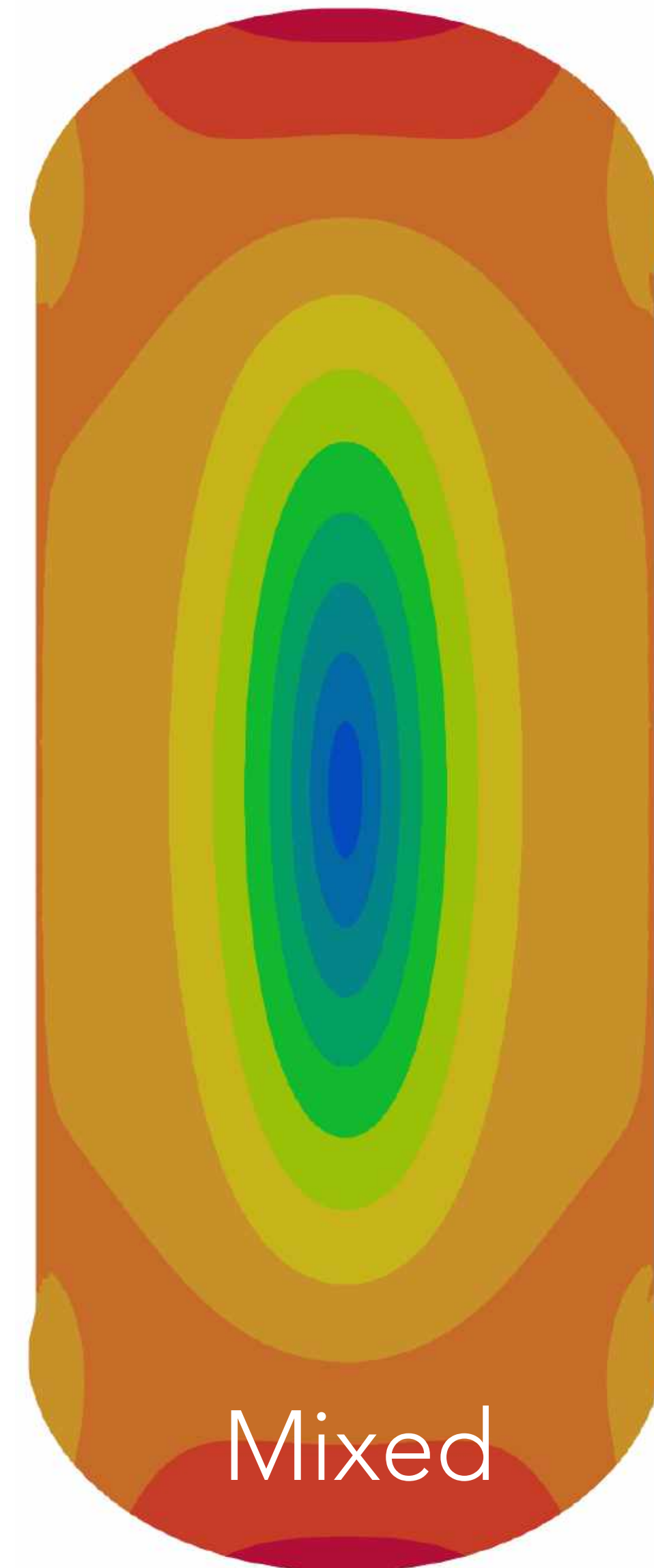
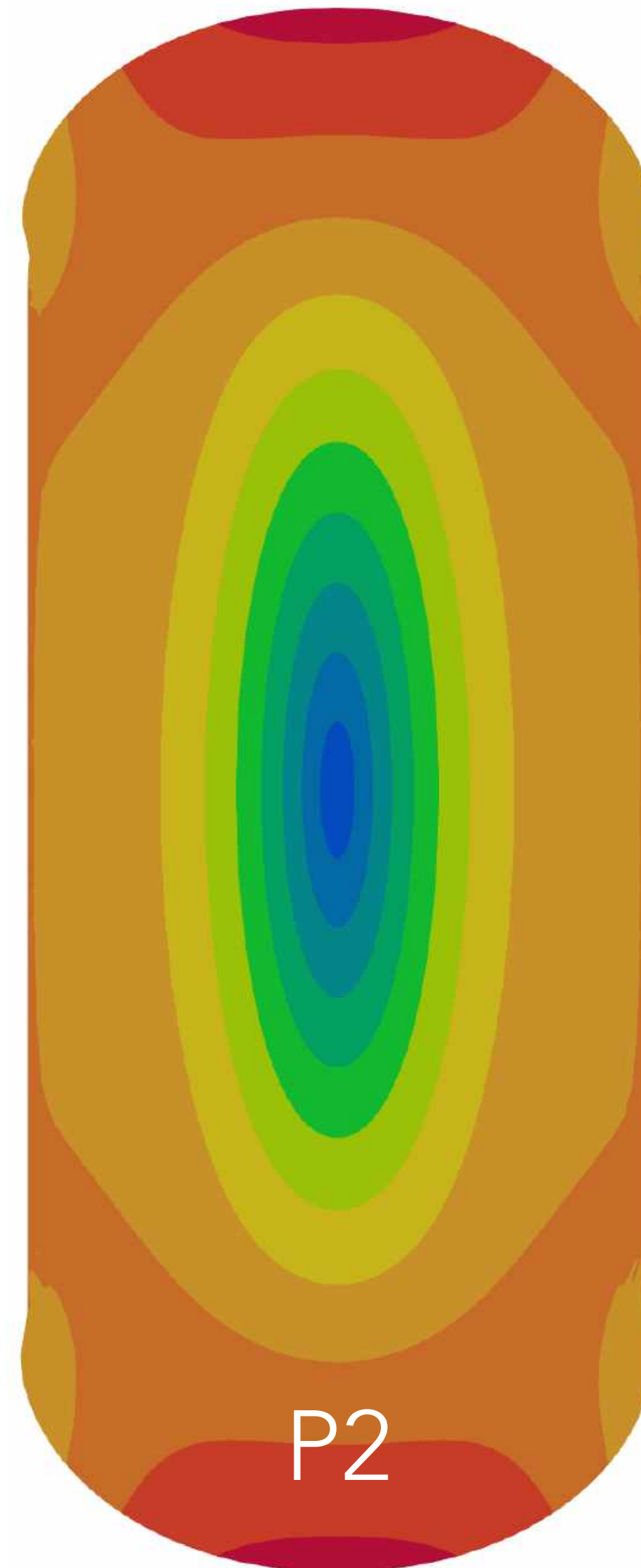
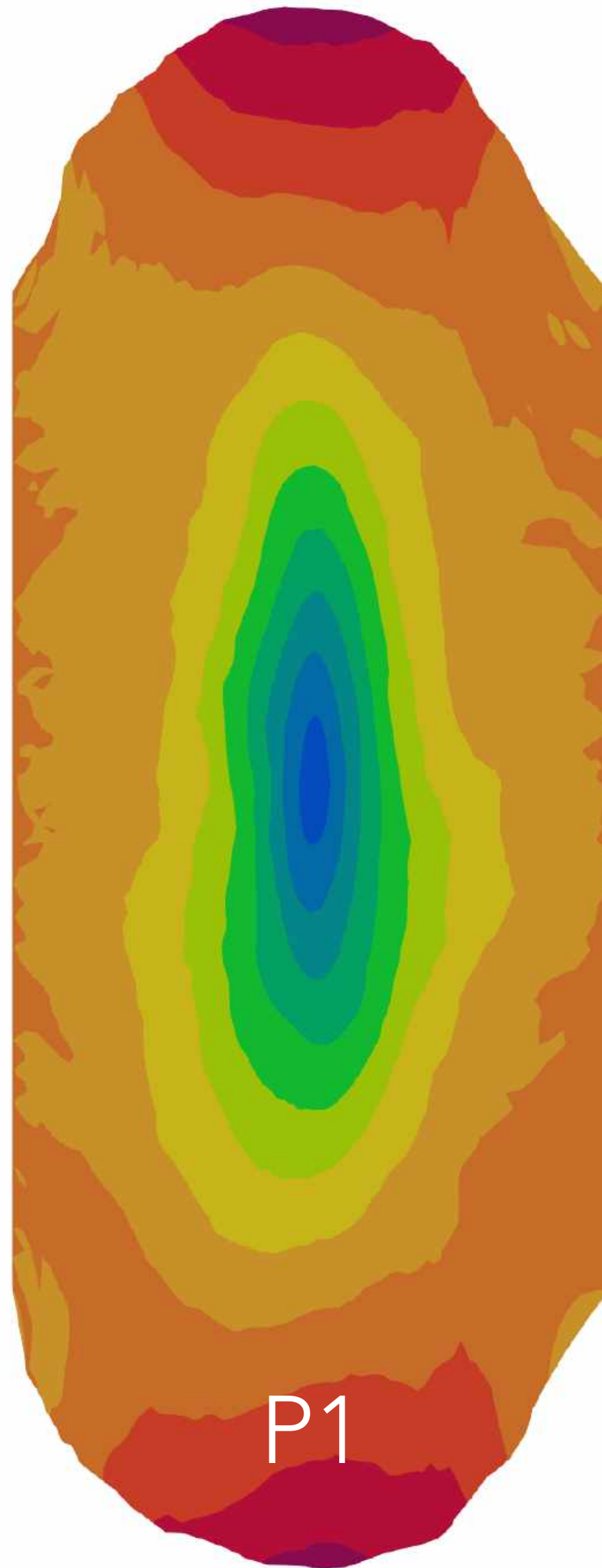
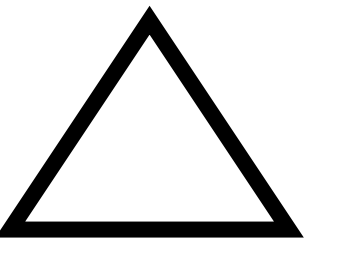
# Interactive Plot

<https://polyfem.github.io/tet-vs-hex/plot.html>



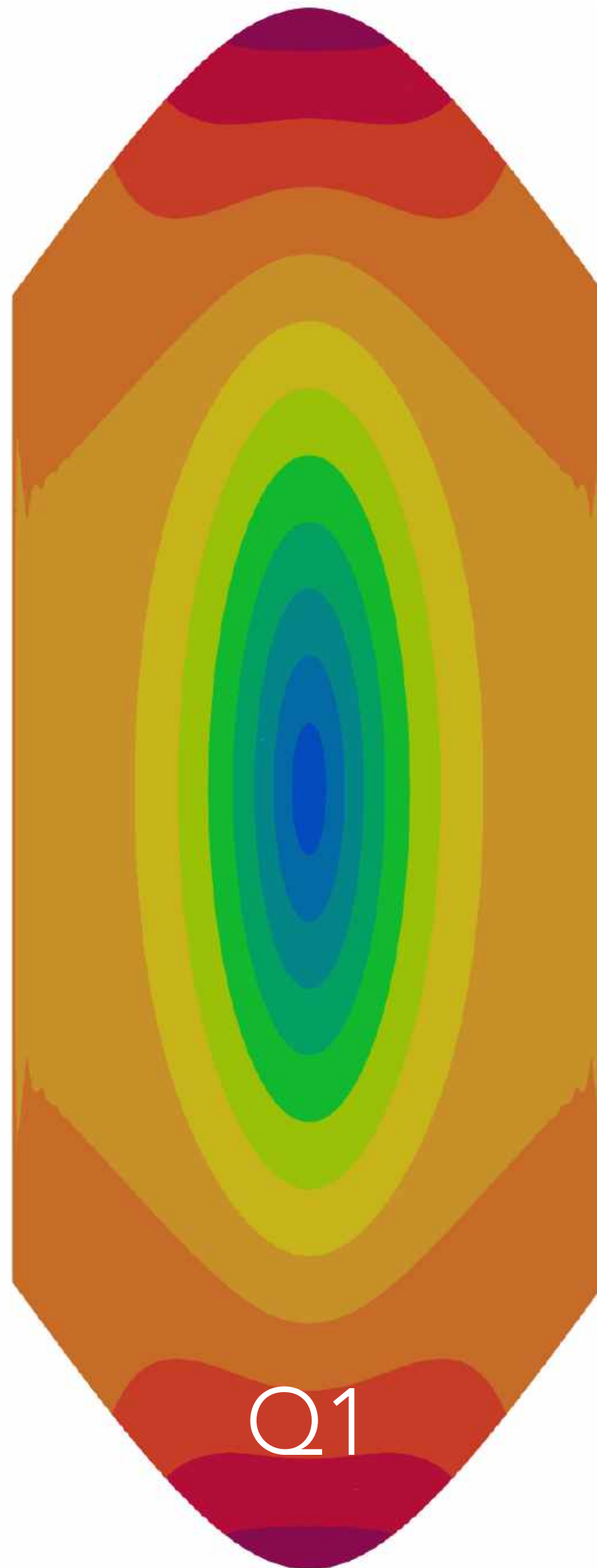
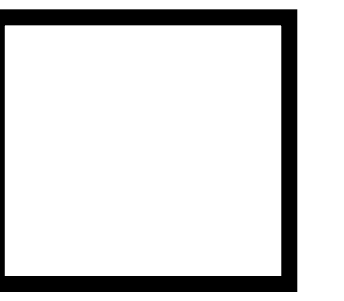


# Incompressible

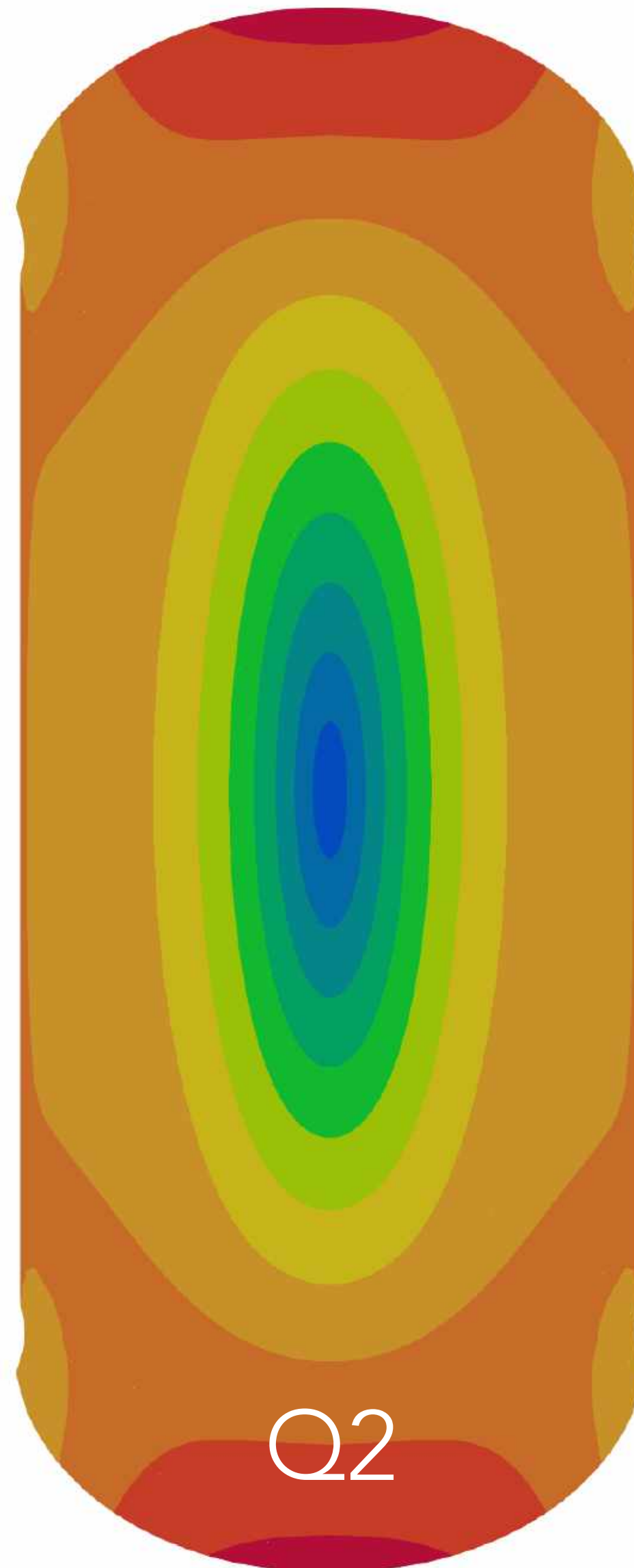




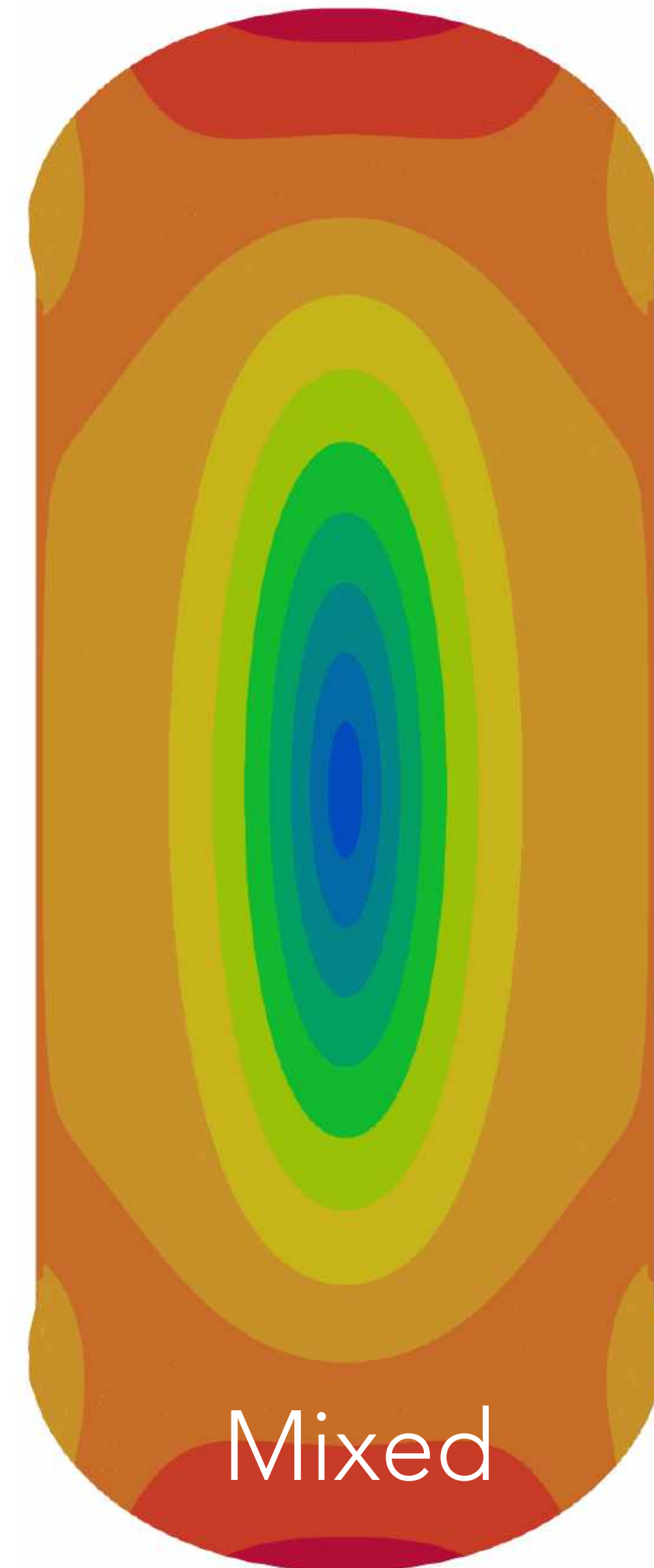
# Incompressible



Q1



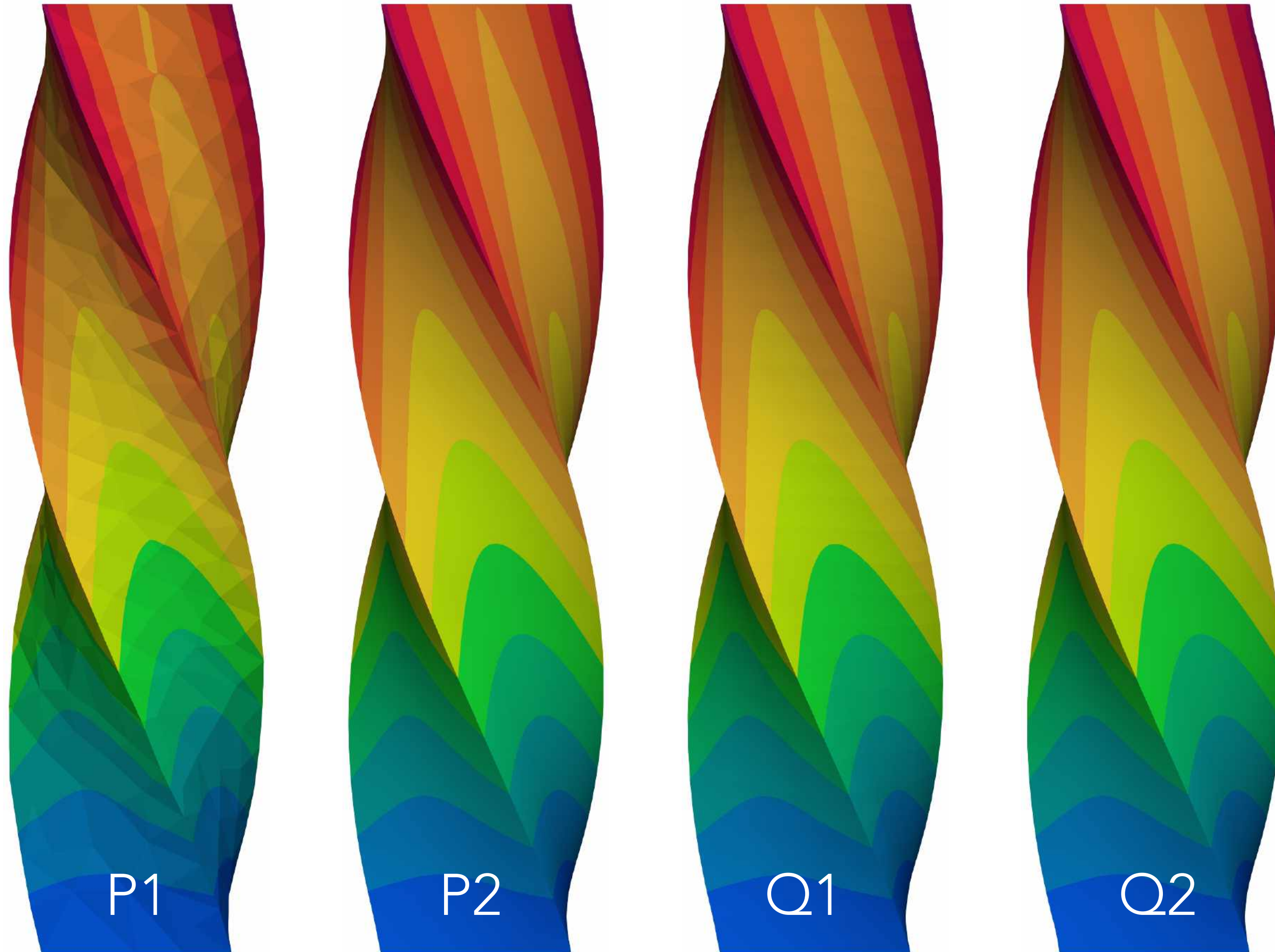
Q2



Mixed

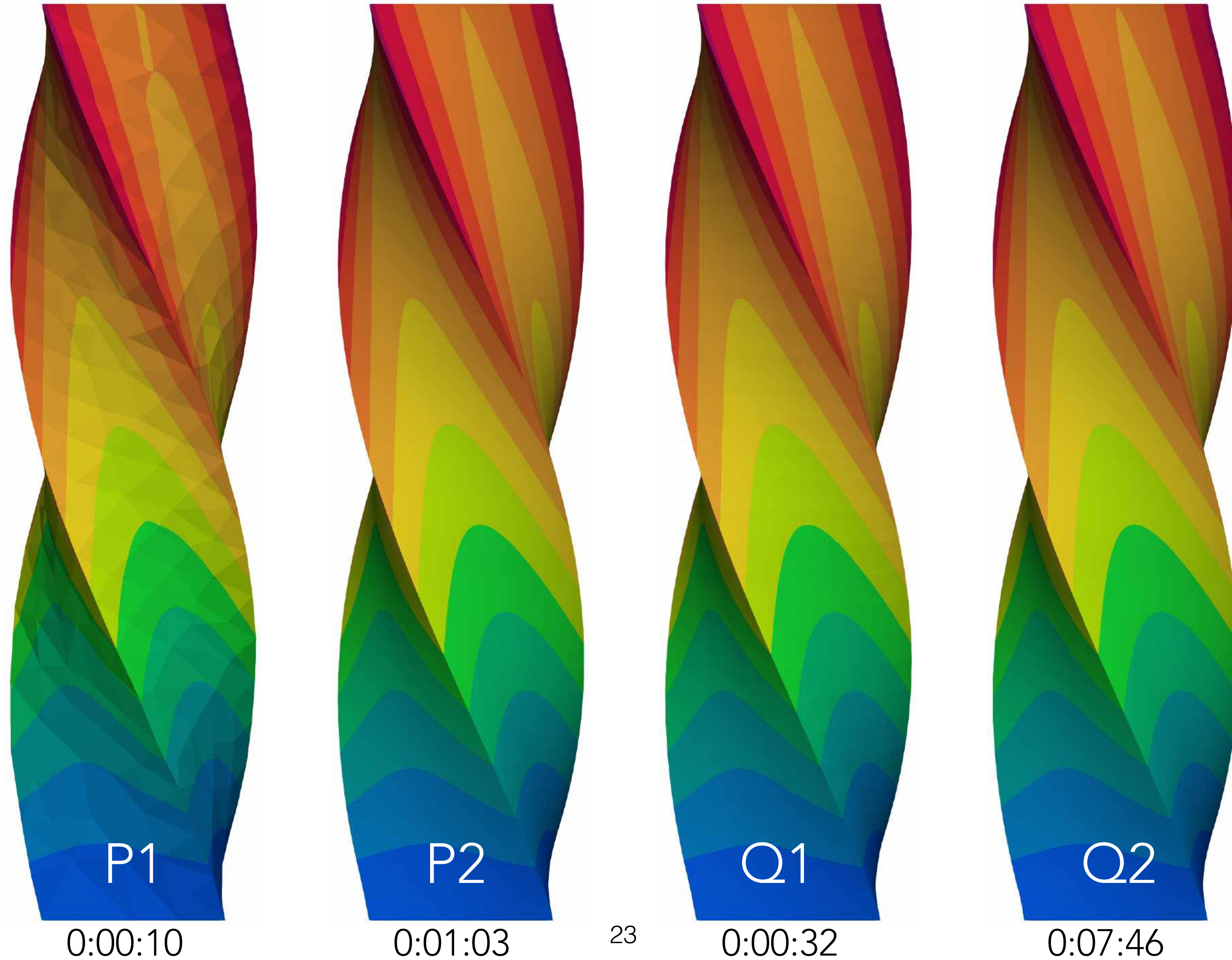


# Neo-Hooke – Coarse



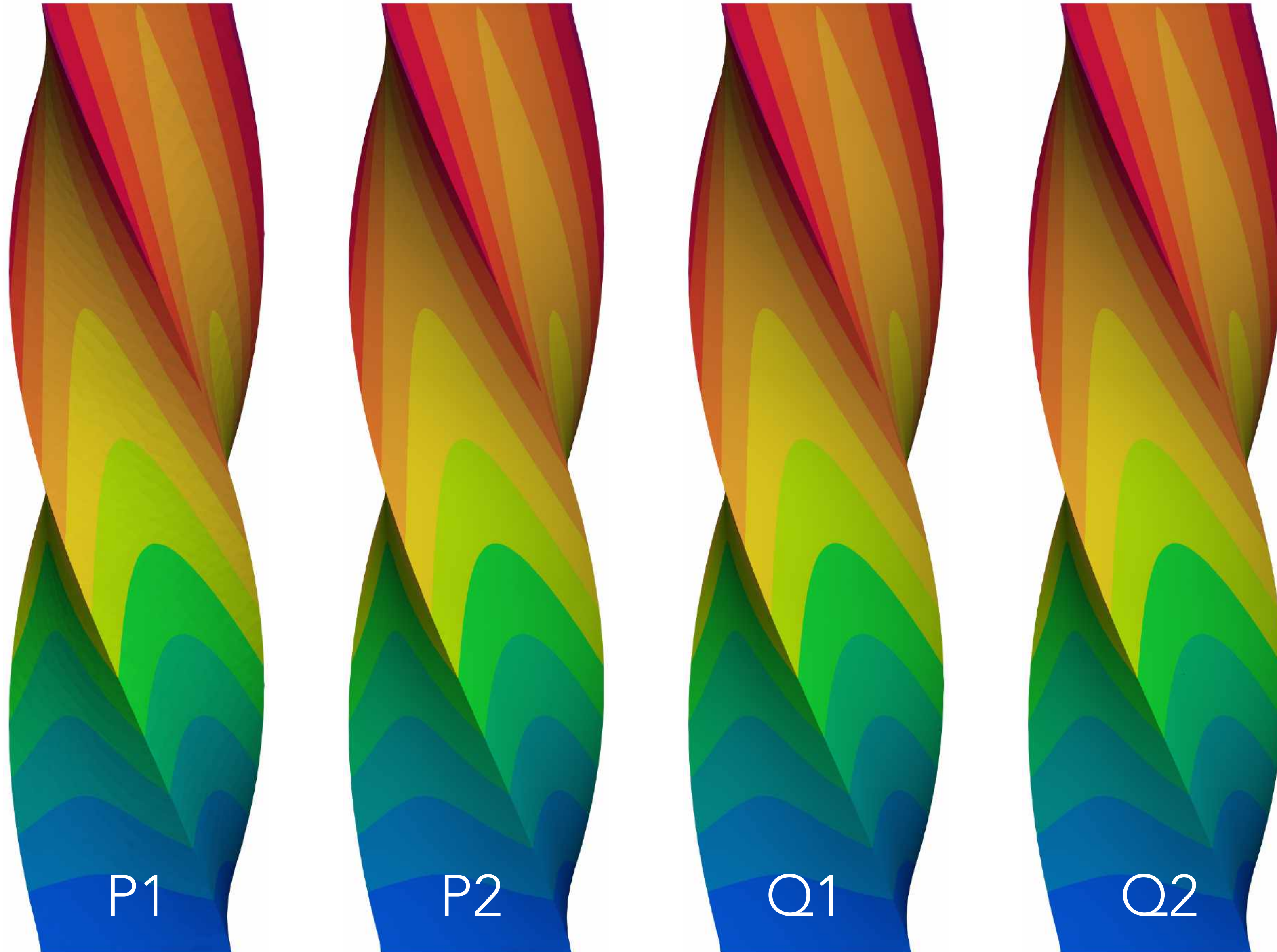


# Neo-Hooke – Coarse



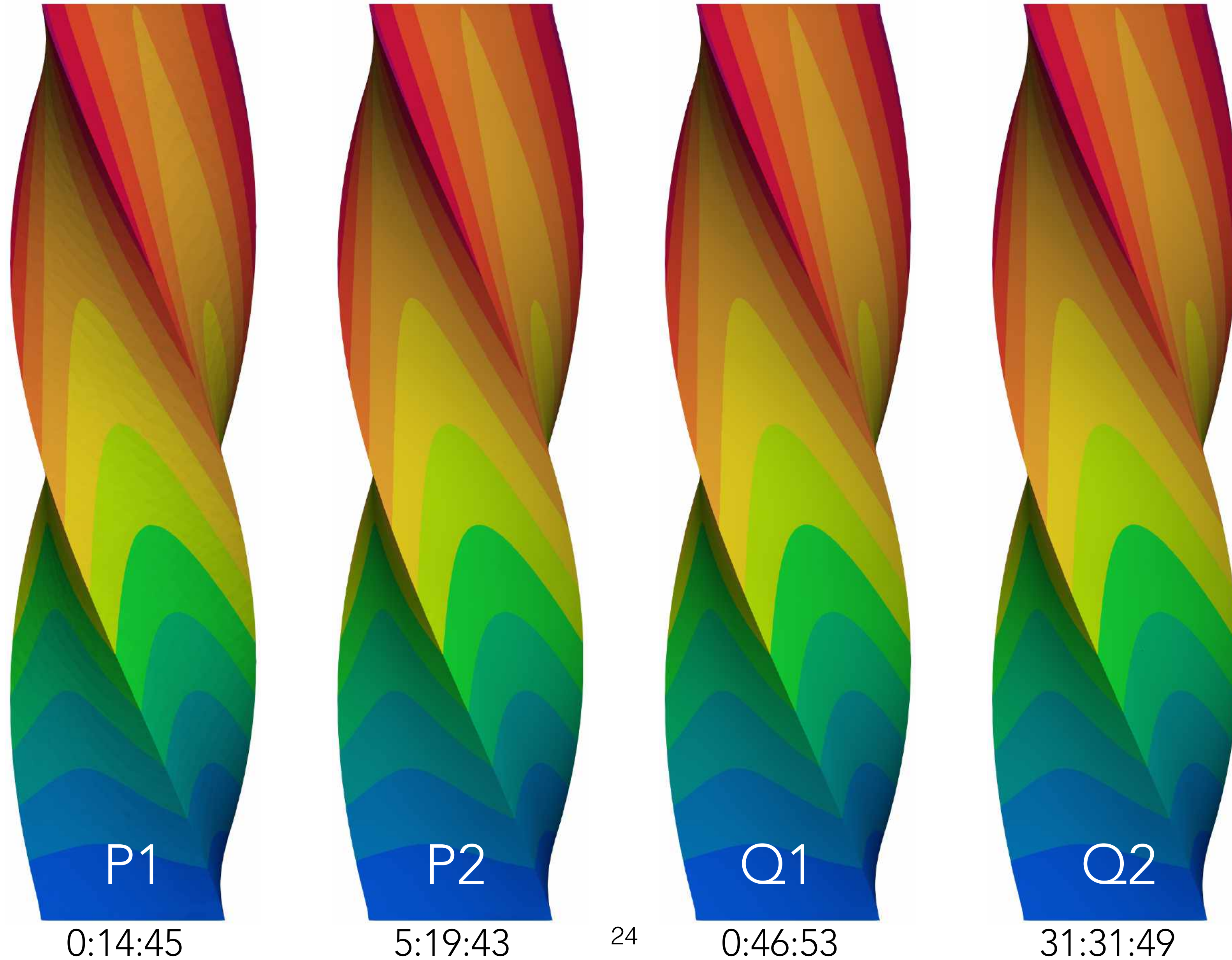


# Neo-Hooke – Dense



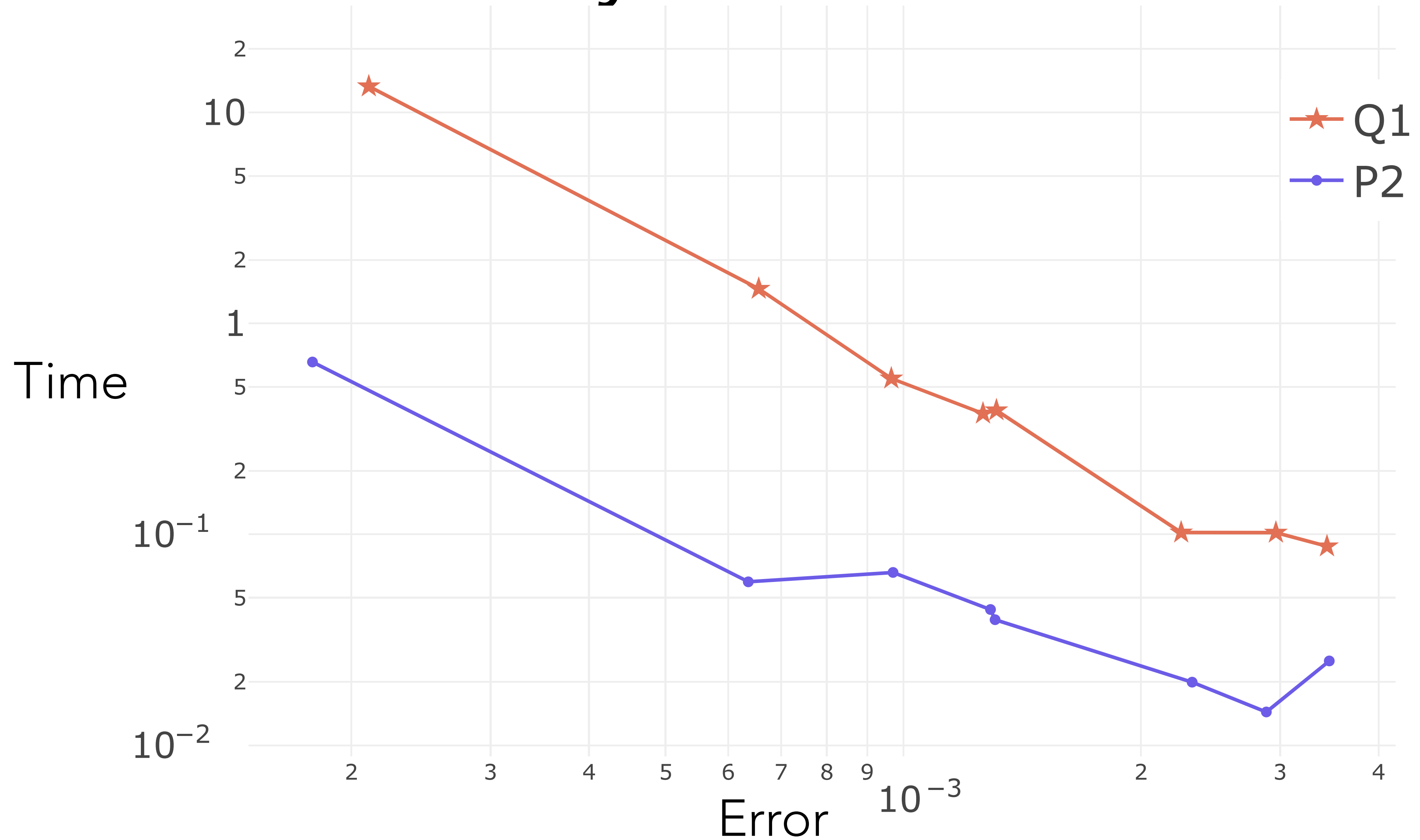


# Neo-Hooke – Dense



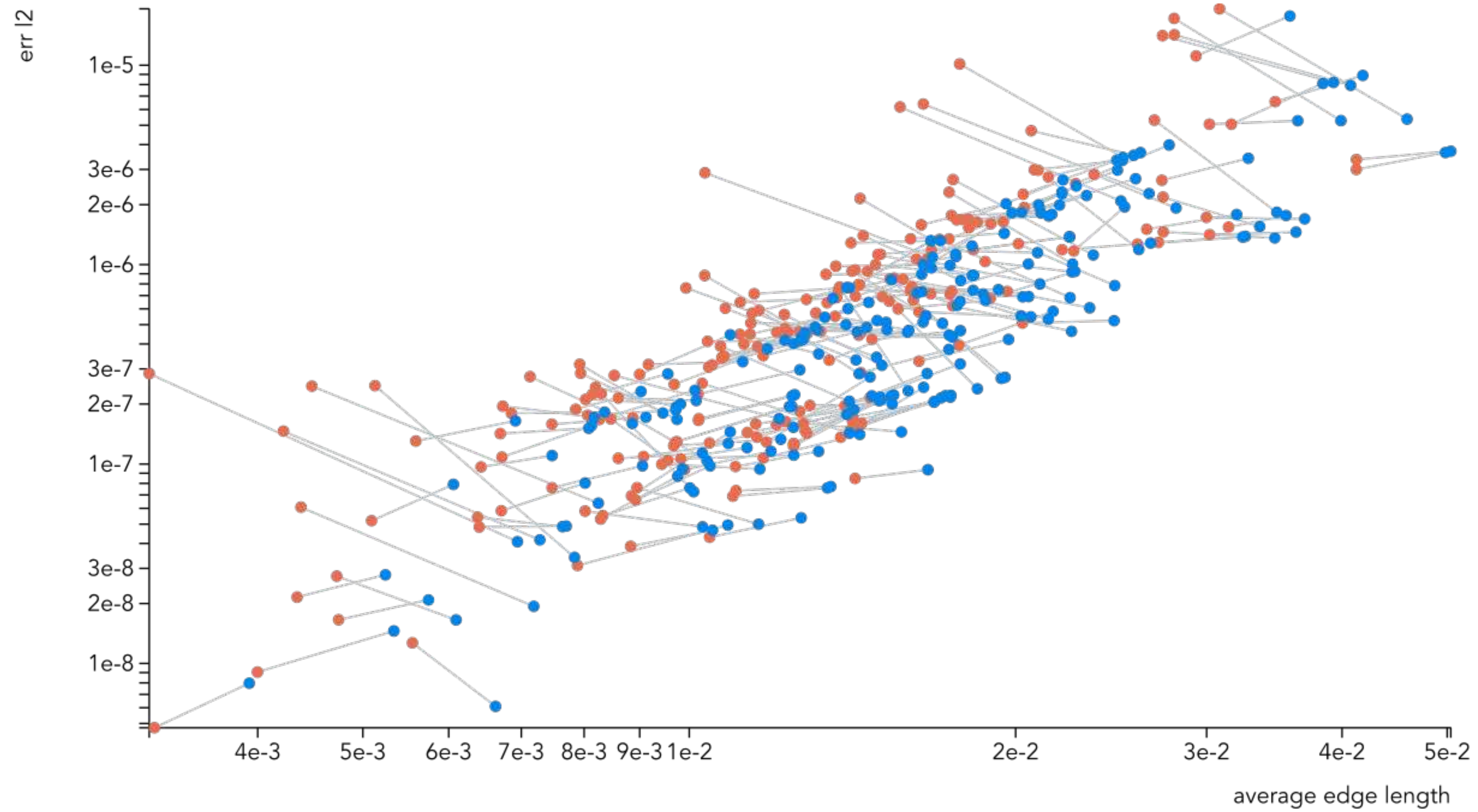


# Elasticity – Bended Bar



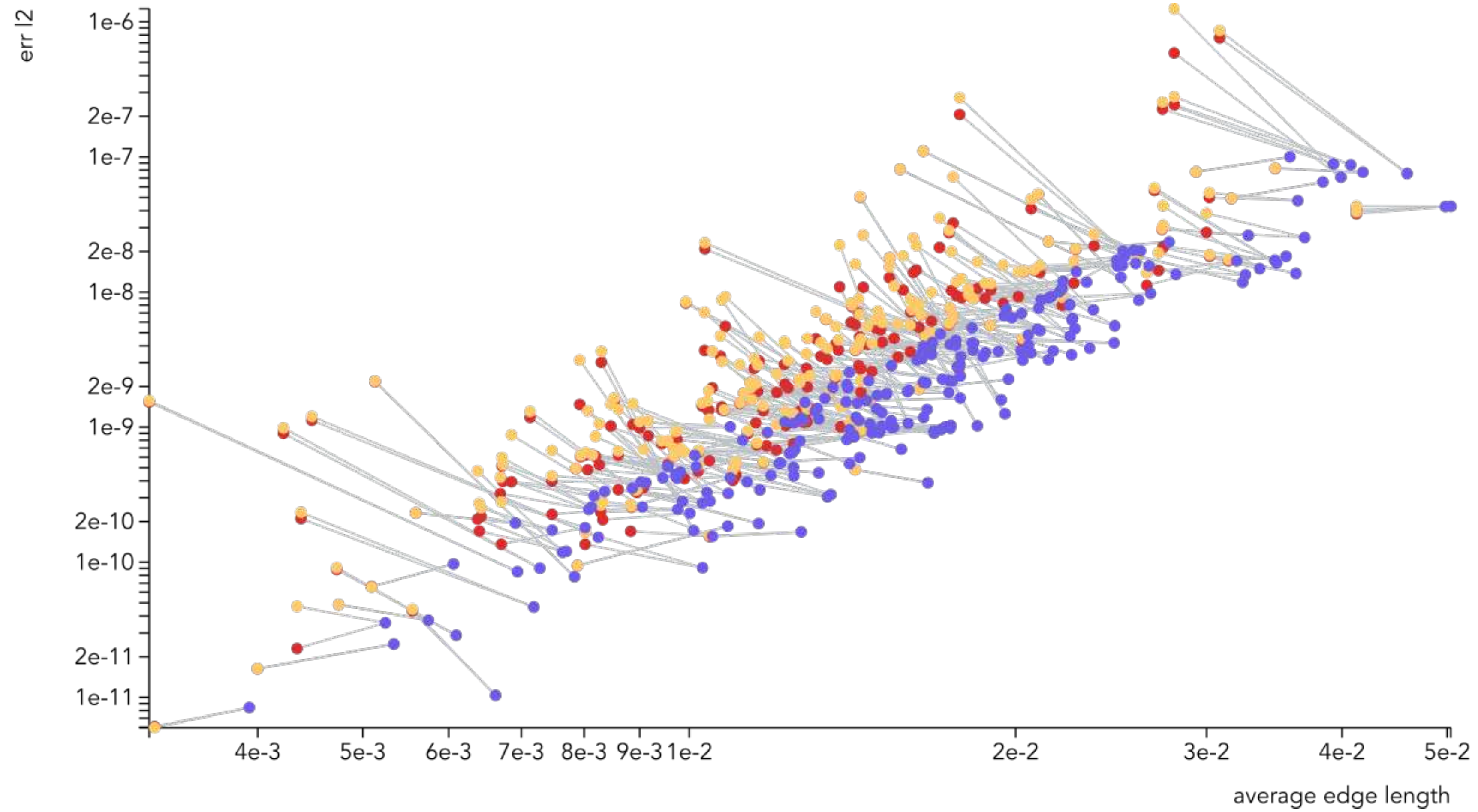


# Hexalab – no-flips



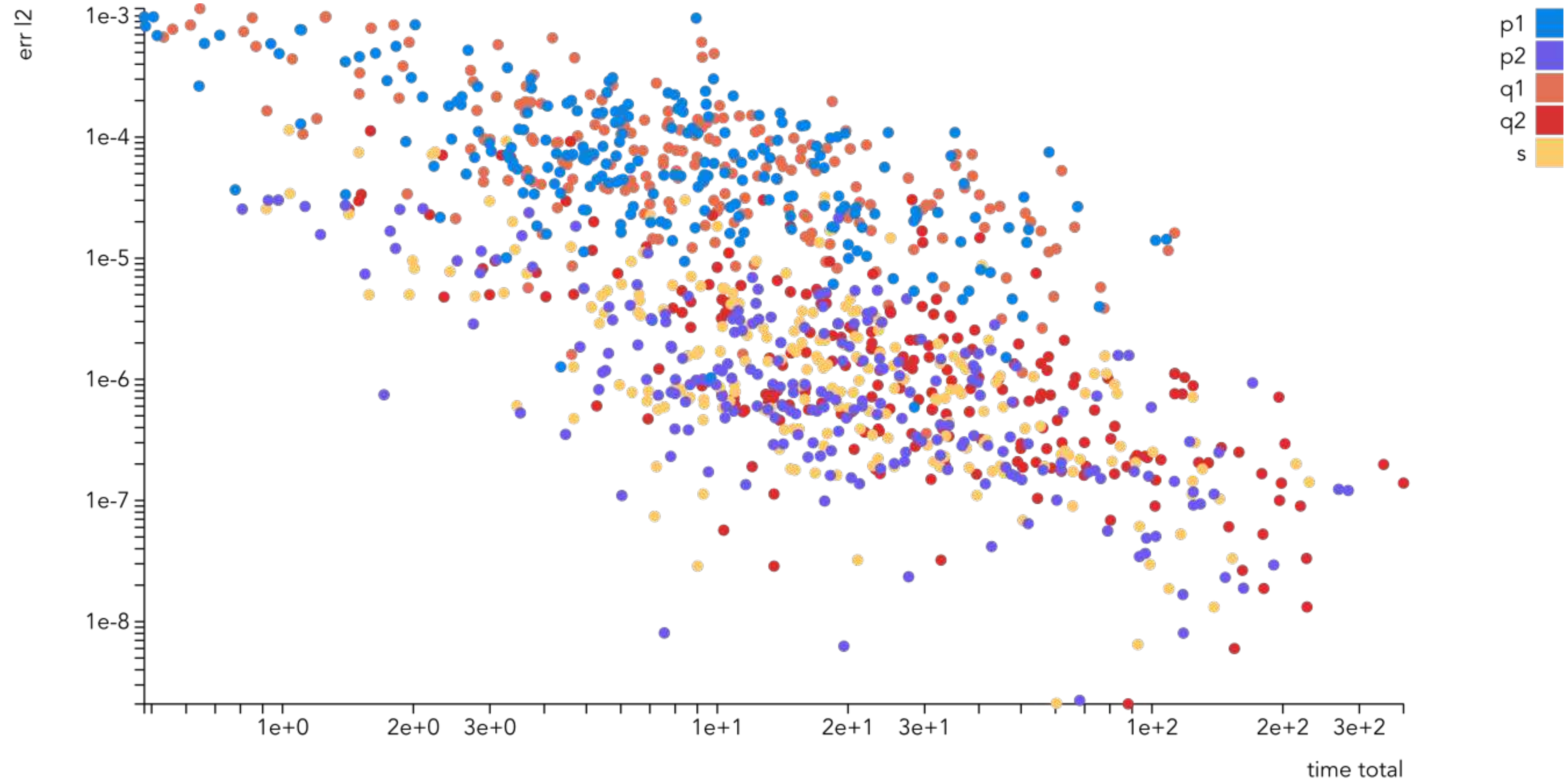


# Hexalab – no-flips



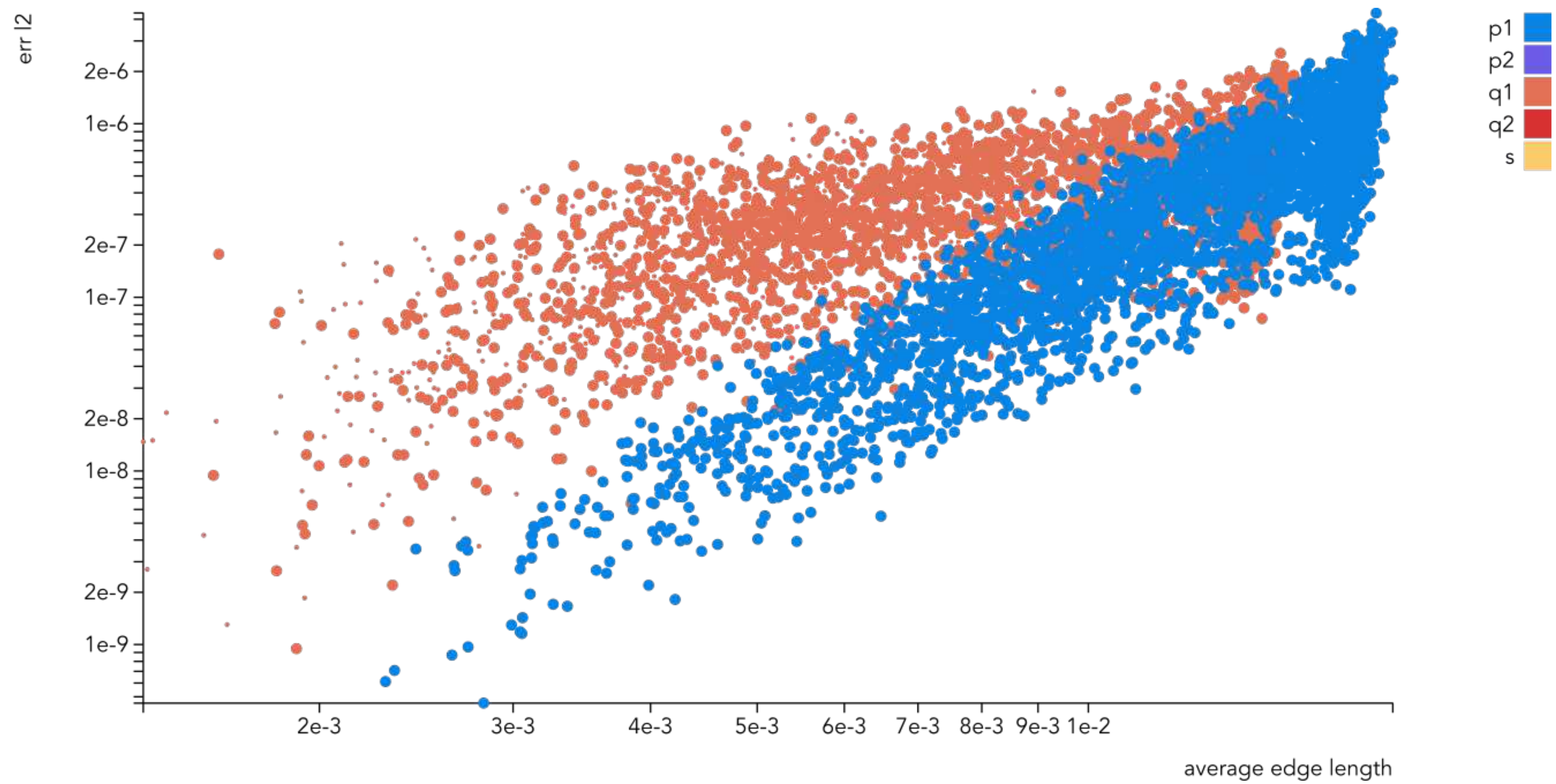


# Hexalab – no-flips



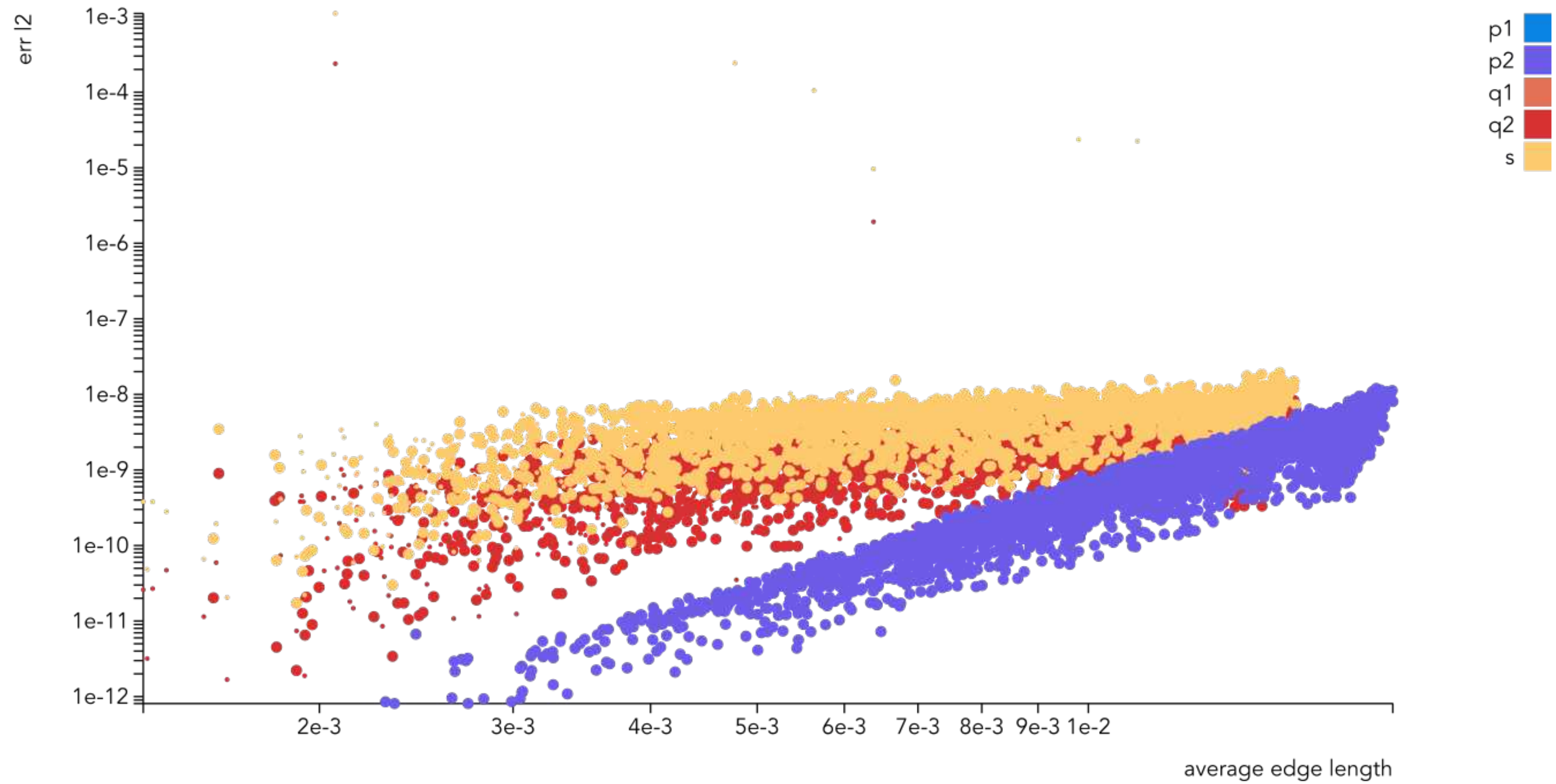


# Thingi10k



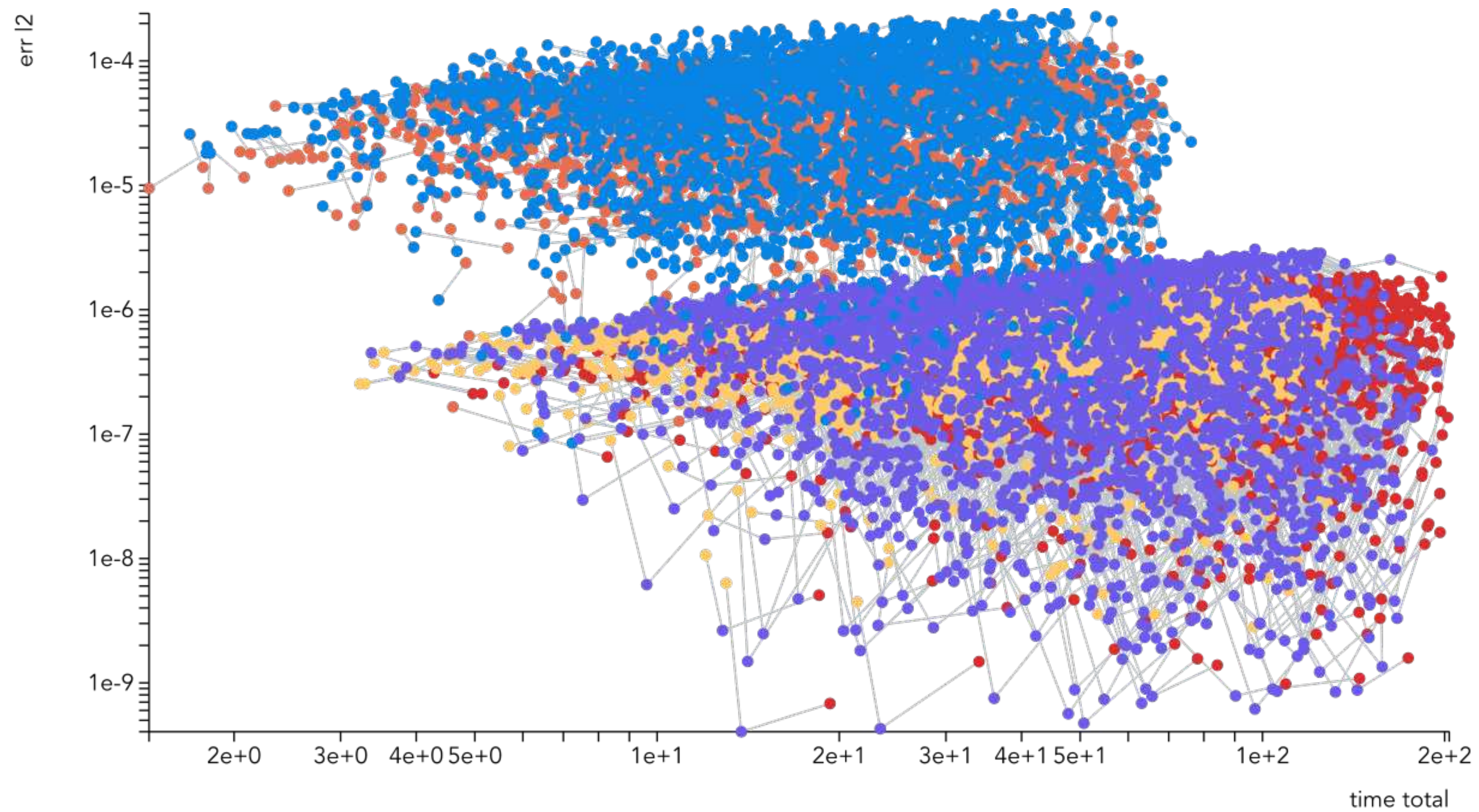


# Thingi10k



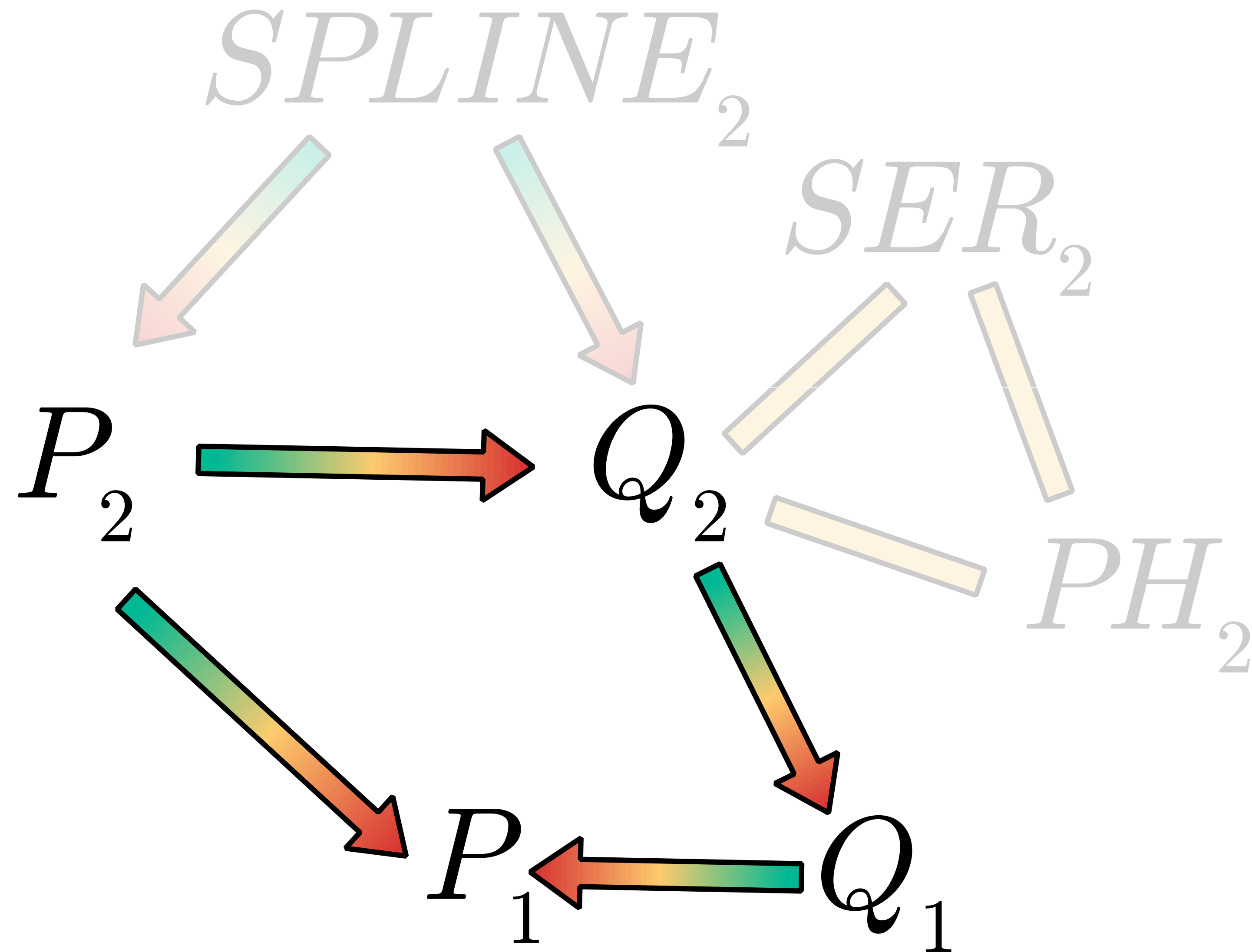


# Thingi10k



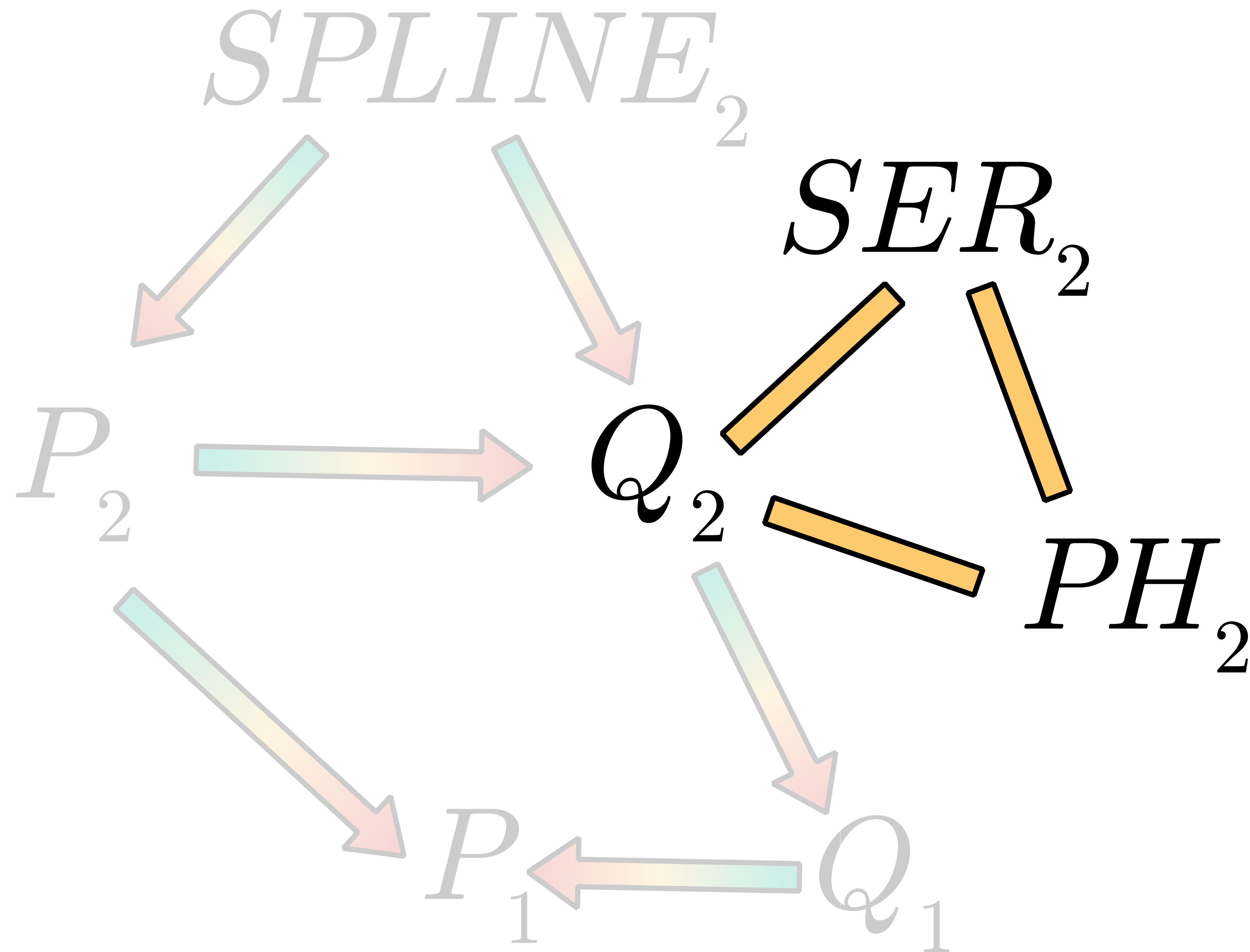


# Element Summary



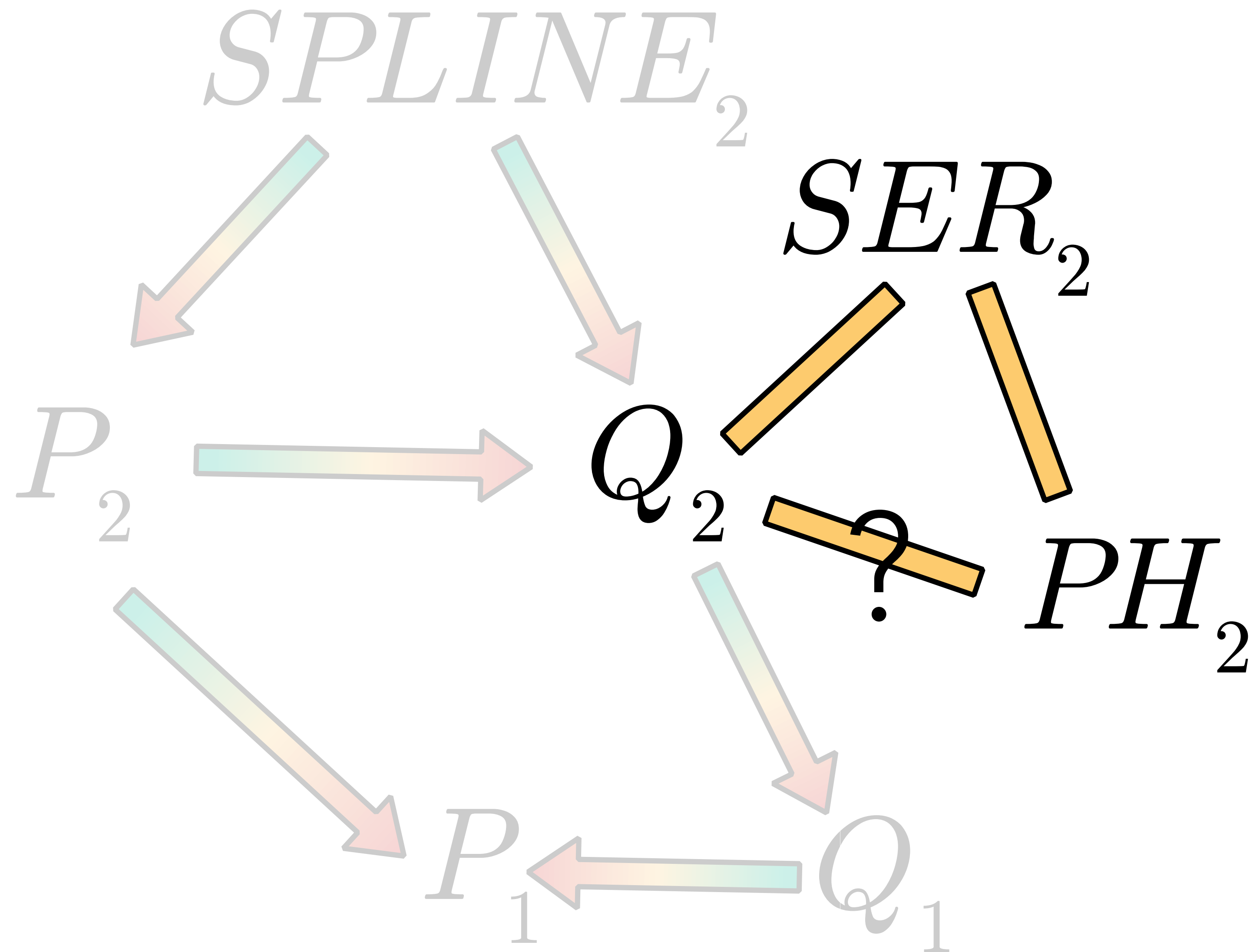


# Element Summary



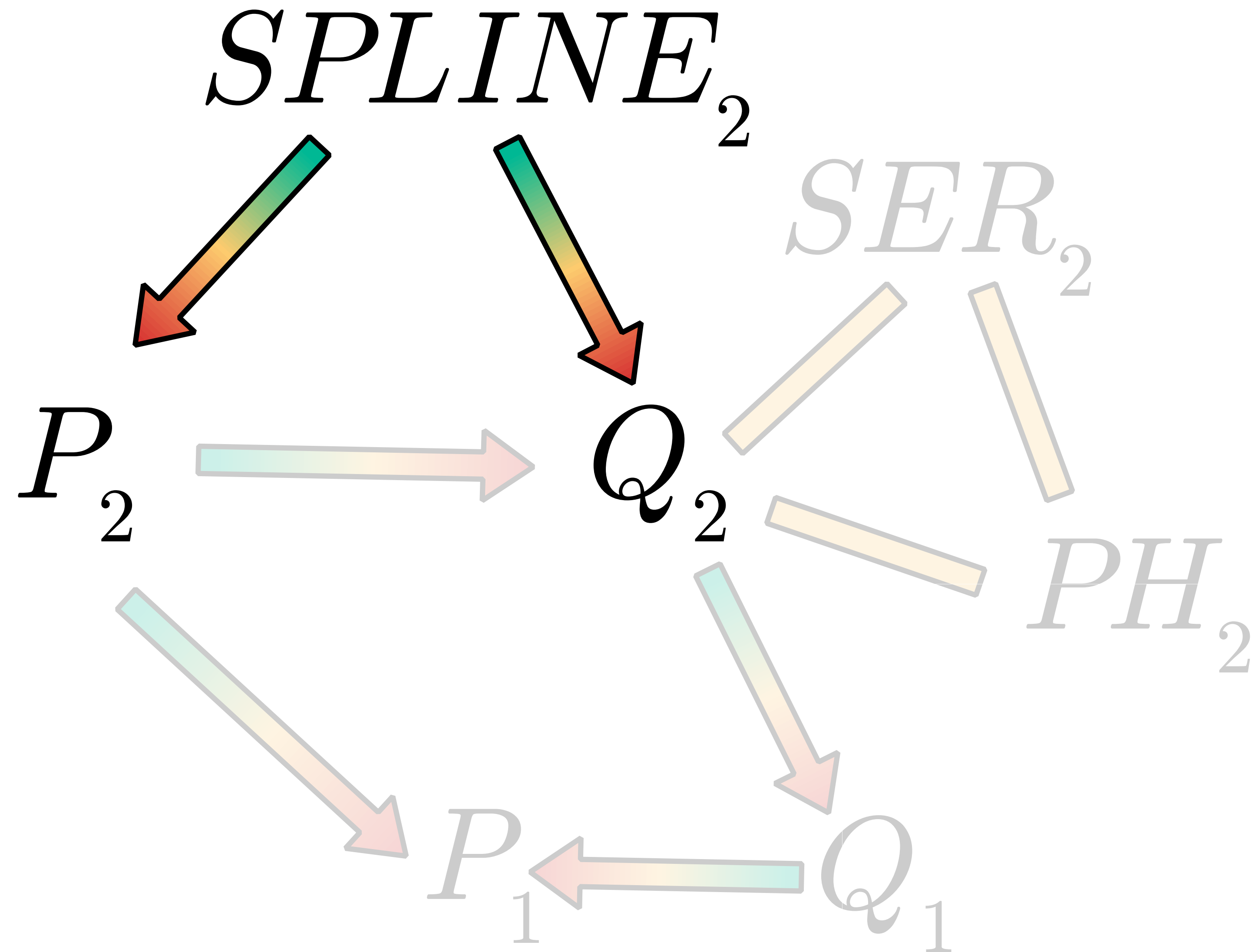


# Element Summary



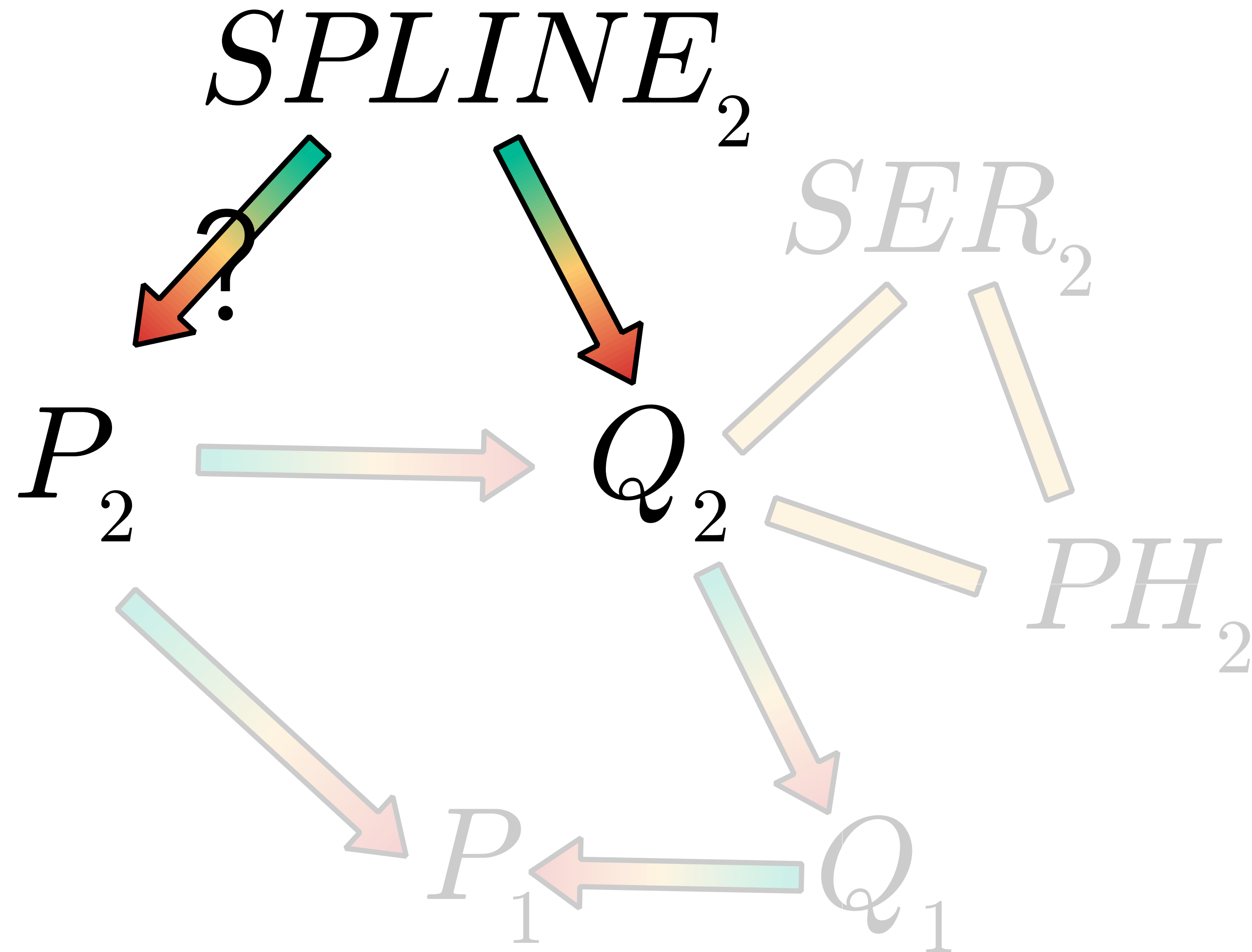


# Element Summary



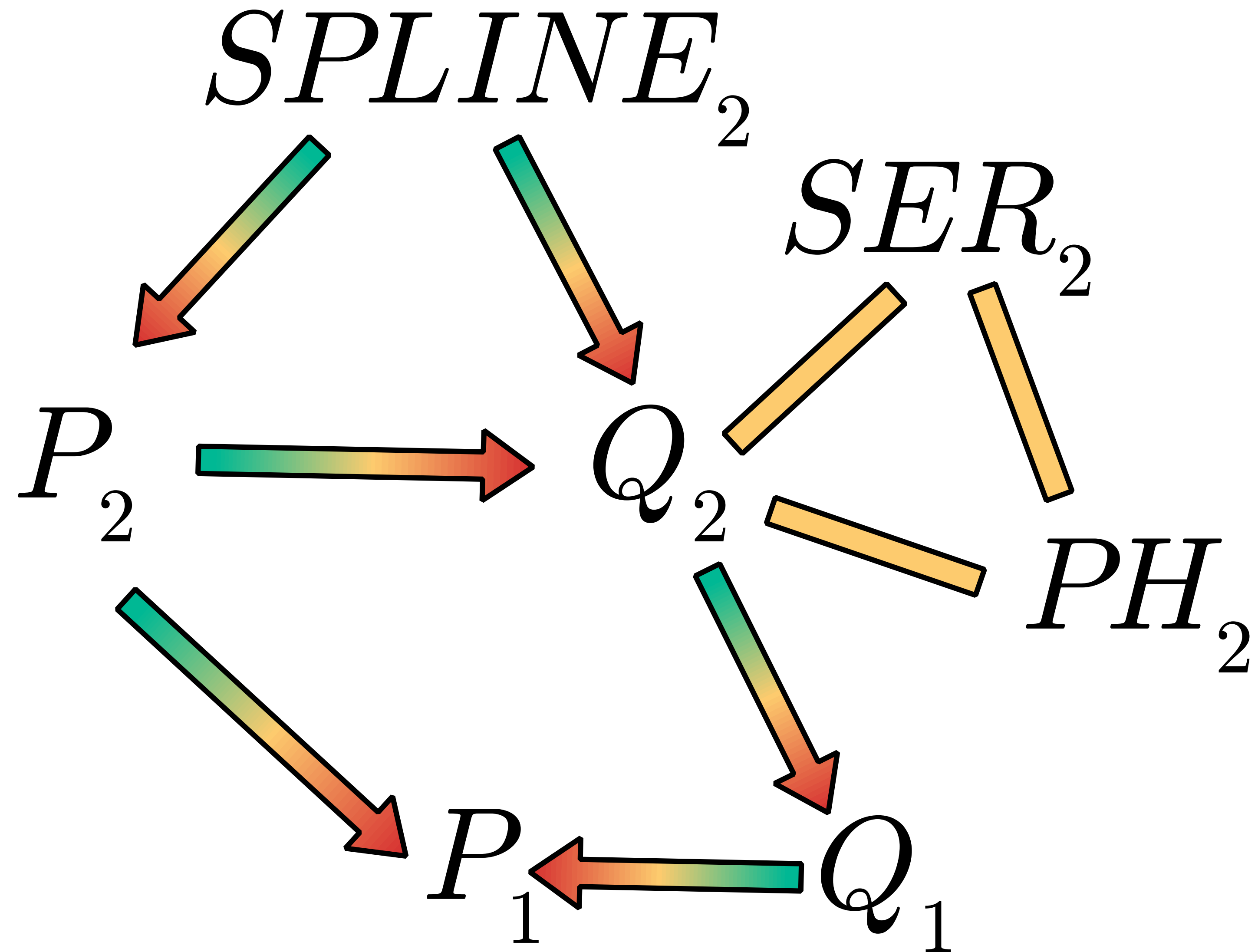


# Element Summary



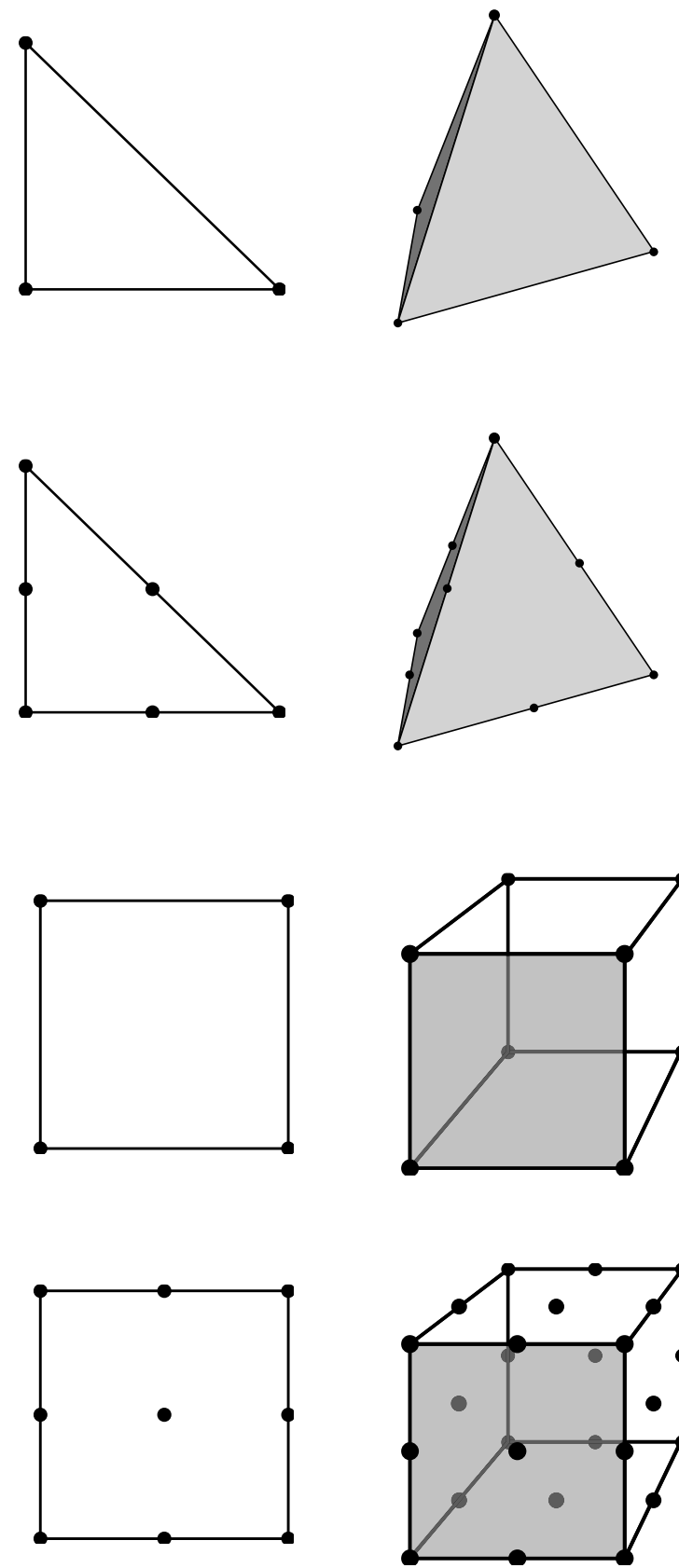


# Element Summary

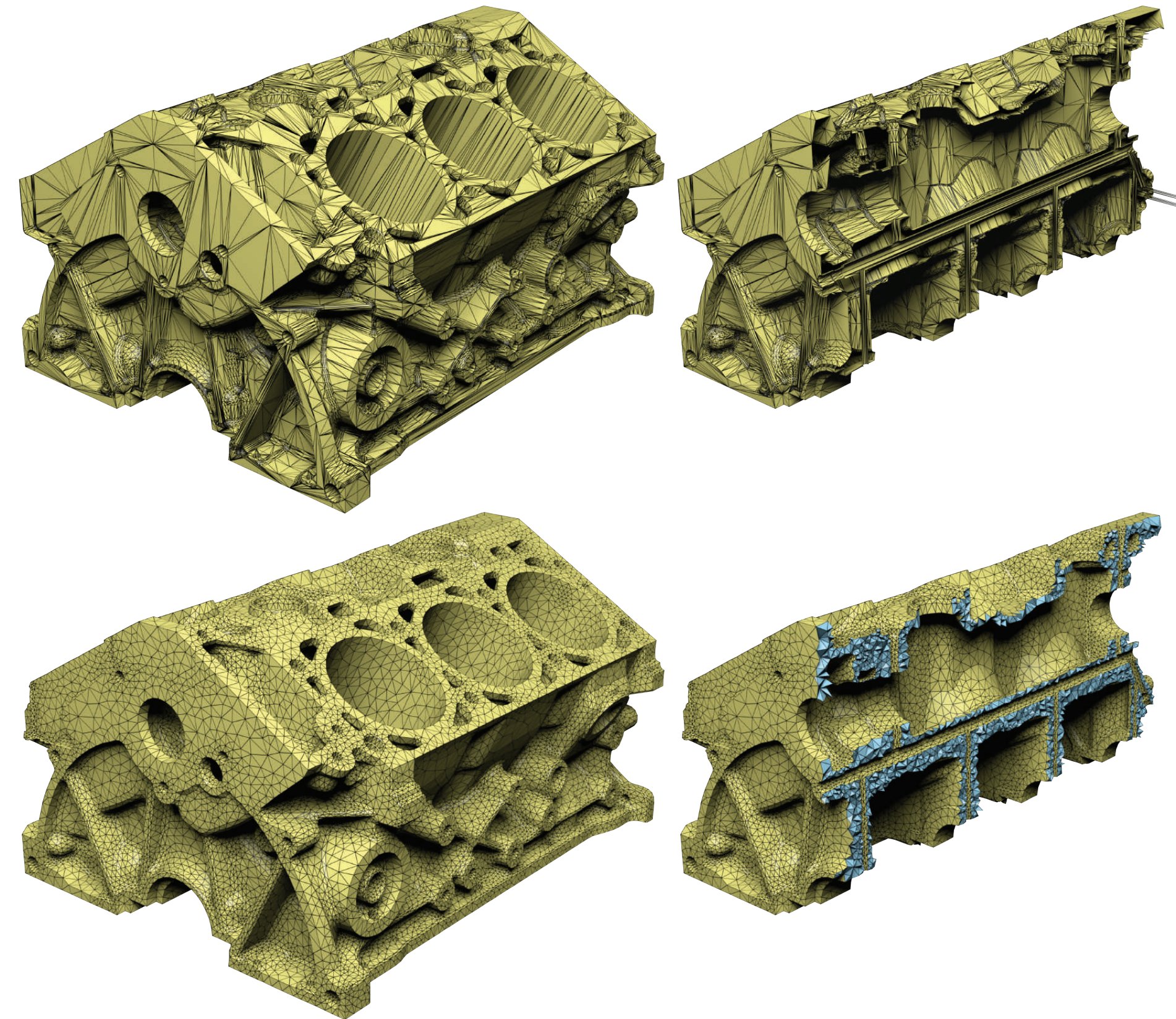




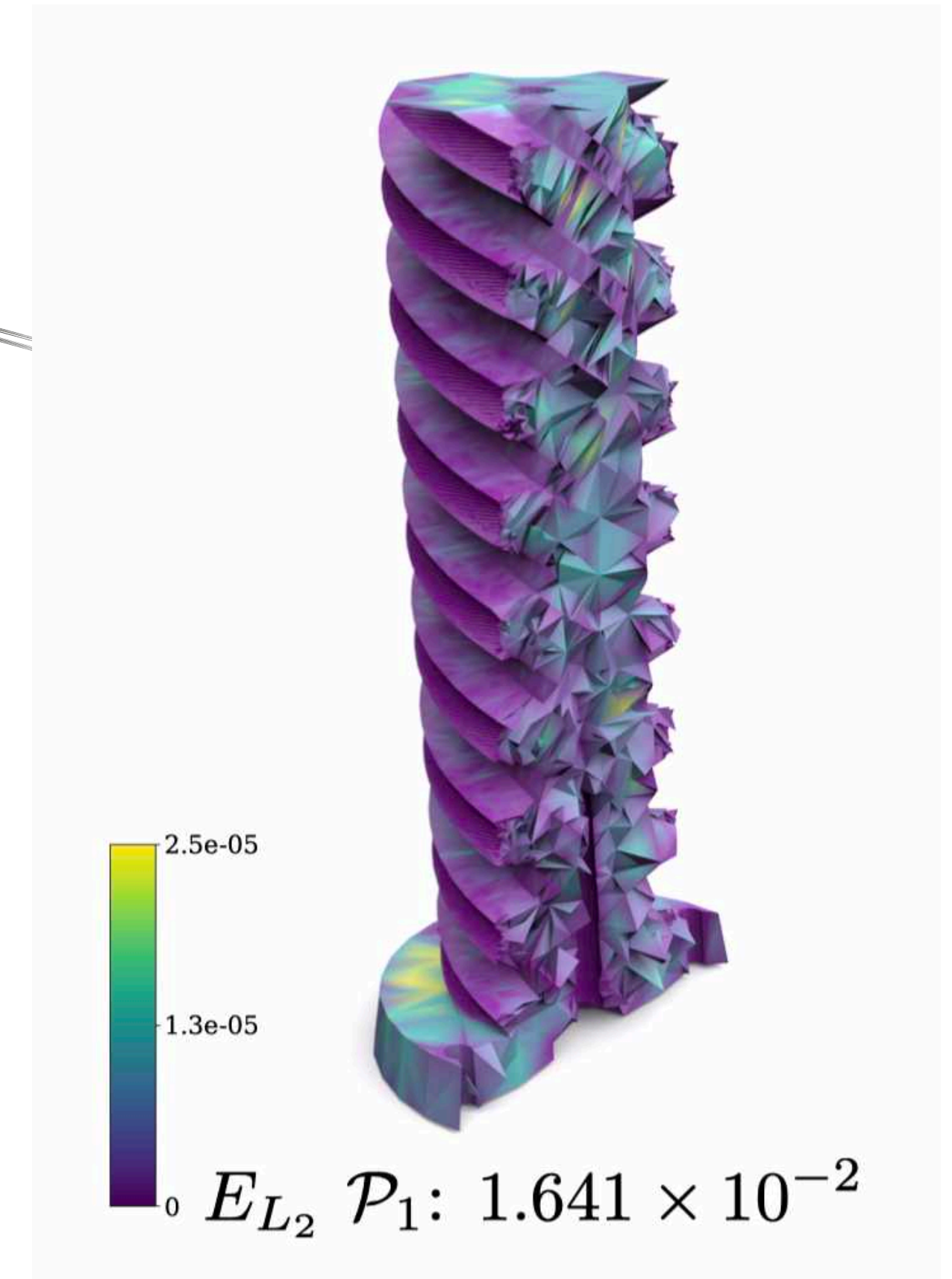
# Overview



Which discretization provides lower running time for a fixed accuracy?



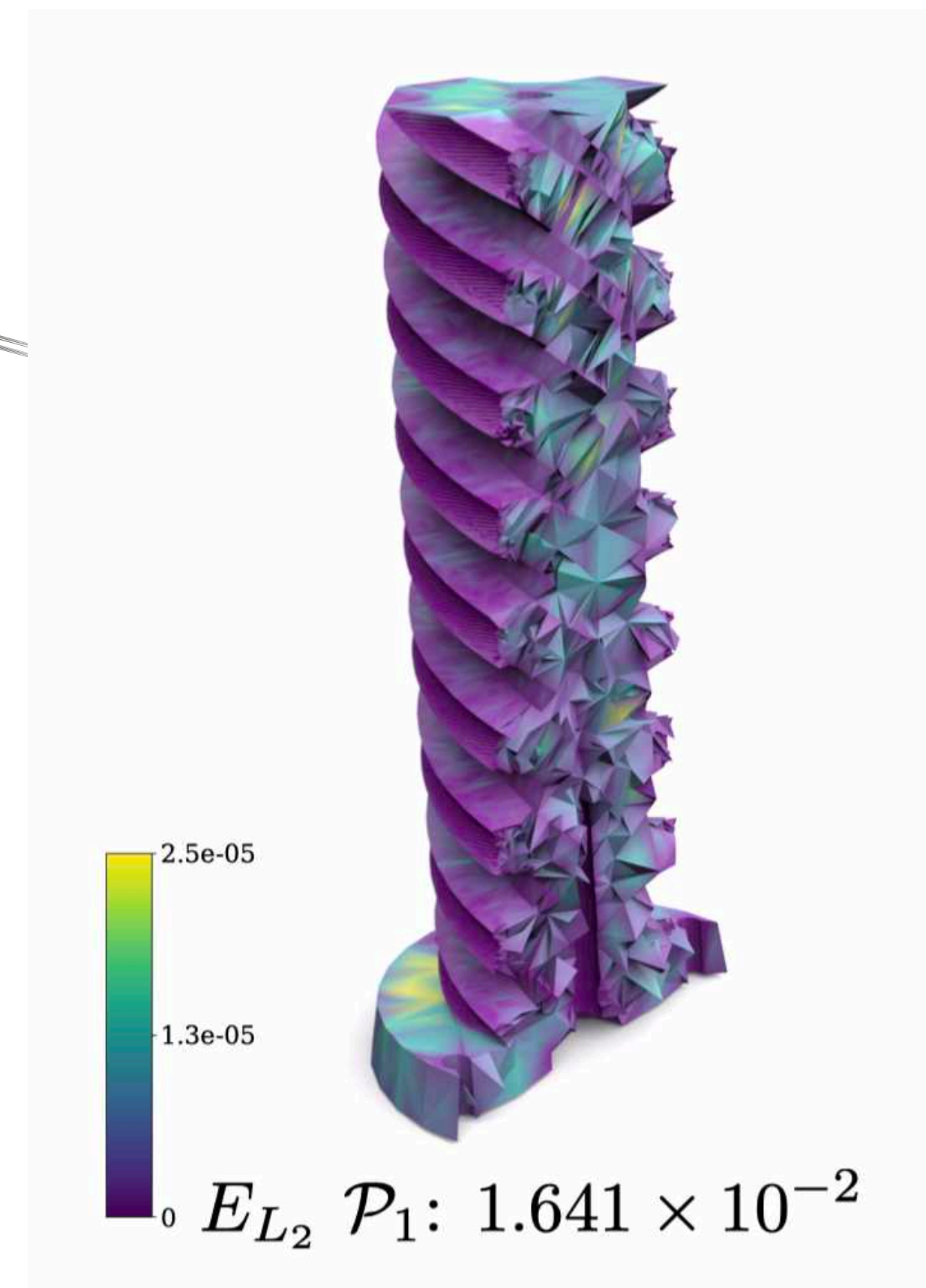
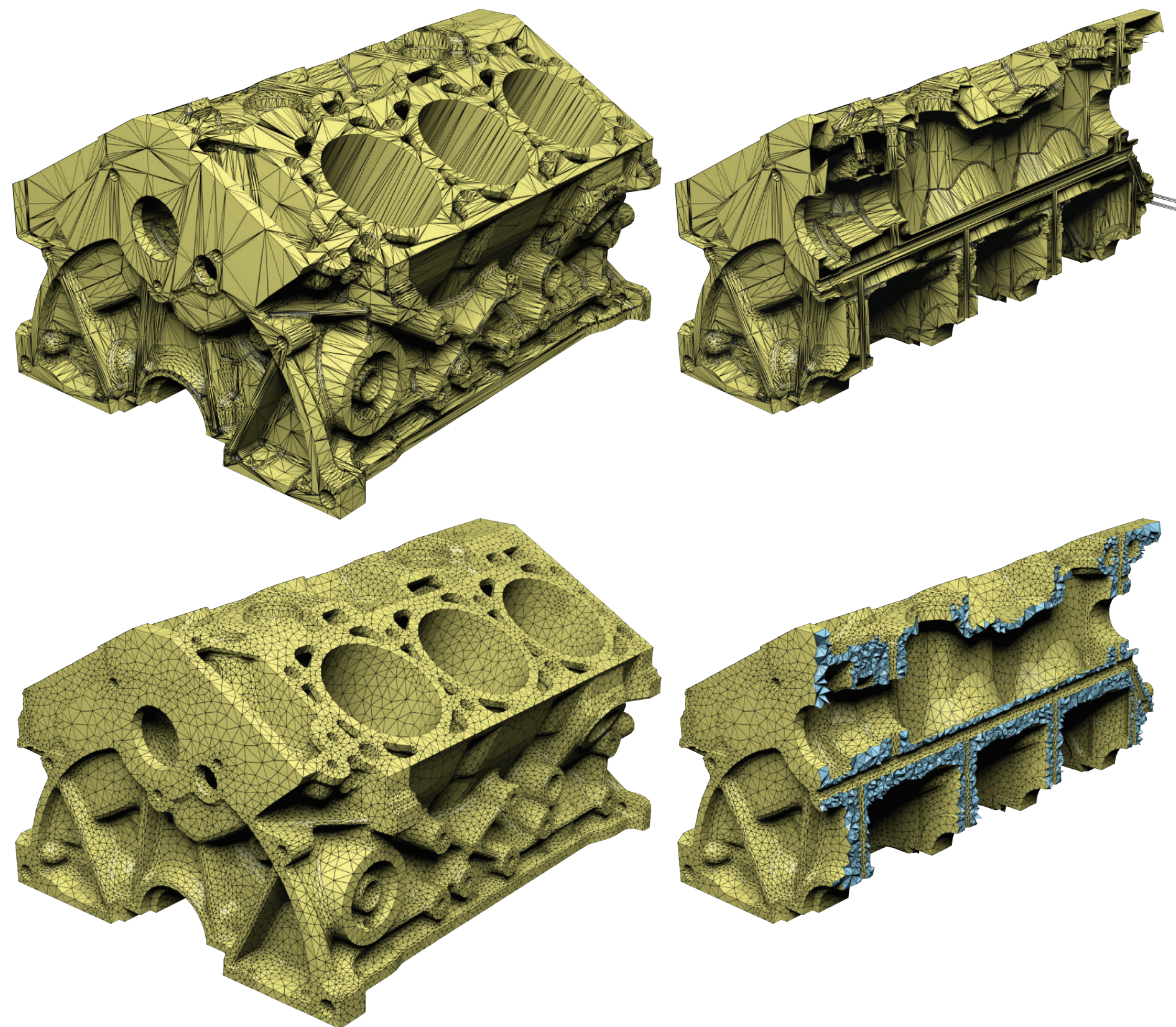
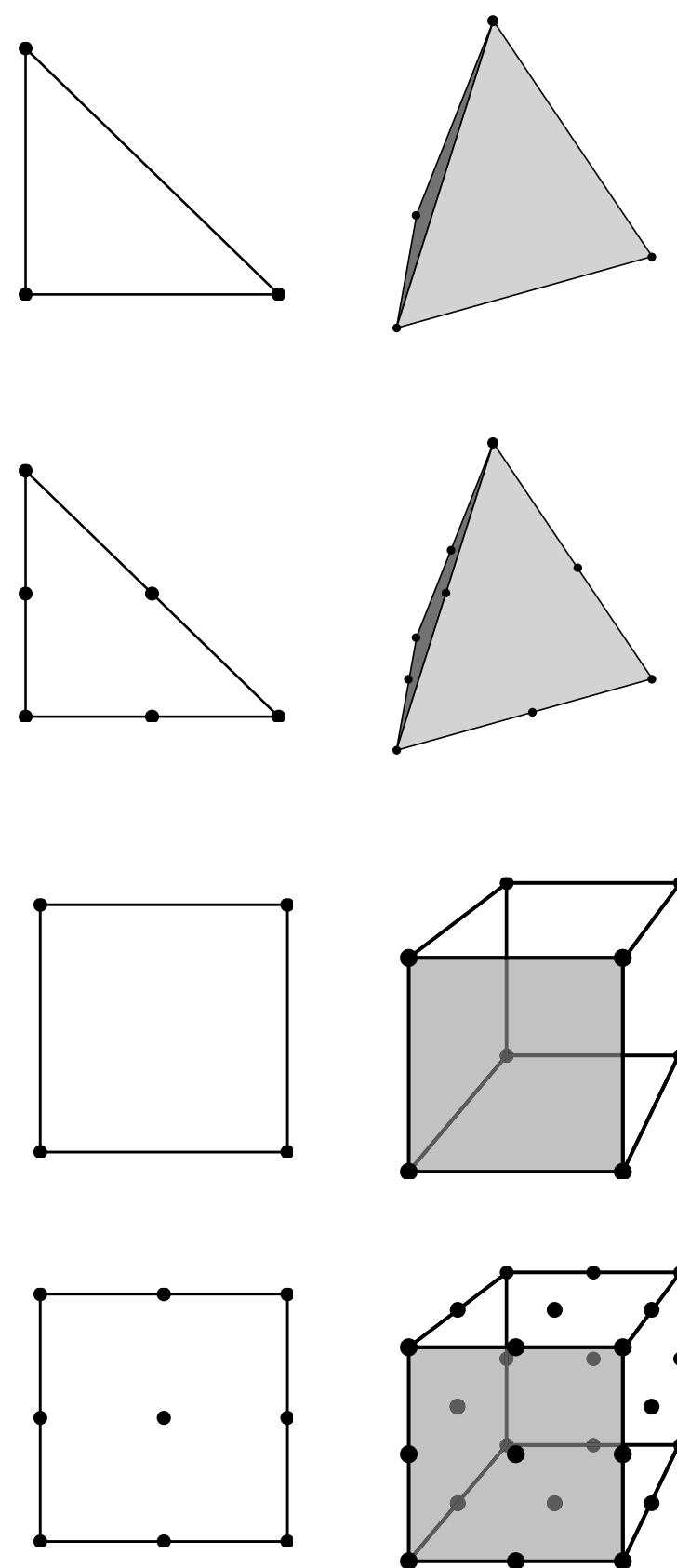
Can you mesh robustly without any assumption on the input?



Does mesh quality affect the accuracy of the FEM solution?



# Overview



Which discretization provides lower running time for a fixed accuracy?

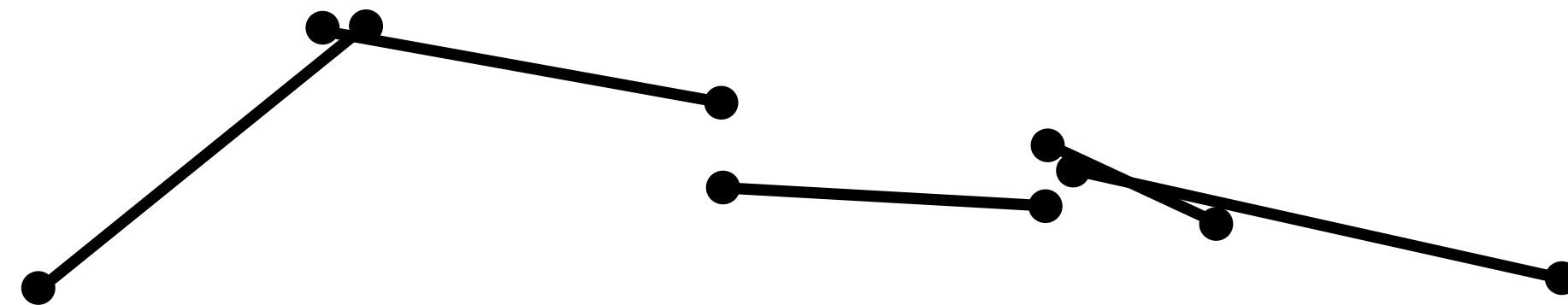
Can you mesh robustly without any assumption on the input?

Does mesh quality affect the accuracy of the FEM solution?



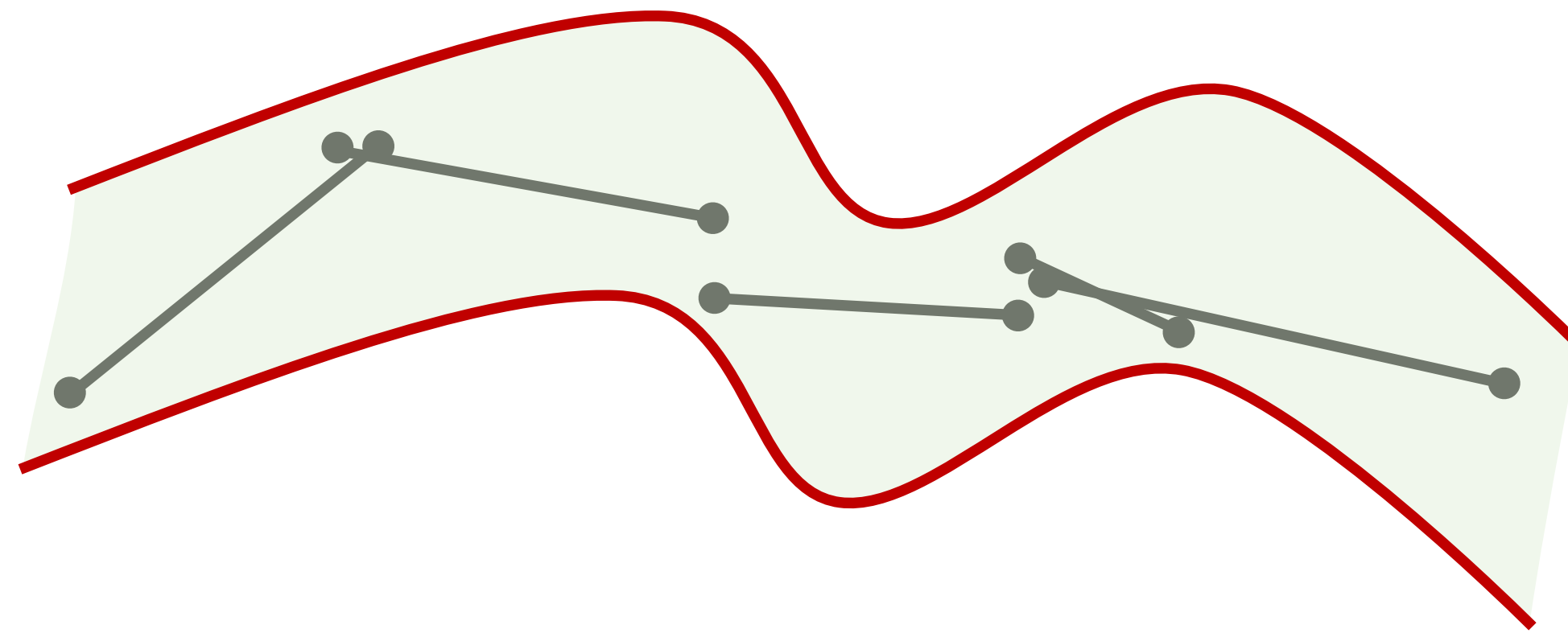
# Envelope

# Envelope

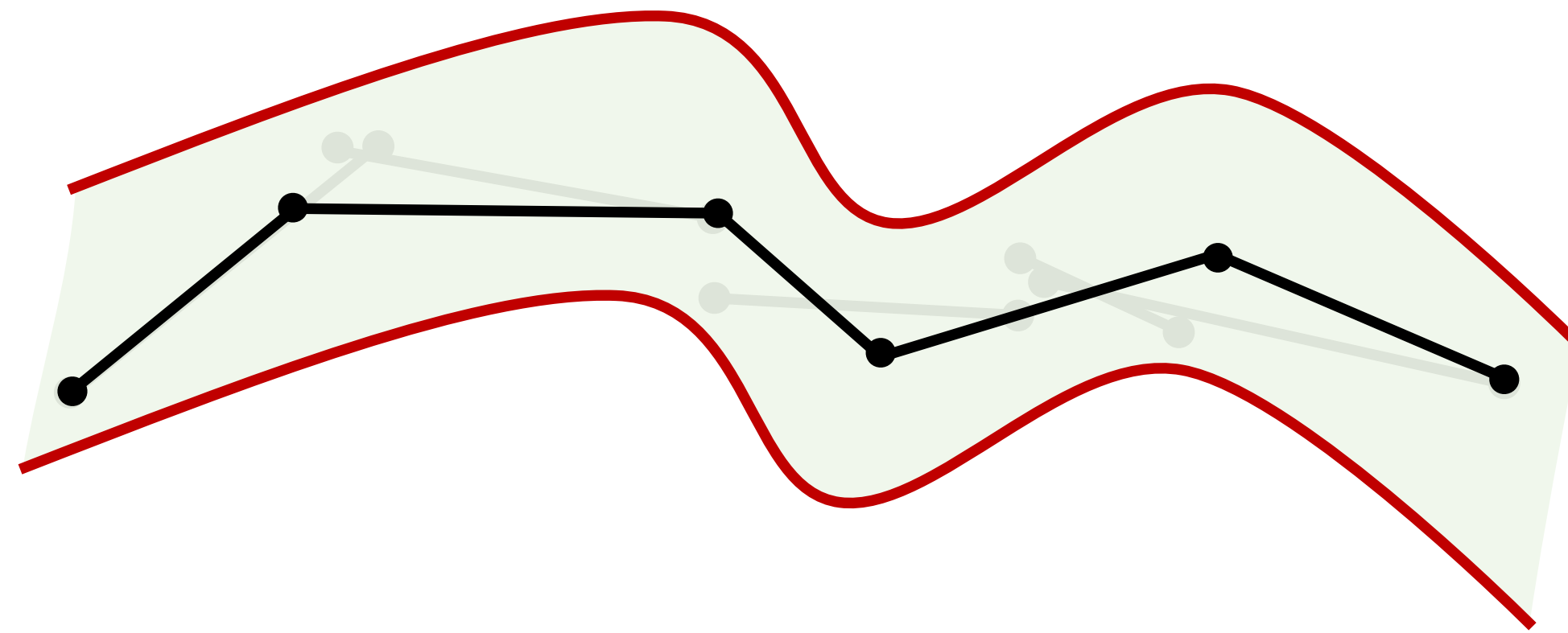




# Envelope

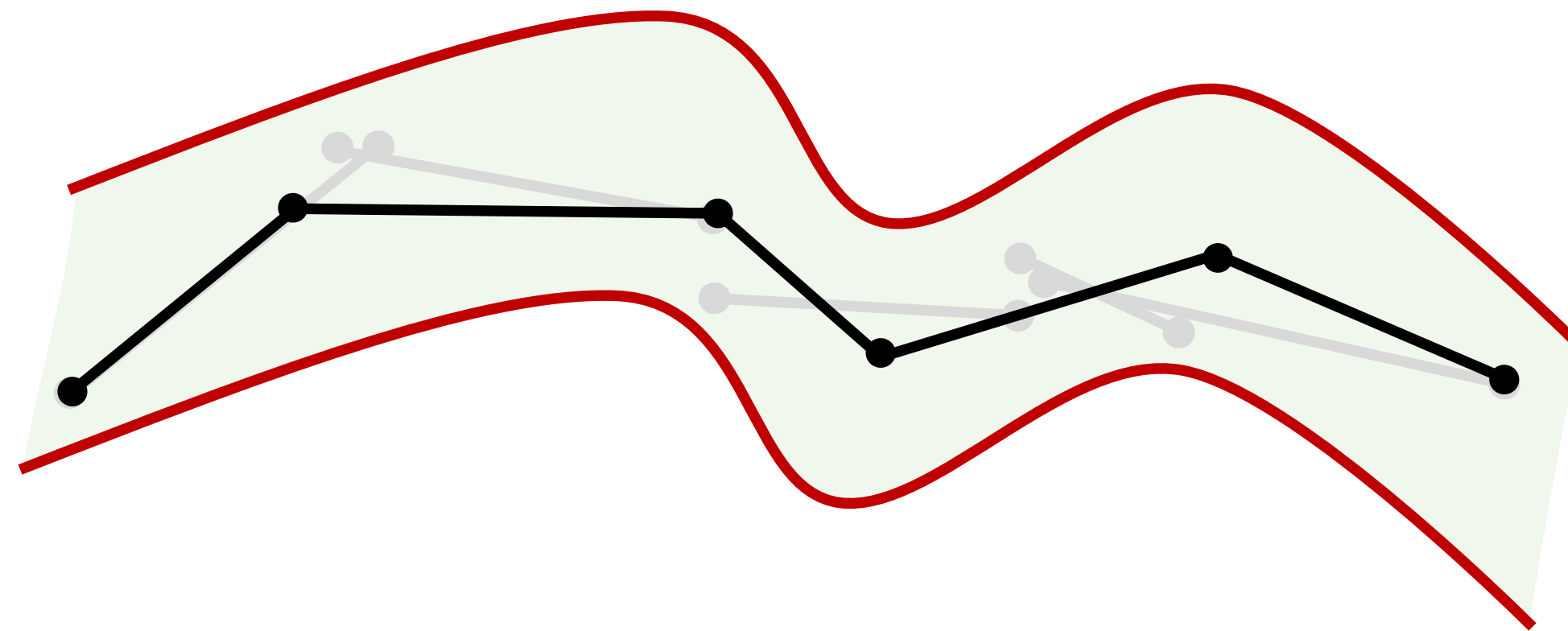


# Envelope



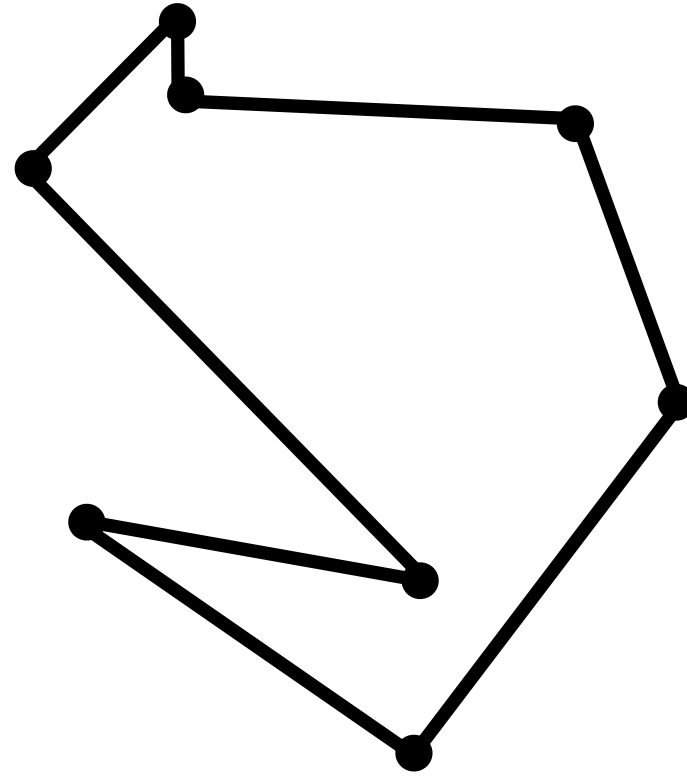


# Envelope



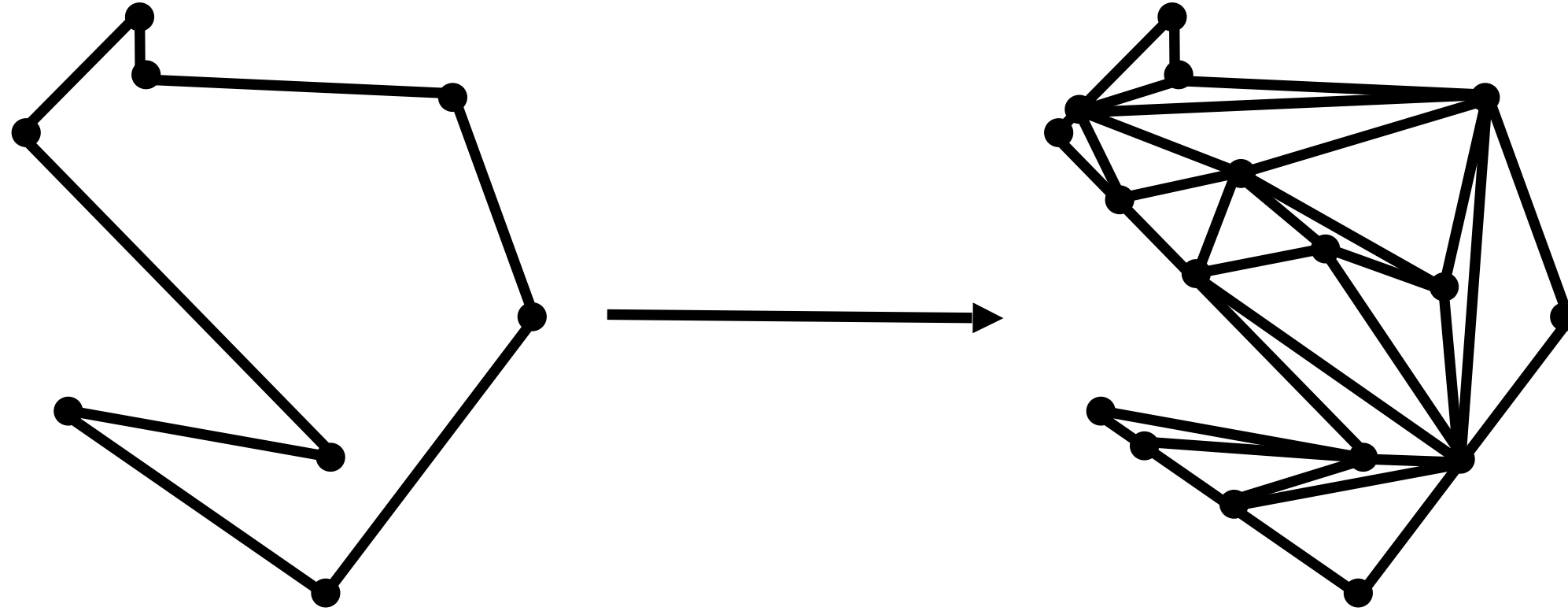


# Validity First

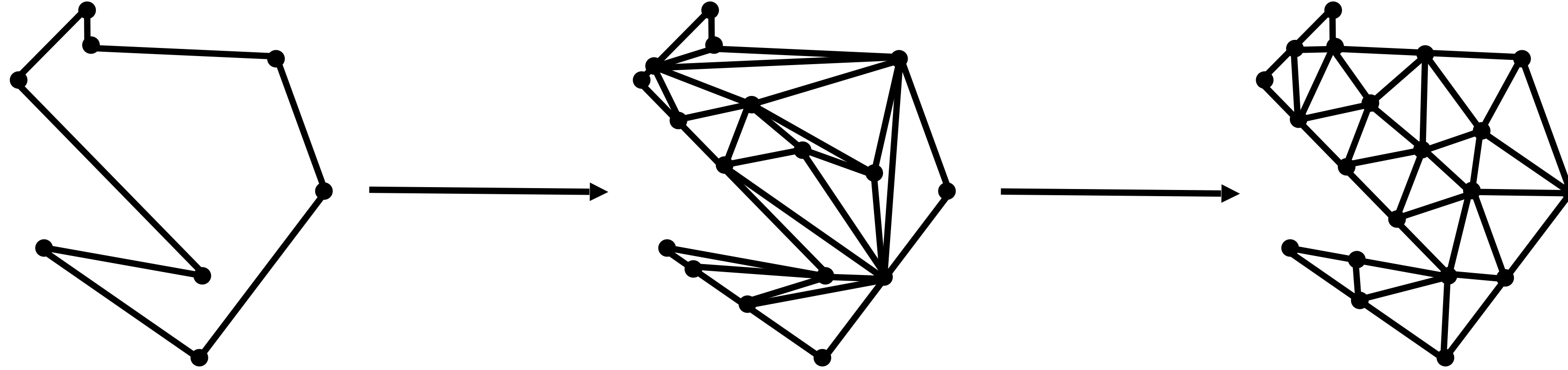




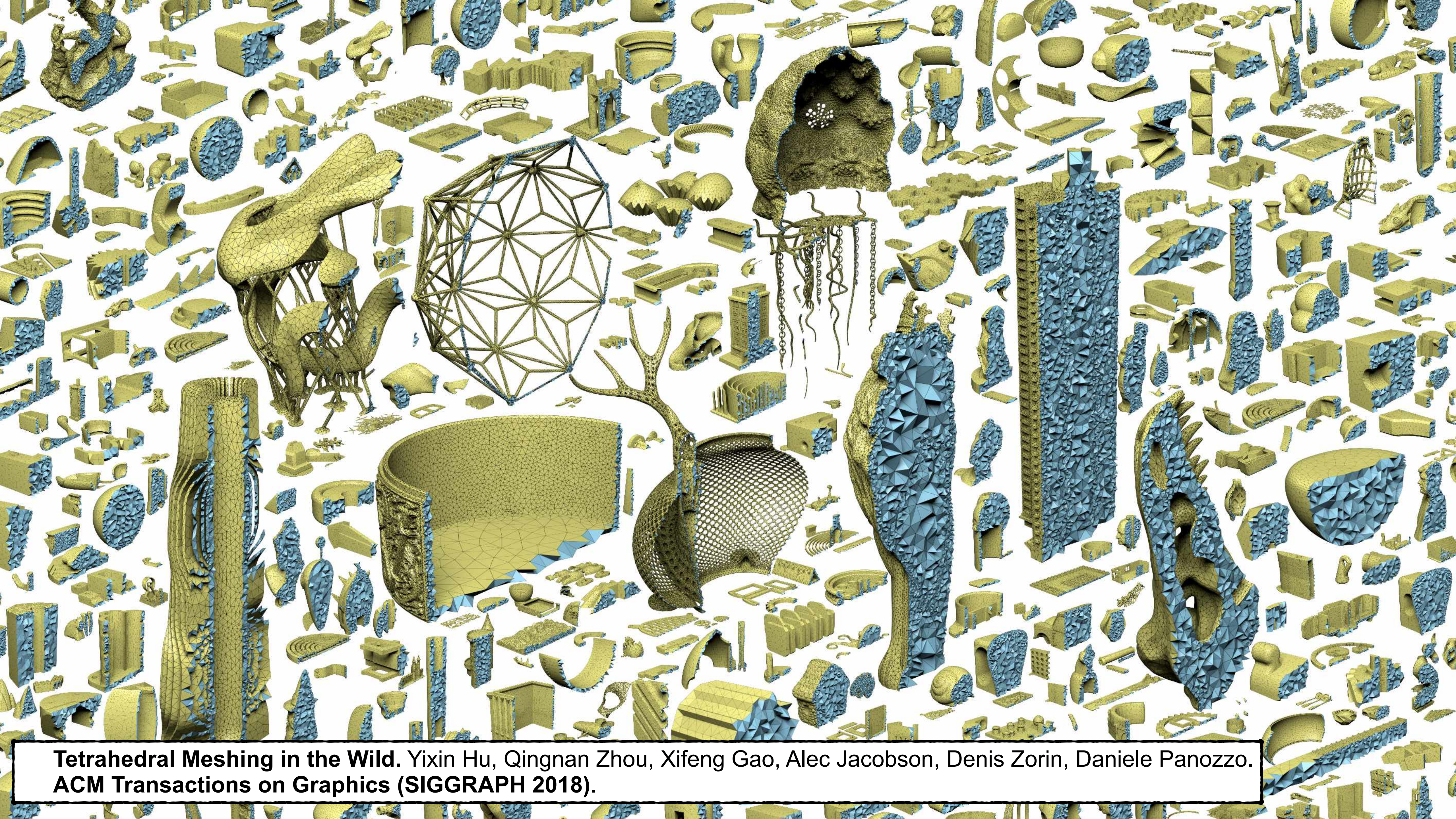
# Validity First



# Validity First







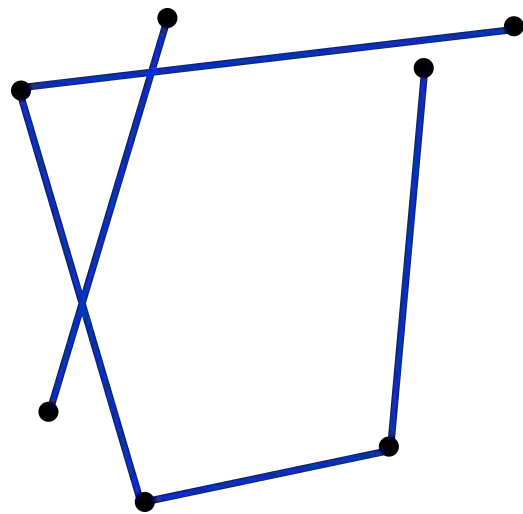
**Tetrahedral Meshing in the Wild.** Yixin Hu, Qingnan Zhou, Xifeng Gao, Alec Jacobson, Denis Zorin, Daniele Panozzo.  
ACM Transactions on Graphics (SIGGRAPH 2018).



# Tetrahedral Meshing in the Wild



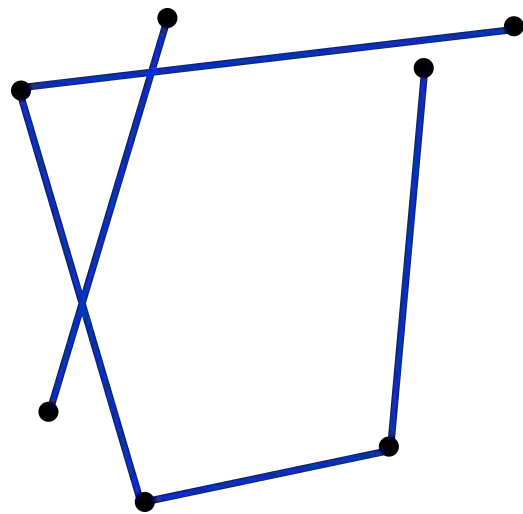
# Tetrahedral Meshing in the Wild



Input



# Tetrahedral Meshing in the Wild



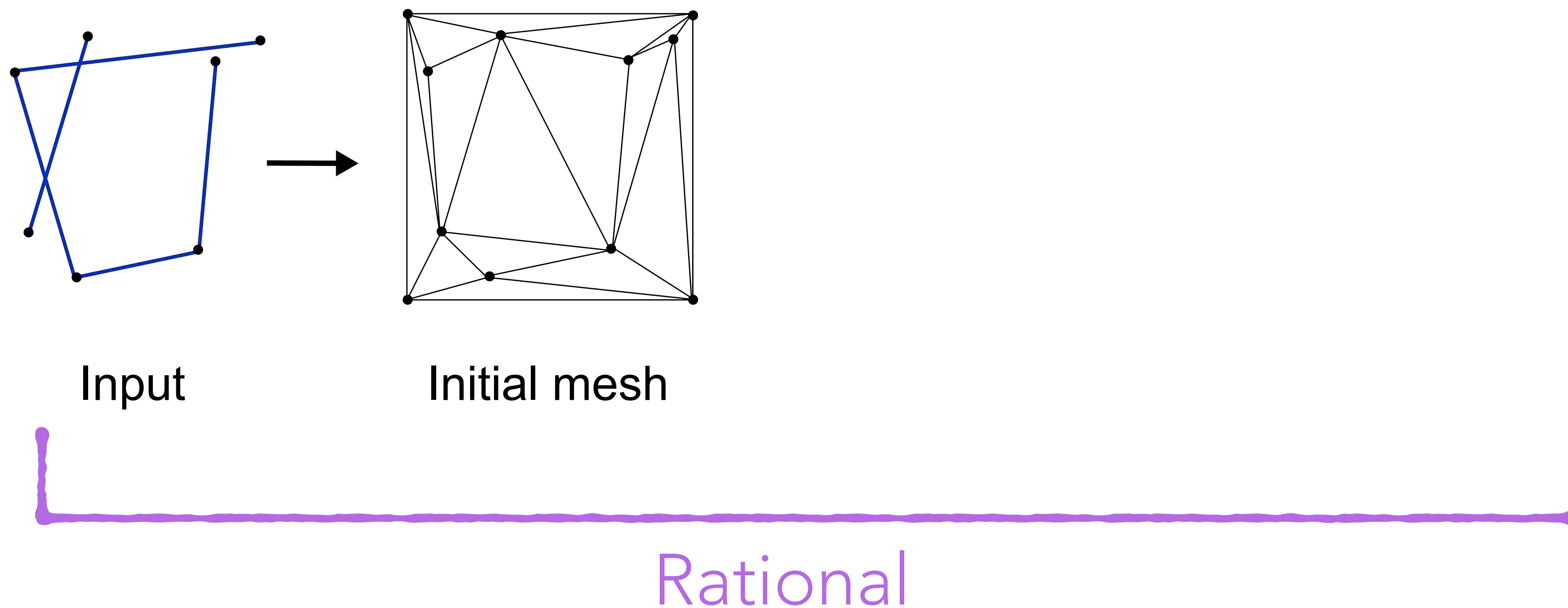
Input



Rational

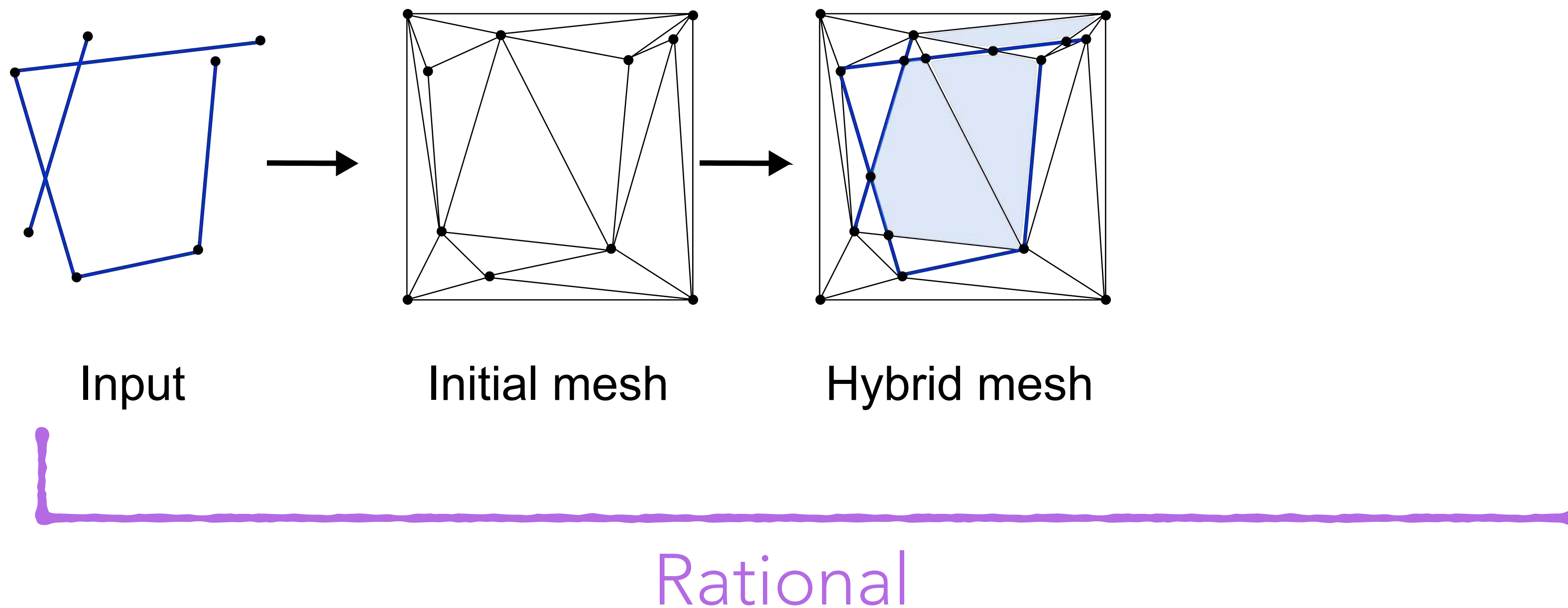


# Tetrahedral Meshing in the Wild



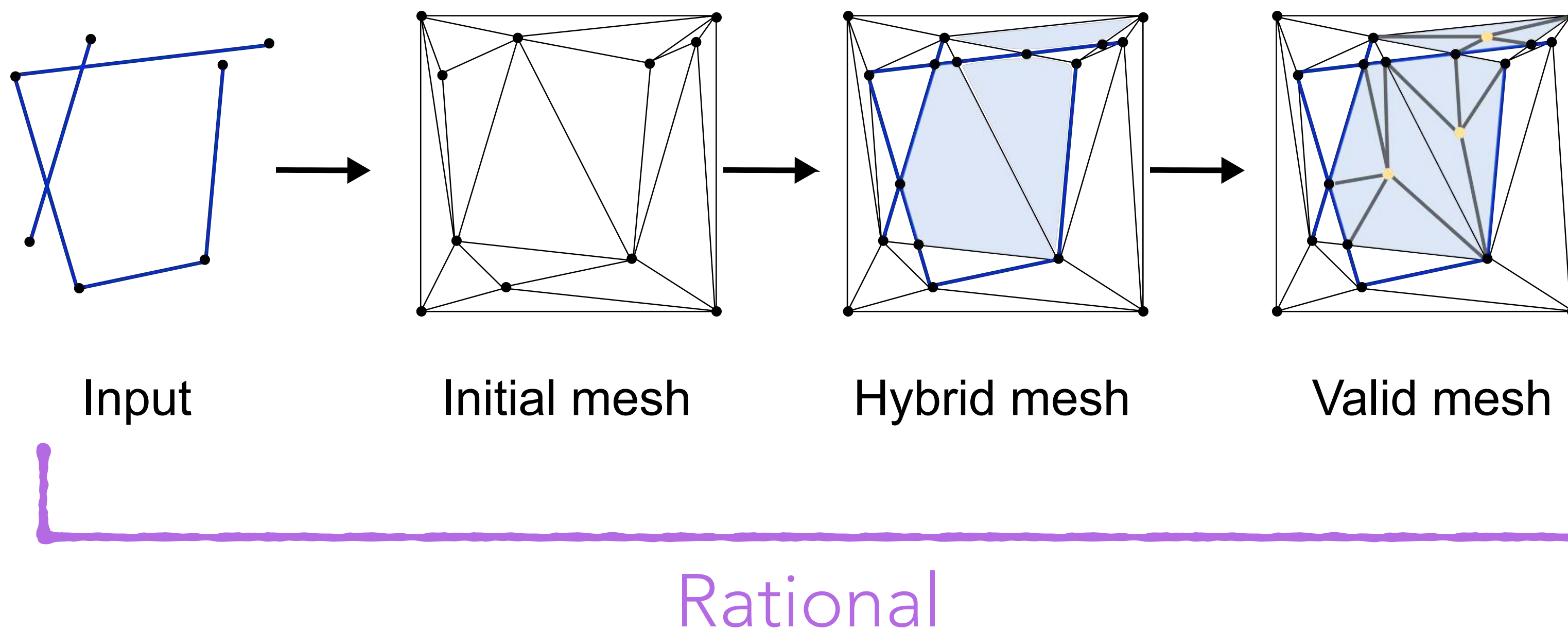


# Tetrahedral Meshing in the Wild



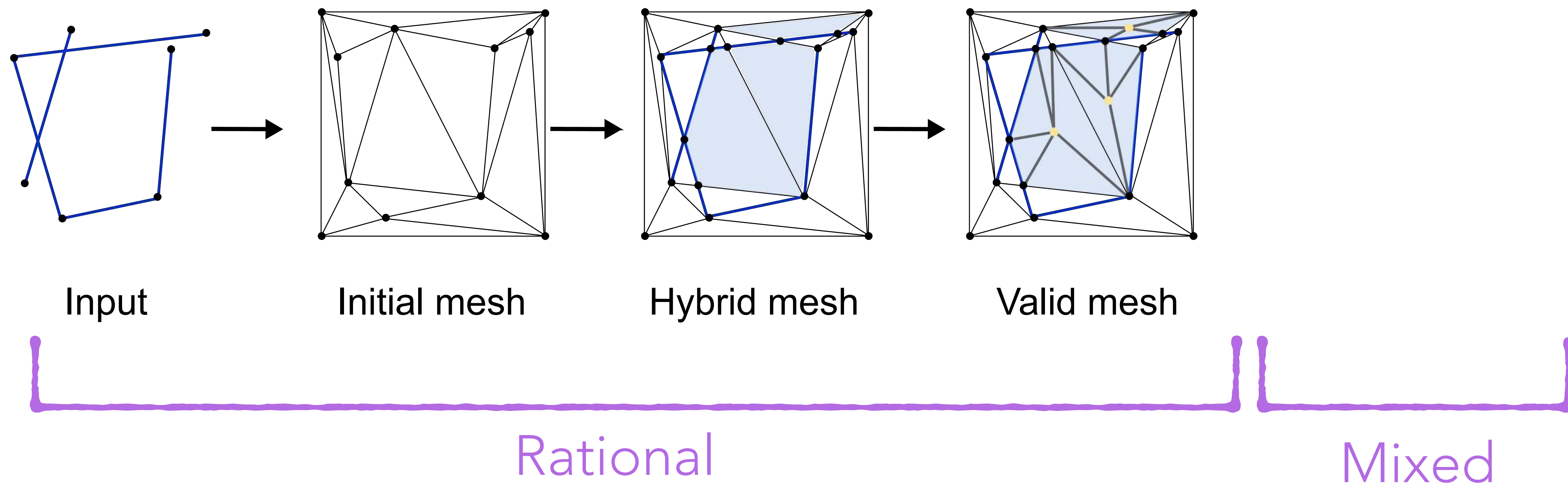


# Tetrahedral Meshing in the Wild

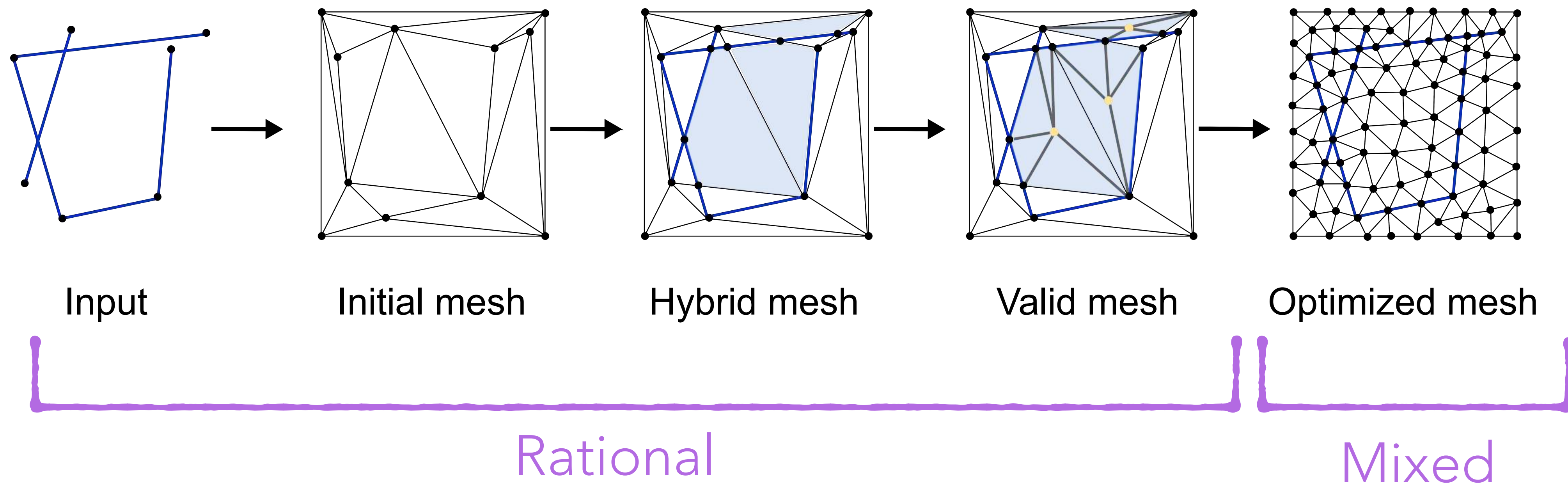




# Tetrahedral Meshing in the Wild

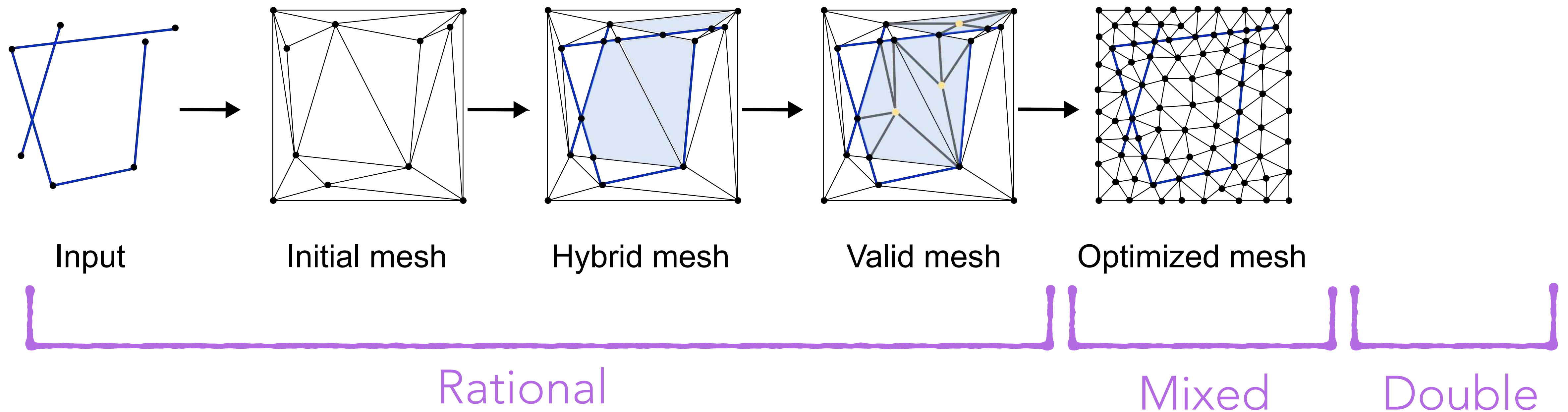


# Tetrahedral Meshing in the Wild

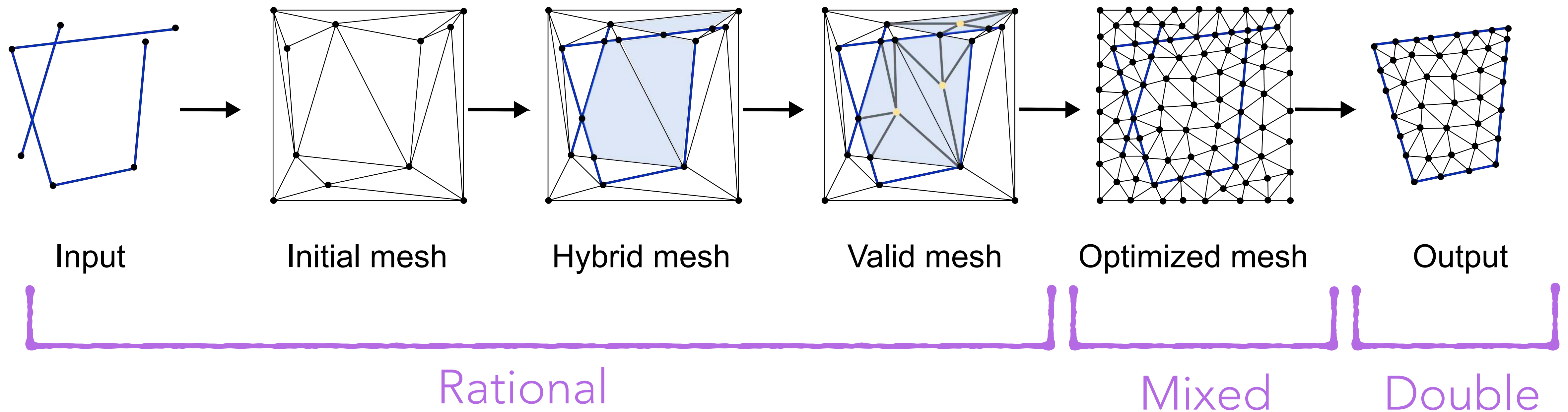




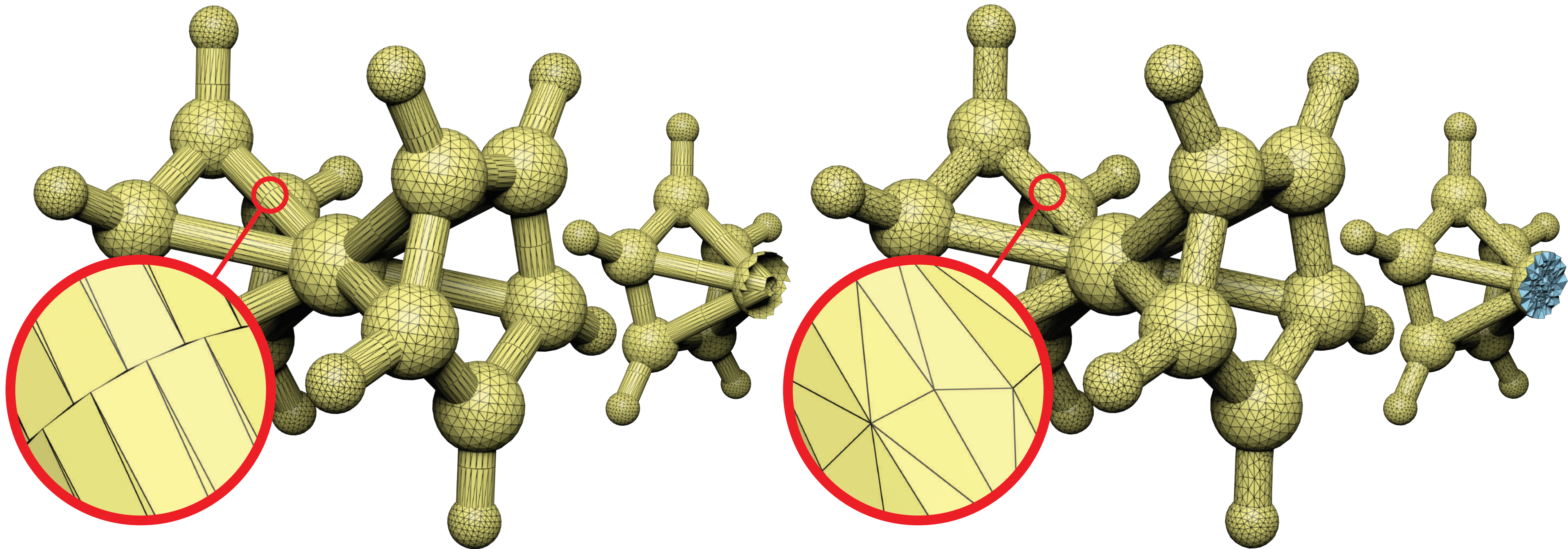
# Tetrahedral Meshing in the Wild



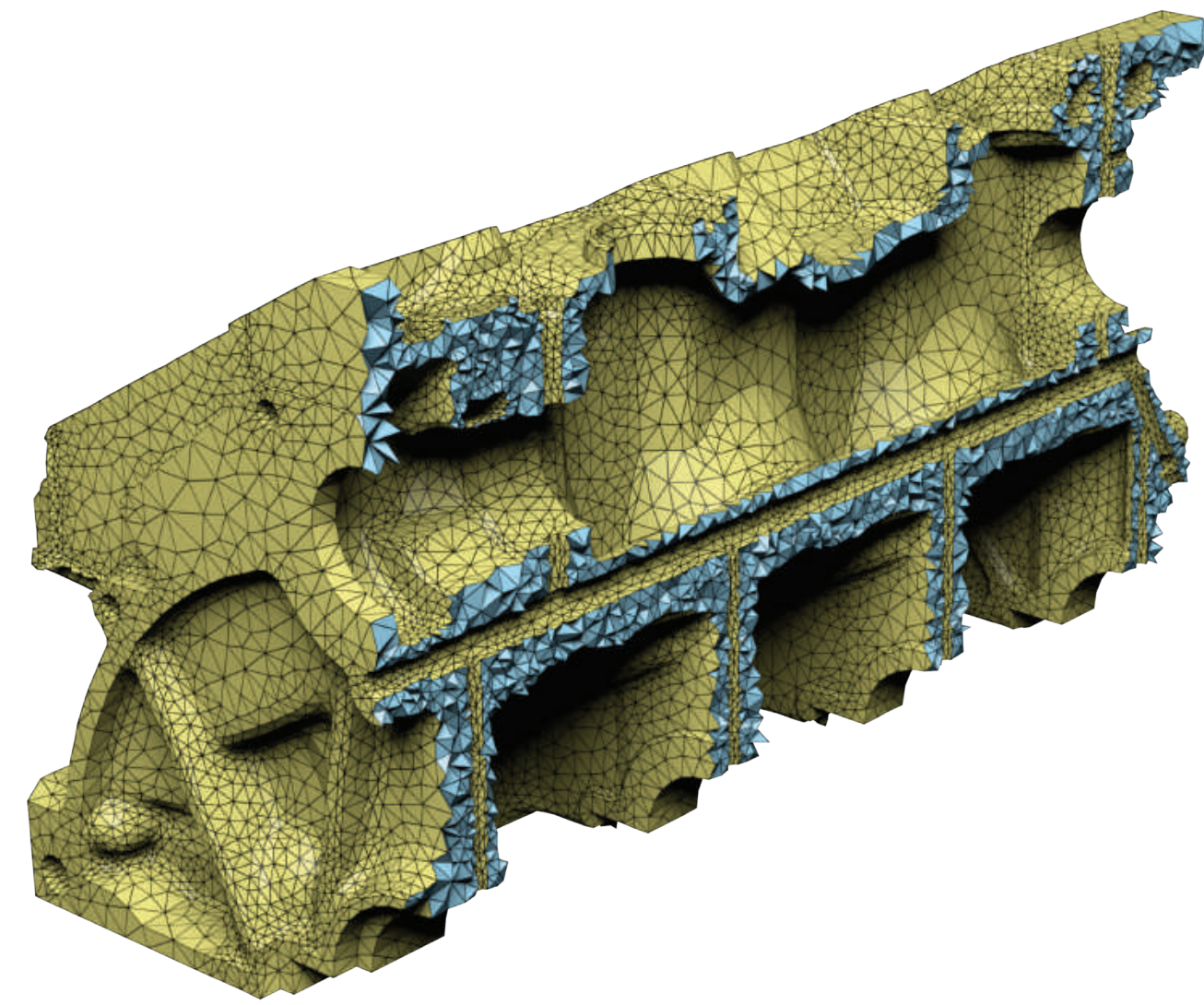
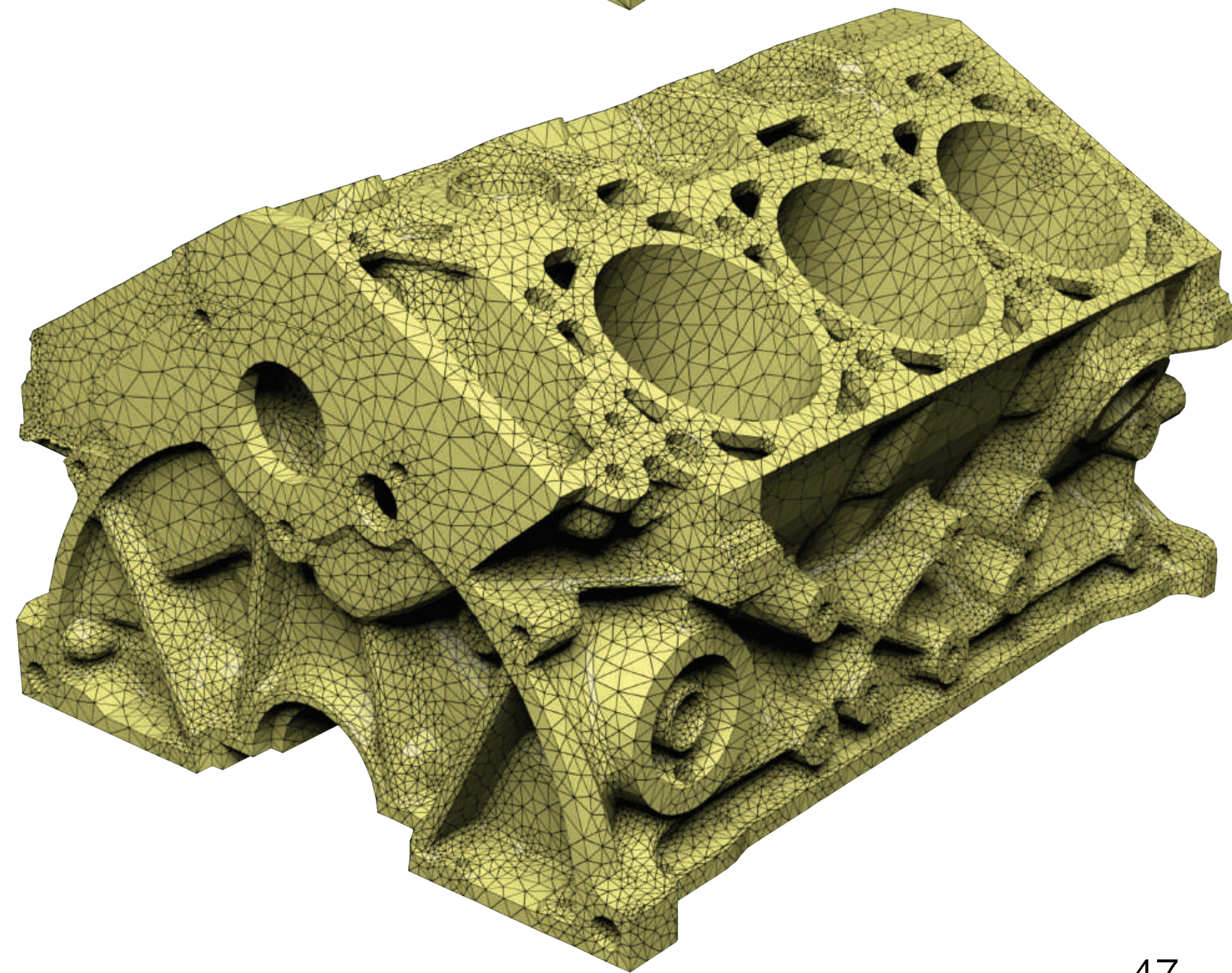
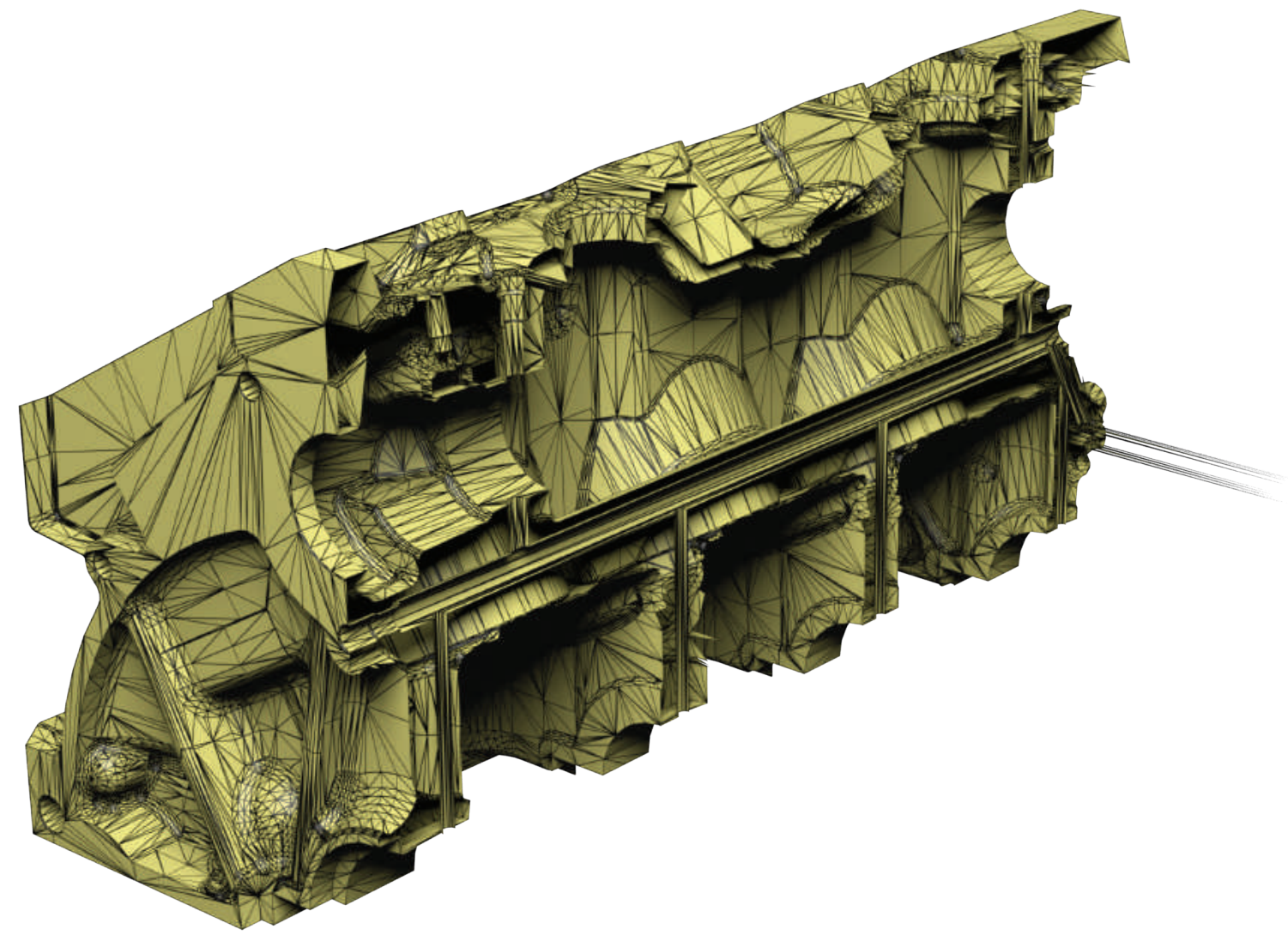
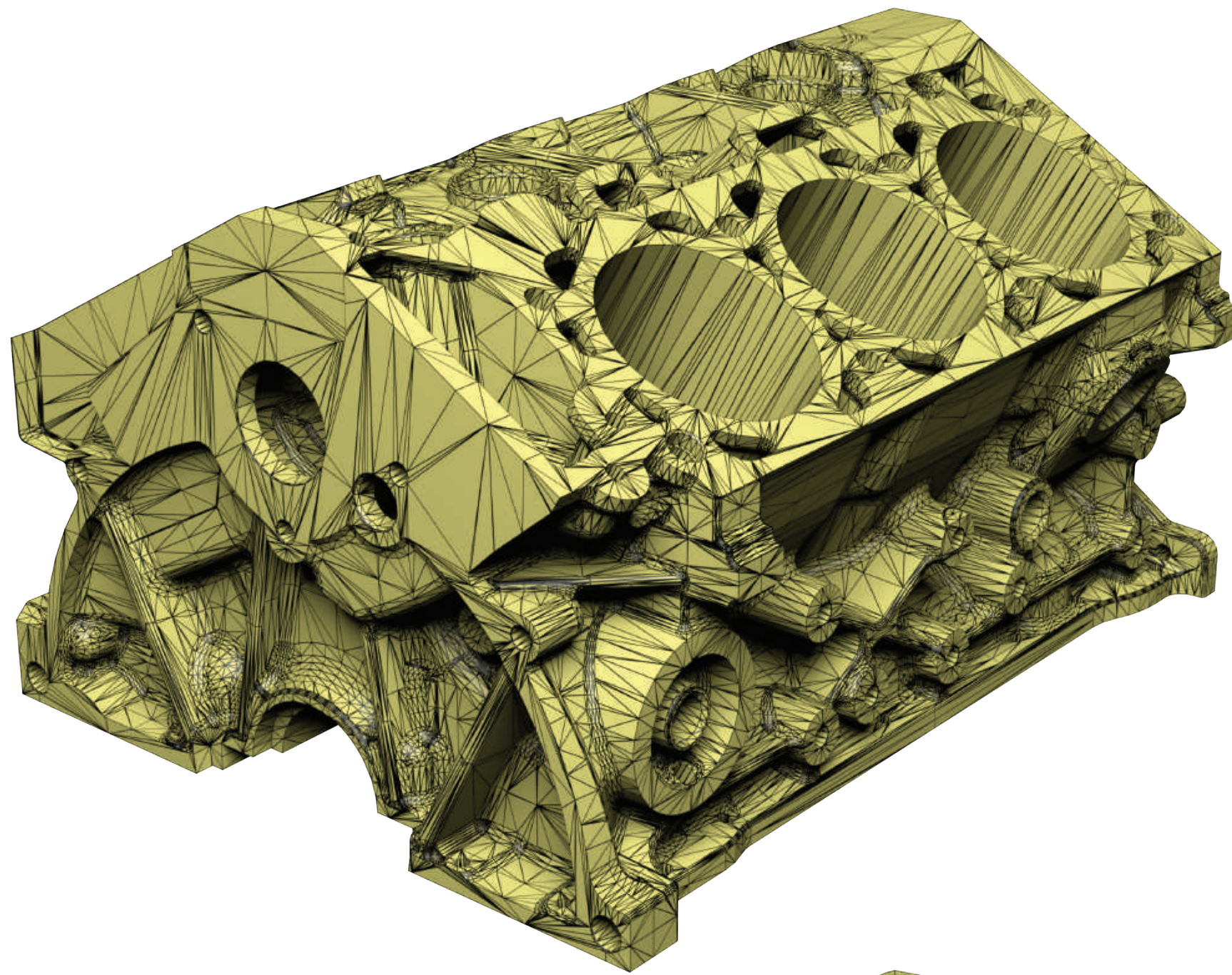
# Tetrahedral Meshing in the Wild



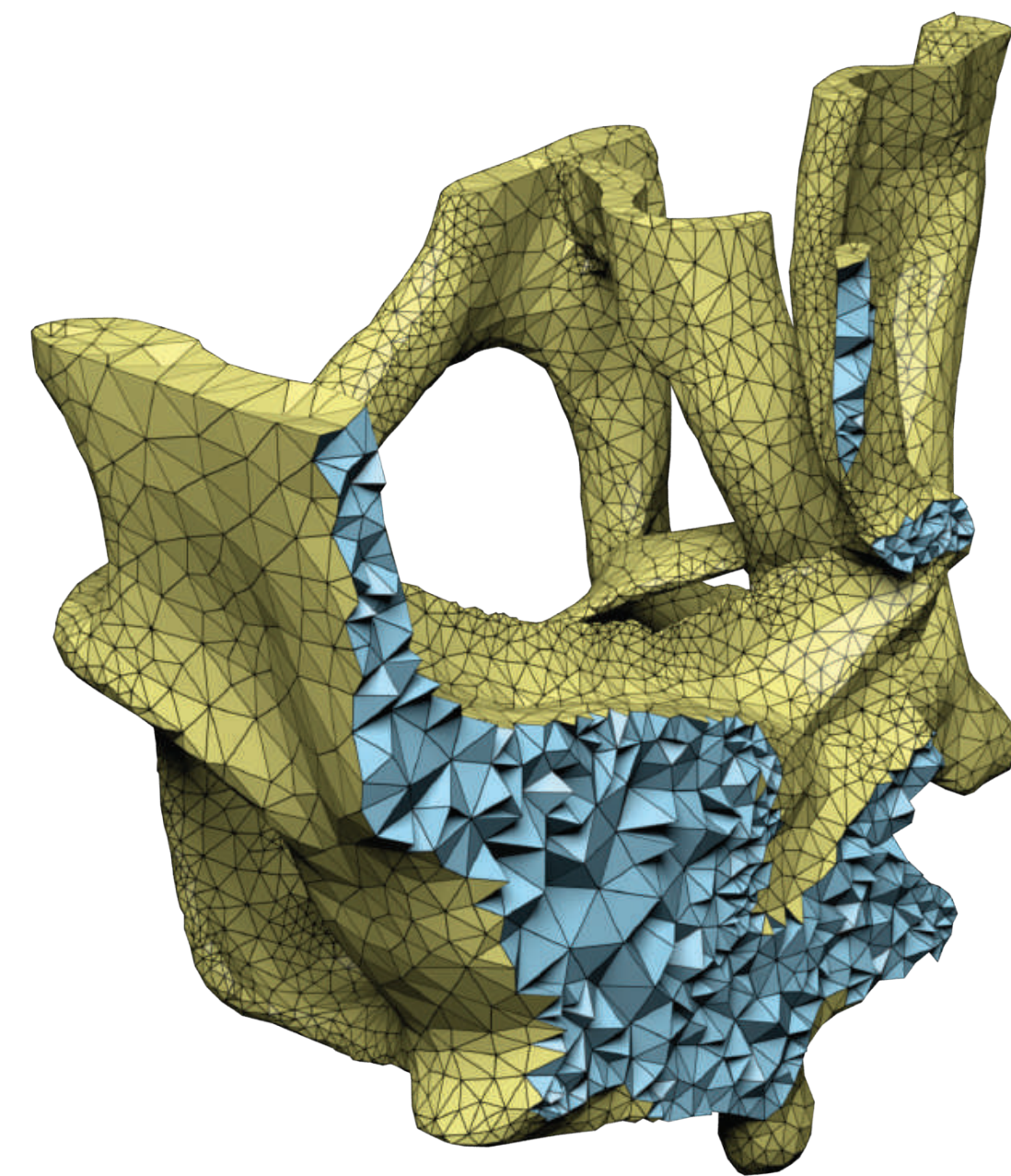
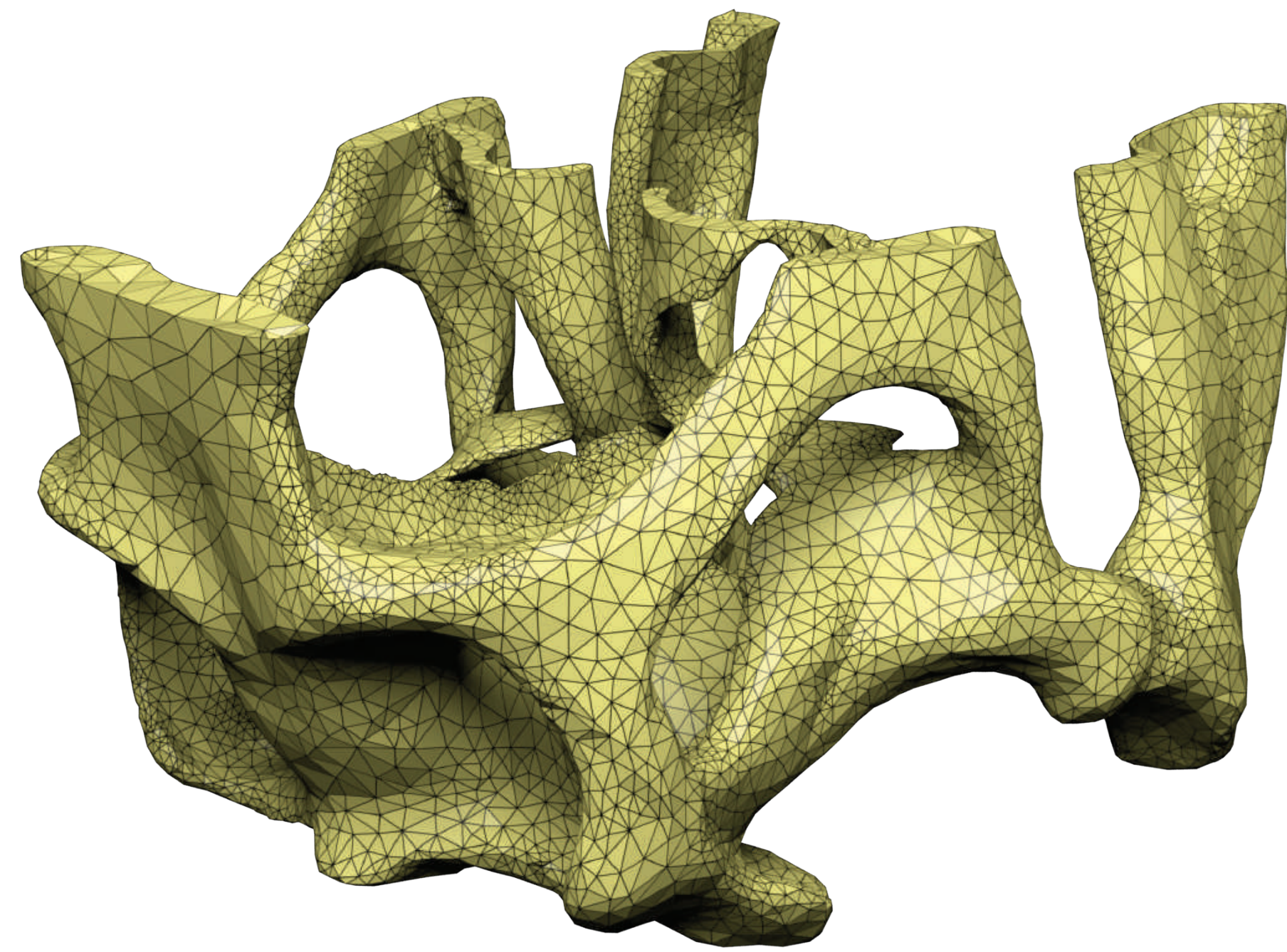
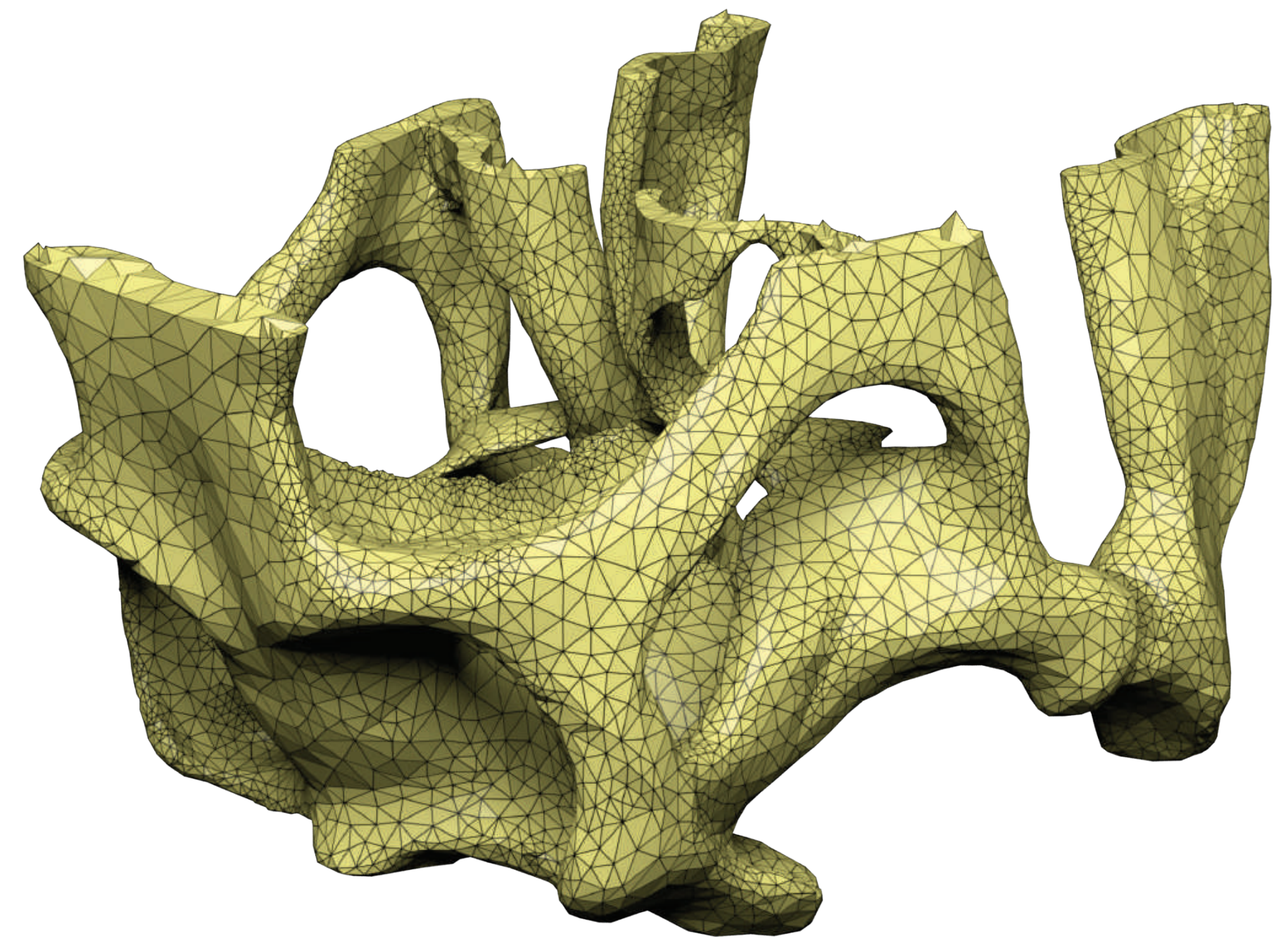
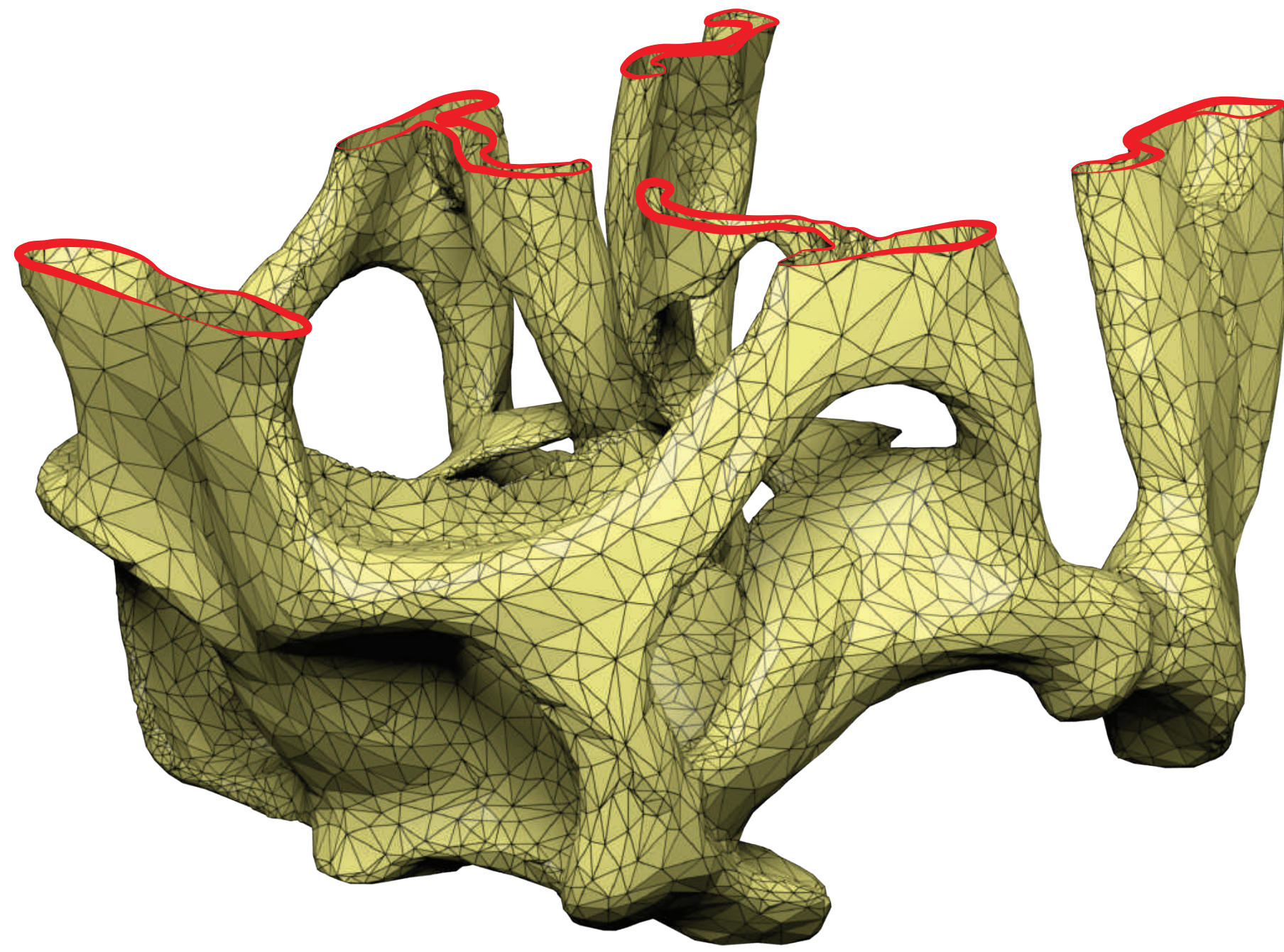




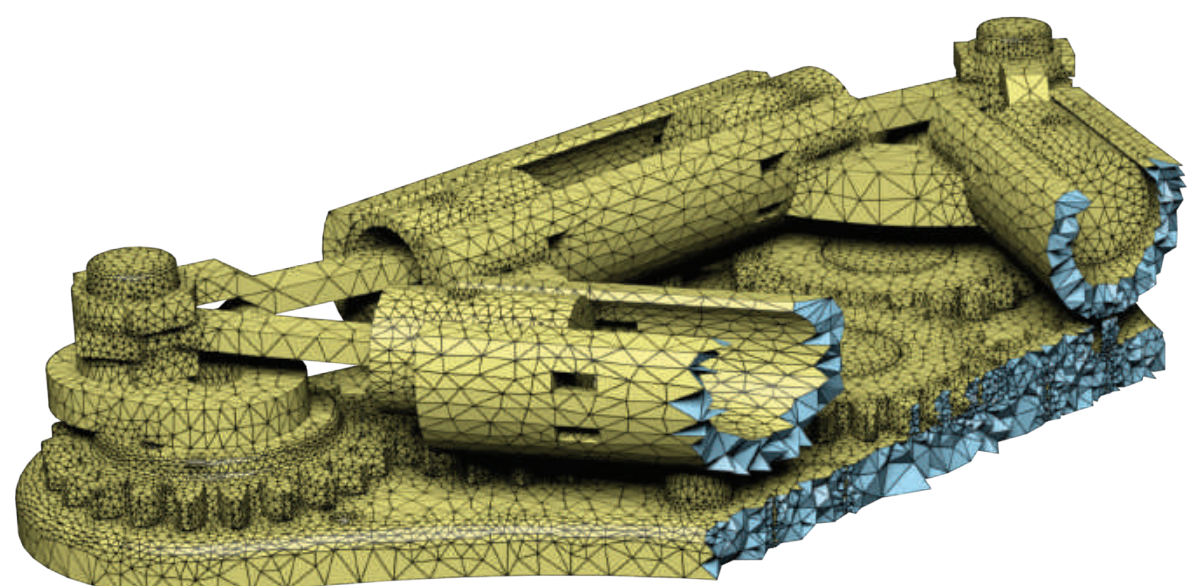
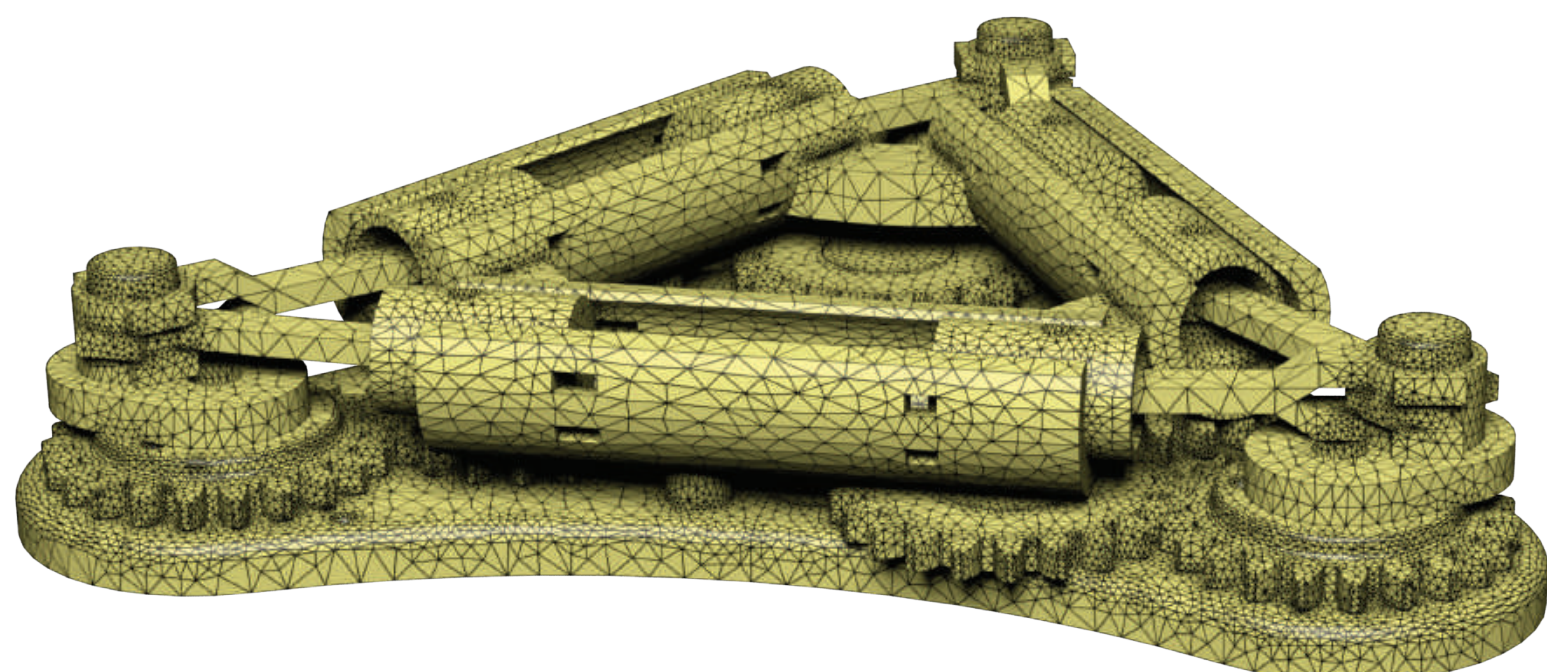
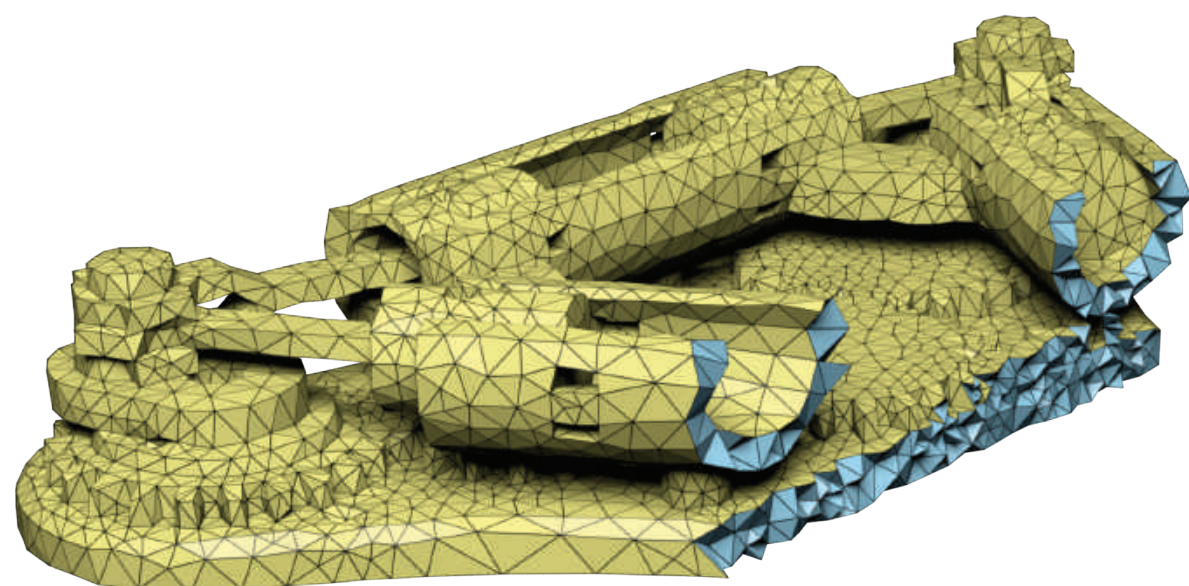
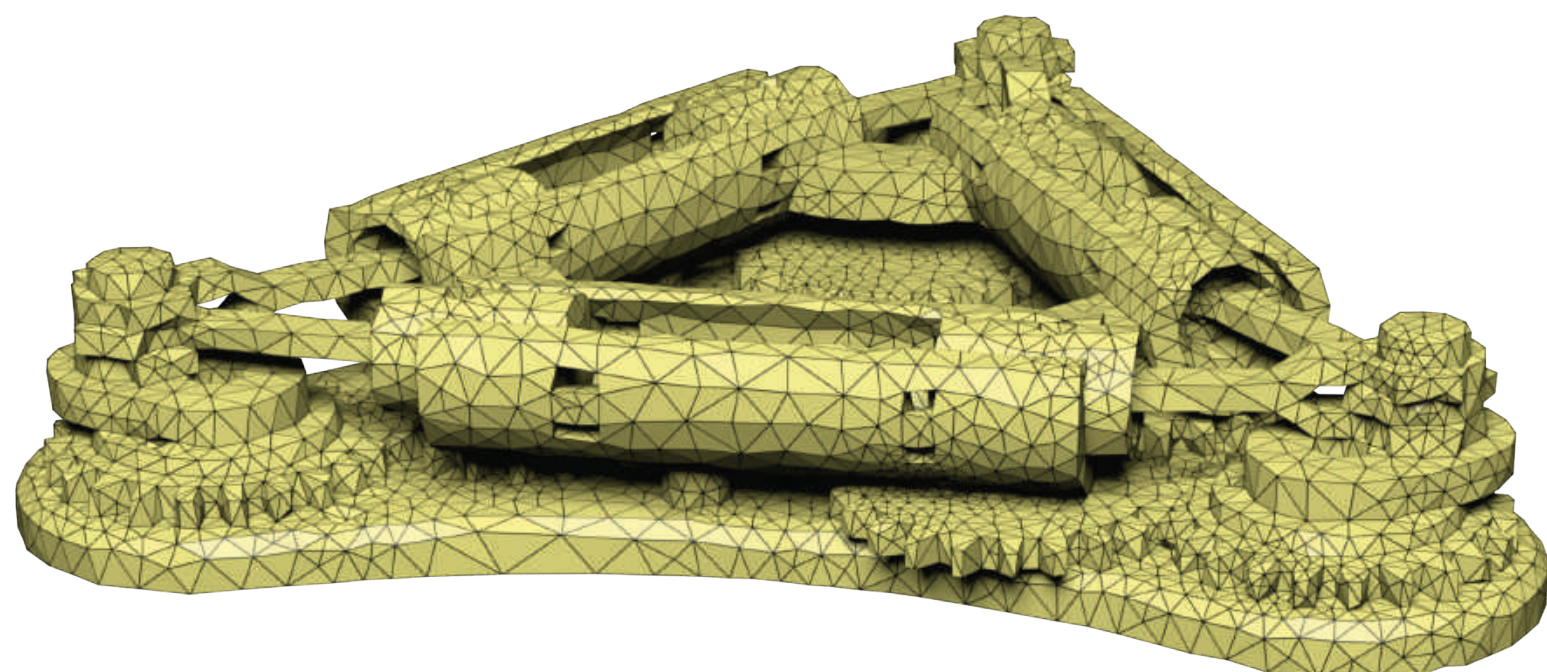
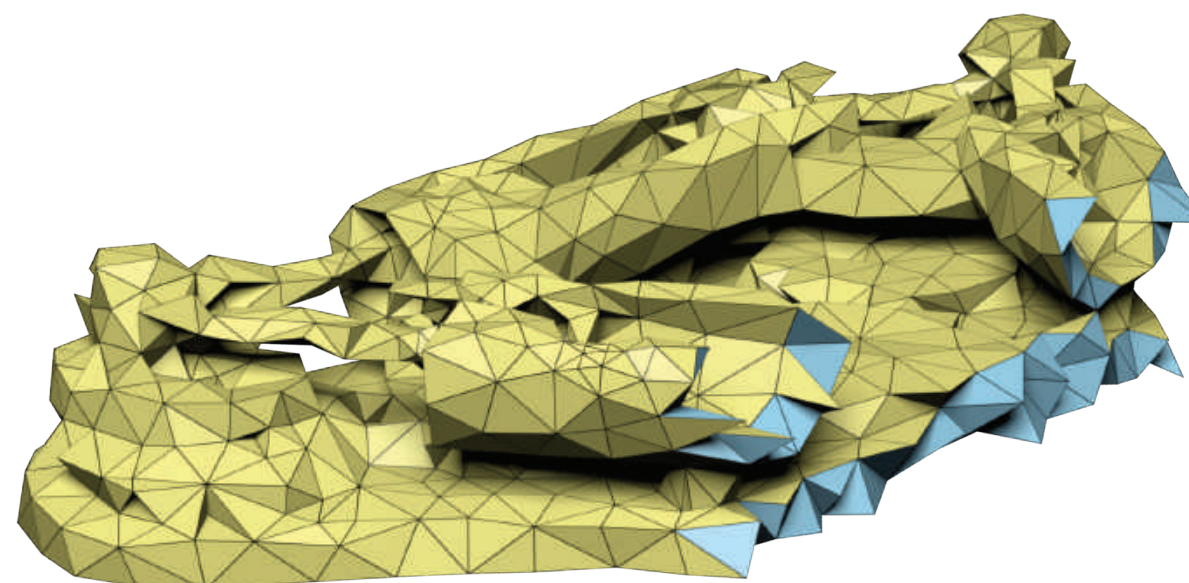
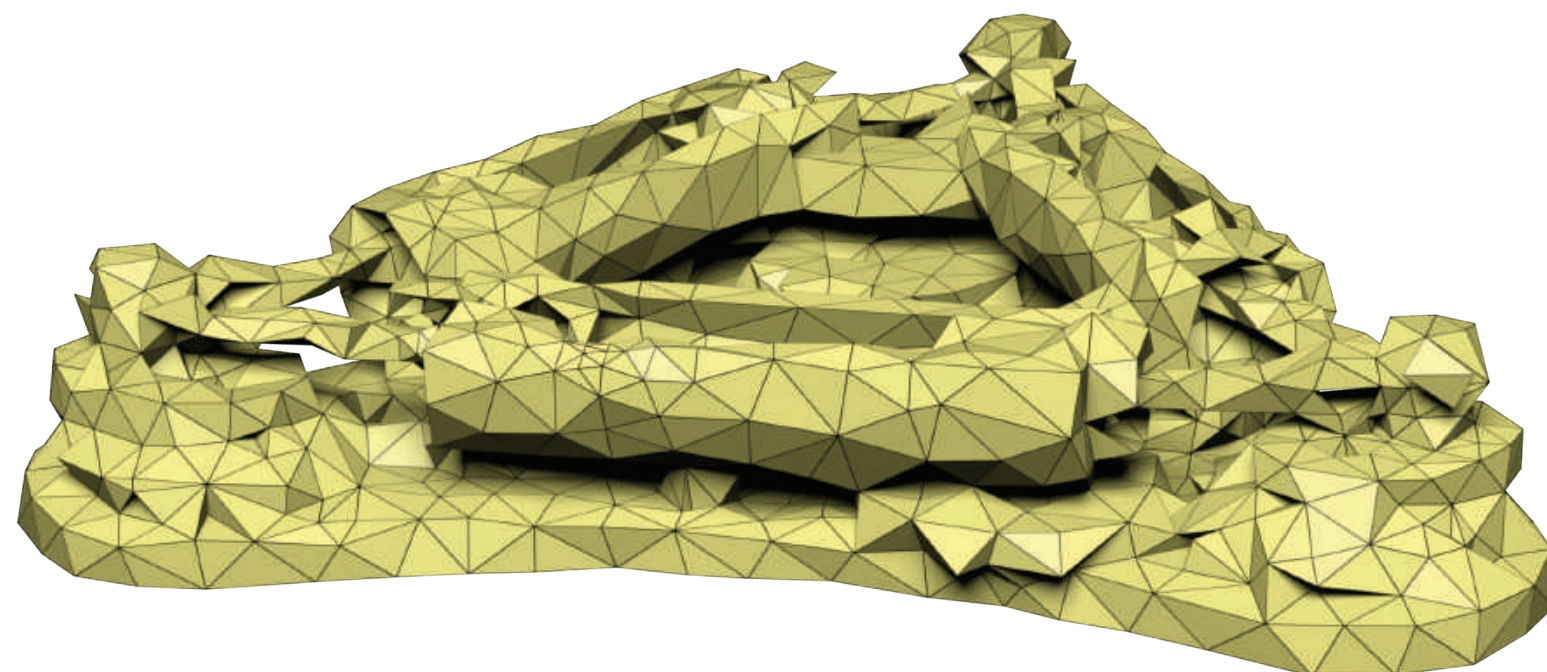
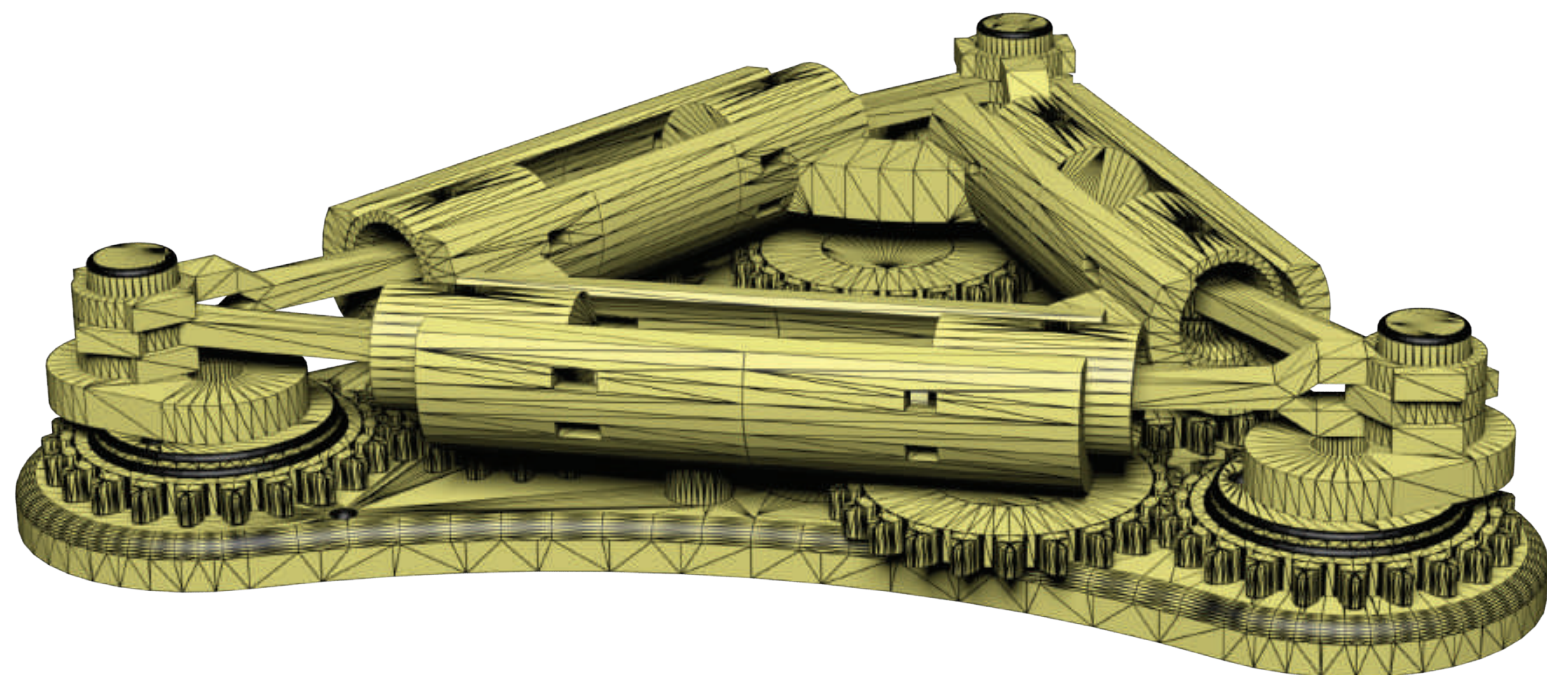




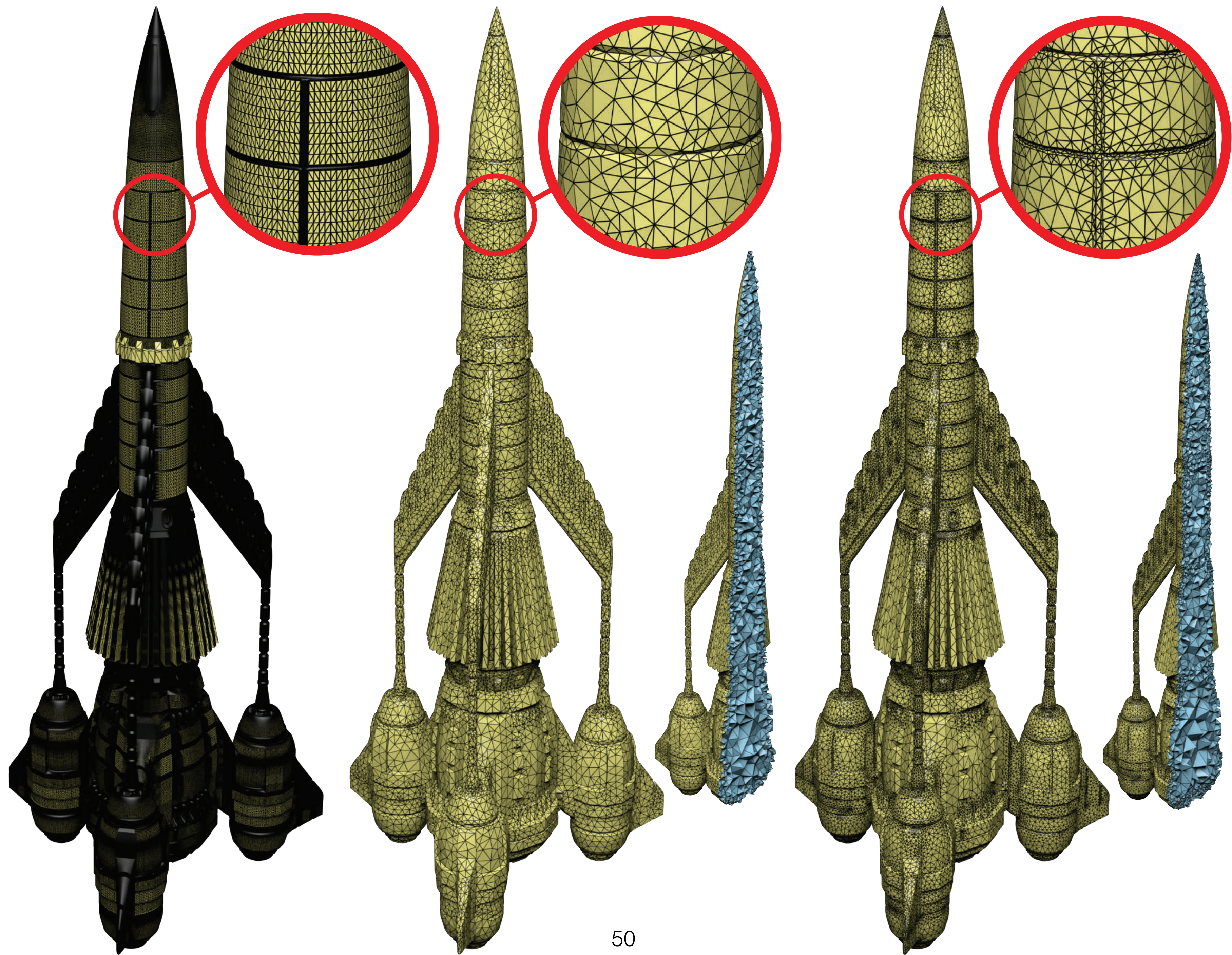




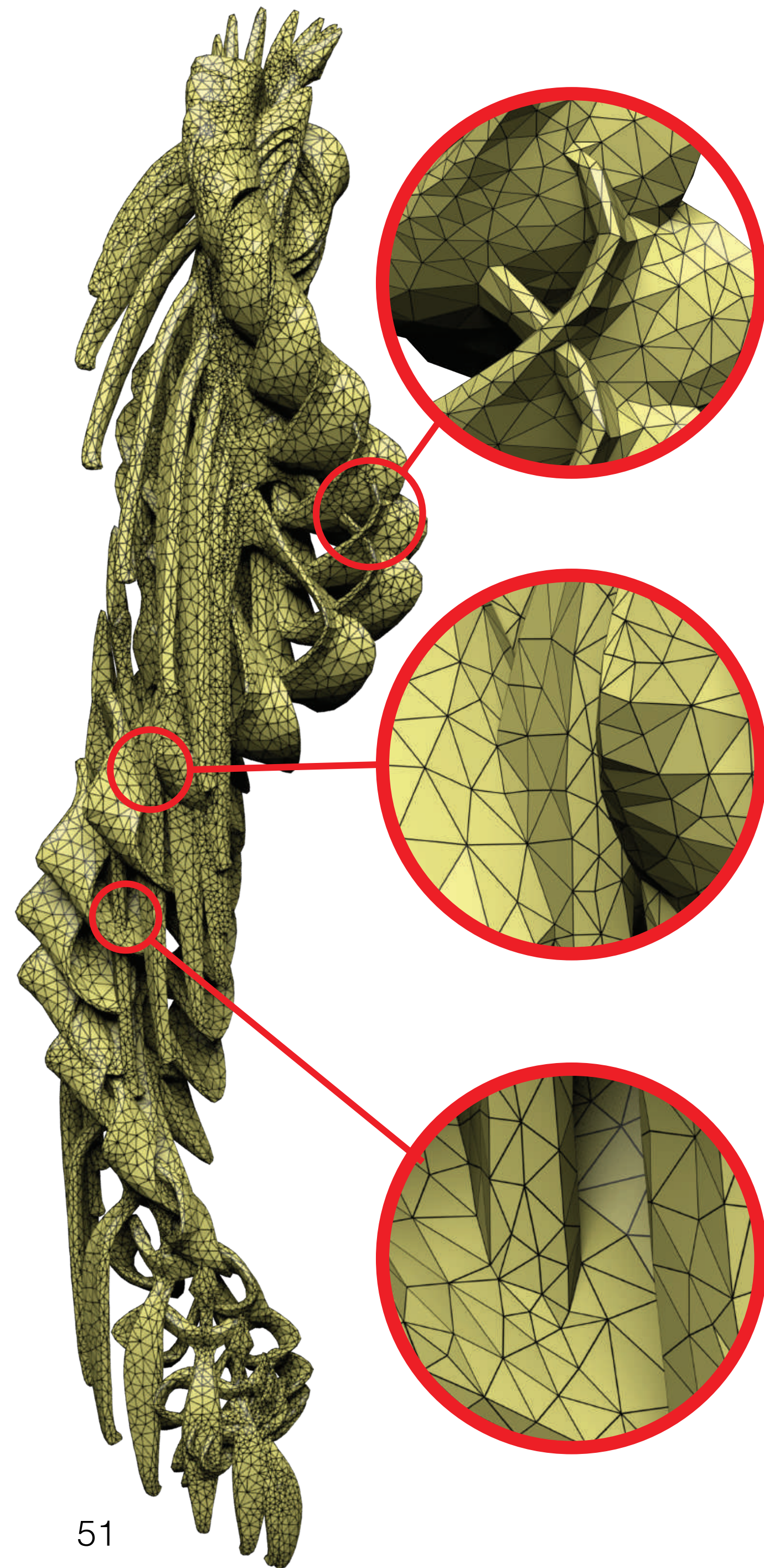
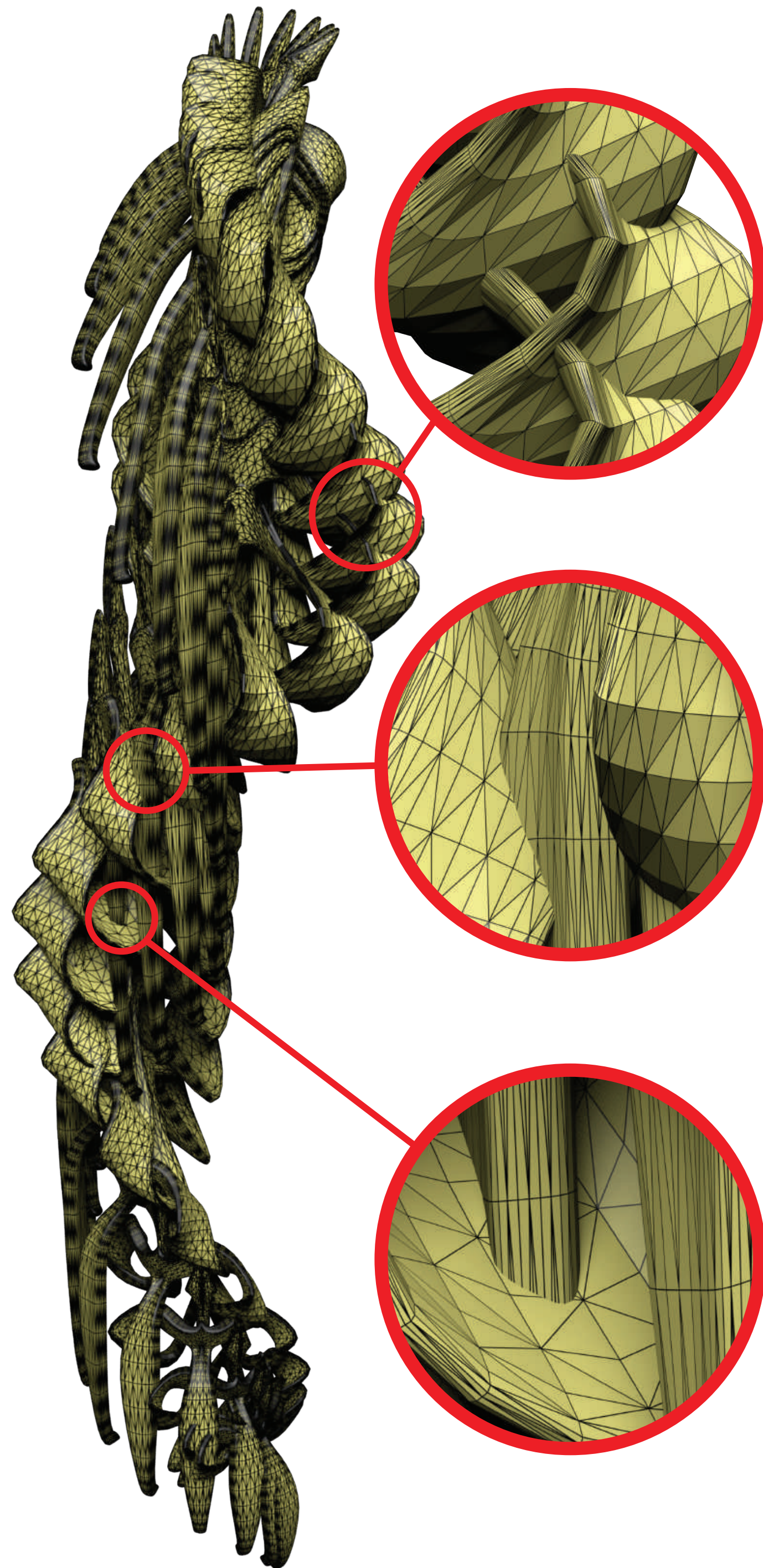




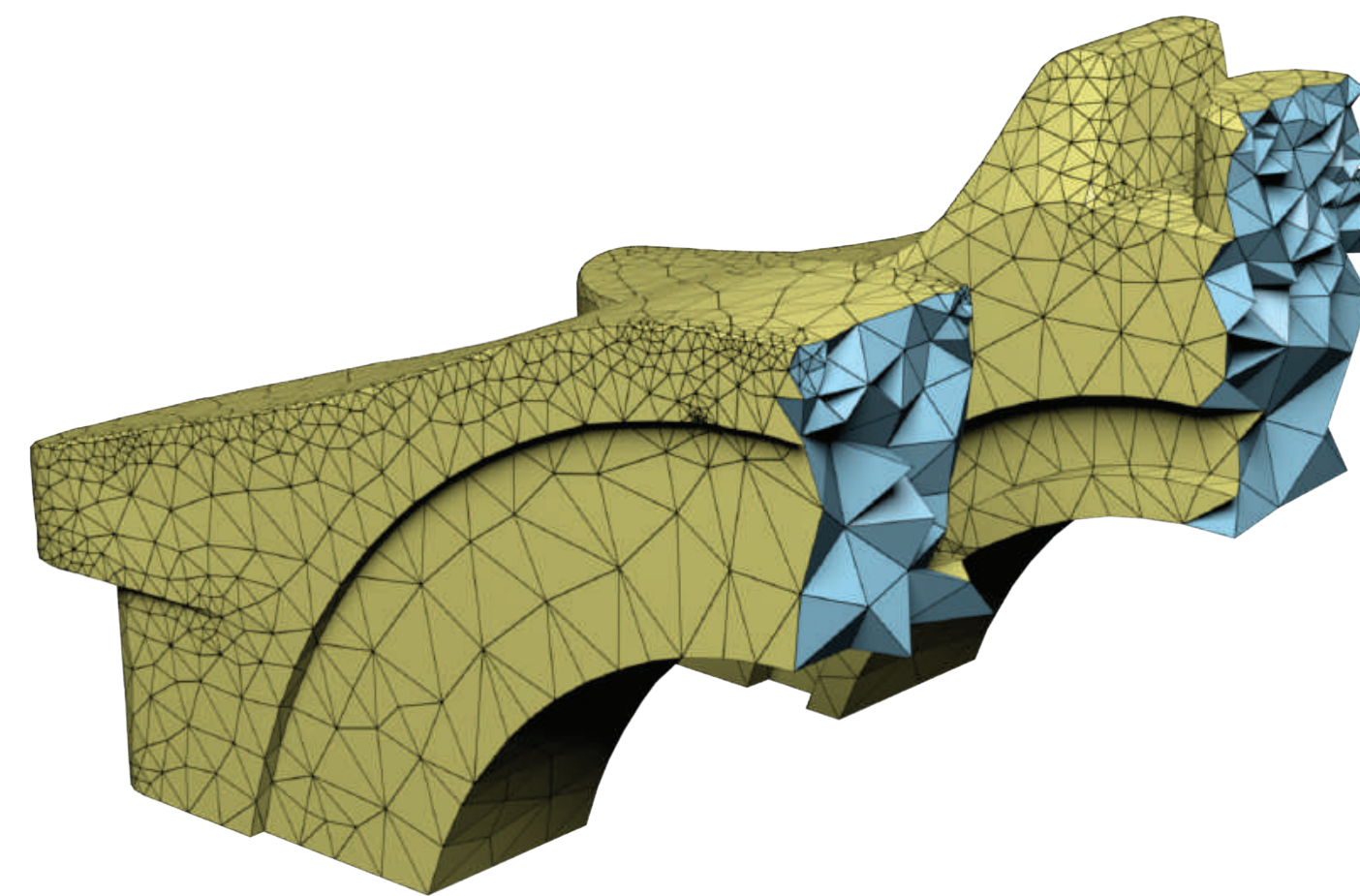
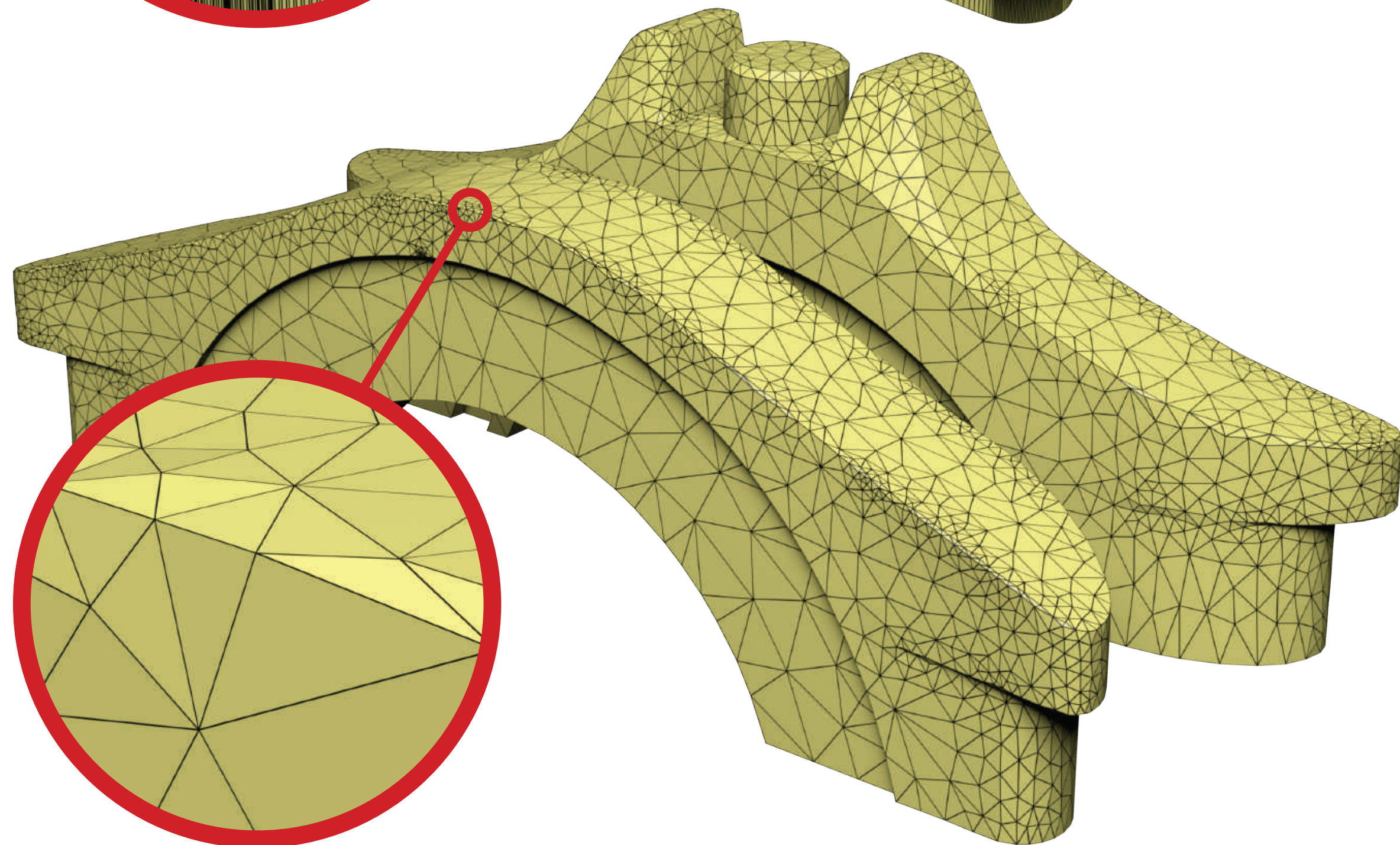
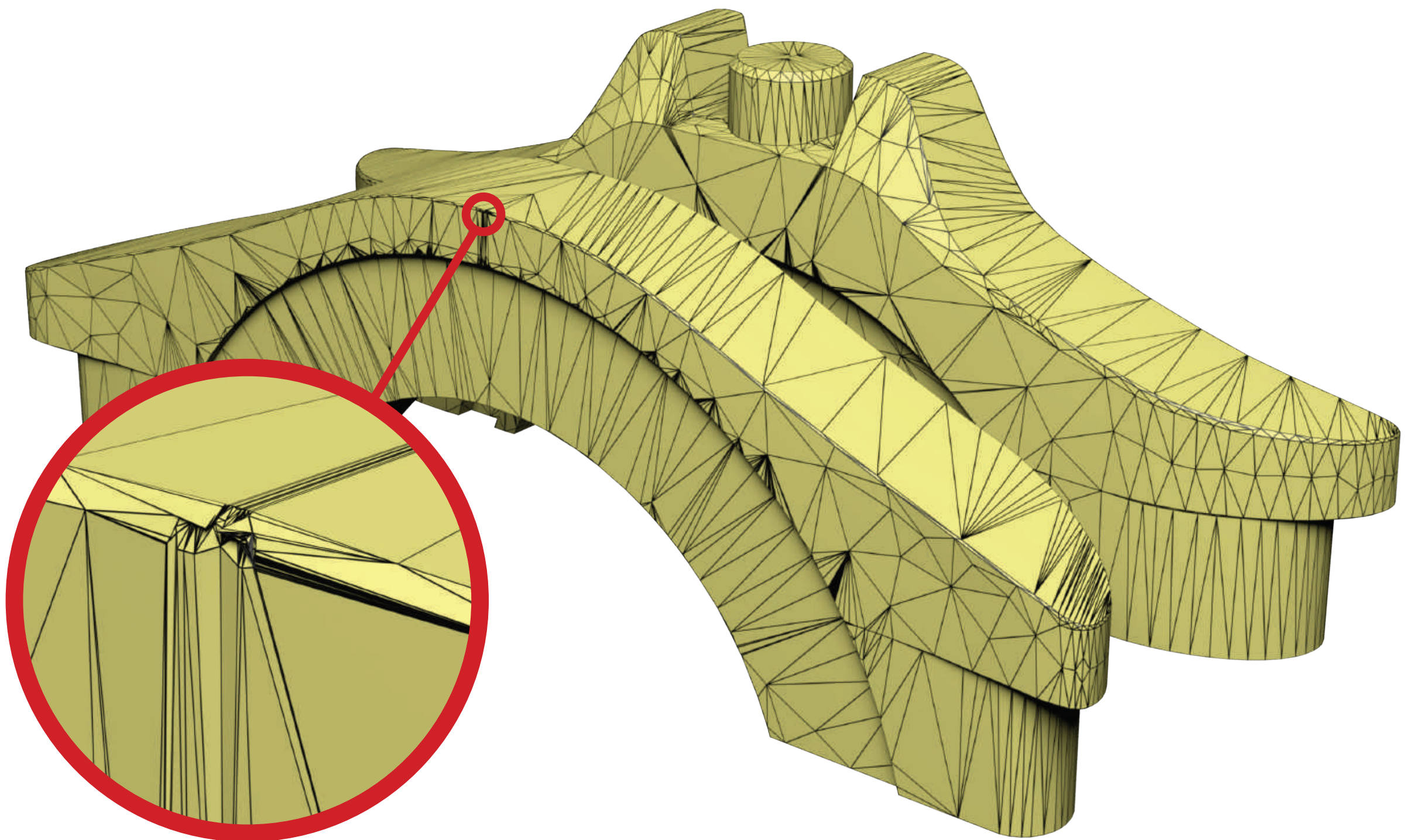
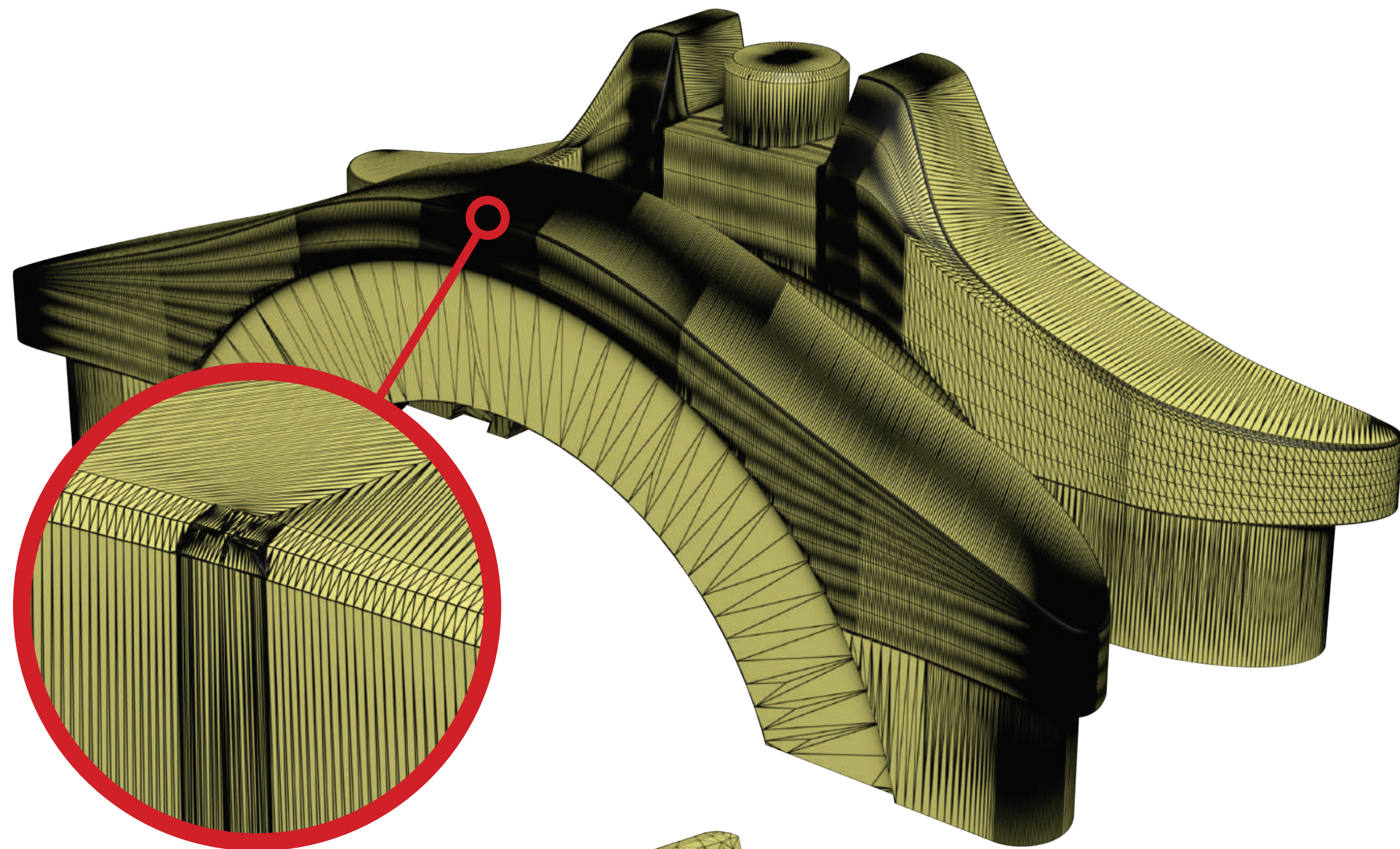




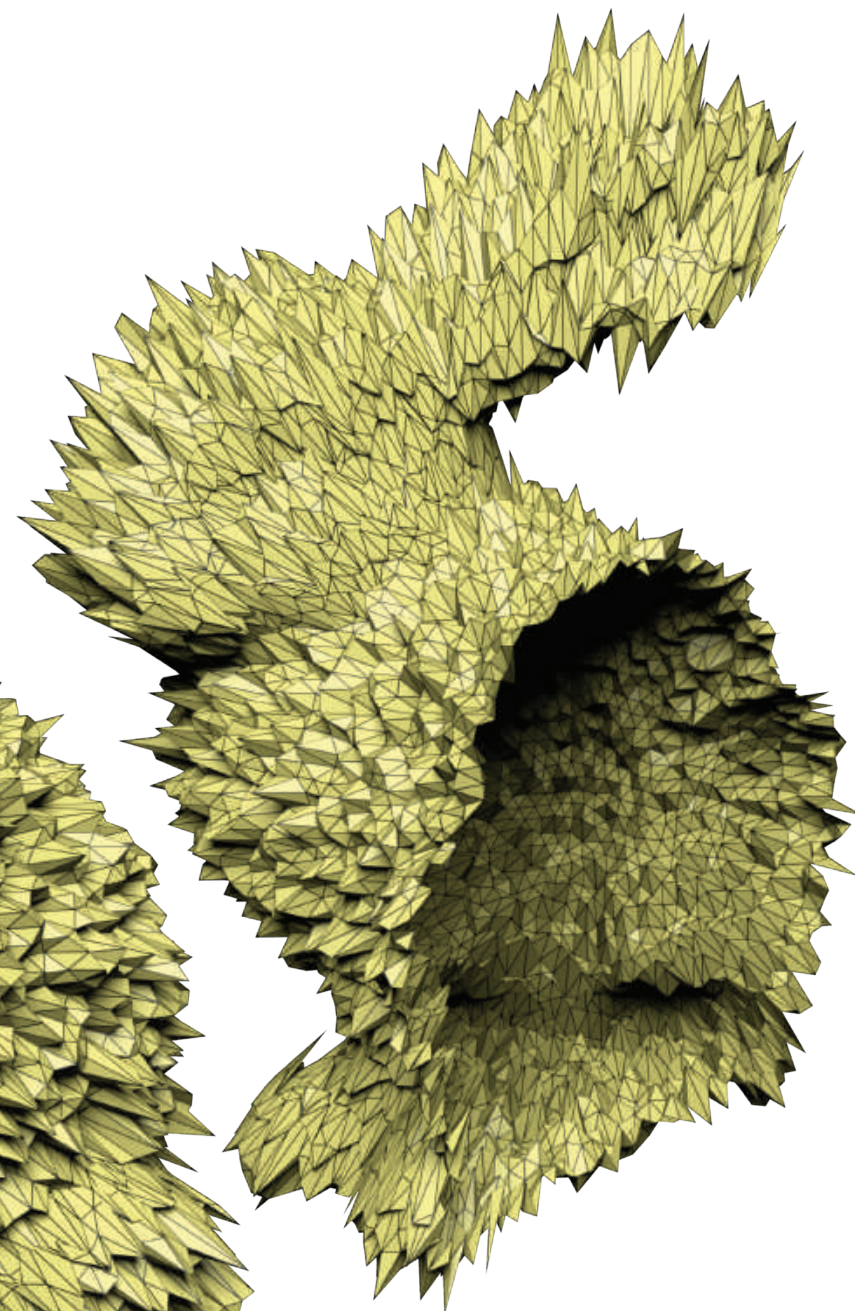
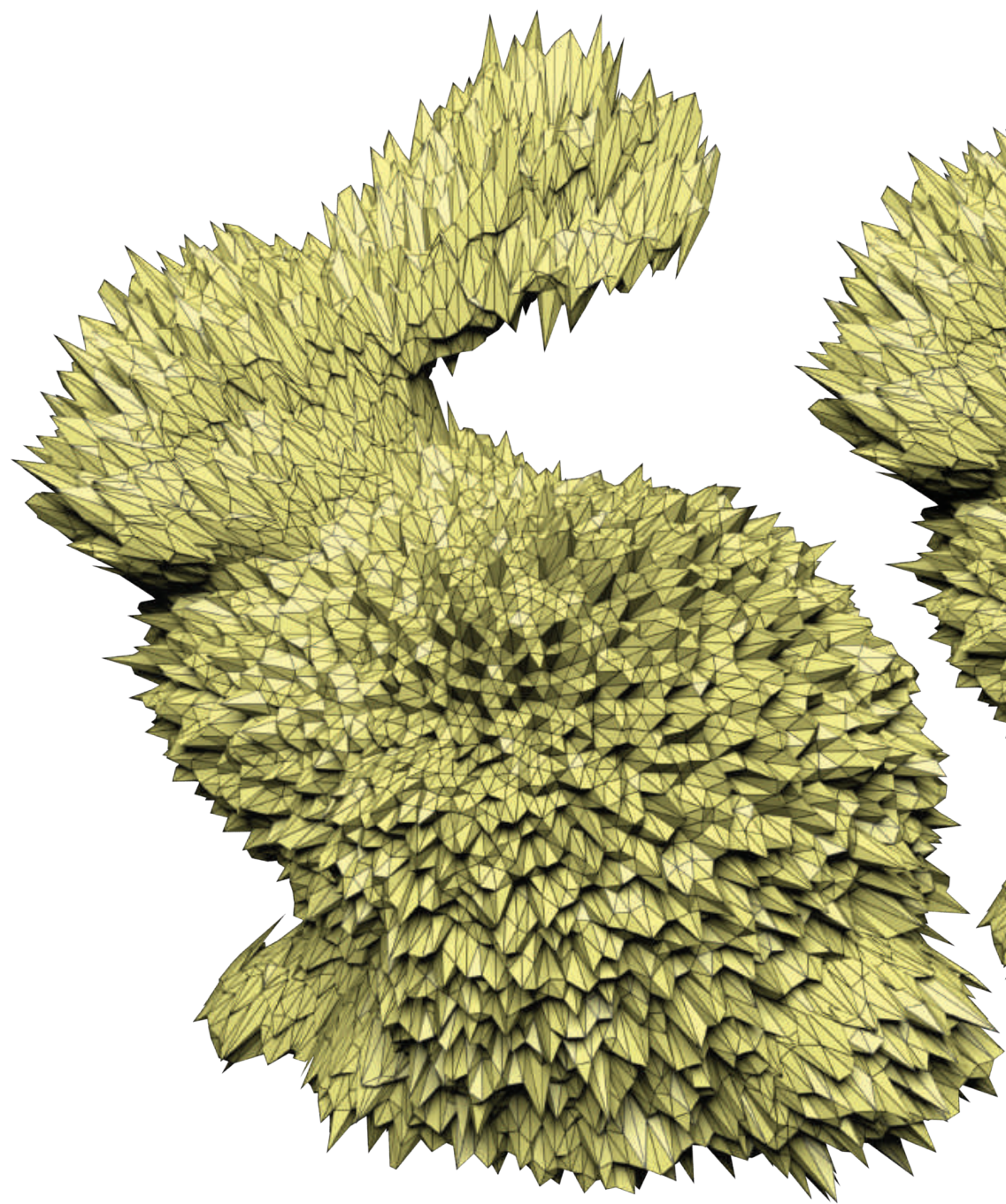




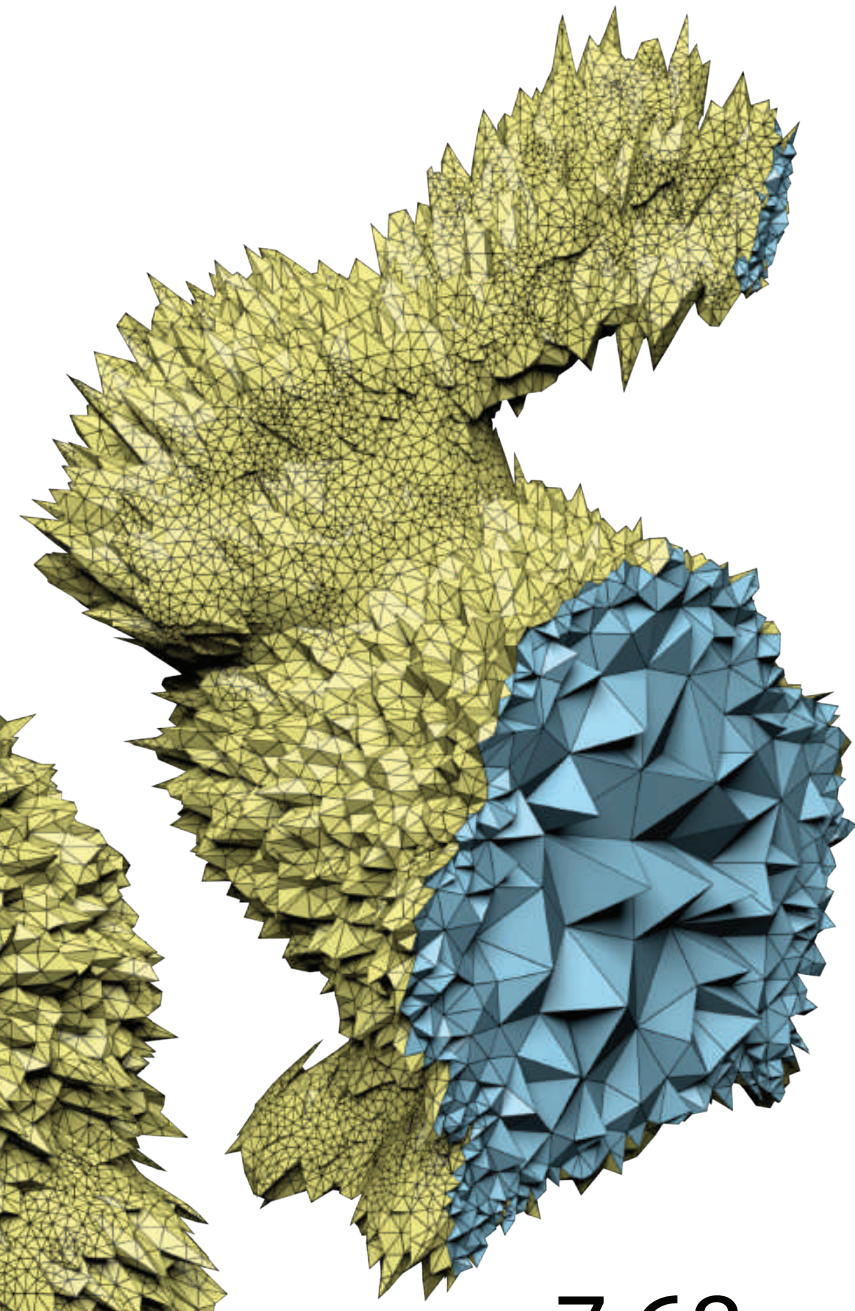
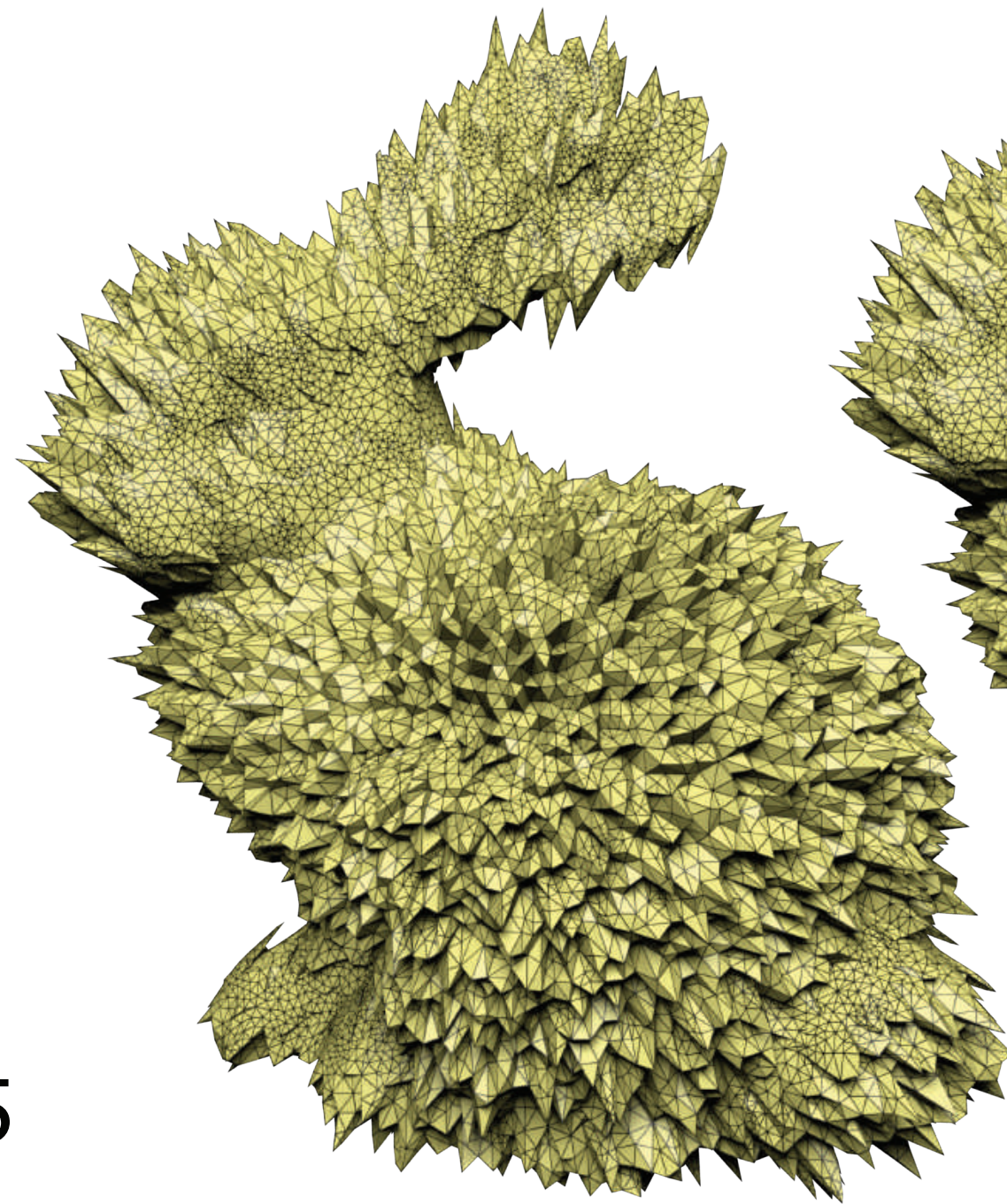




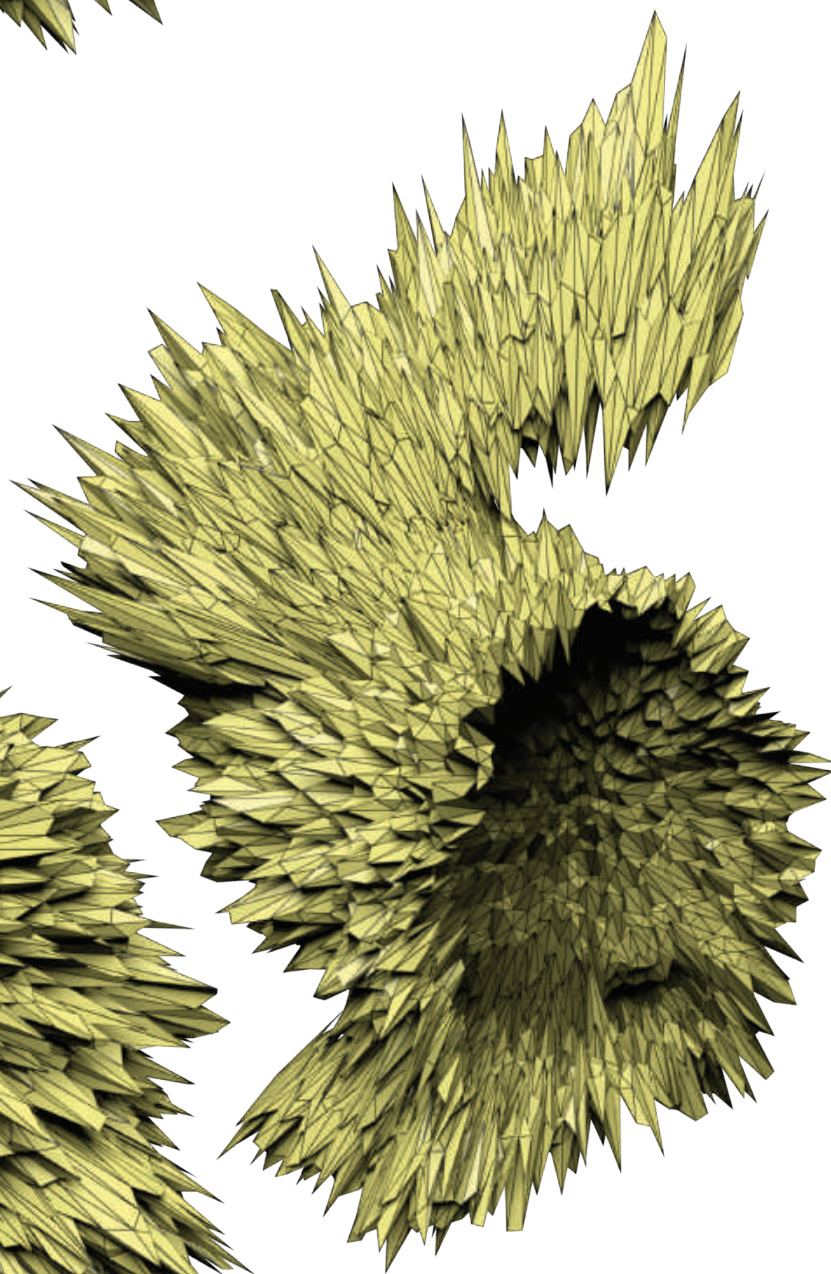
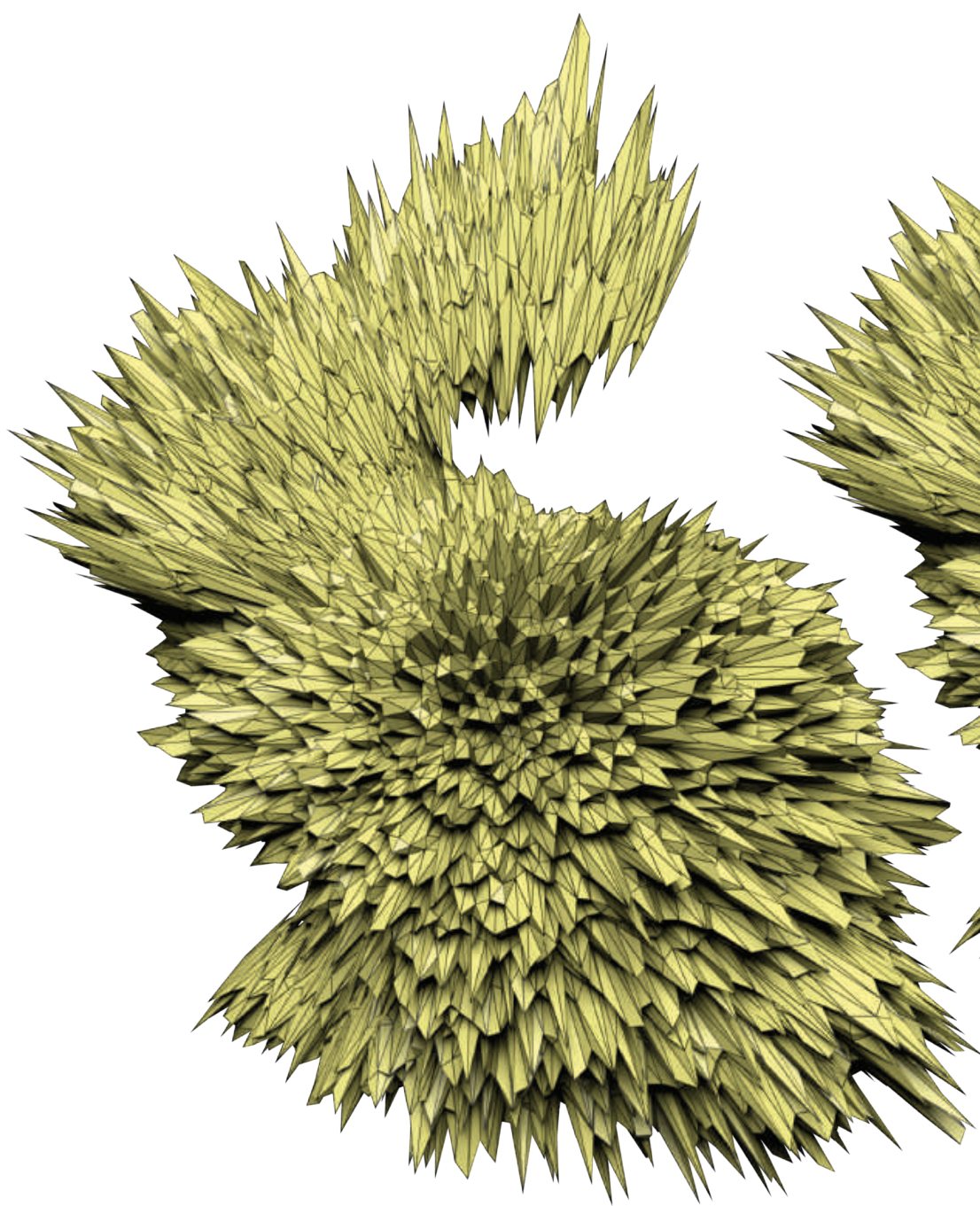




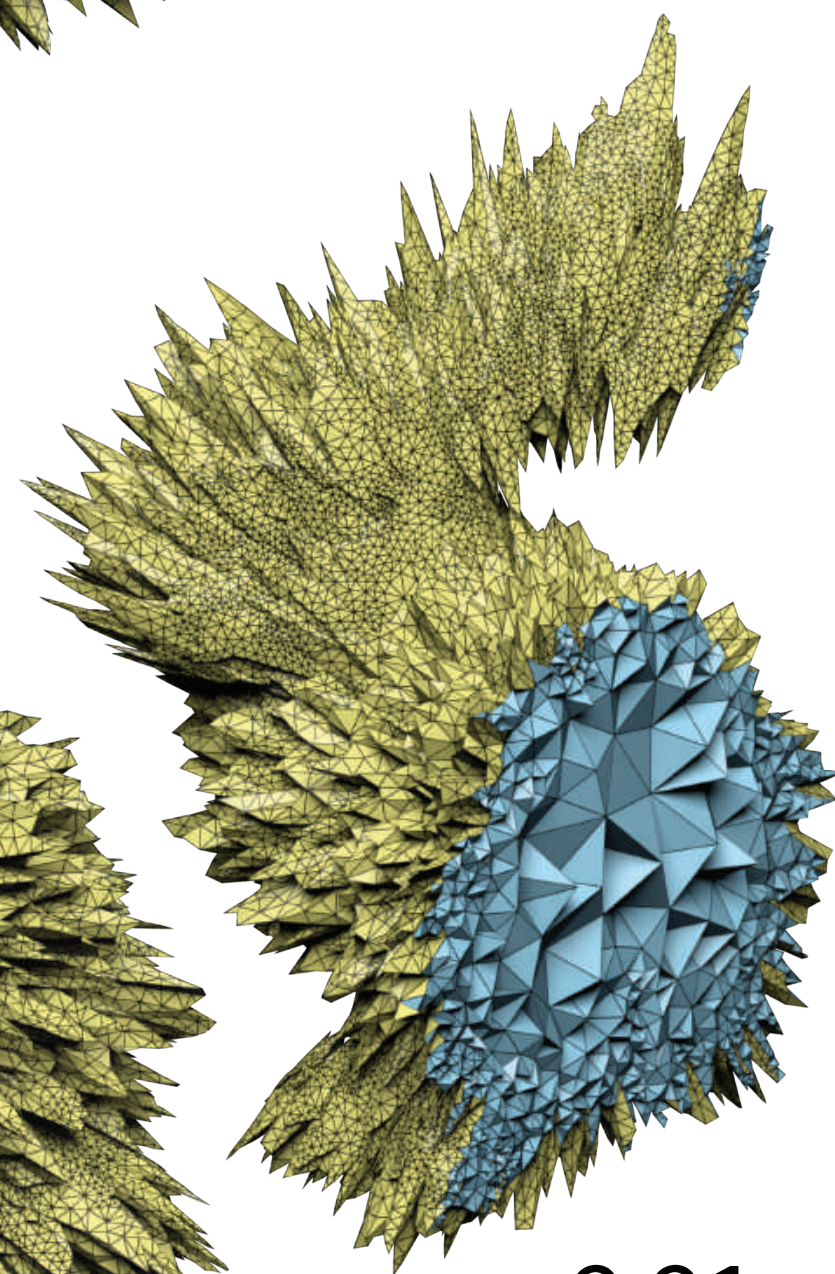
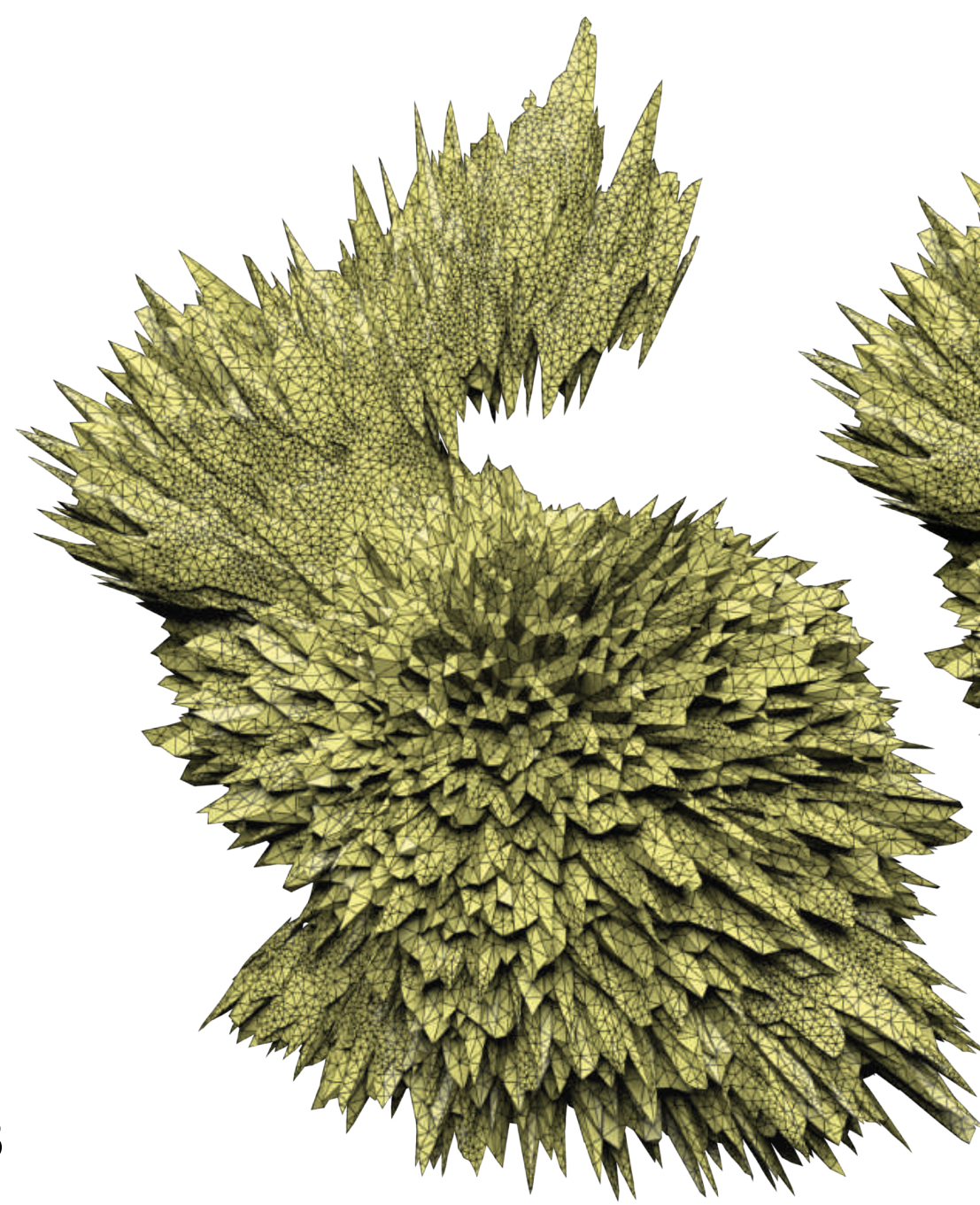
noise = 0.05



7.68

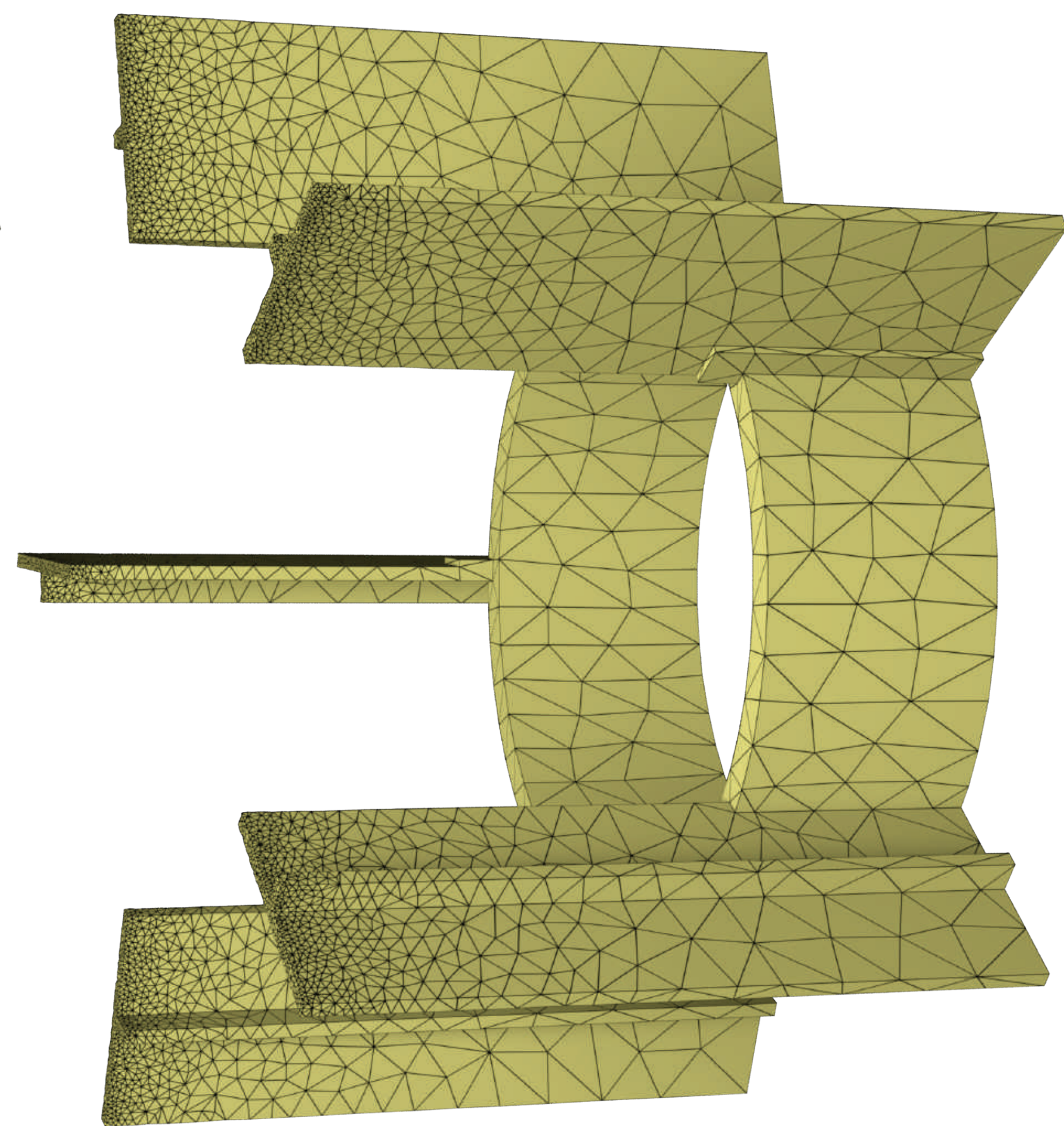
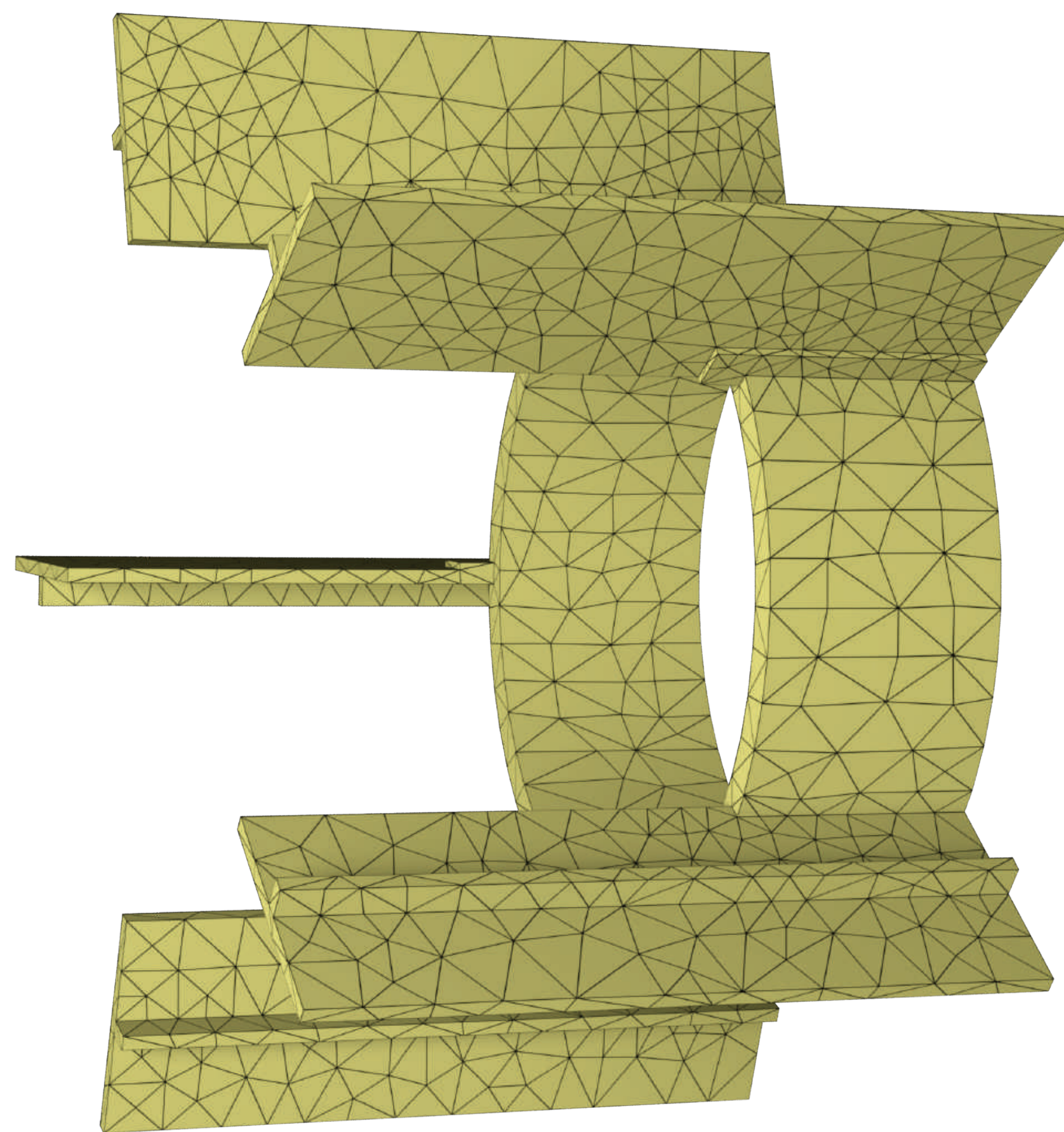


noise = 0.1<sub>53</sub>



9.91







Input

CGAL

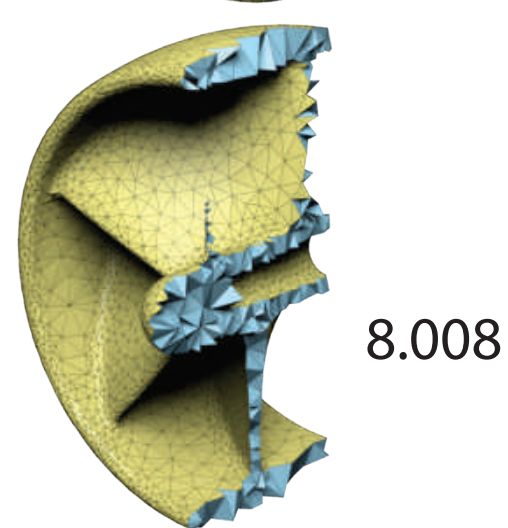
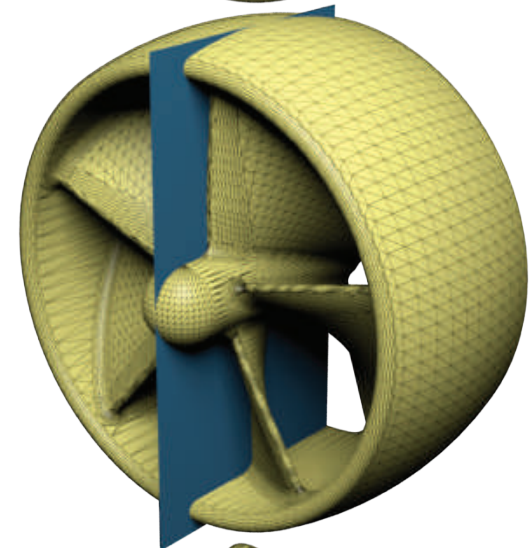
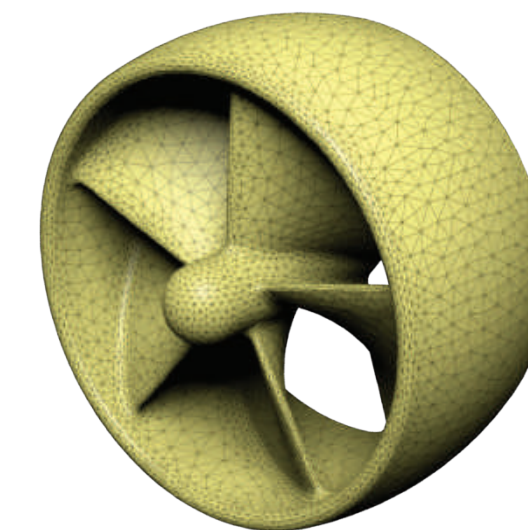
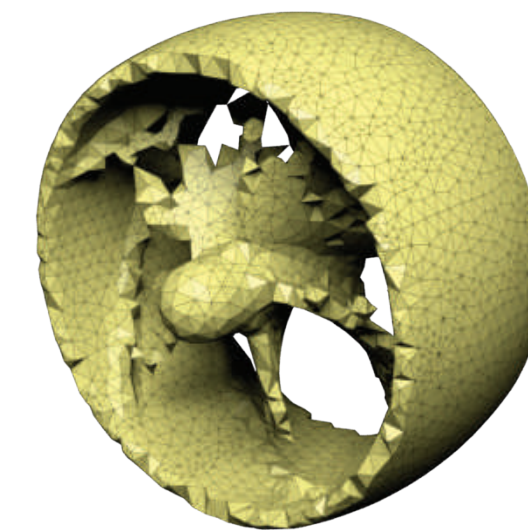
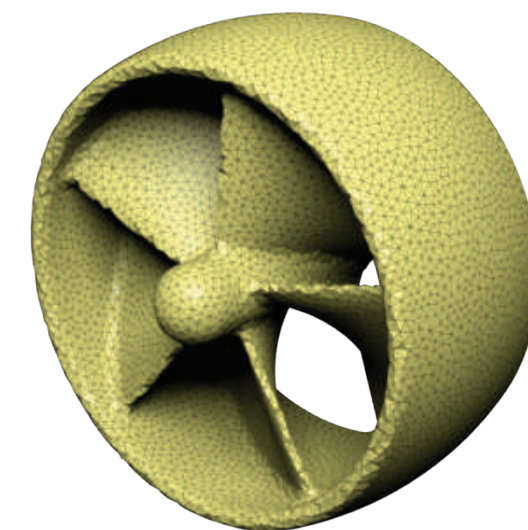
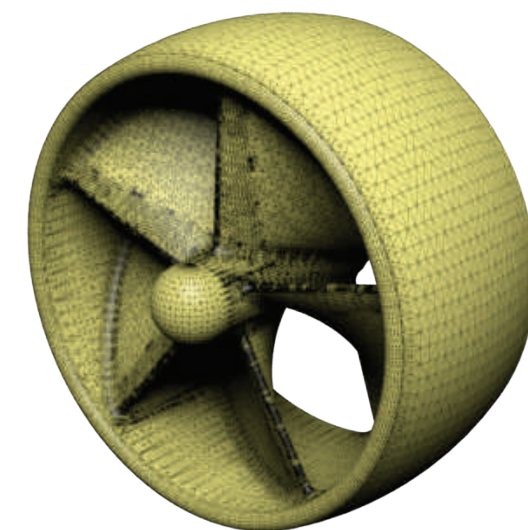
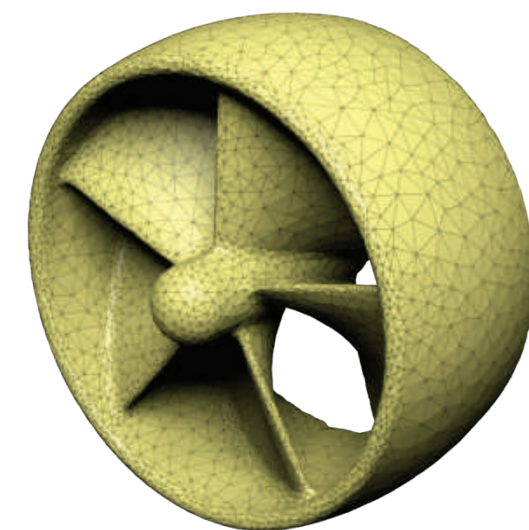
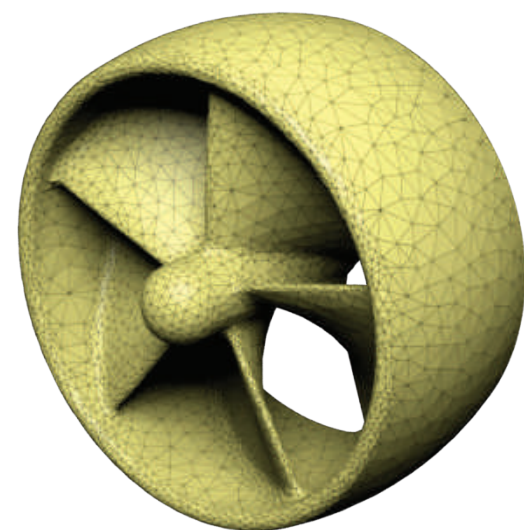
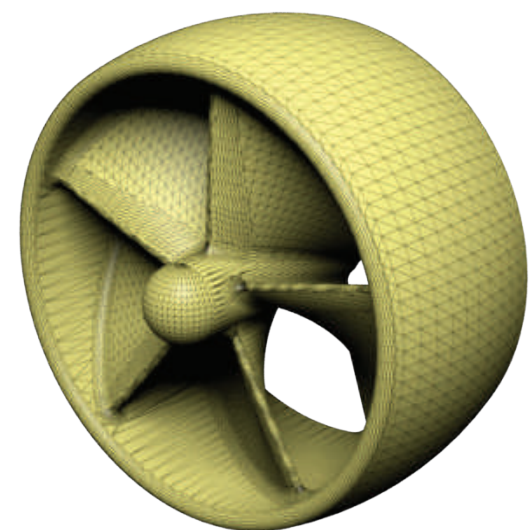
CGAL - no features

TetGen

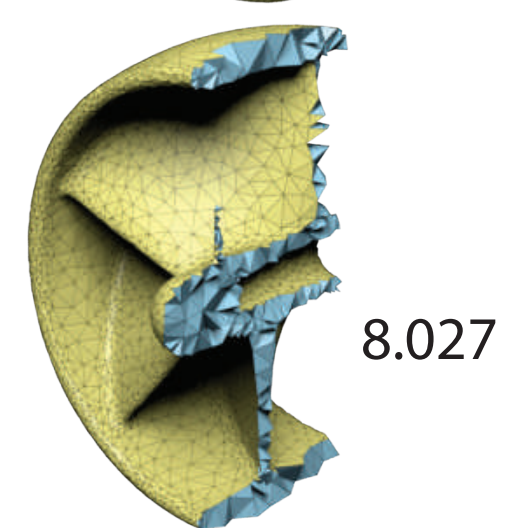
DelPSC

Quartet

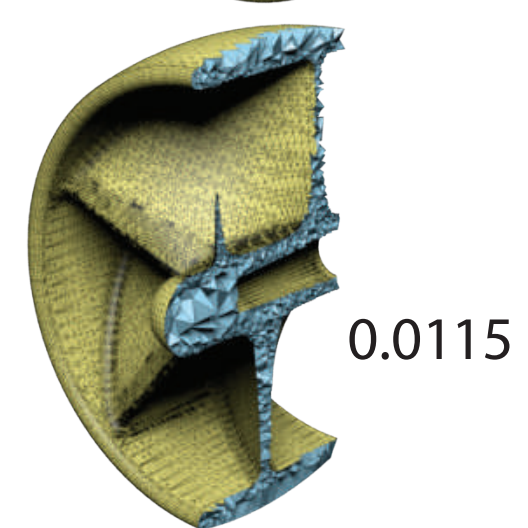
Ours



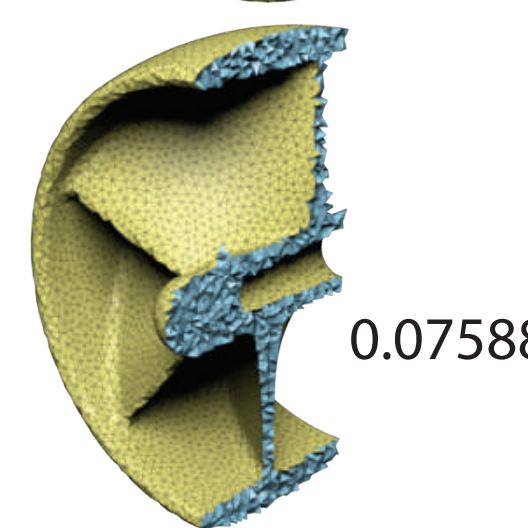
8.008



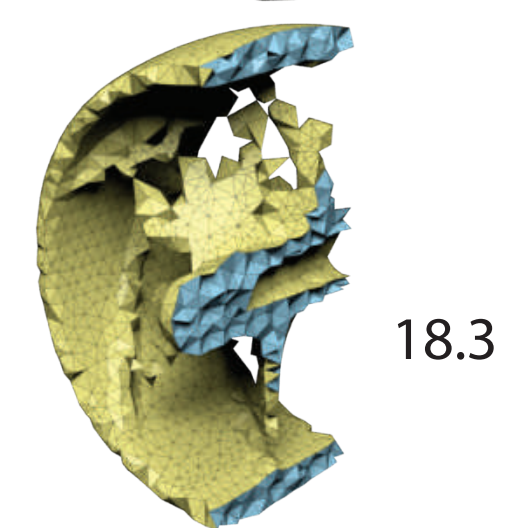
8.027



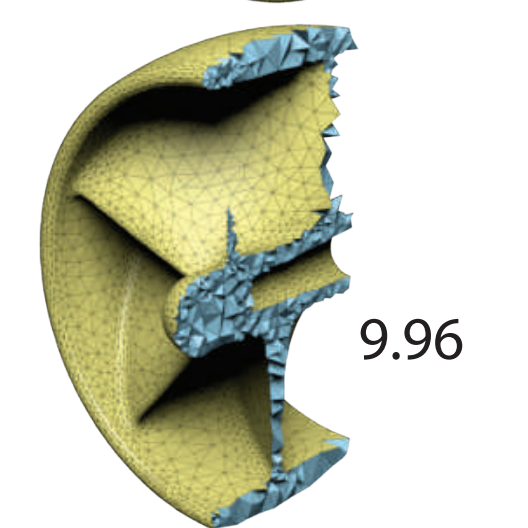
0.0115



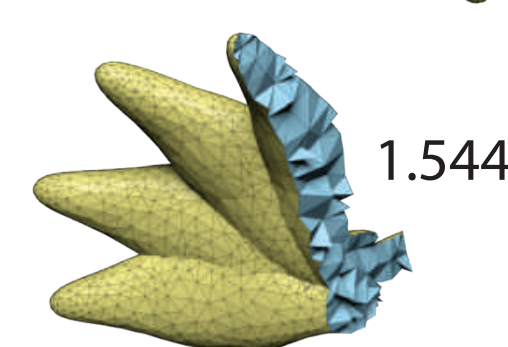
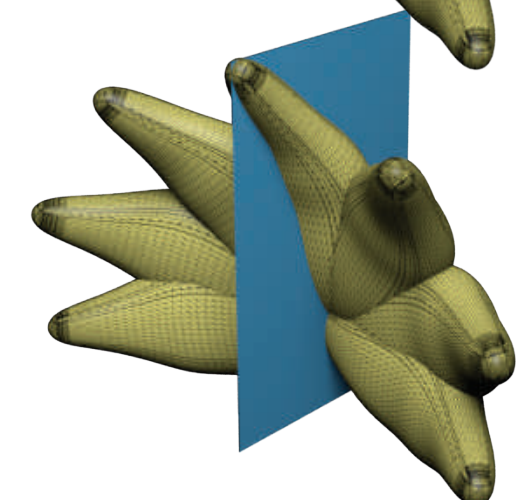
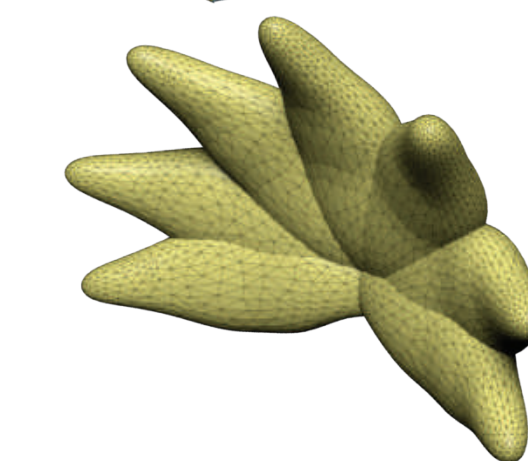
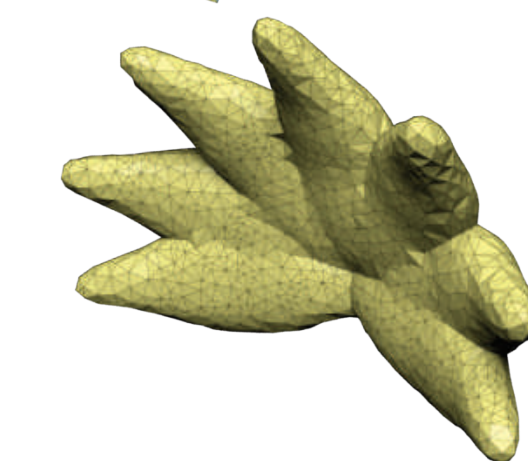
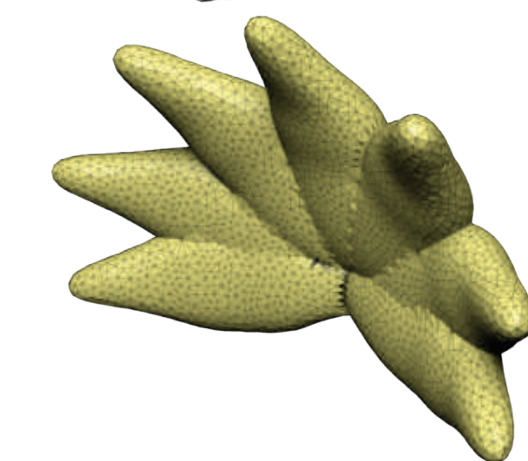
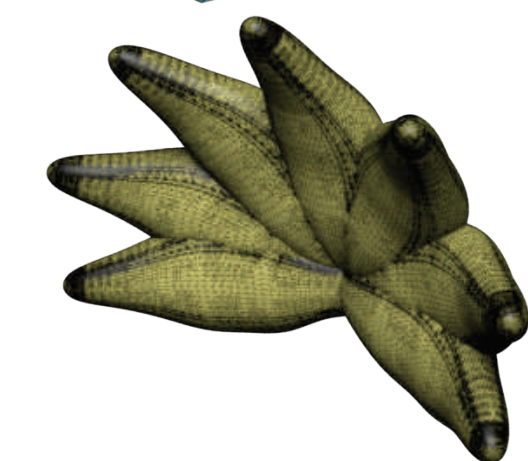
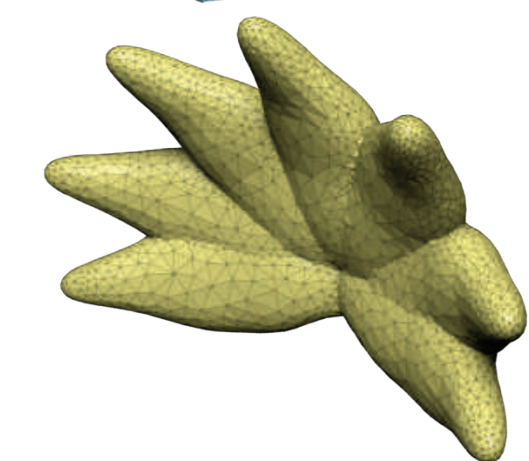
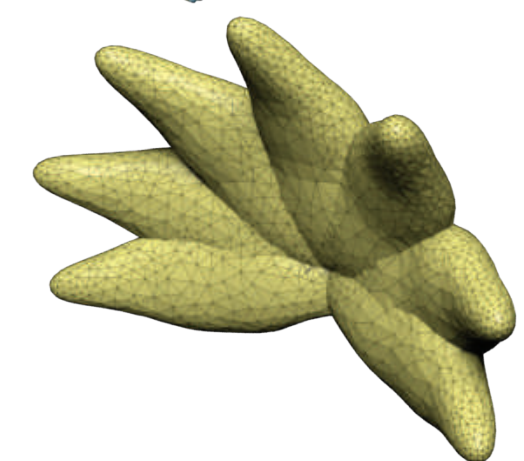
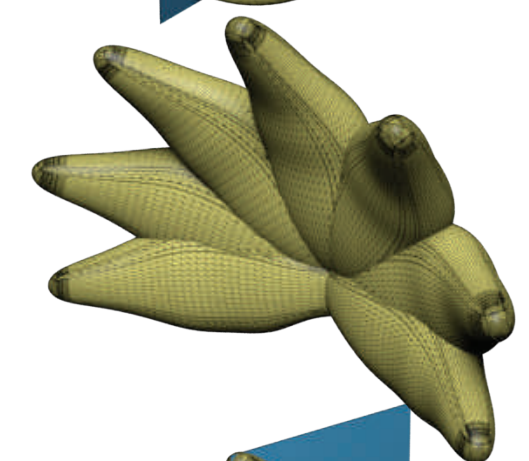
0.07588



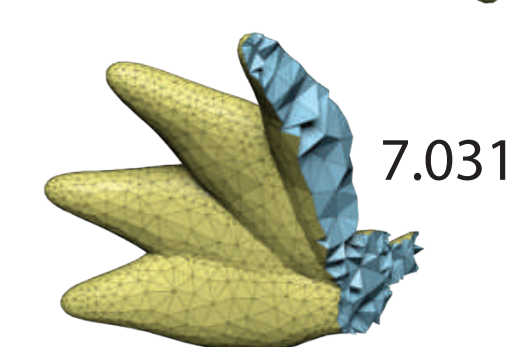
18.3



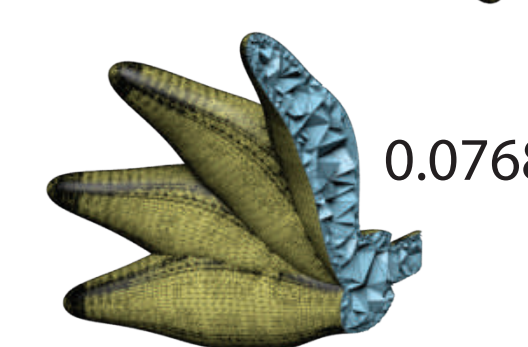
9.96



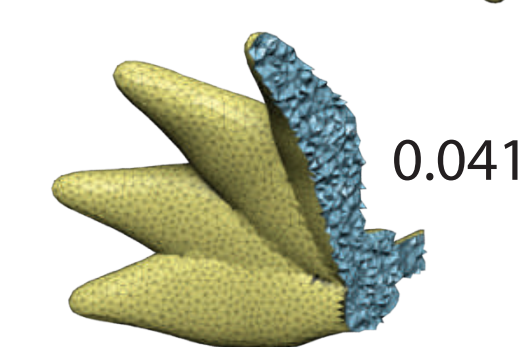
1.544



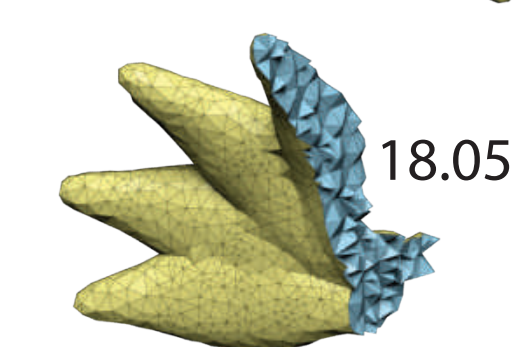
7.031



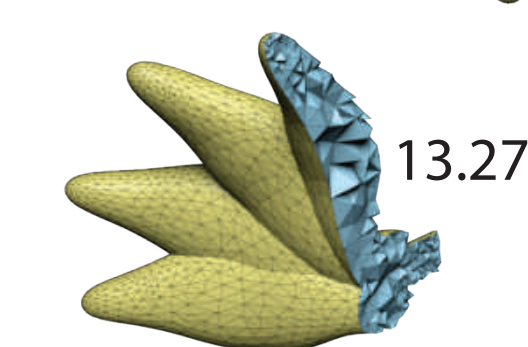
0.07683



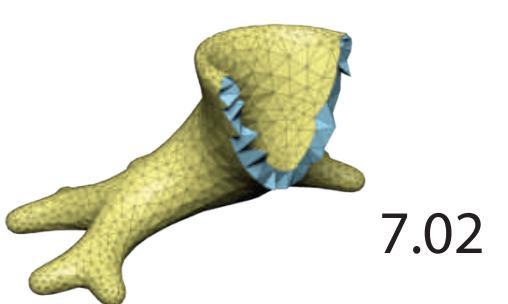
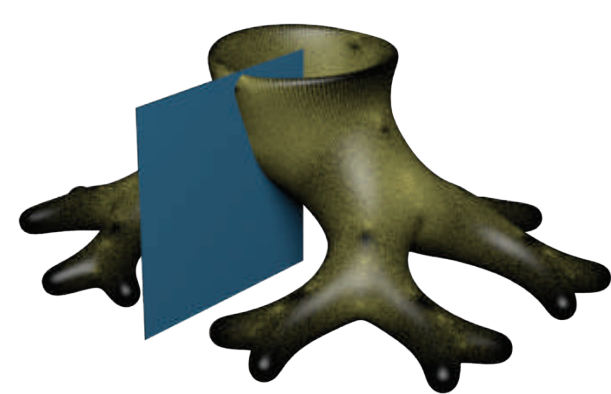
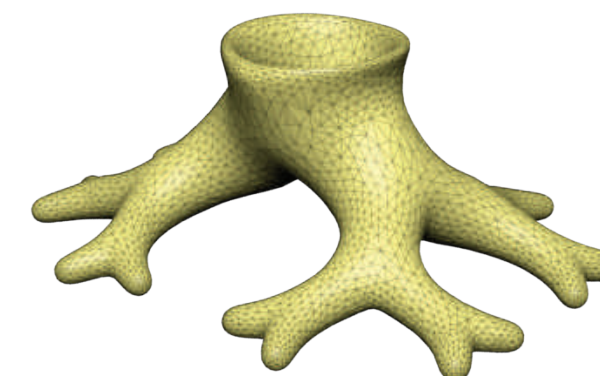
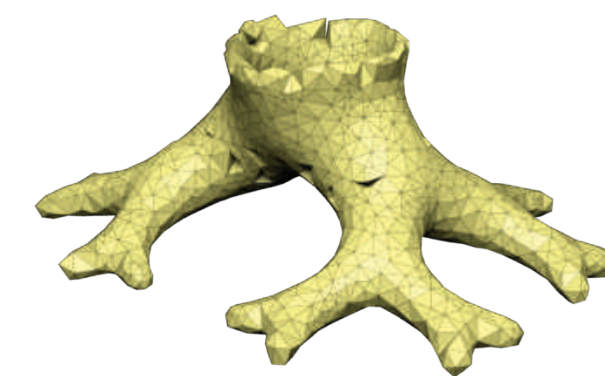
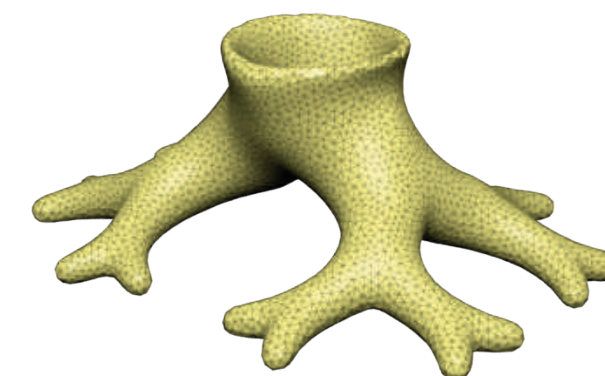
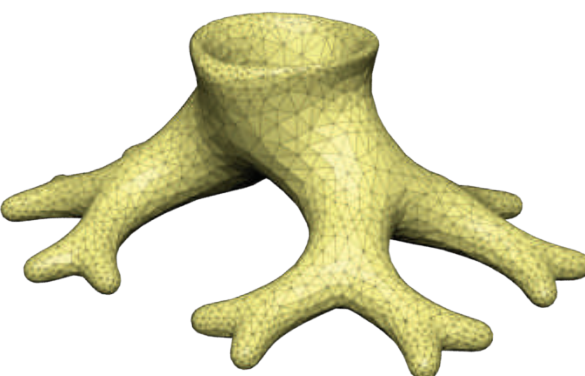
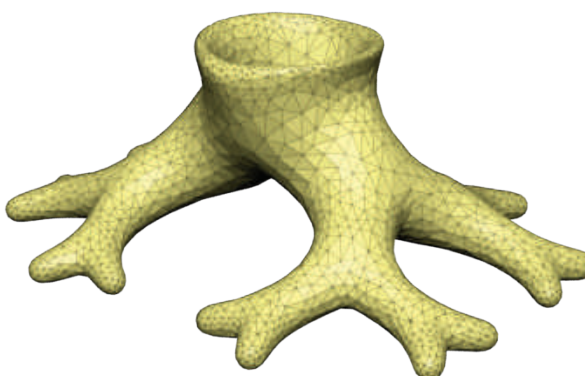
0.04171



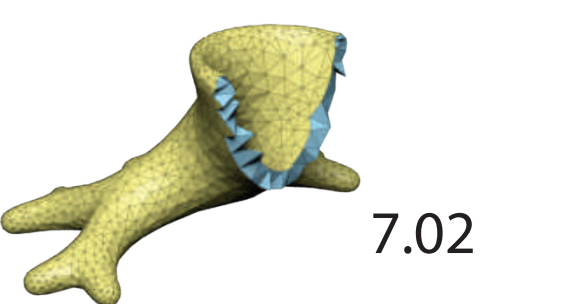
18.05



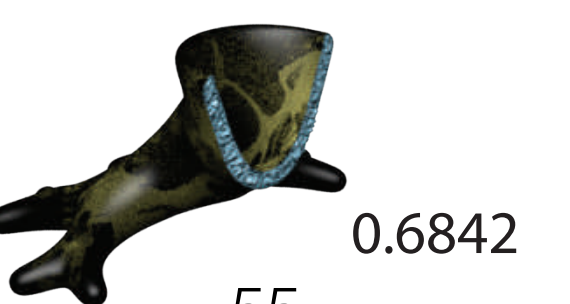
13.27



7.02

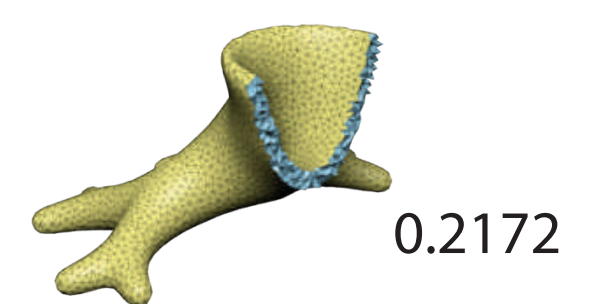


7.02

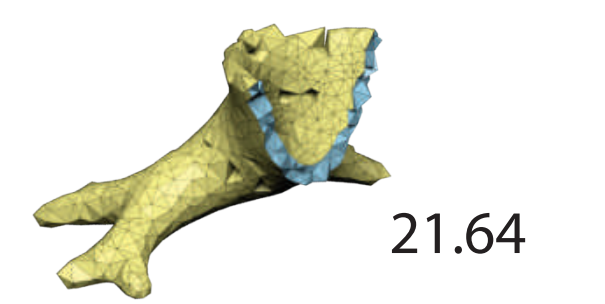


0.6842

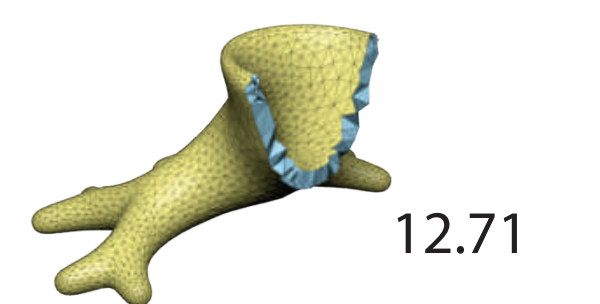
55



0.2172

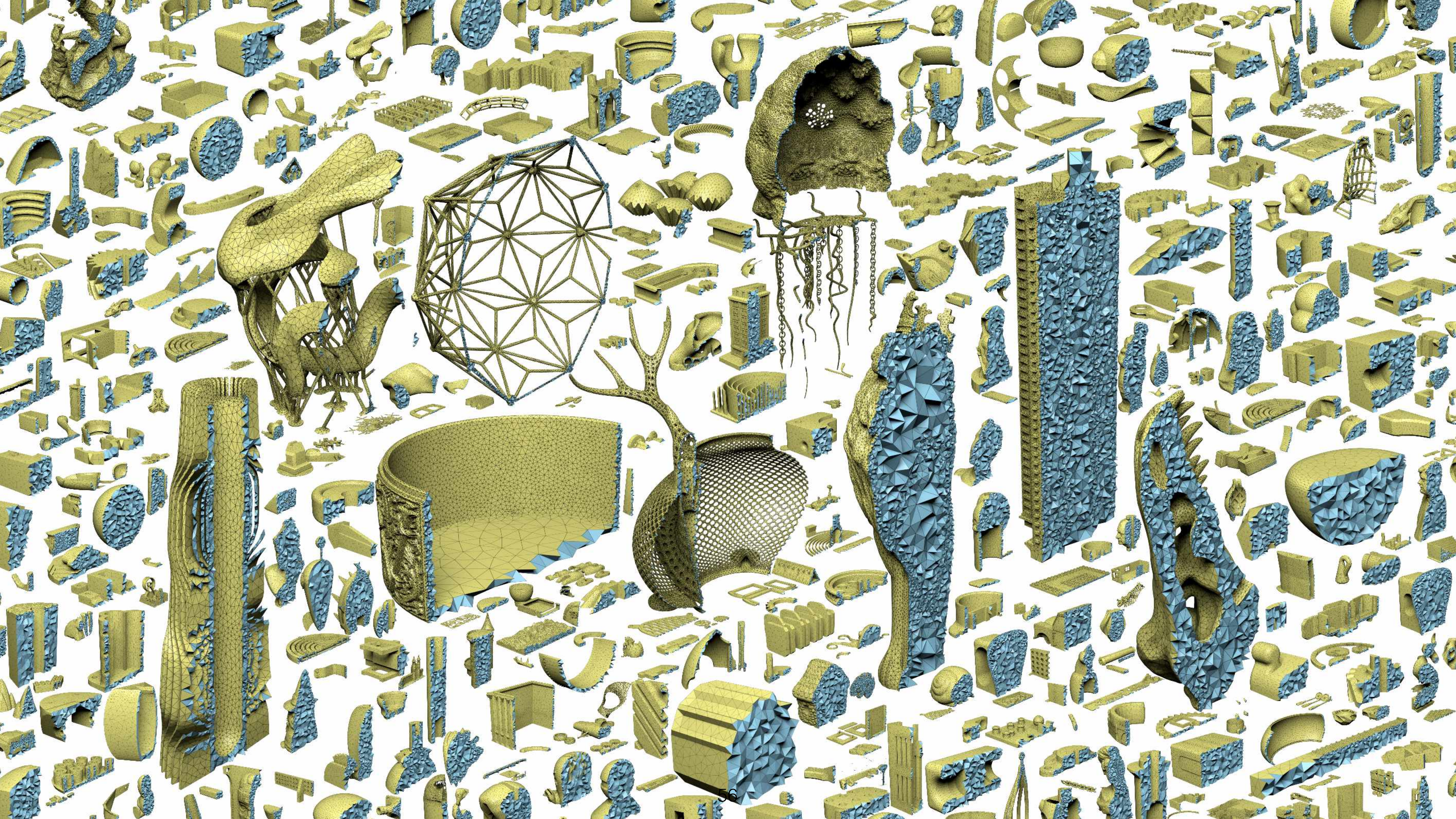


21.64

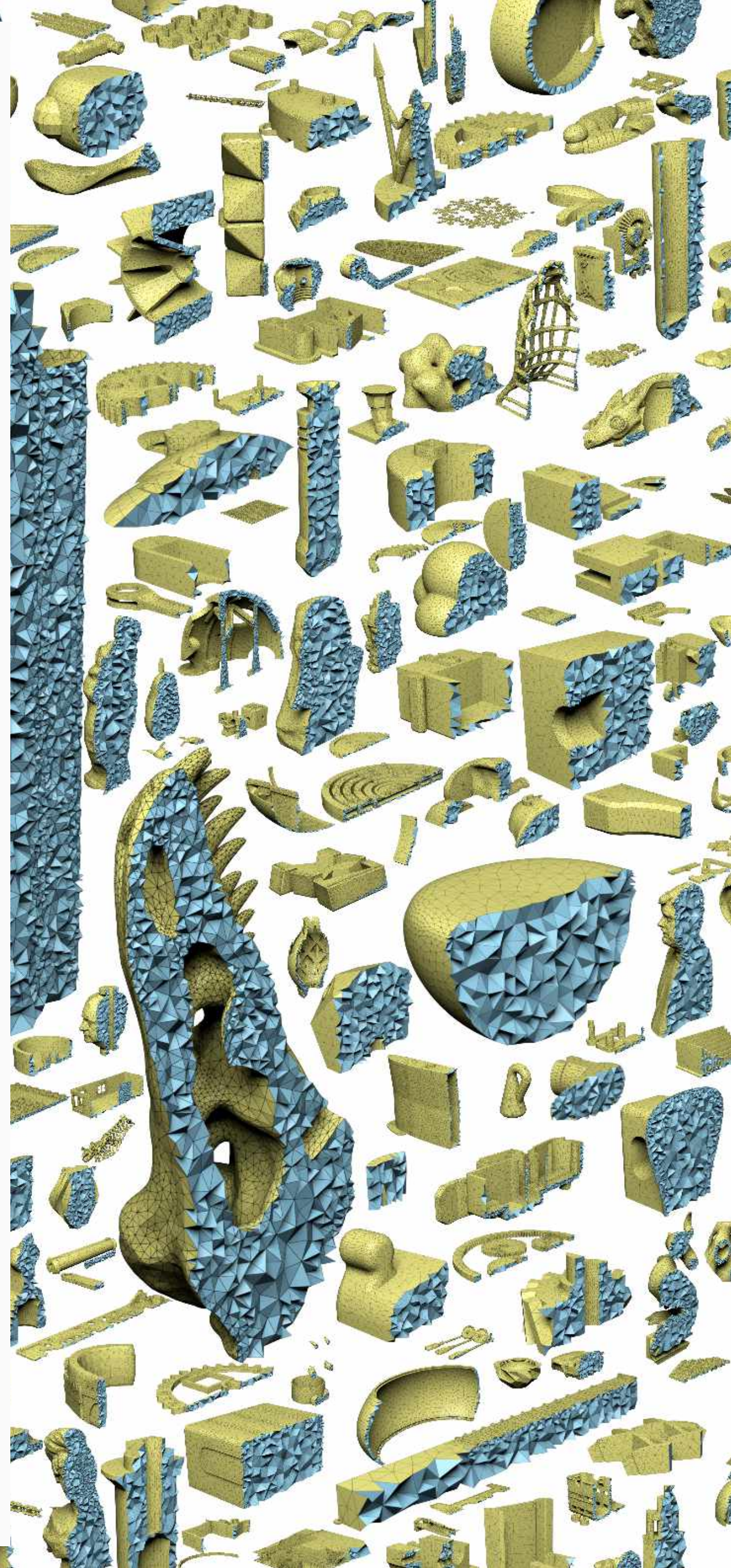
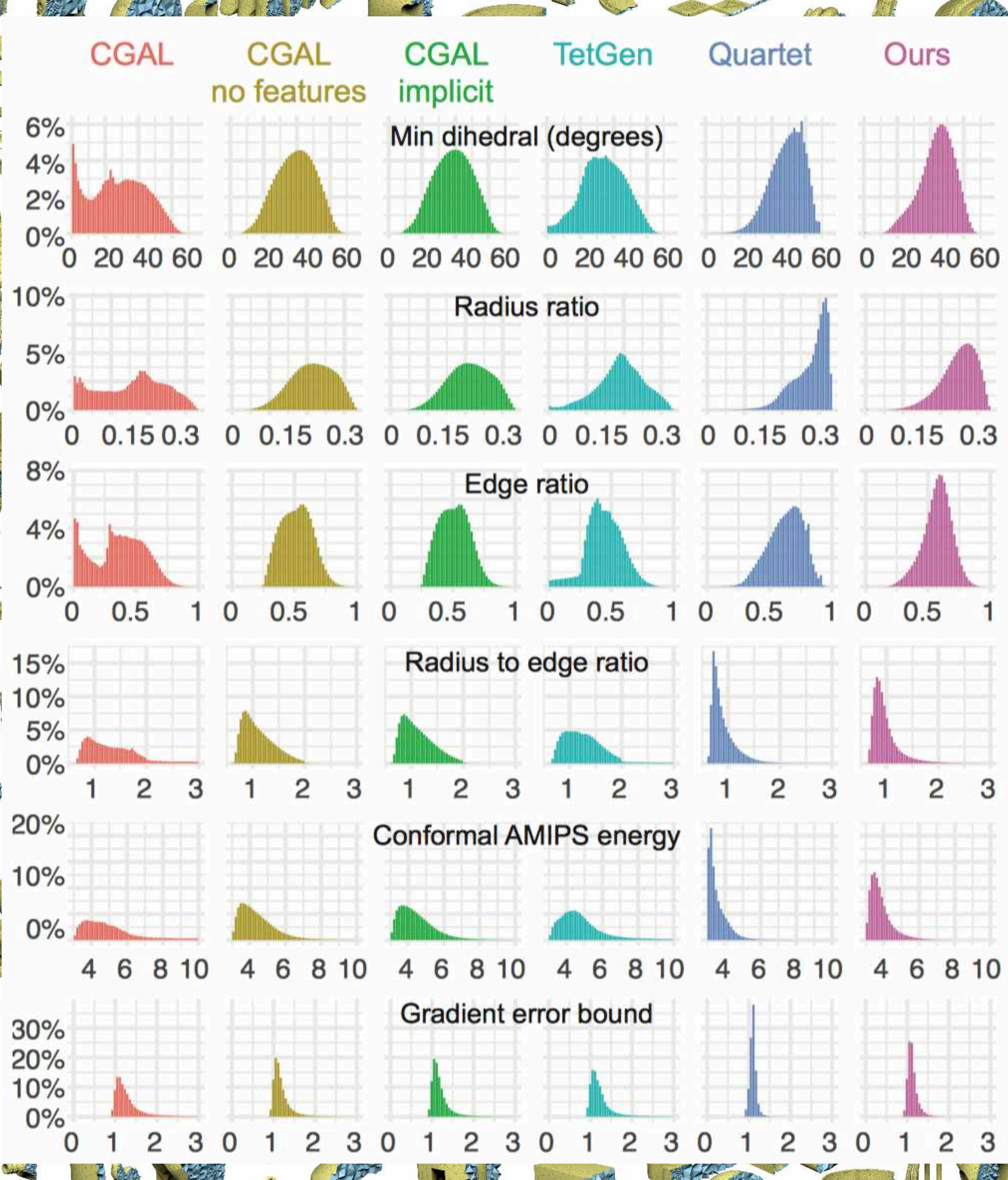
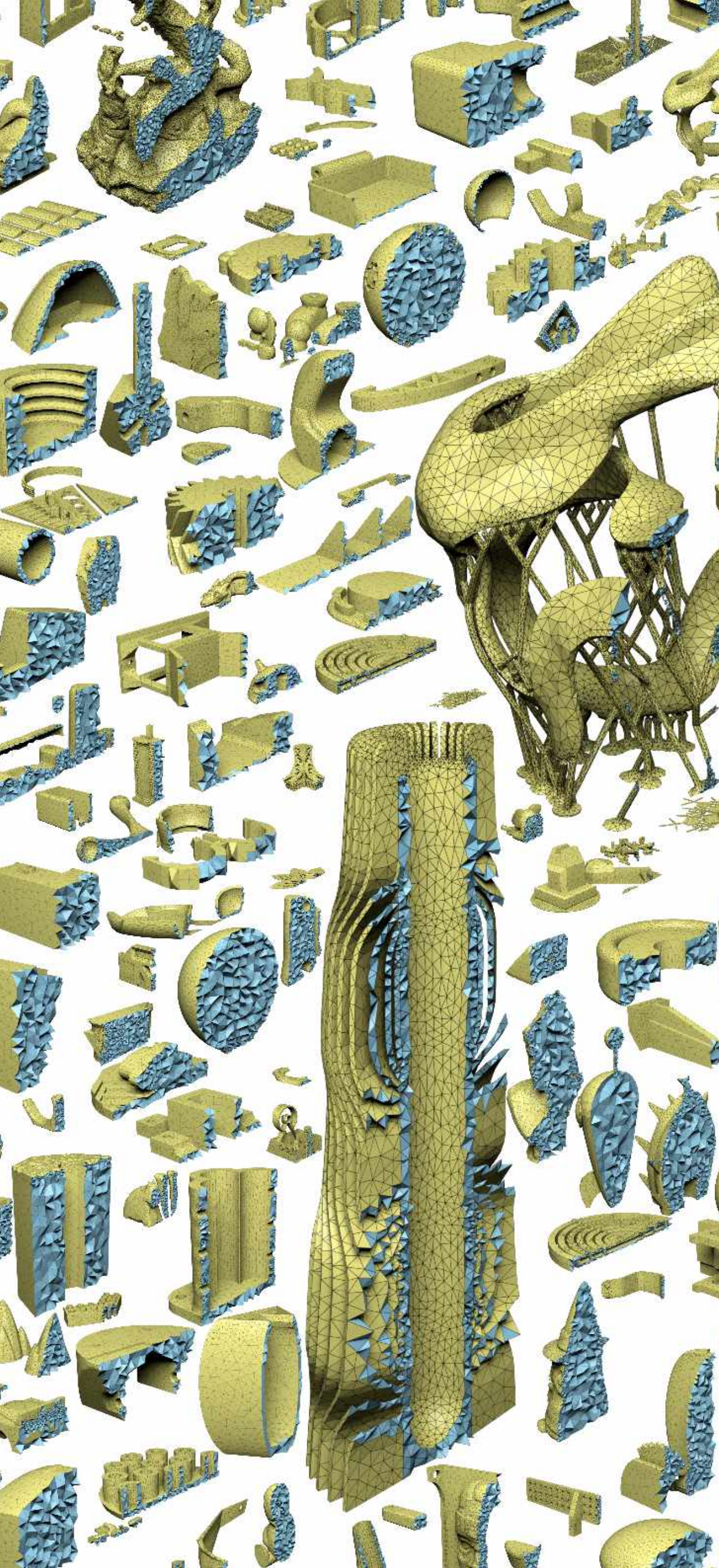


12.71

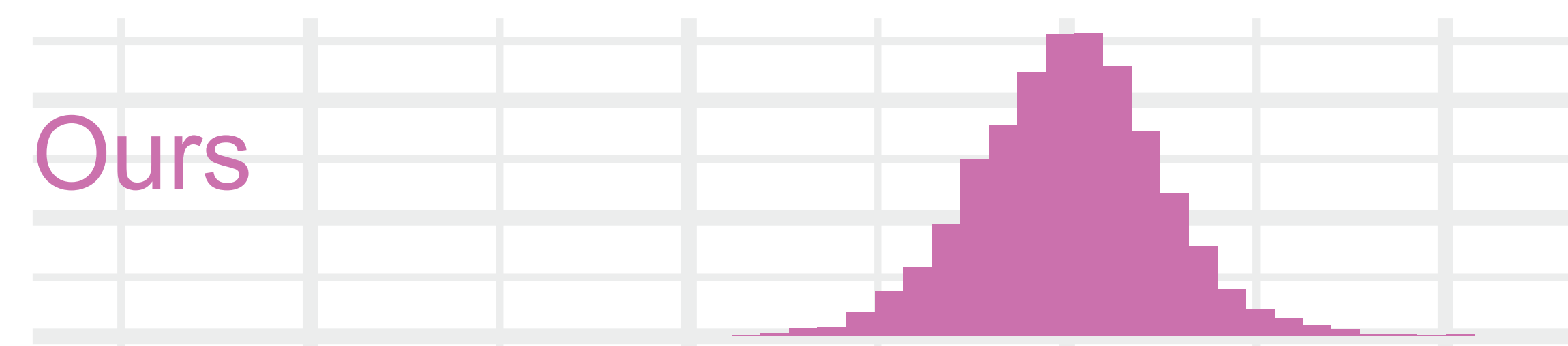
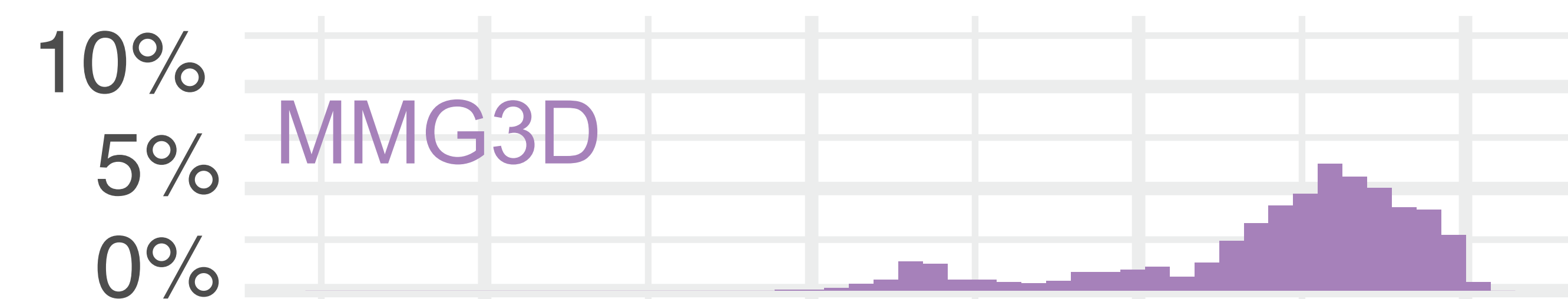
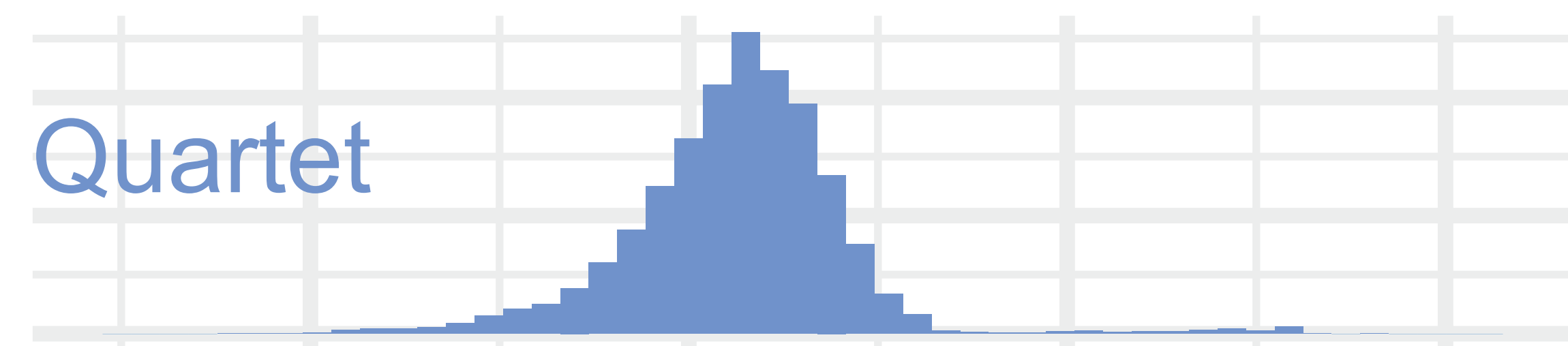
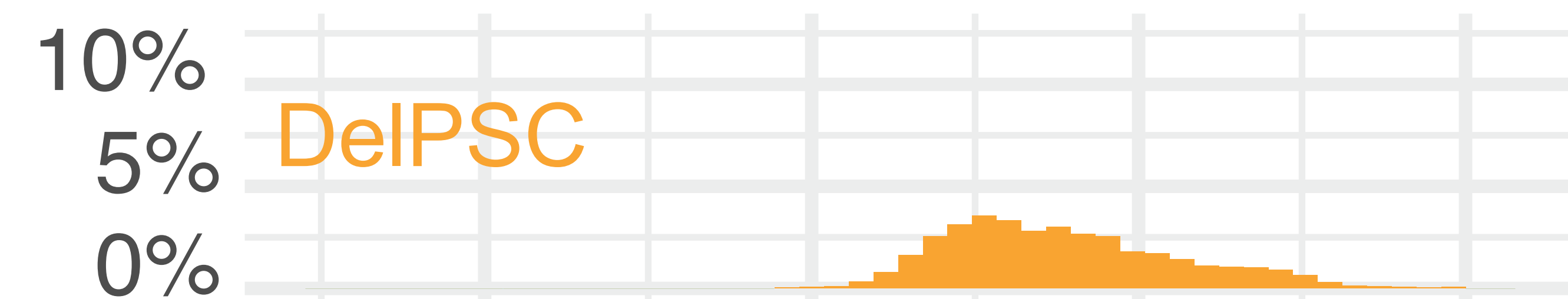
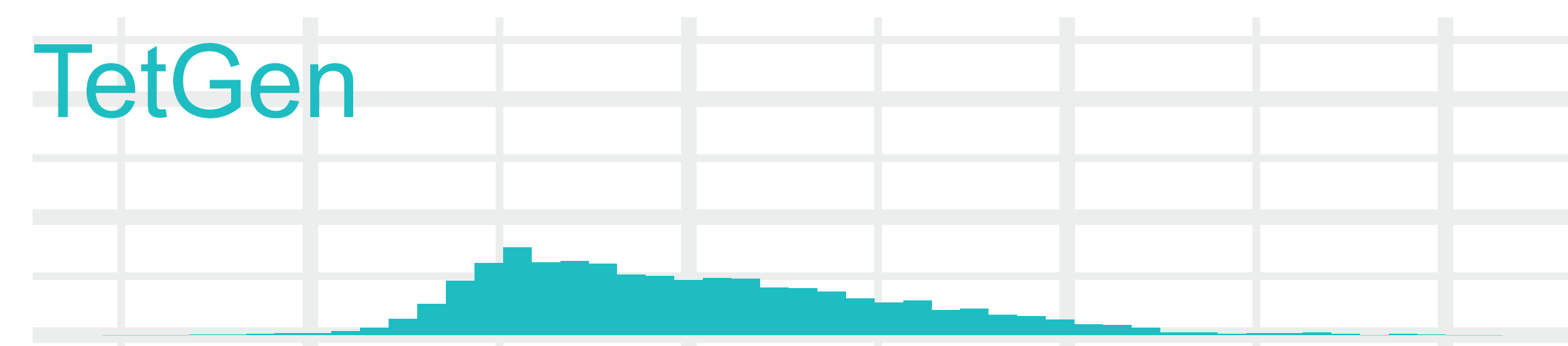
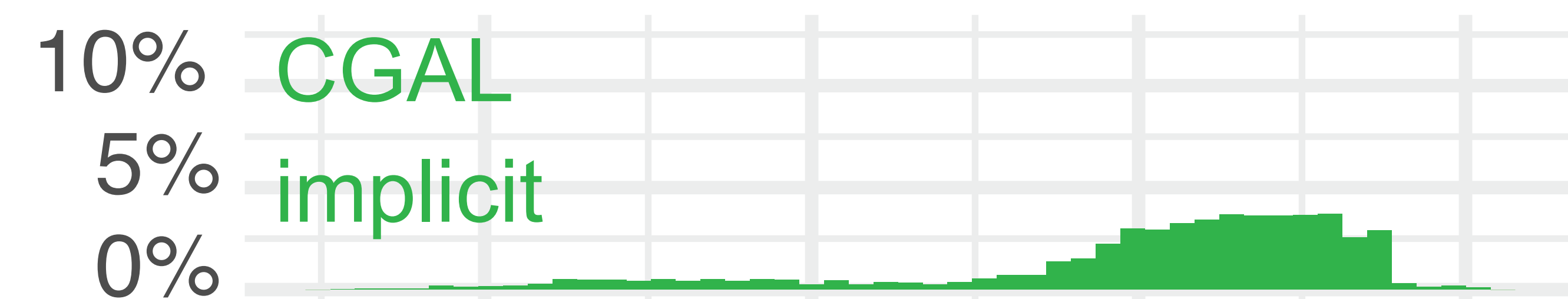
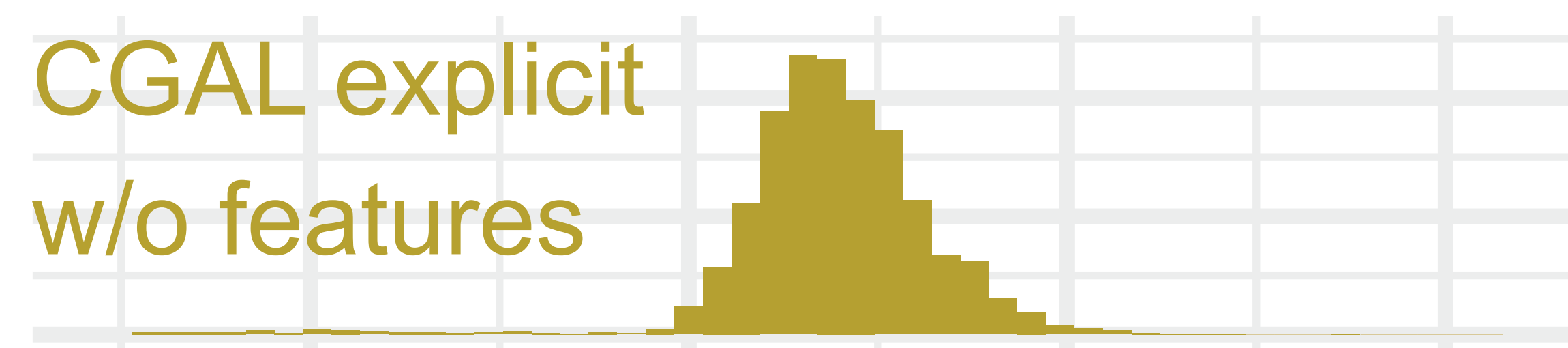
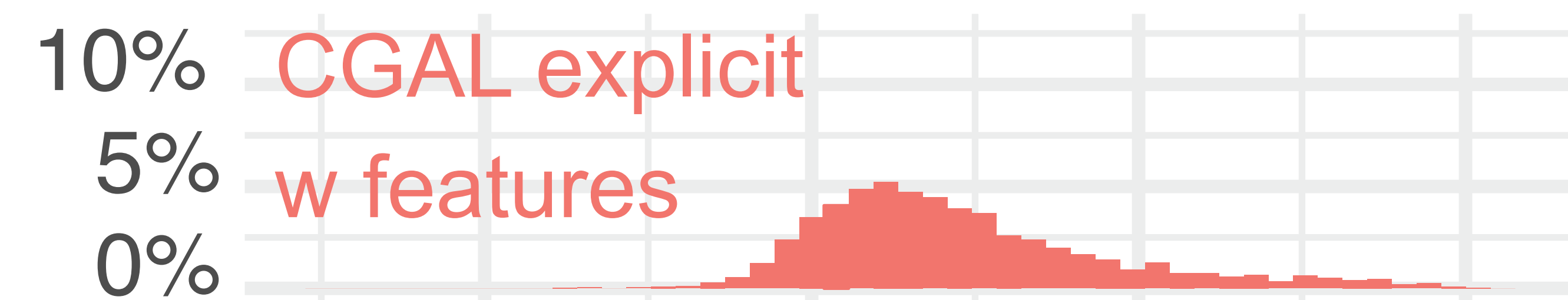












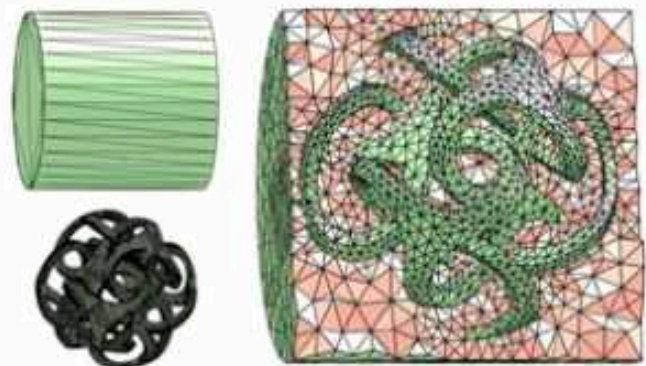
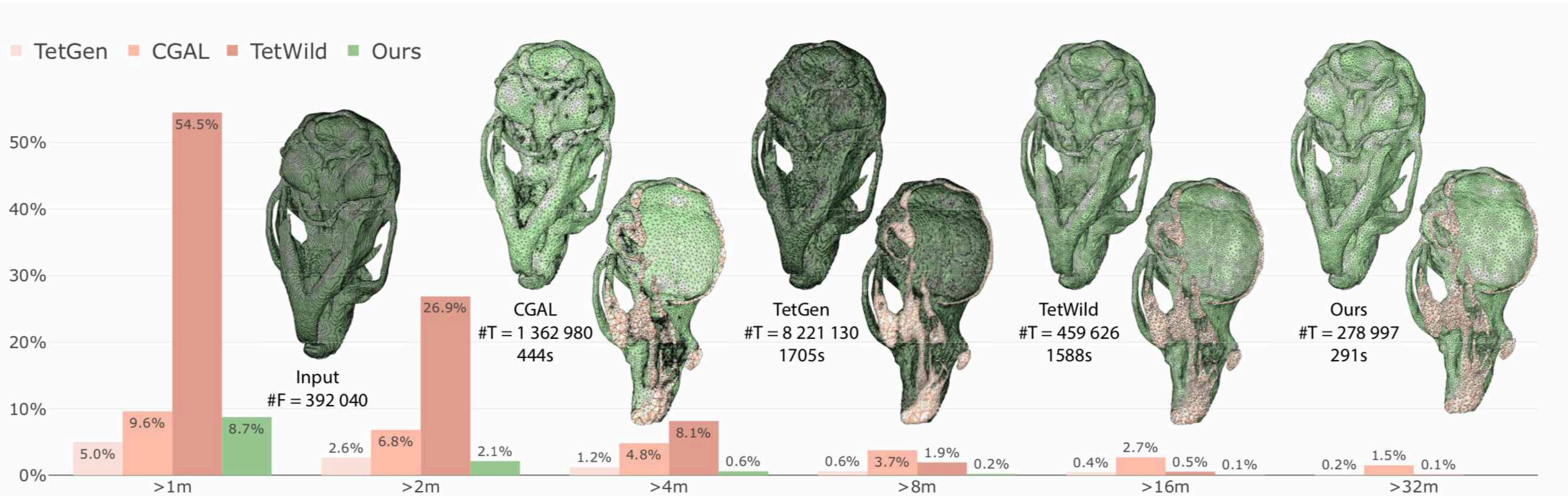
1e-2 1e+0 1e+2 1e+4

1e-2 1e+0 1e+2 1e+4

Time in seconds



# Floating Point Version



## Fast Tetrahedral Meshing in the Wild

[Yixin Hu](#), [Teseo Schneider](#), Bolun Wang, [Denis Zorin](#), [Daniele Panozzo](#),

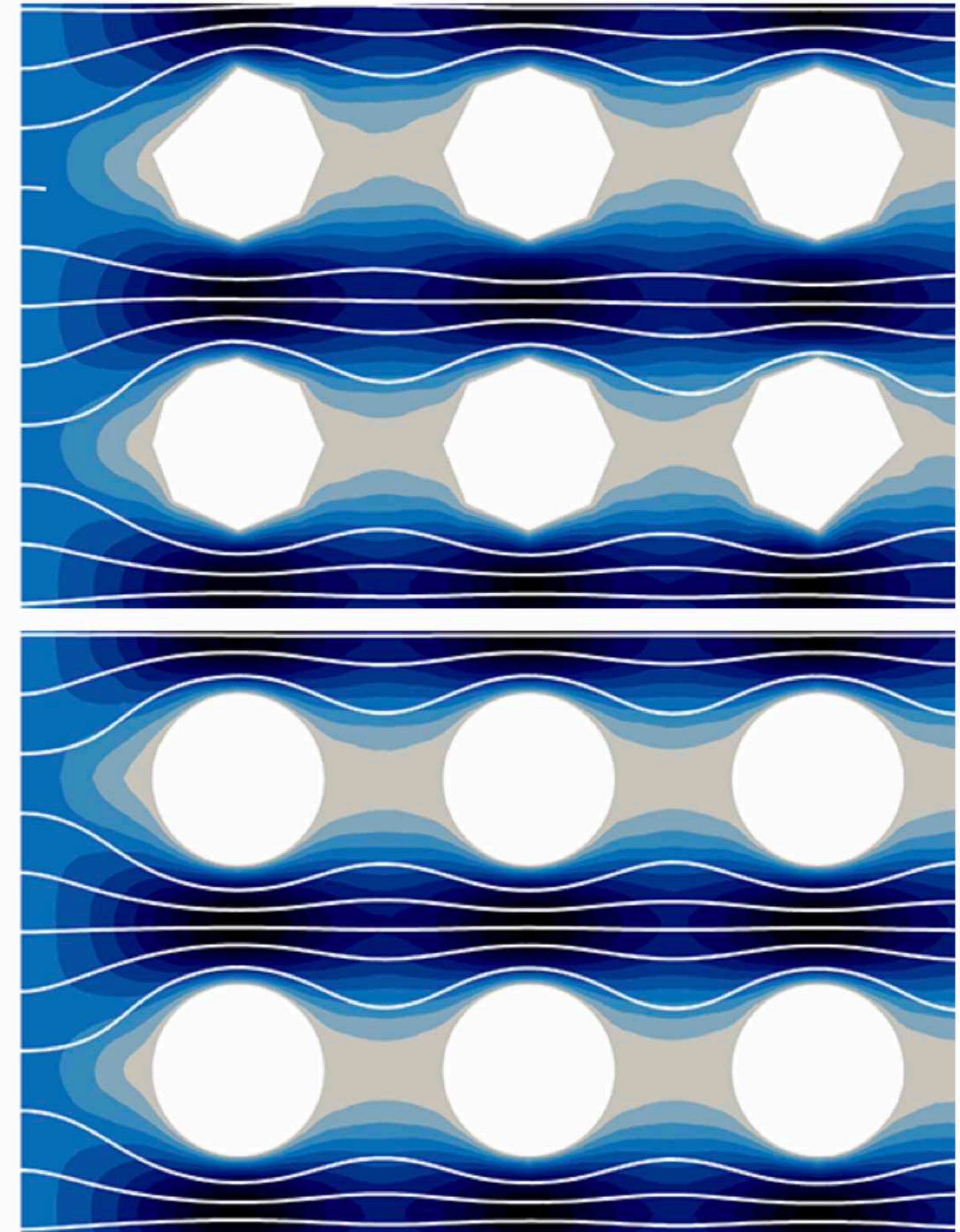
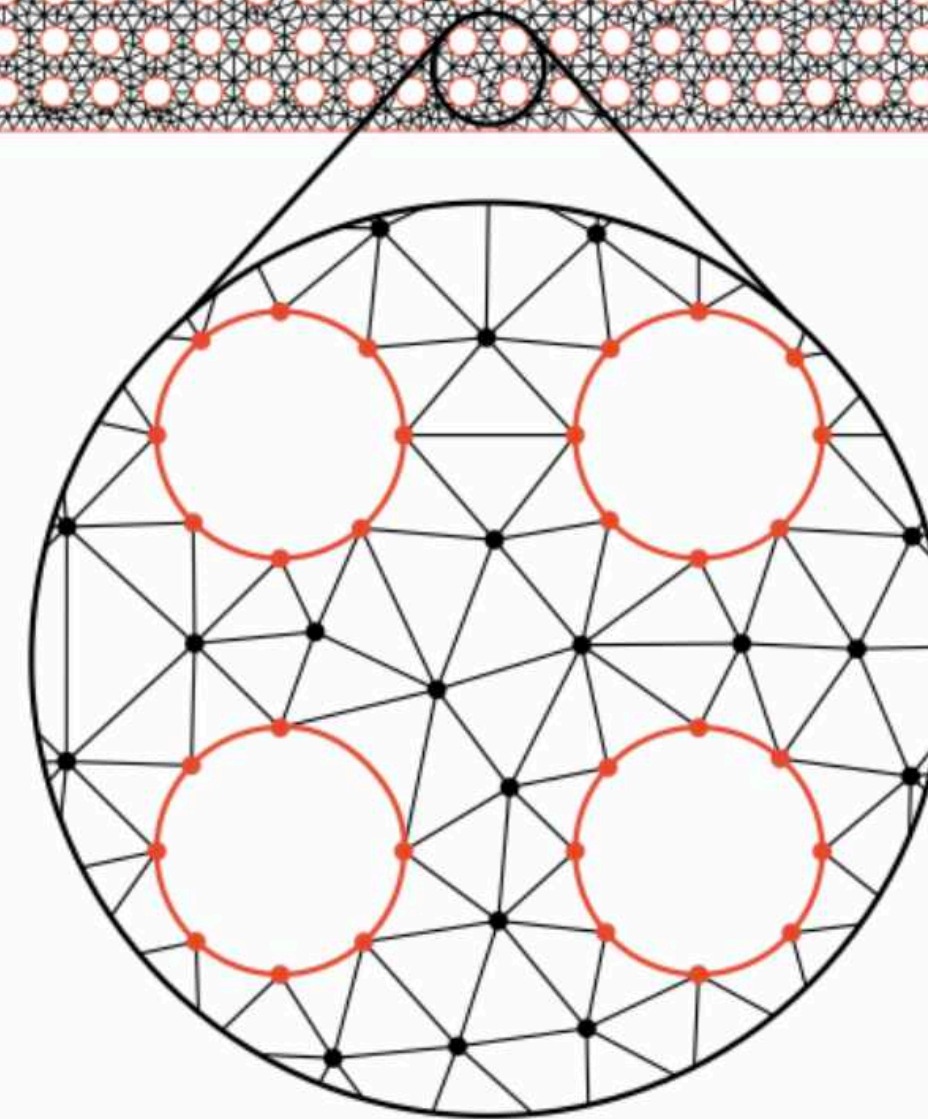
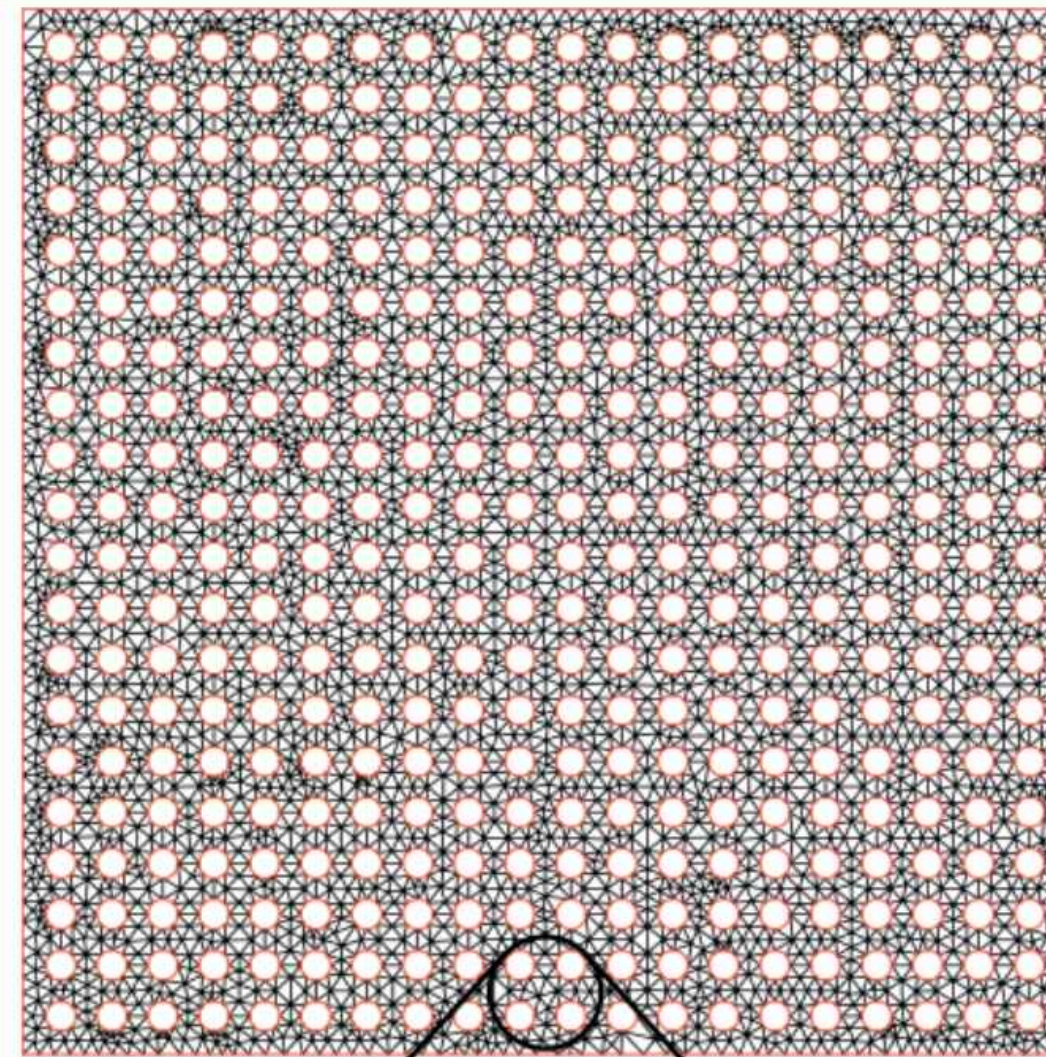
Arxiv, 2019

[\[Paper\]](#) [\[Code\]](#)



# High-Order Geometric Map

- We can also create meshes with a high-order geometric map to reproduce a set of input Bezier curves
- The runtime increases by 2%
- We are working on a version that can take CAD models as input and reproduce the input NURBS



TriWild: Robust Triangulation with Curve Constraints

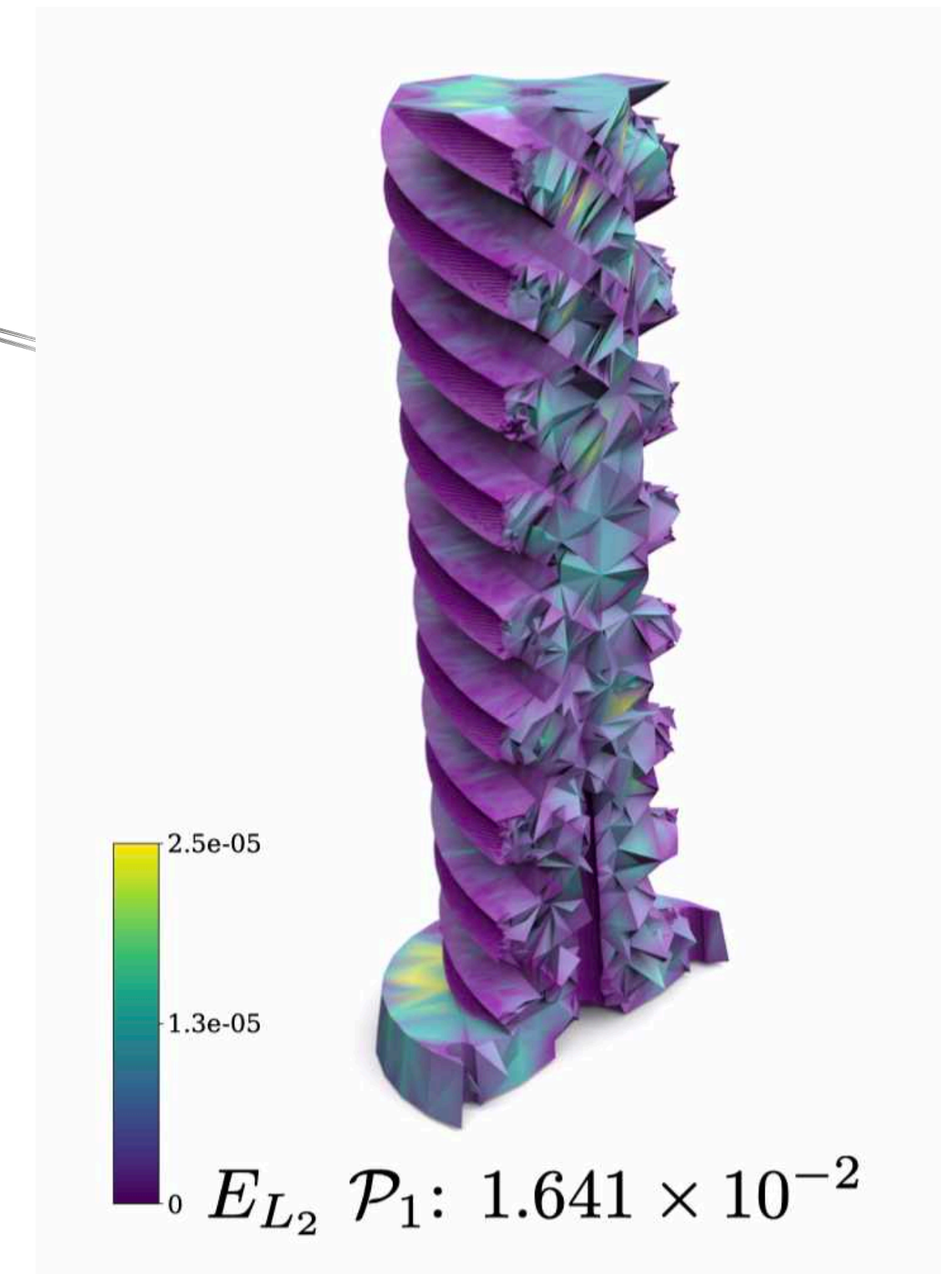
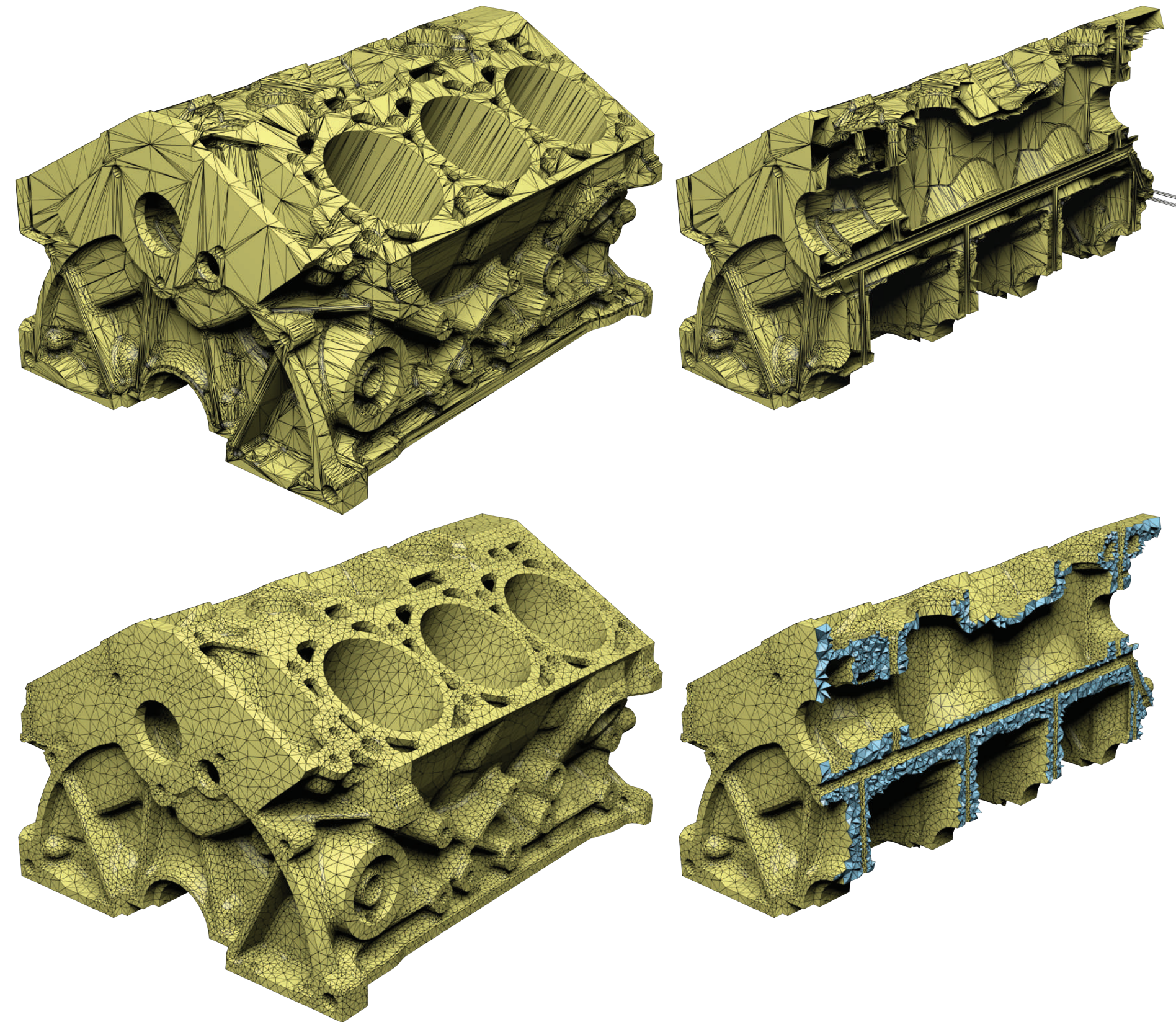
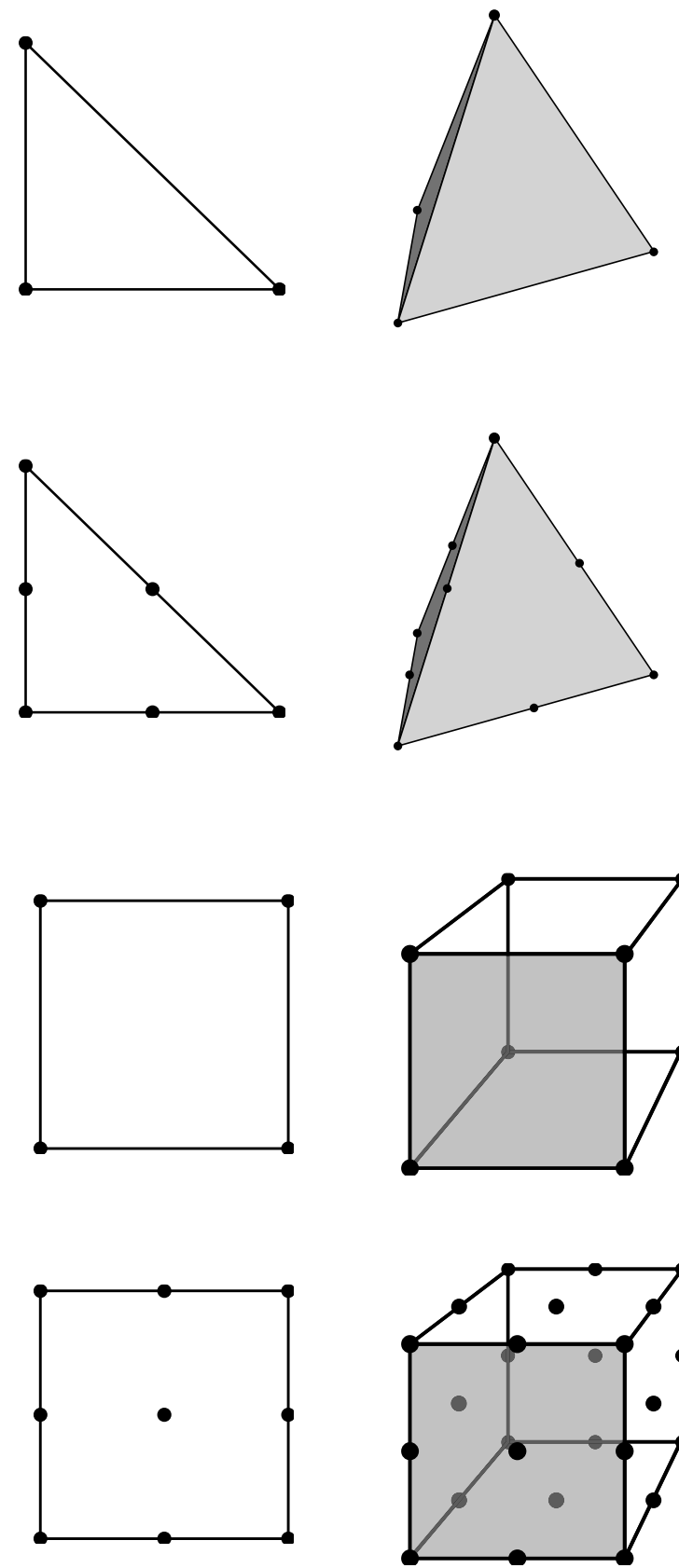
Yixin Hu, Teseo Schneider, Xifeng Gao, Qingnan Zhou, Alec Jacobson, Denis Zorin, Daniele Panozzo,

ACM Transaction on Graphics (SIGGRAPH), 2019

[\[Paper\]](#) [\[Code\]](#) [\[Data\]](#)



# Overview



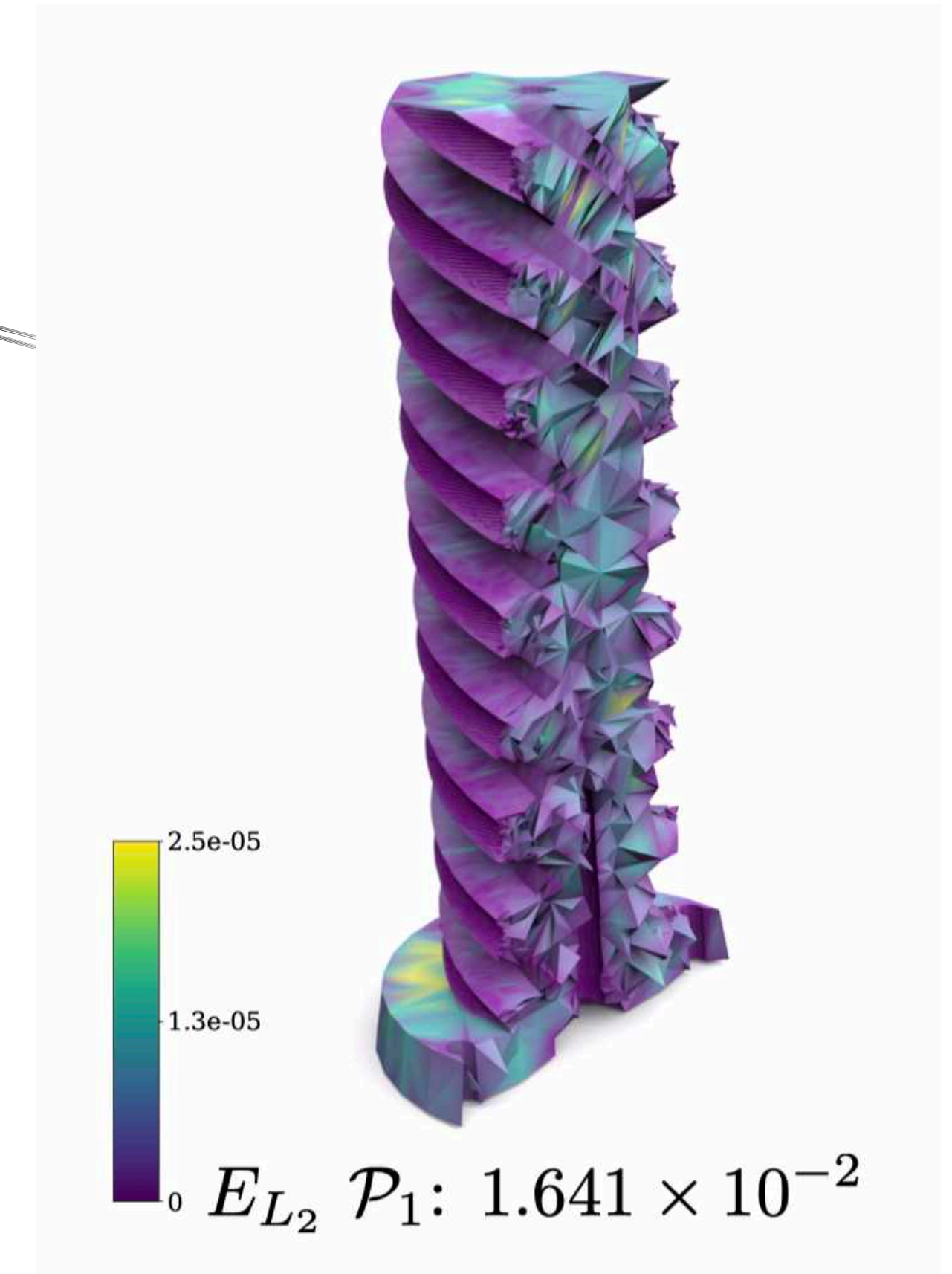
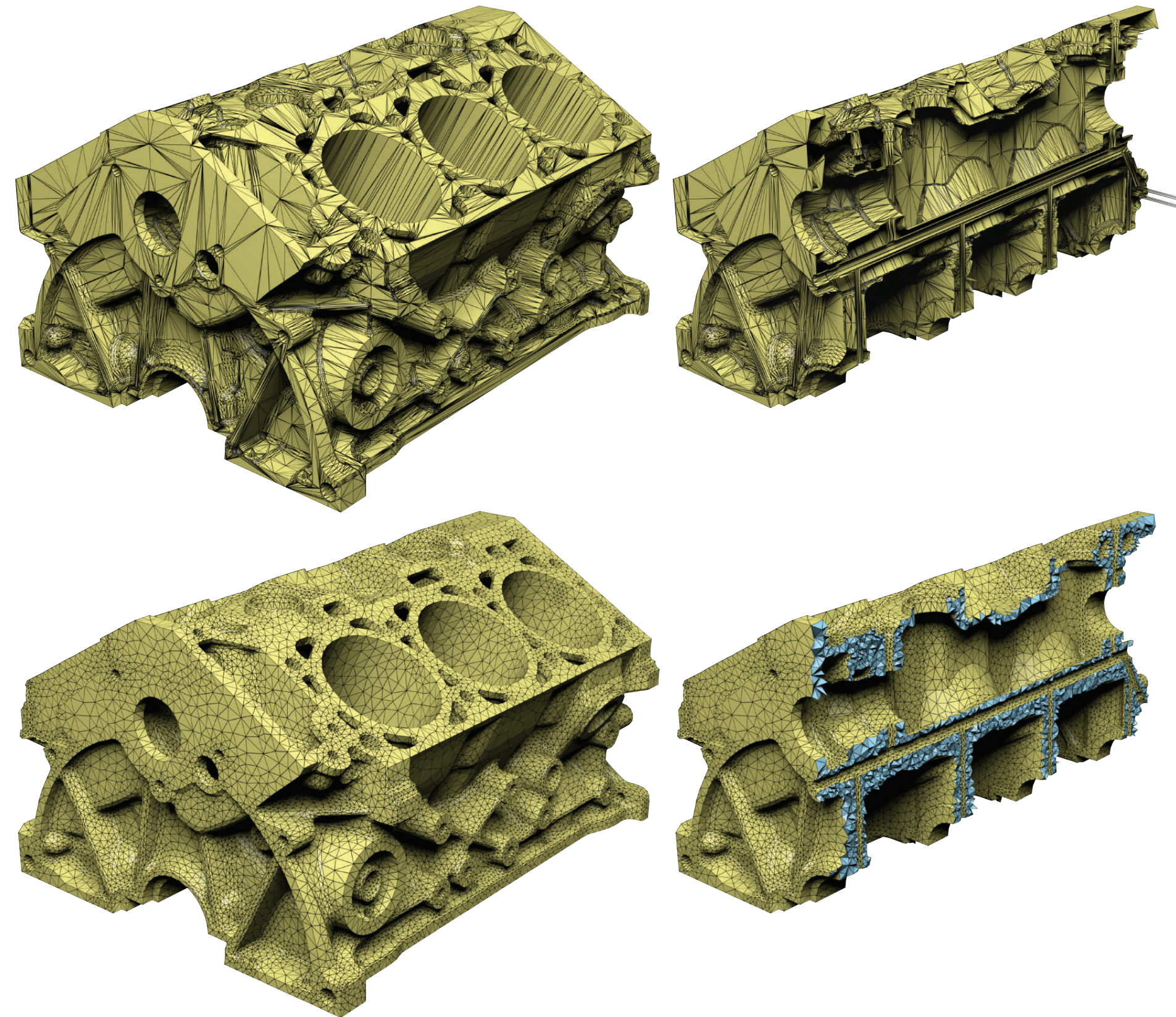
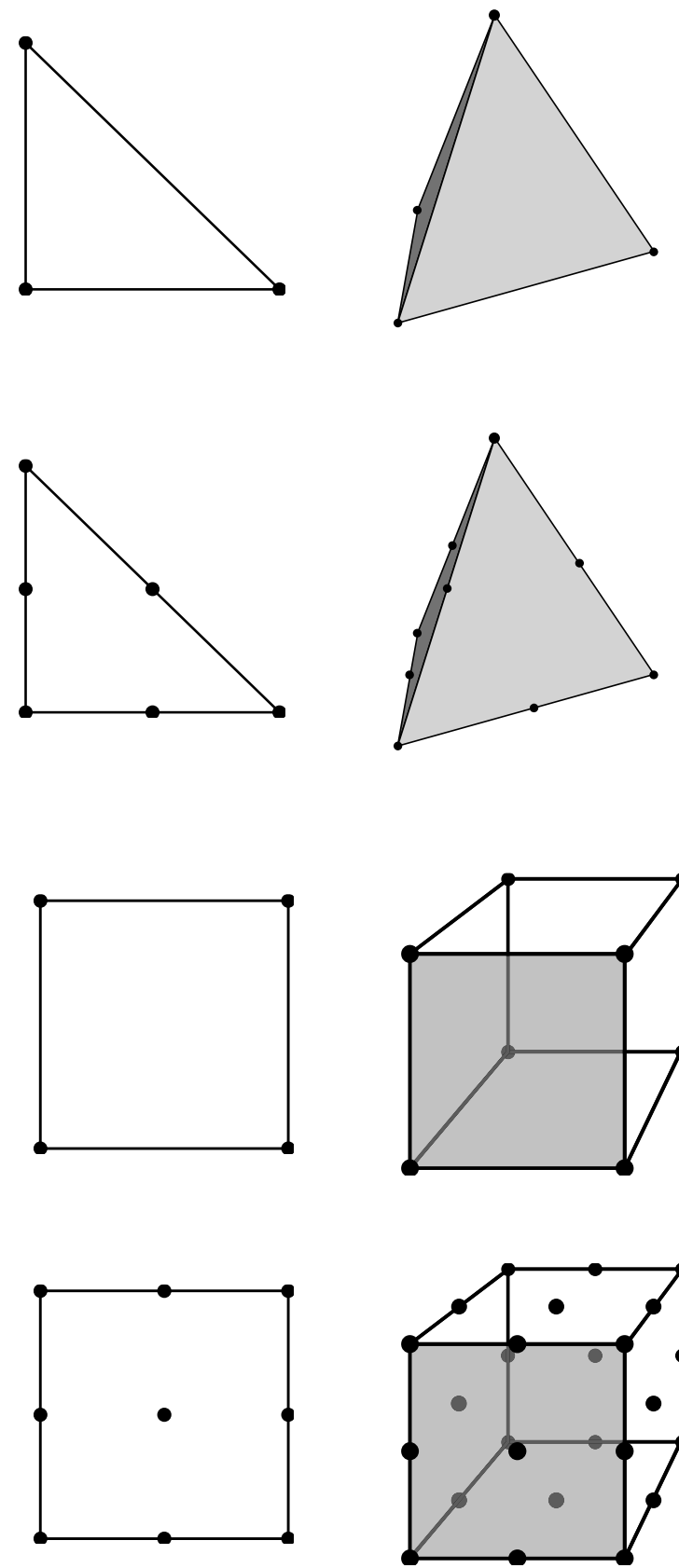
Which discretization provides lower running time for a fixed accuracy?

Can you mesh robustly without any assumption on the input?

Does mesh quality affect the accuracy of the FEM solution?



# Overview



Which discretization provides lower running time for a fixed accuracy?

Can you mesh robustly without any assumption on the input?

Does mesh quality affect the accuracy of the FEM solution?



# Does Quality Matter?

$$\Delta u = f, \quad f = 12x^2$$

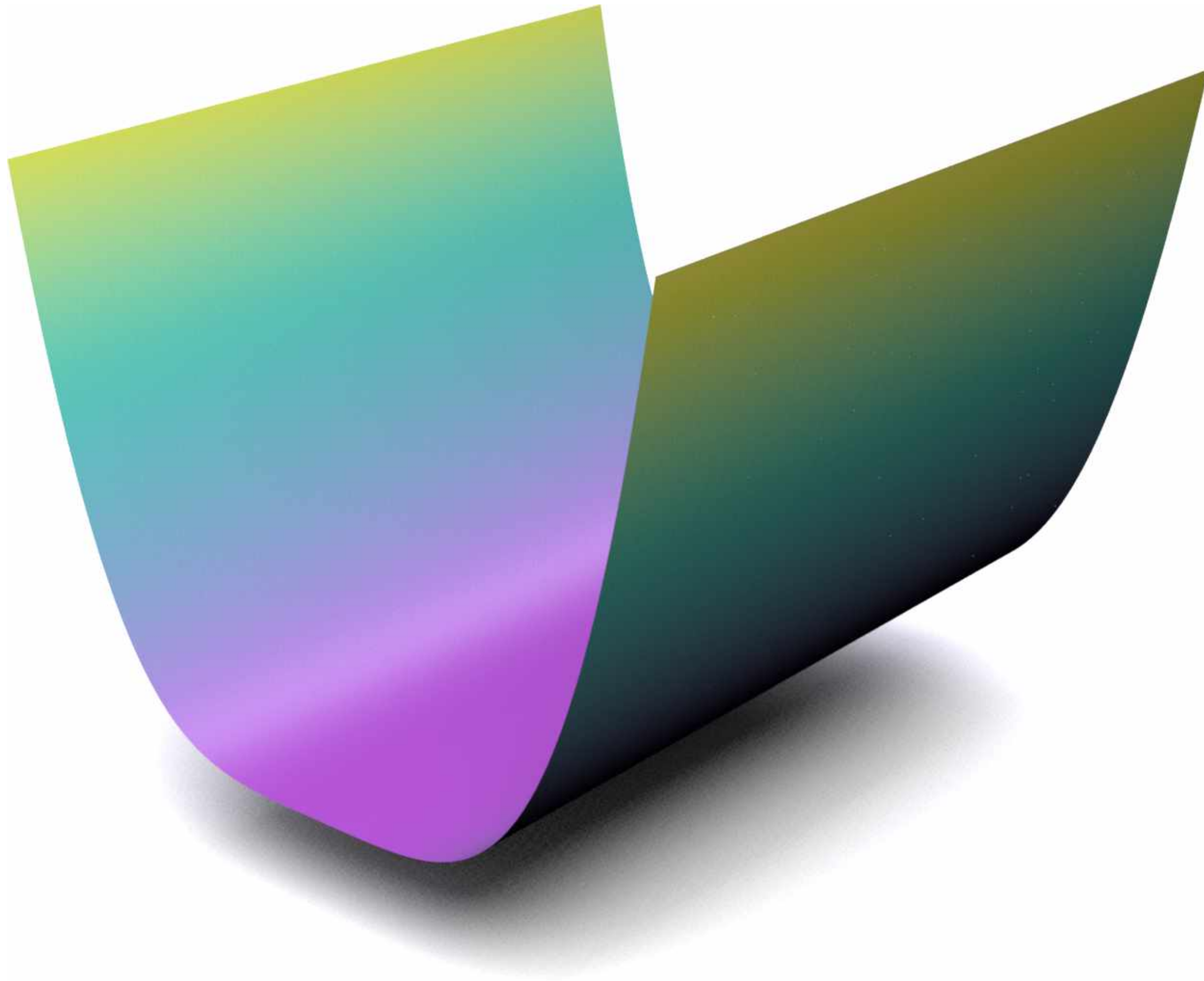


# Does Quality Matter?

$$\Delta u = 12x^2$$

Solution

$$u = x^4$$



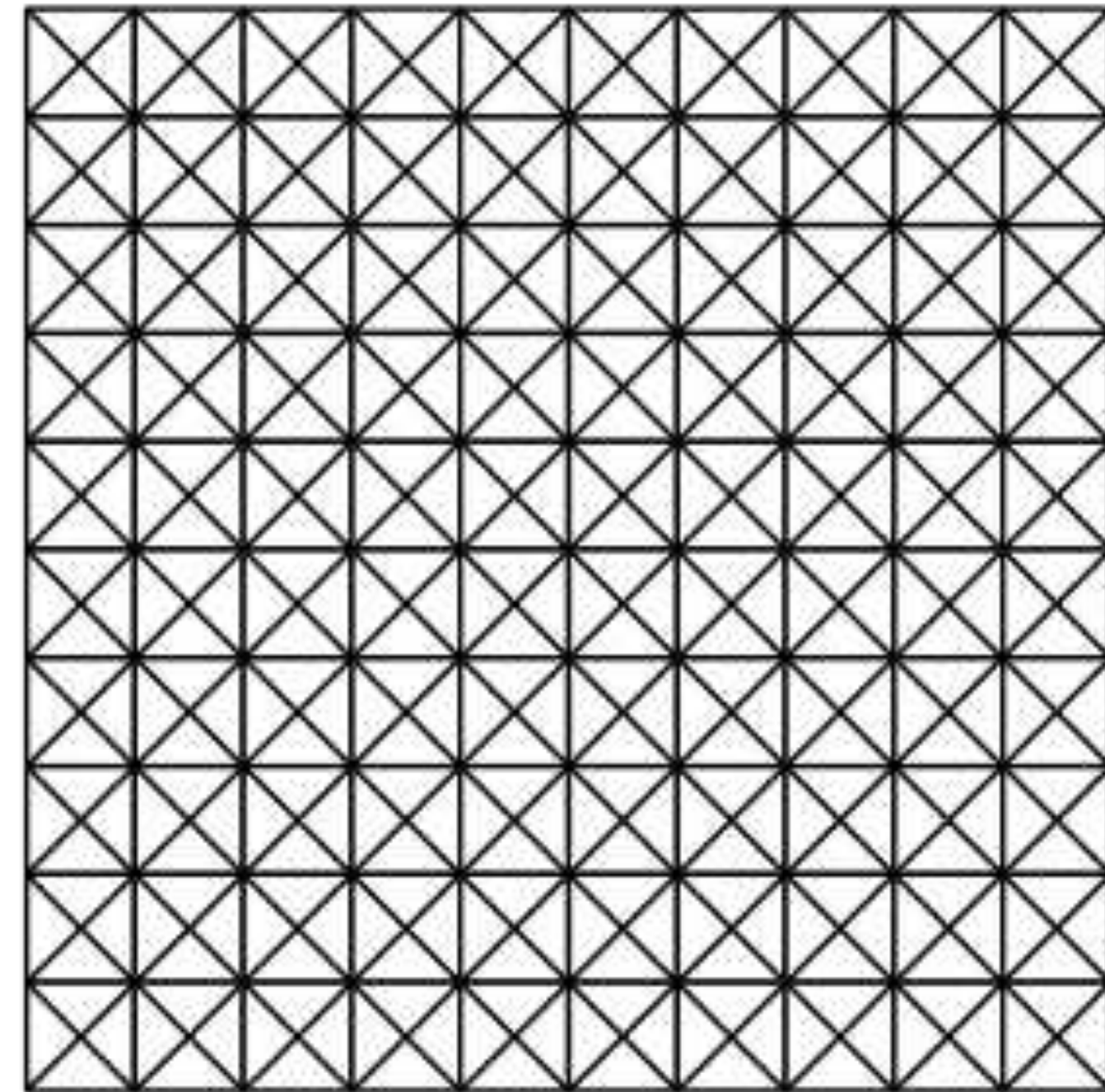
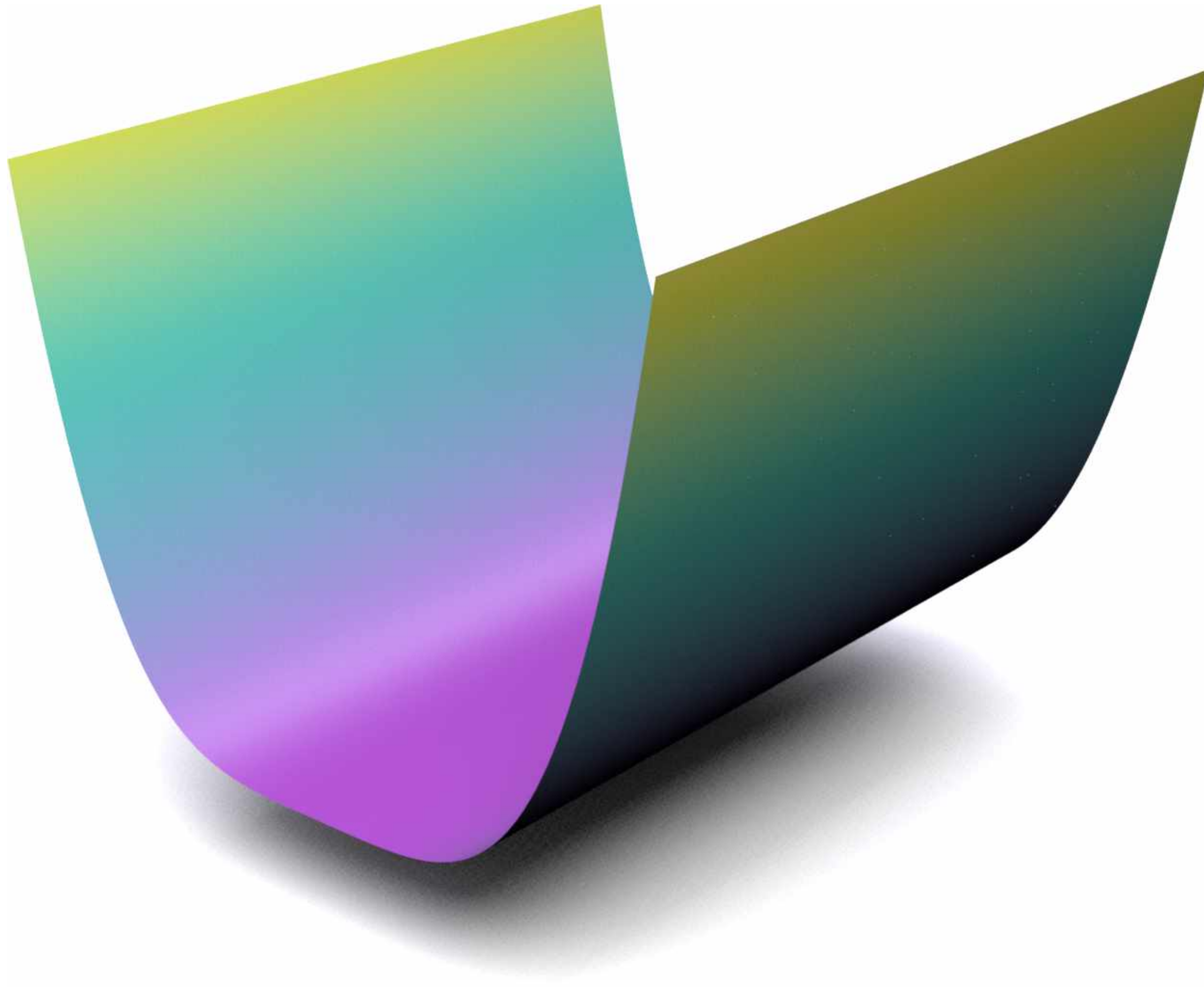


# Does Quality Matter?

$$\Delta u = 12x^2$$

Solution

$$u = x^4$$

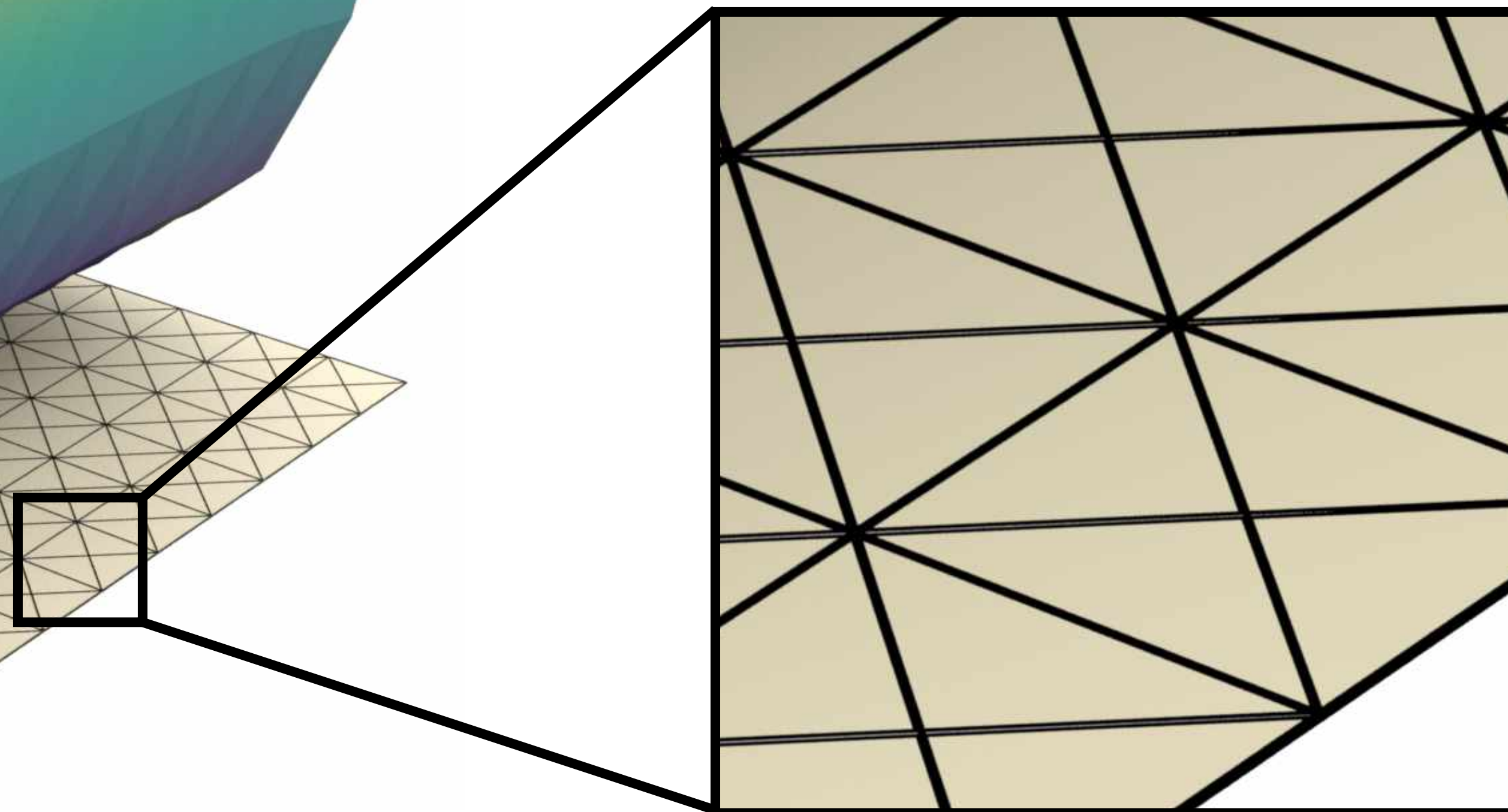
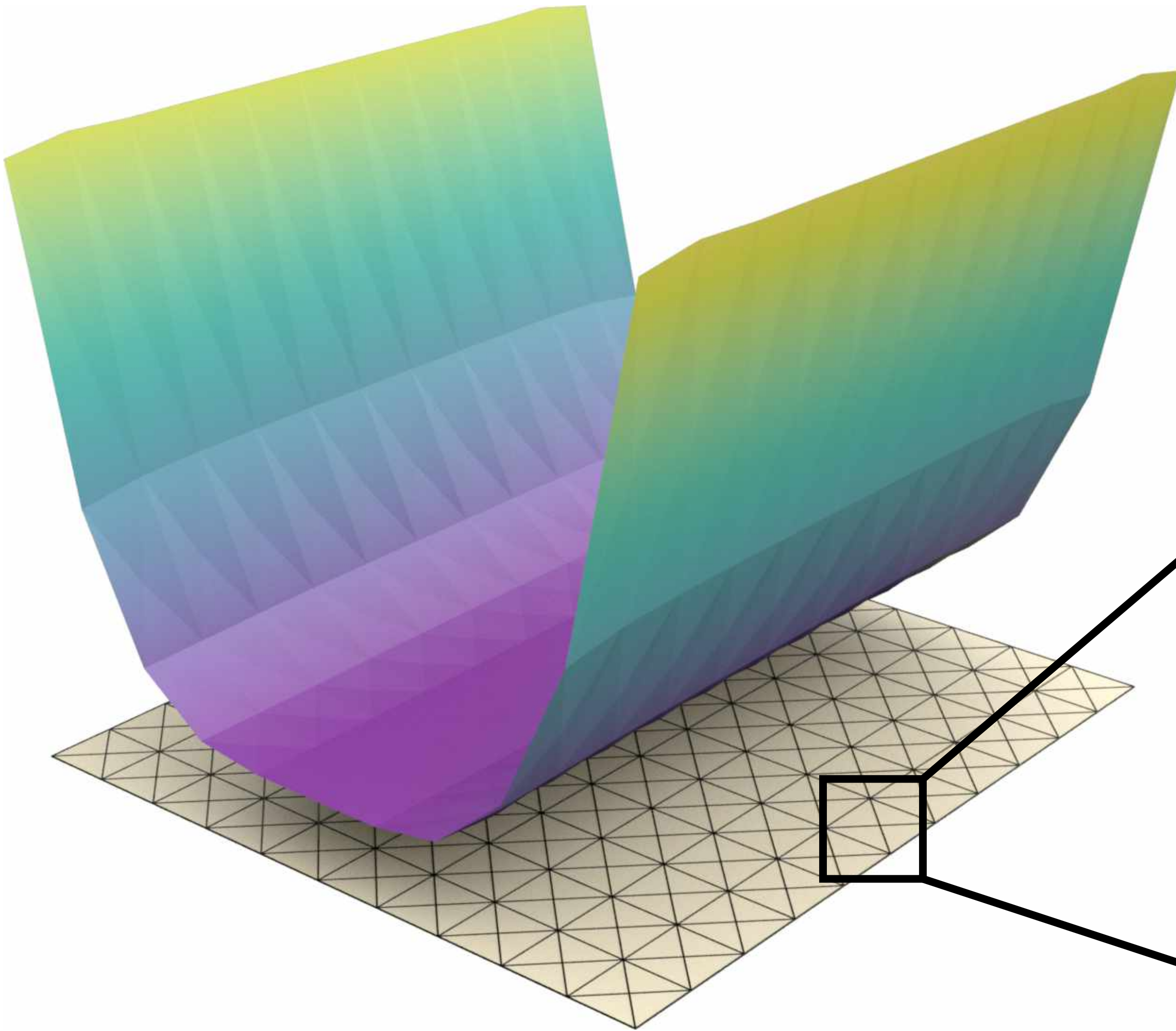
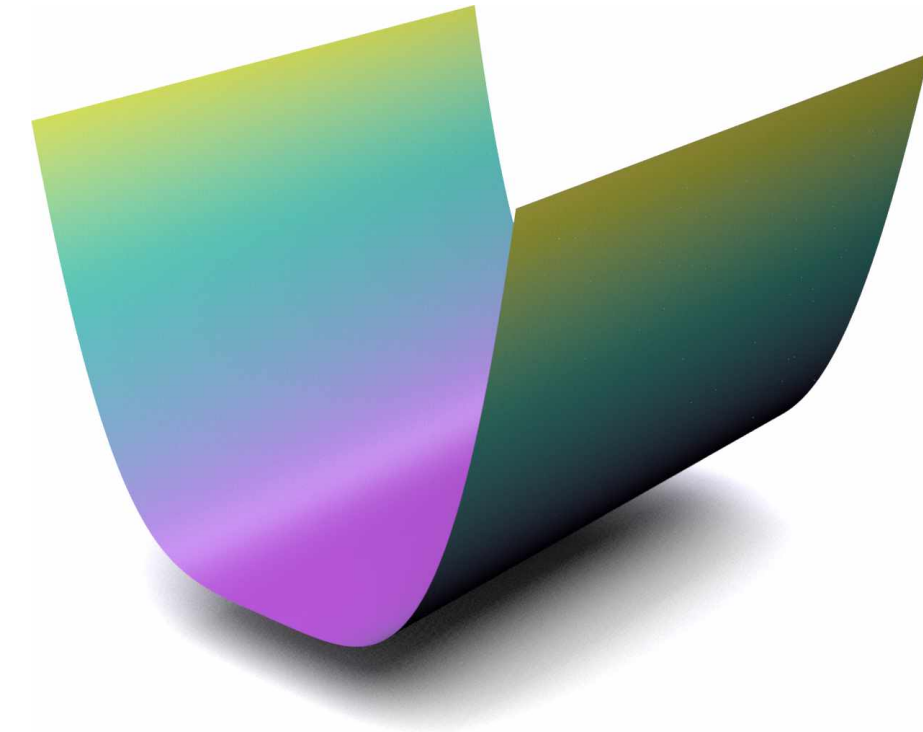




# Does Quality Matter?

$$\Delta u = 12x^2$$

$$u = x^4$$

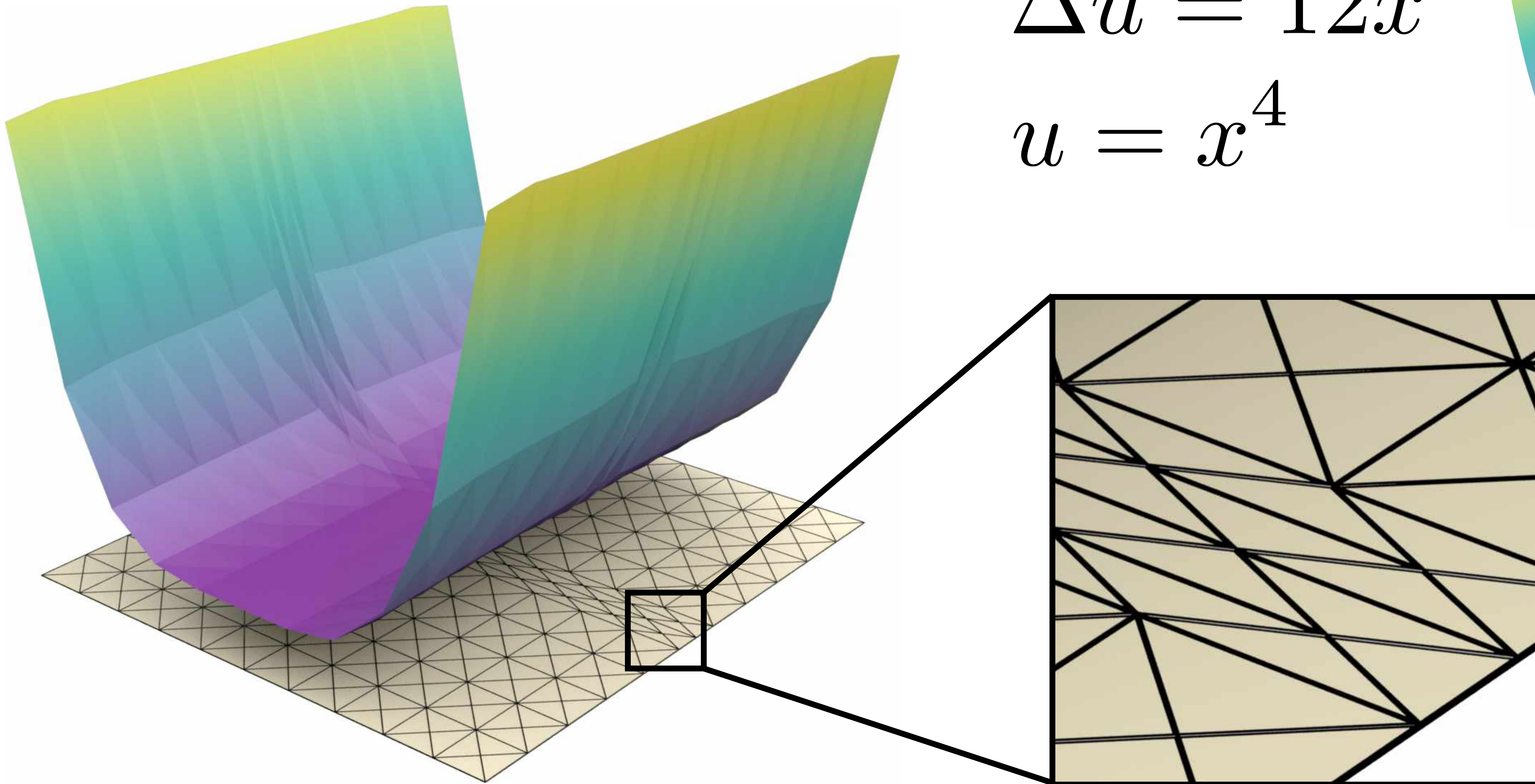
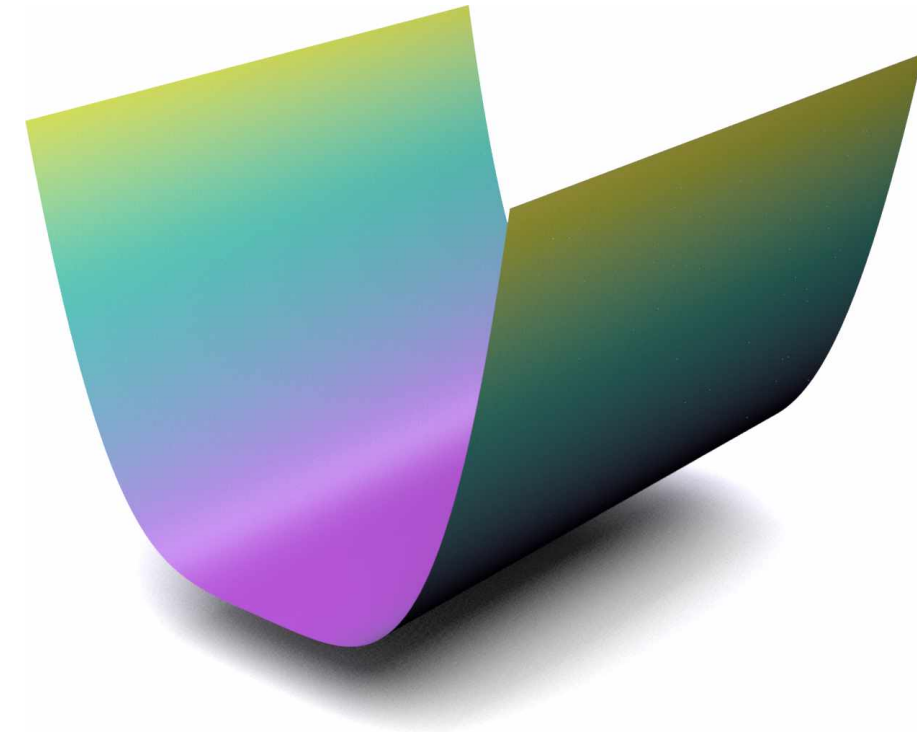




# Does Quality Matter?

$$\Delta u = 12x^2$$

$$u = x^4$$

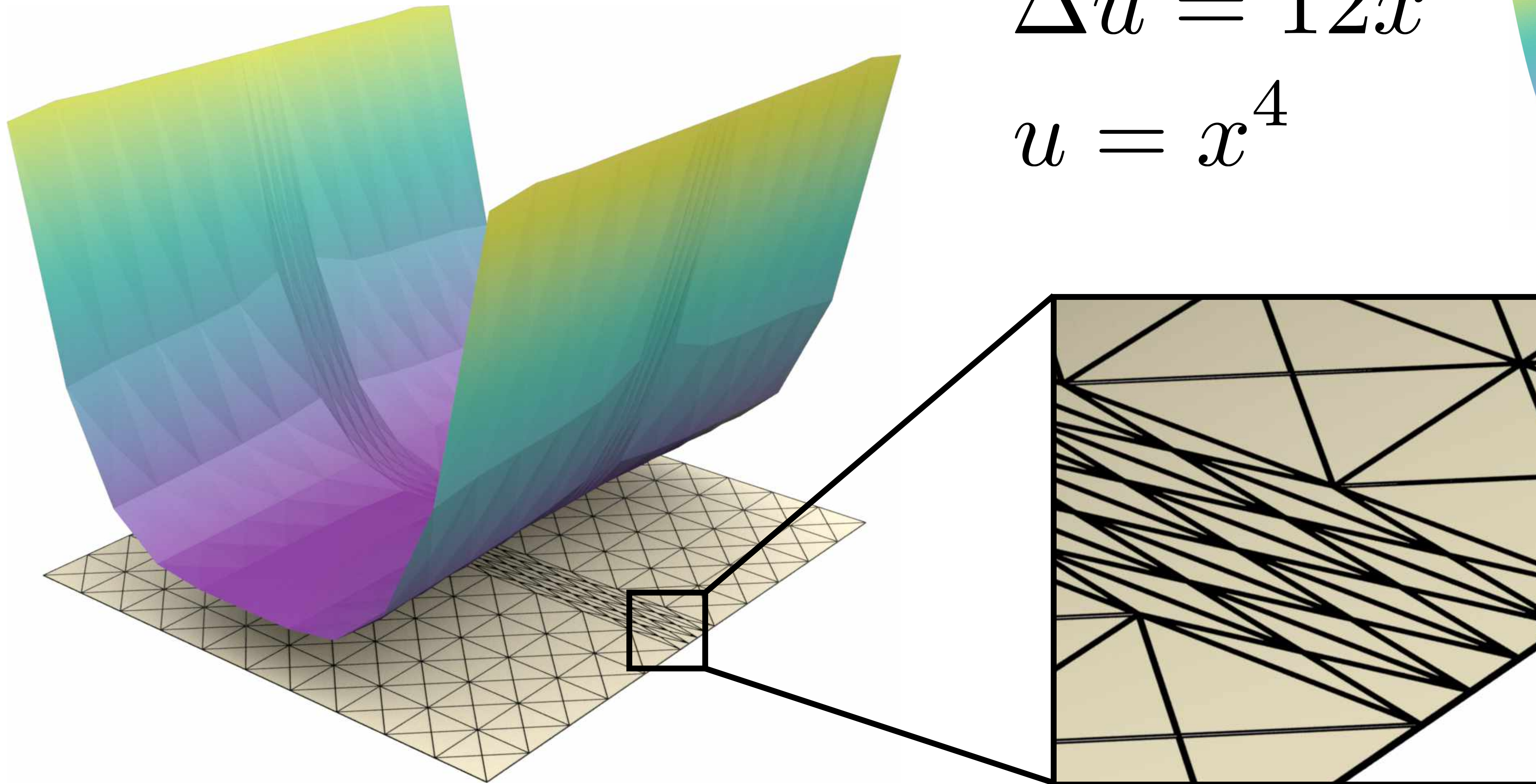
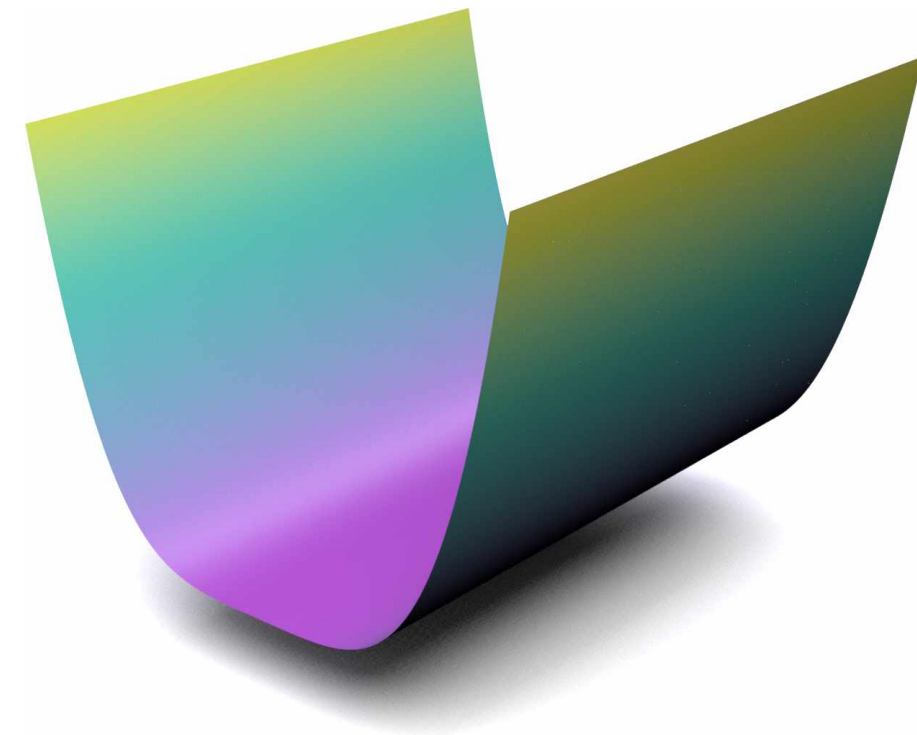




# Does Quality Matter?

$$\Delta u = 12x^2$$

$$u = x^4$$

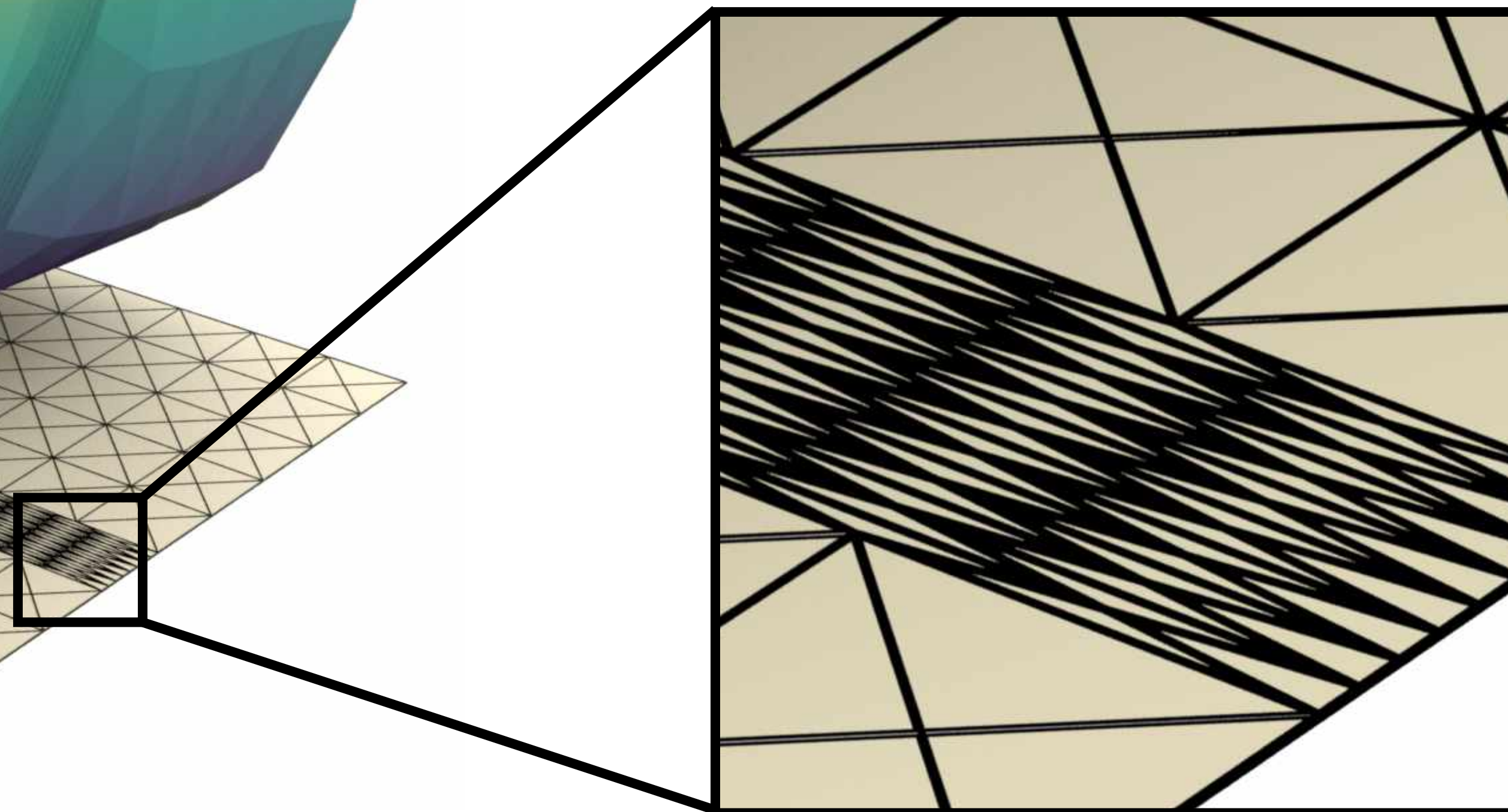
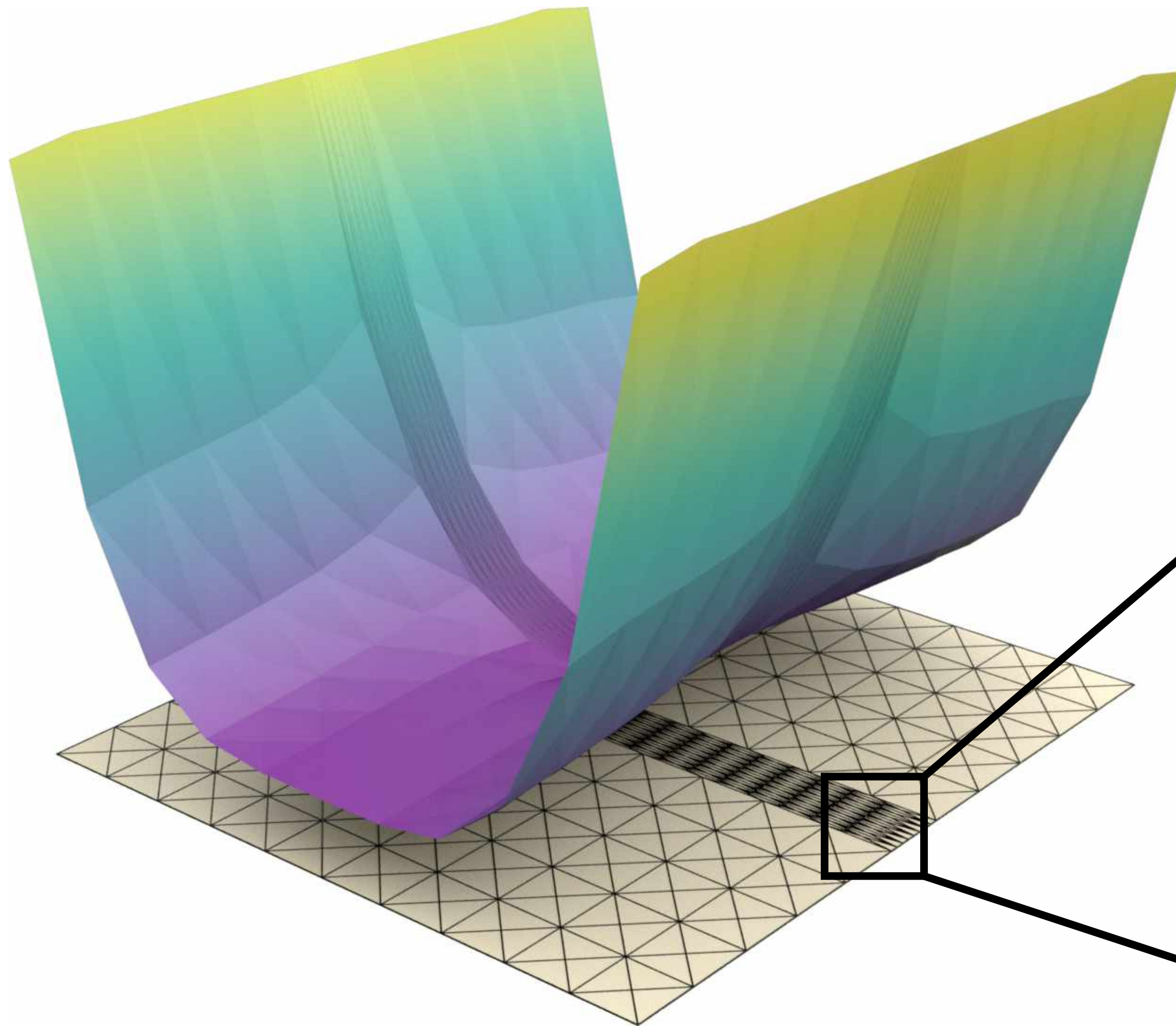
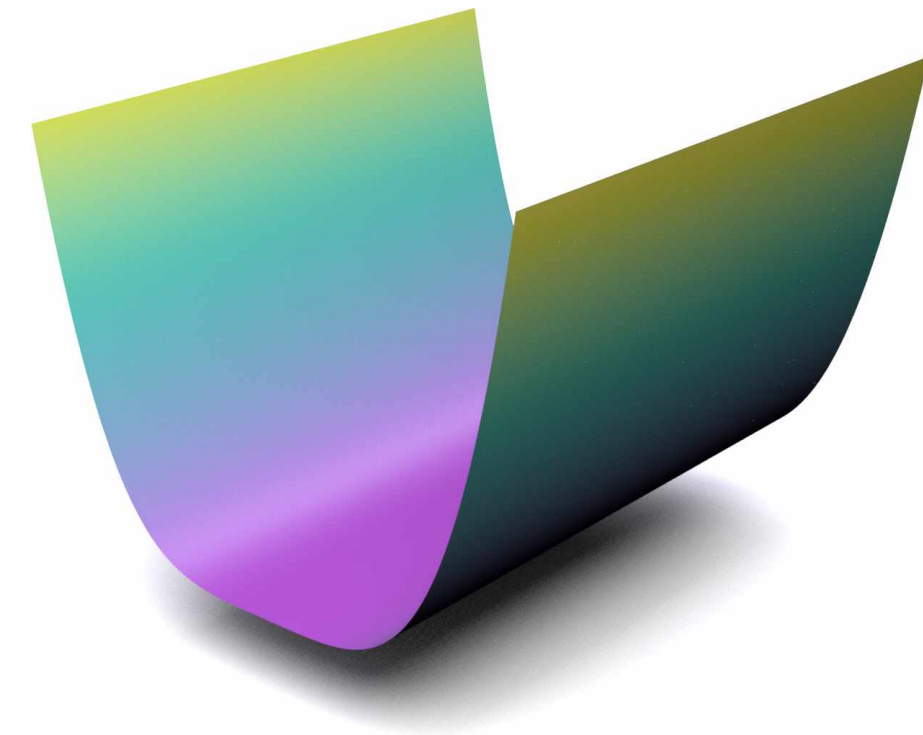




# Does Quality Matter?

$$\Delta u = 12x^2$$

$$u = x^4$$

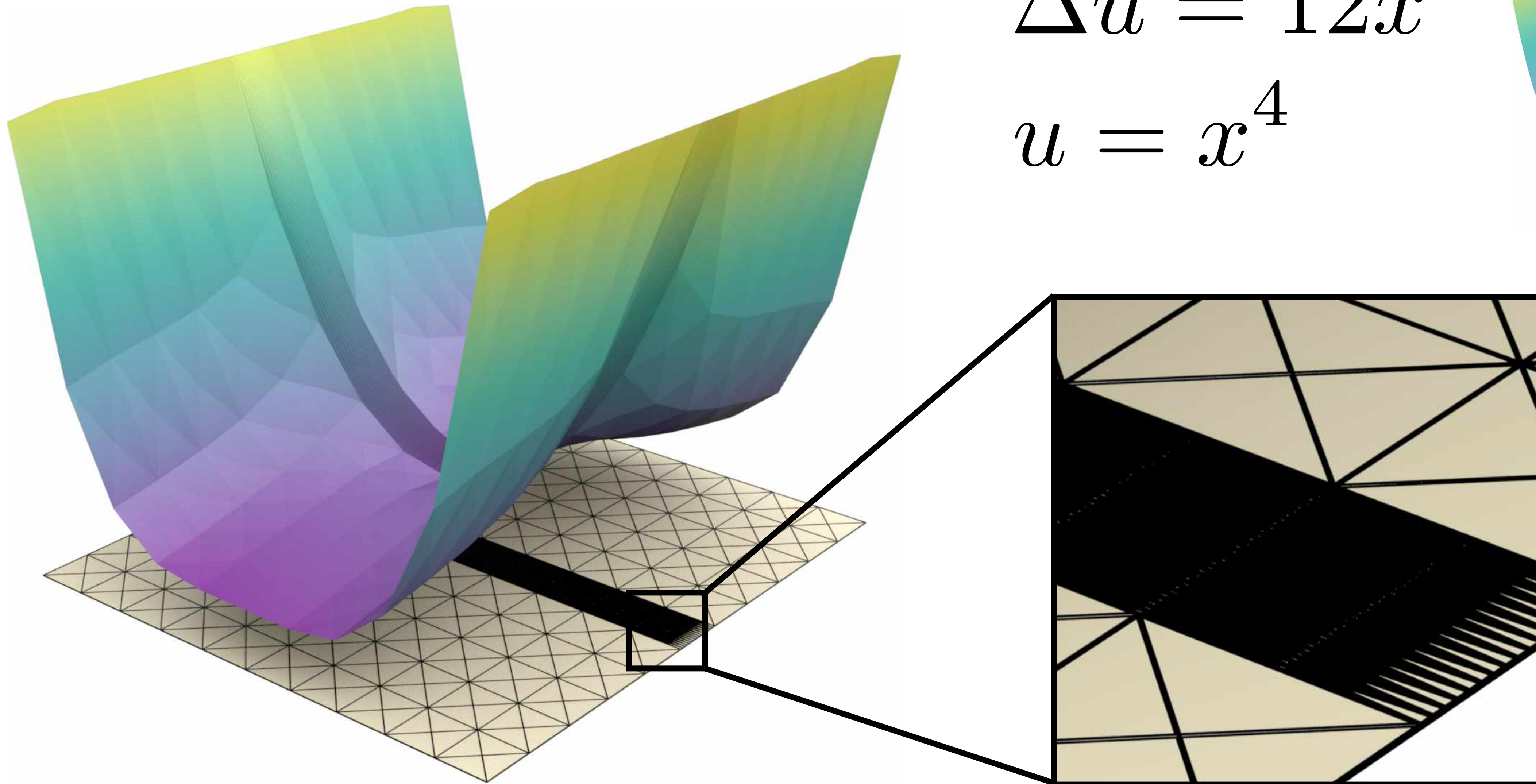
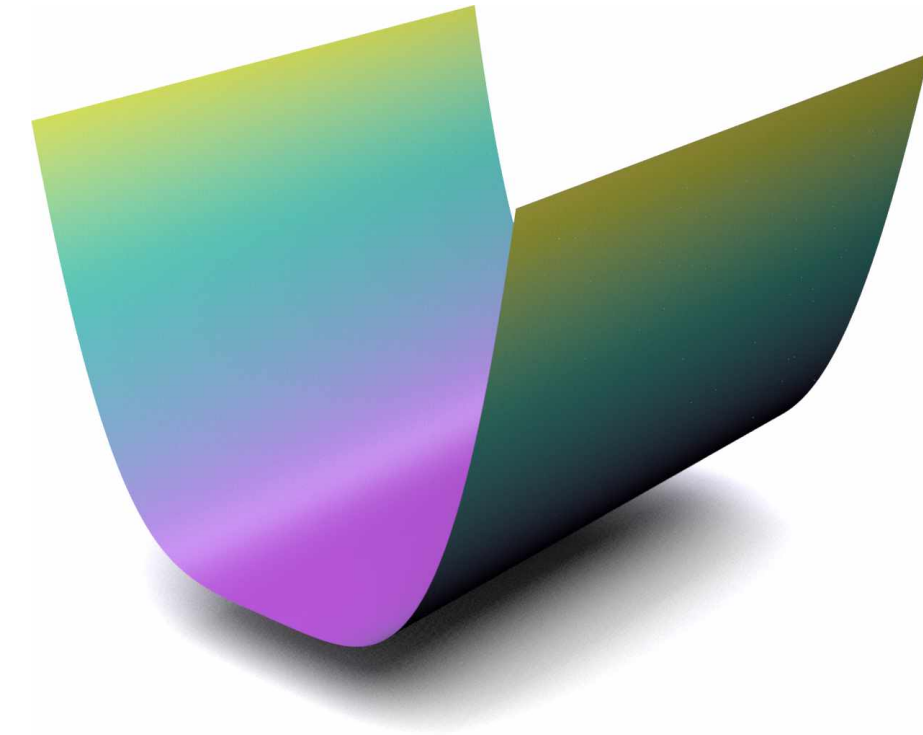




# Does Quality Matter?

$$\Delta u = 12x^2$$

$$u = x^4$$

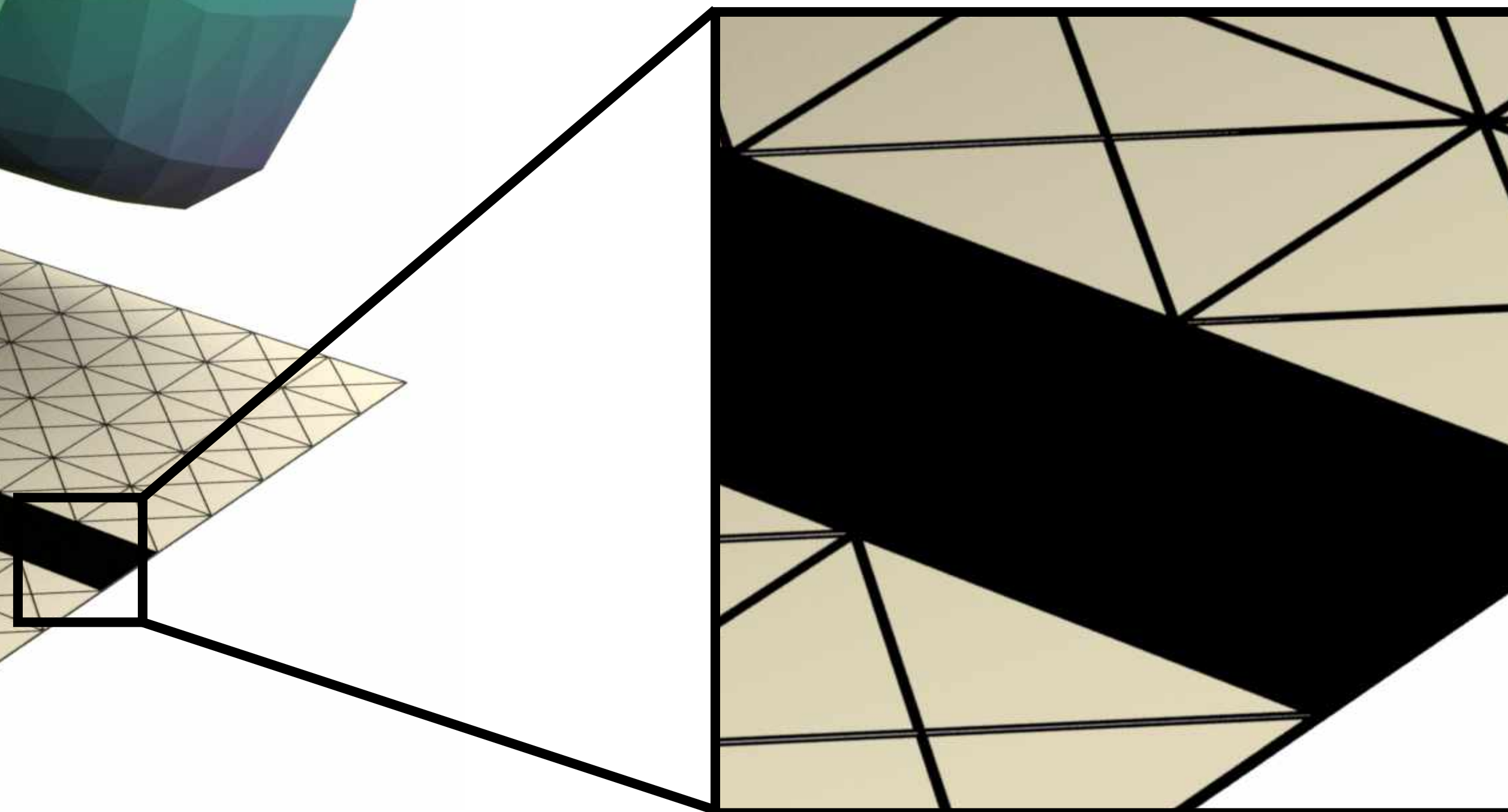
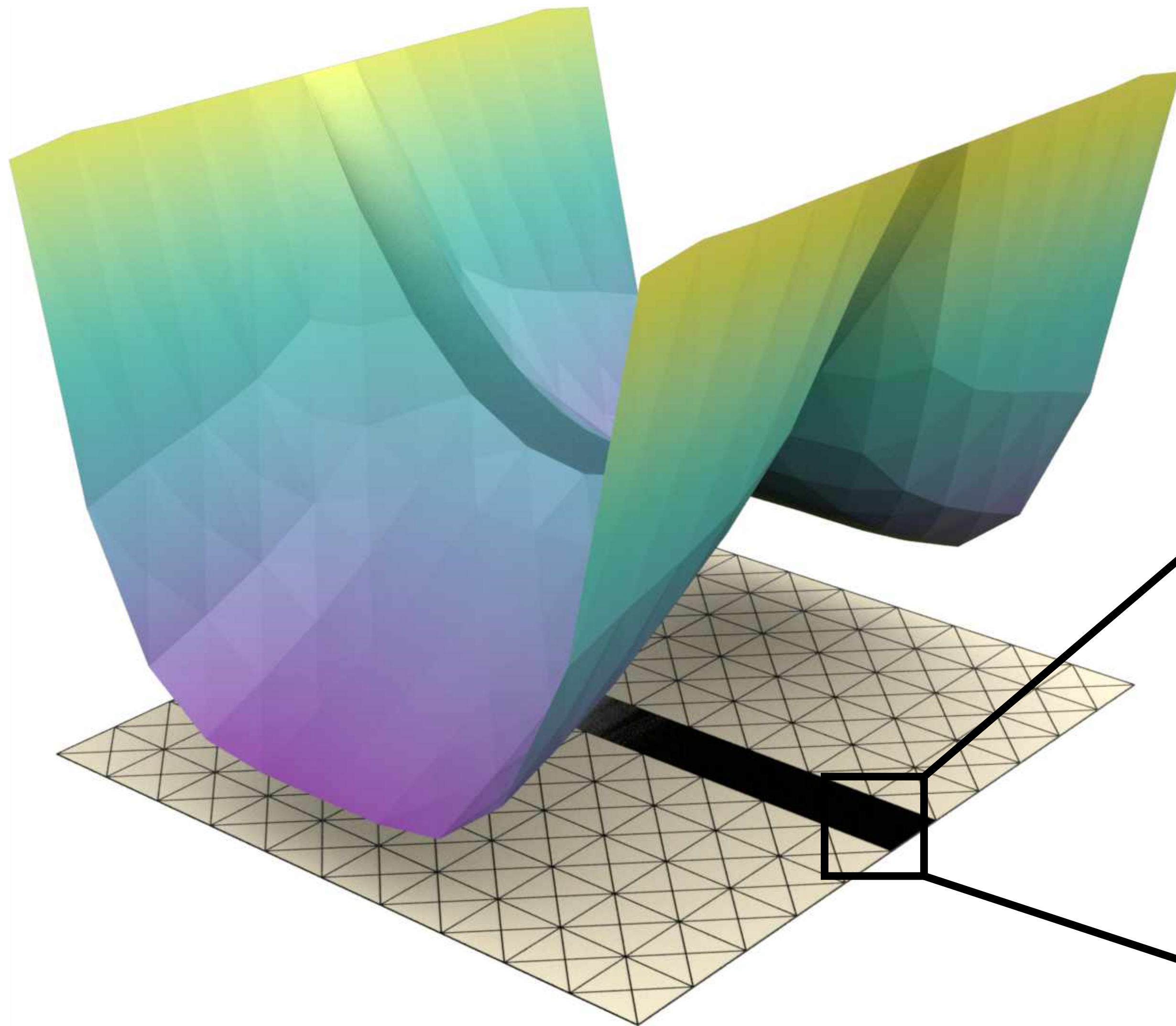
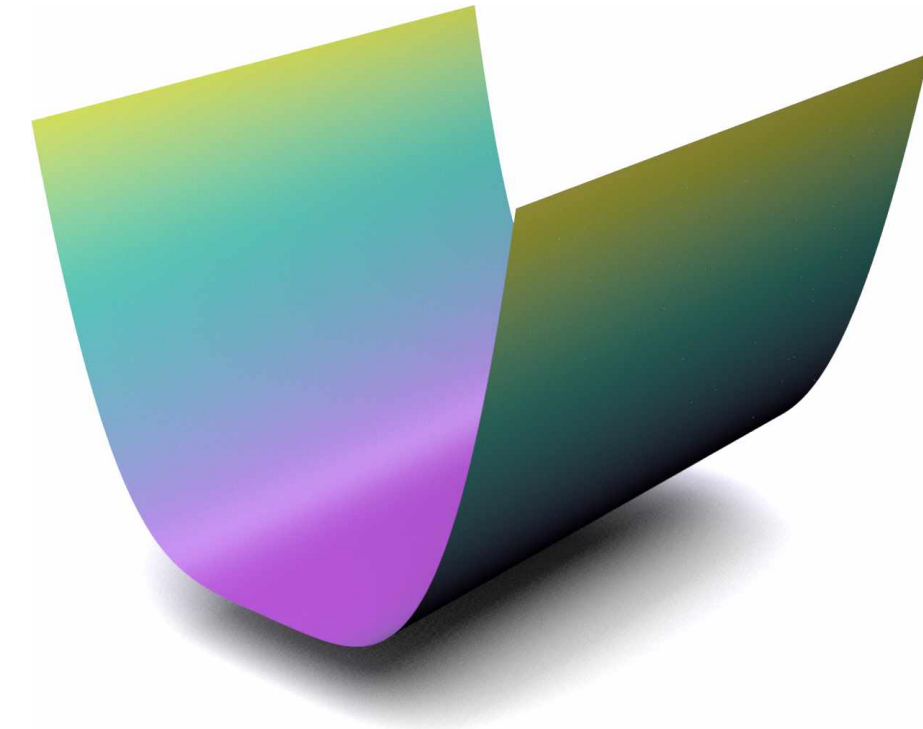




# Does Quality Matter?

$$\Delta u = 12x^2$$

$$u = x^4$$

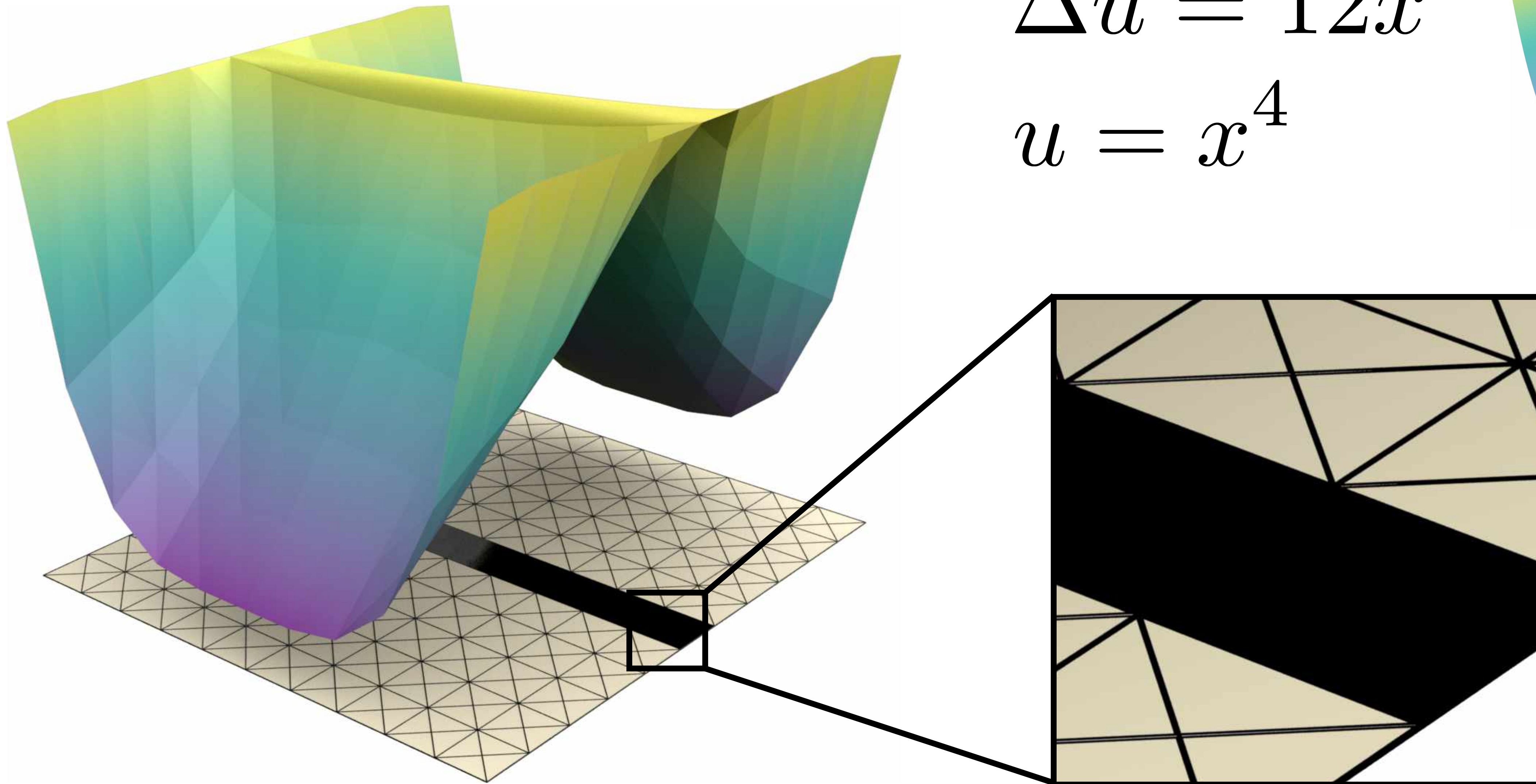
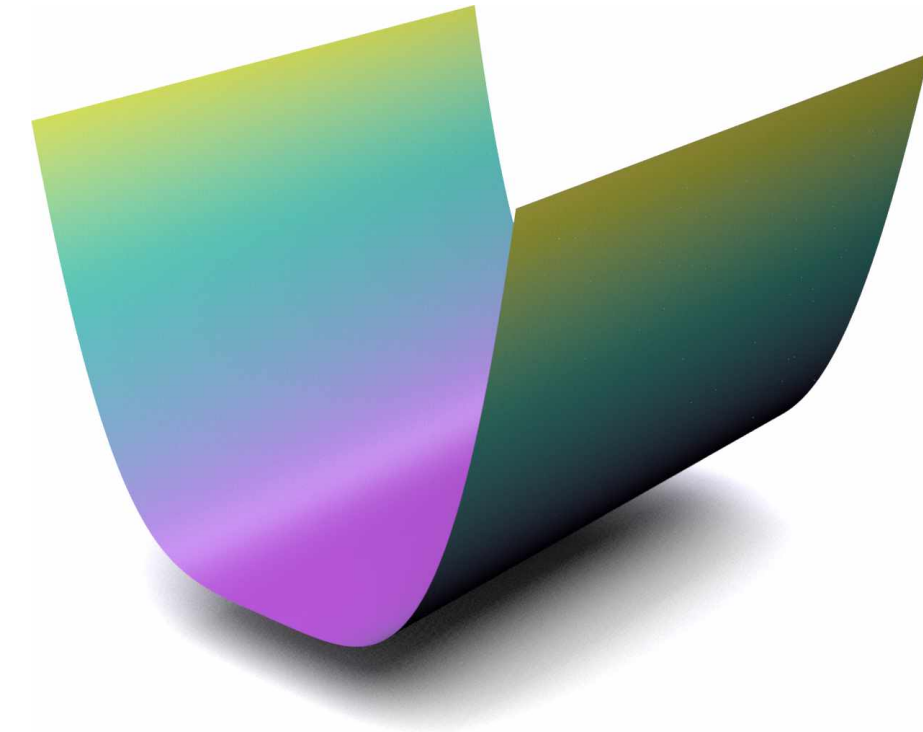




# Does Quality Matter?

$$\Delta u = 12x^2$$

$$u = x^4$$

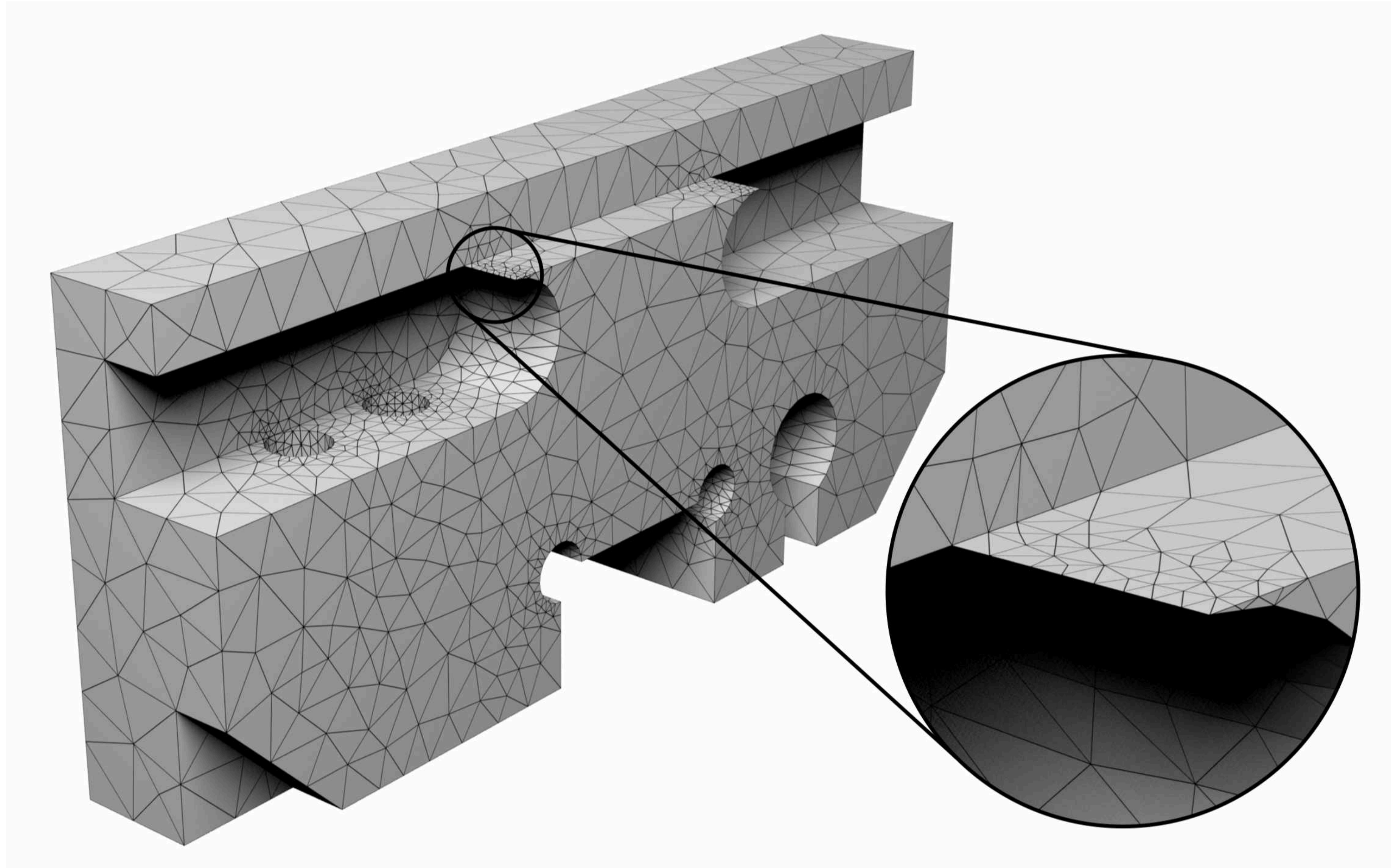




# No Problem, Let's Remesh!



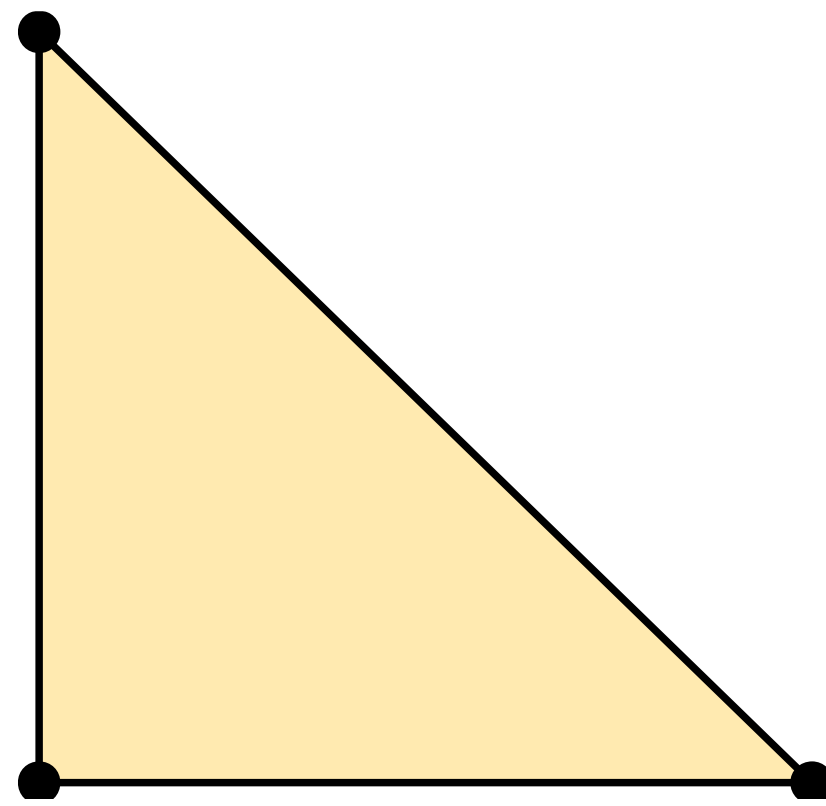
# No Problem, Let's Remesh!



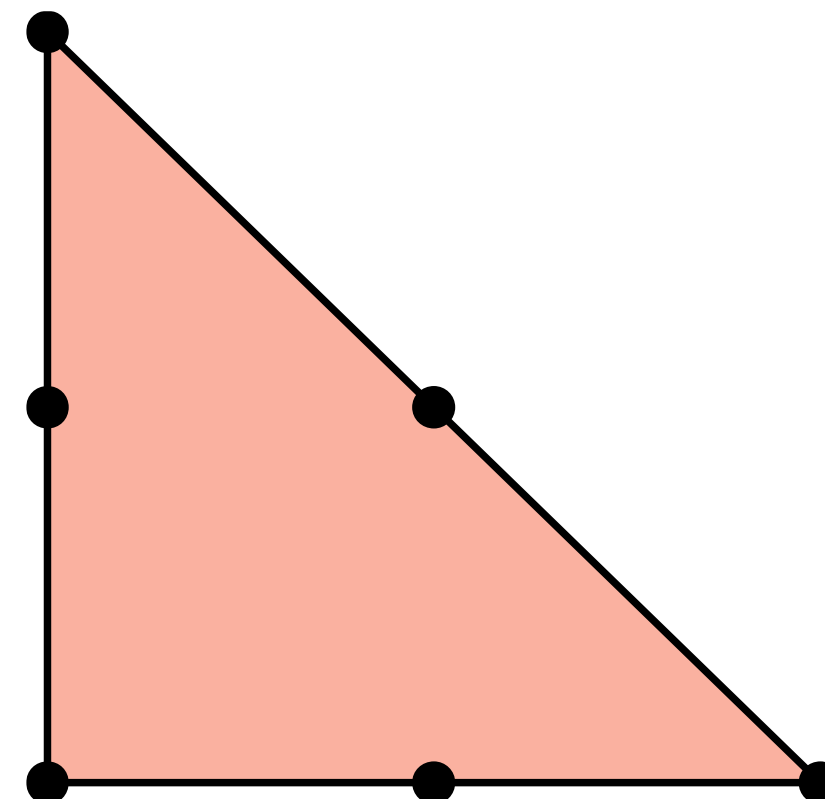


# Our Solution

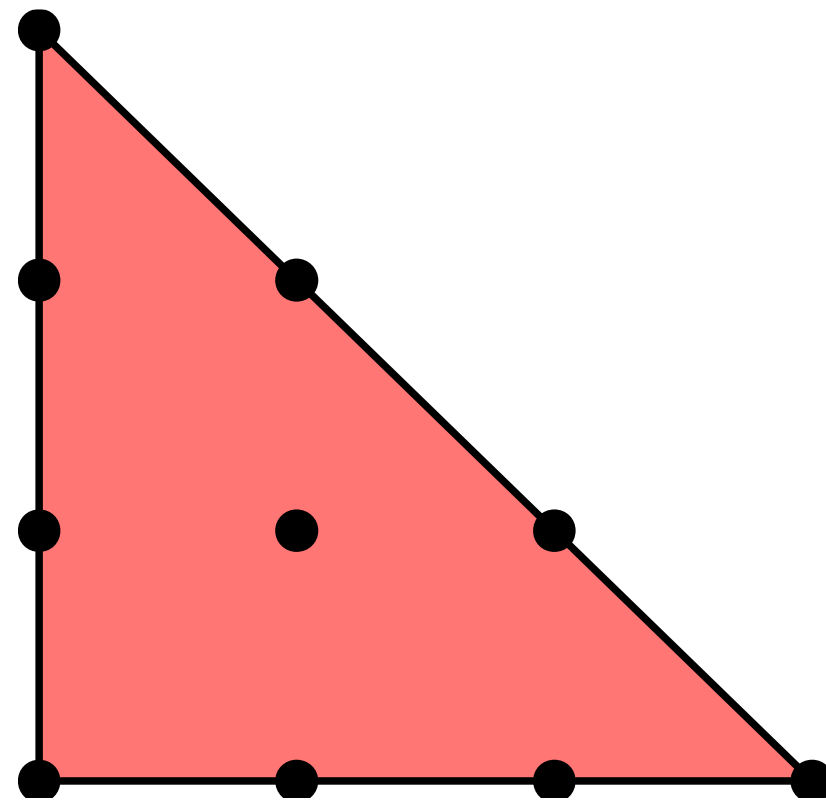
- Locally increase the order of elements



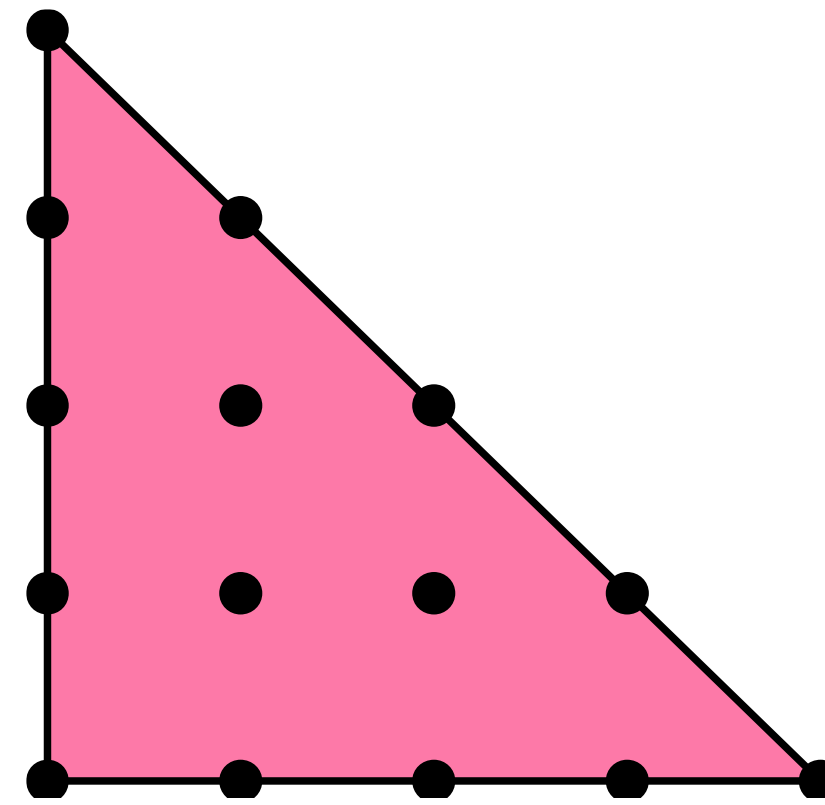
Linear



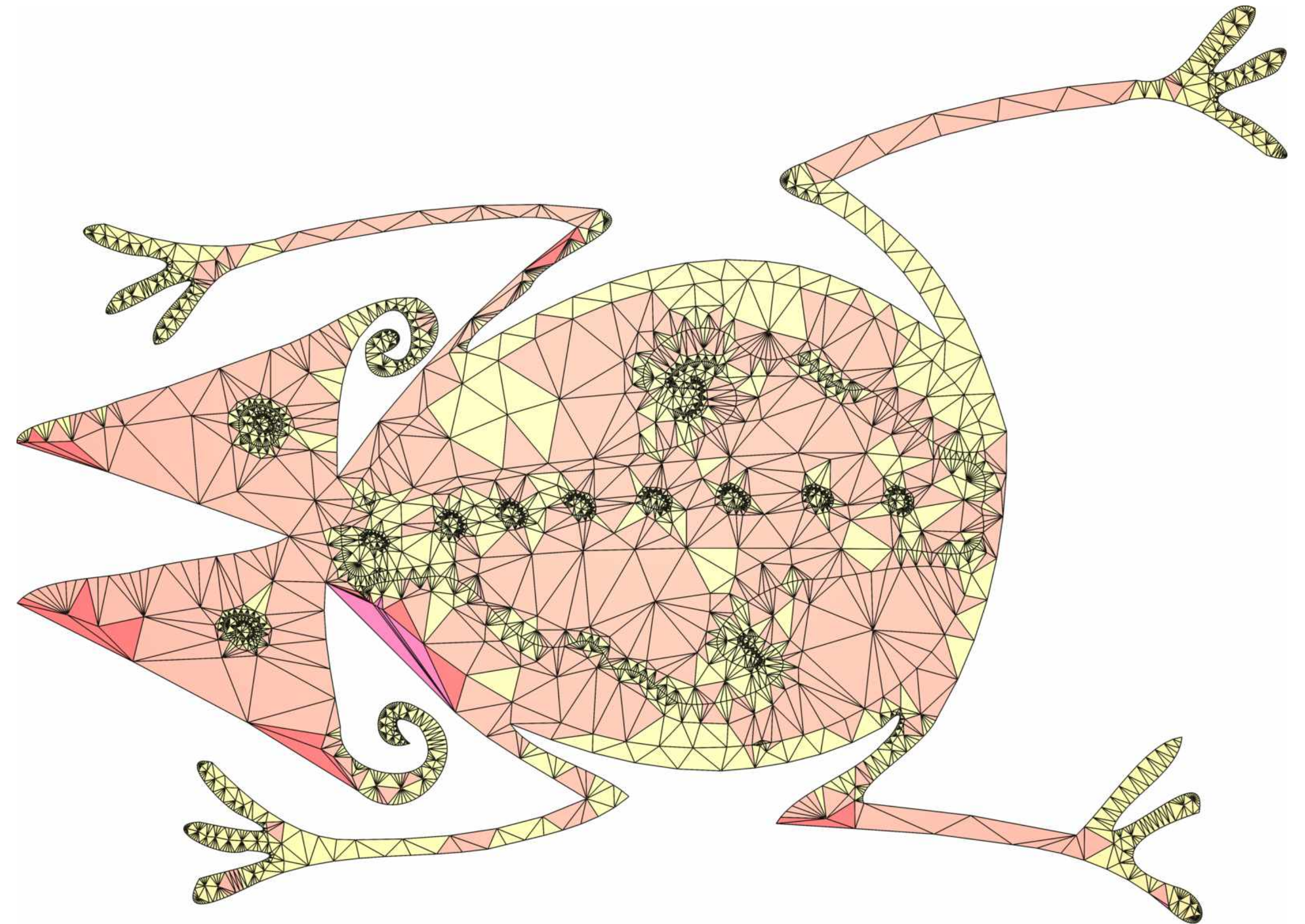
Quadratic



Cubic



Quartic





# Refinement



# Refinement

- A posteriori h-refinement
  - Increase the mesh resolution locally  
[\[Wu 01\]](#), [\[Simnett 09\]](#), [\[Wicke 10\]](#), [\[Pfaff 14\]](#), ...



# Refinement

- A posteriori h-refinement
  - Increase the mesh resolution locally  
[\[Wu 01\]](#), [\[Simnett 09\]](#), [\[Wicke 10\]](#), [\[Pfaff 14\]](#), ...
- A posteriori p-refinement
  - Solve, then increase order where necessary  
[\[Babuška 94\]](#), [\[Kaufmann 13\]](#), [\[Bargteil 14\]](#), [\[Edwards 14\]](#), ...



# Refinement

- A posteriori h-refinement
  - Increase the mesh resolution locally  
[\[Wu 01\]](#), [\[Simnett 09\]](#), [\[Wicke 10\]](#), [\[Pfaff 14\]](#), ...
- A posteriori p-refinement
  - Solve, then increase order where necessary  
[\[Babuška 94\]](#), [\[Kaufmann 13\]](#), [\[Bargteil 14\]](#), [\[Edwards 14\]](#), ...
- Ours is a priori p-refinement
  - We increase order only based on the input



# Overview

$$k = \frac{\ln \left( B \hat{h}^{\hat{k}+1} \frac{\sigma_E^2}{\hat{\sigma}^2} \right) - \ln h_E}{\ln h_E}$$

1. Use formula



# Magic Formula

Order of an element

$$\boxed{k} = \frac{\ln \left( B \hat{h}^{\hat{k}+1} \frac{\sigma_E^2}{\hat{\sigma}^2} \right) - \ln h_E}{\ln h_E}$$



# Magic Formula

User parameter, = 3

$$k = \frac{\ln \left( B \hat{h}^{\hat{k}+1} \frac{\sigma_E^2}{\hat{\sigma}^2} \right) - \ln h_E}{\ln h_E}$$



# Magic Formula

Average edge length

$$k = \frac{\ln \left( B \hat{h}^{\hat{k}+1} \frac{\sigma_E^2}{\hat{\sigma}^2} \right) - \ln h_E}{\ln h_E}$$



# Magic Formula

Base order, usually 1

$$k = \frac{\ln \left( B \hat{h}^{\hat{k}+1} \frac{\sigma_E^2}{\hat{\sigma}^2} \right) - \ln h_E}{\ln h_E}$$



# Magic Formula

$$k = \frac{\ln \left( B \hat{h}^{\hat{k}+1} \frac{\sigma_E^2}{\hat{\sigma}^2} \right) - \ln h_E}{\ln h_E}$$

$$\hat{\sigma}_{2D} = \sqrt{3}/6$$

$$\hat{\sigma}_{3D} = \sqrt{6}/12$$



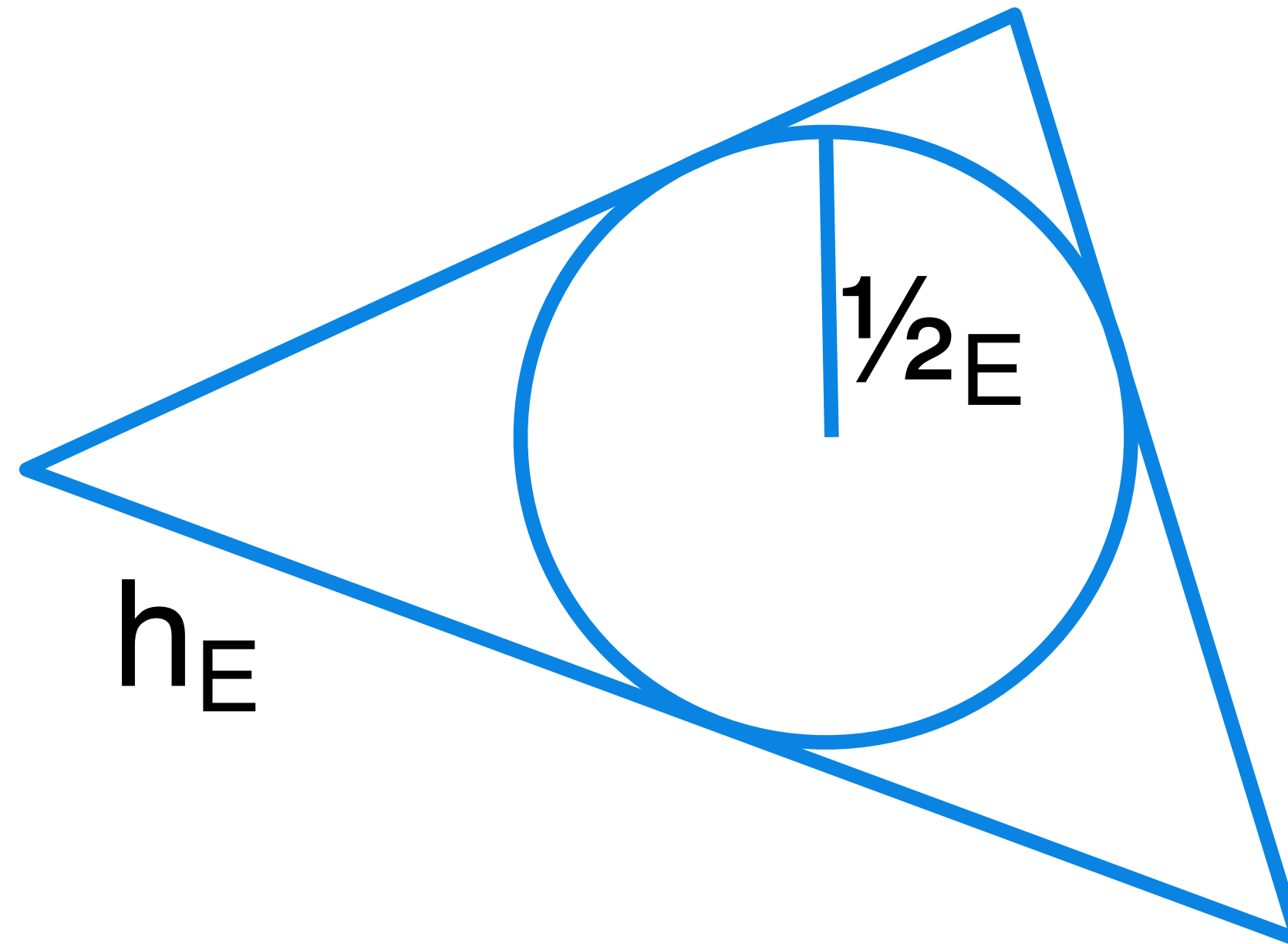
# Magic Formula

$$k = \frac{\ln \left( B \hat{h}^{\hat{k}+1} \frac{\sigma_E^2}{\hat{\sigma}^2} \right) - \ln h_E}{\ln h_E}$$

$$\hat{\sigma}_{2D} = \sqrt{3}/6$$

$$\hat{\sigma}_{3D} = \sqrt{6}/12$$

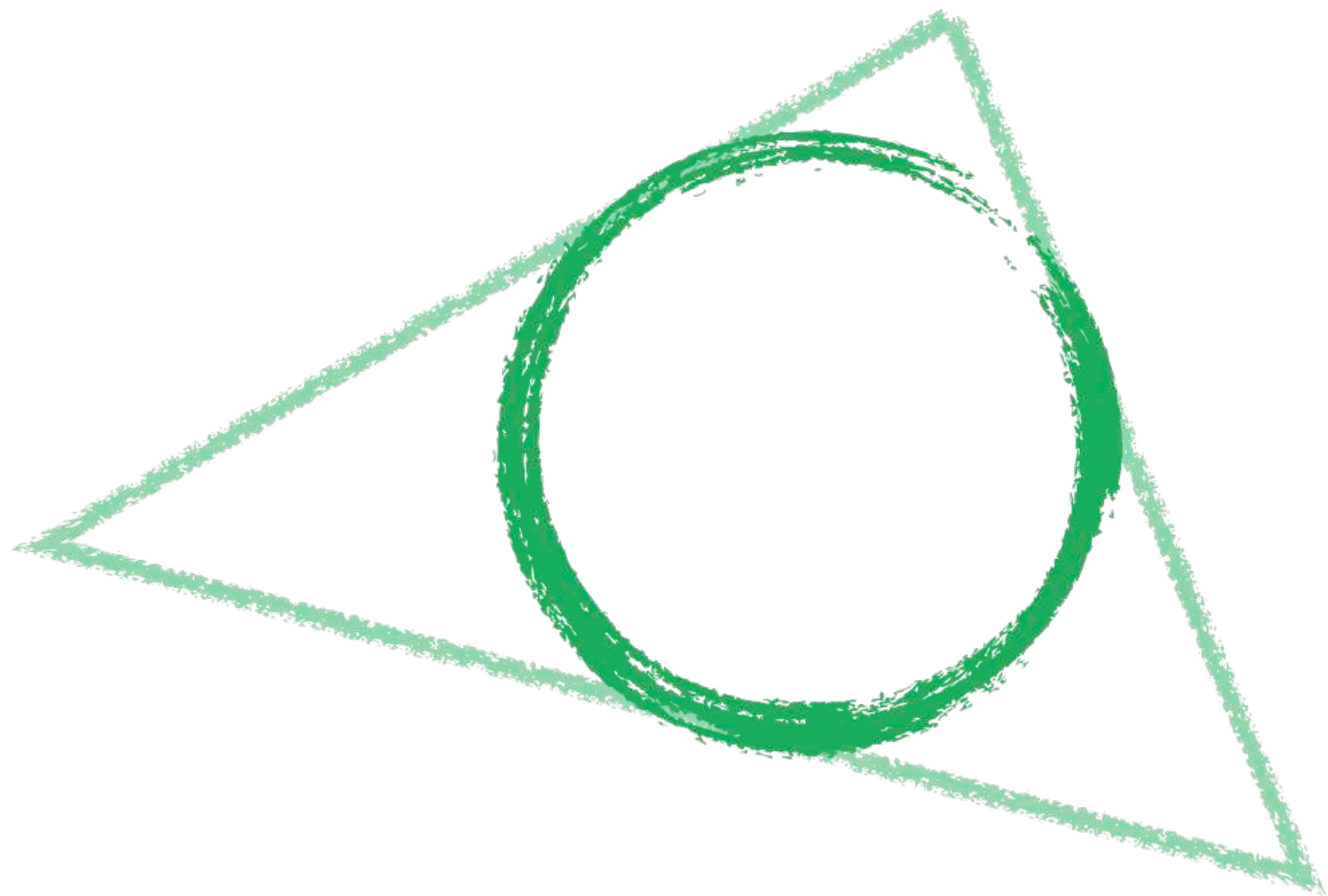
$$\sigma_E = \frac{\rho_E}{h_E}$$





# Magic Formula

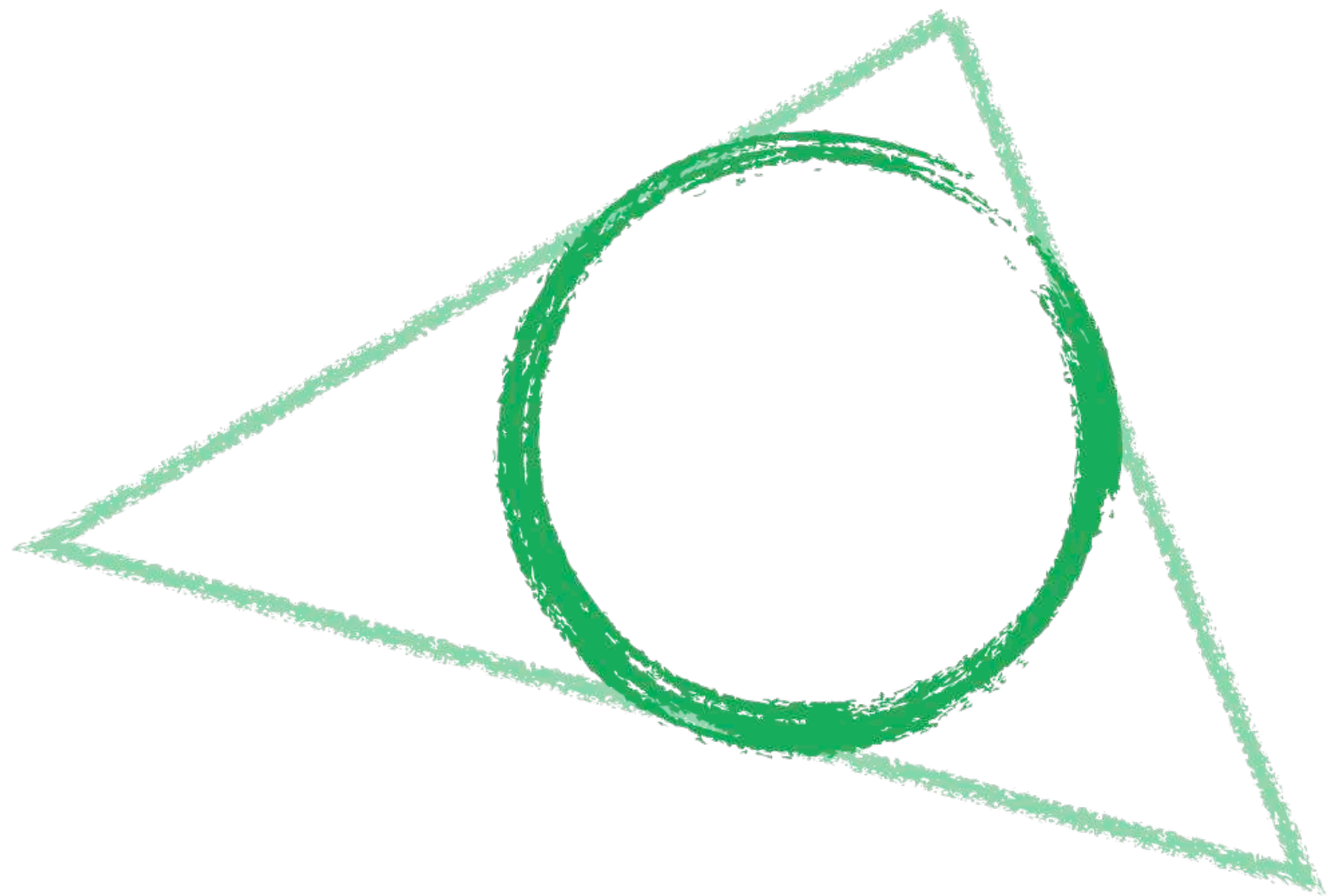
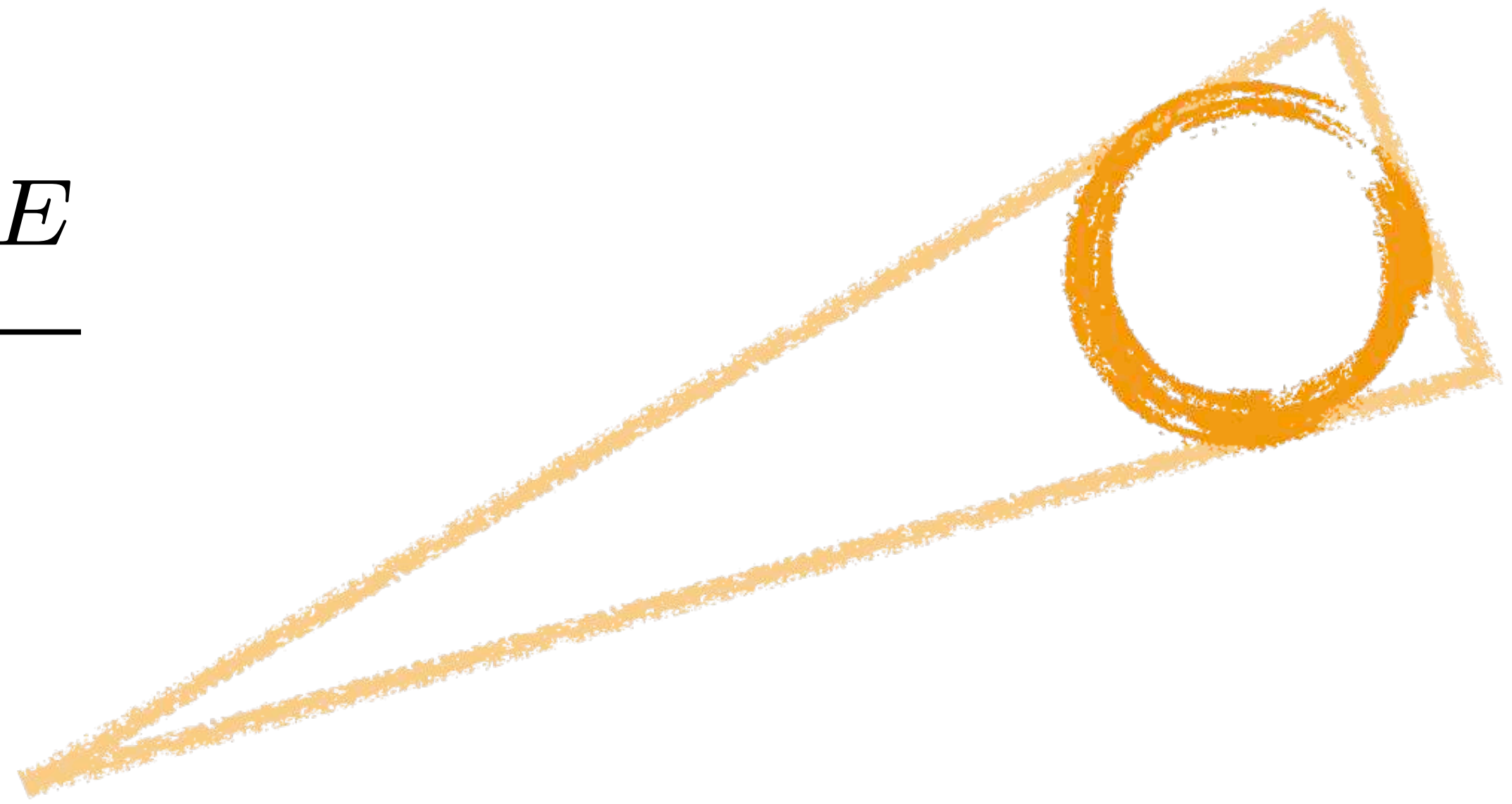
$$k = \frac{\ln \left( B \hat{h}^{\hat{k}+1} \frac{\sigma_E^2}{\hat{\sigma}^2} \right) - \ln h_E}{\ln h_E}$$





# Magic Formula

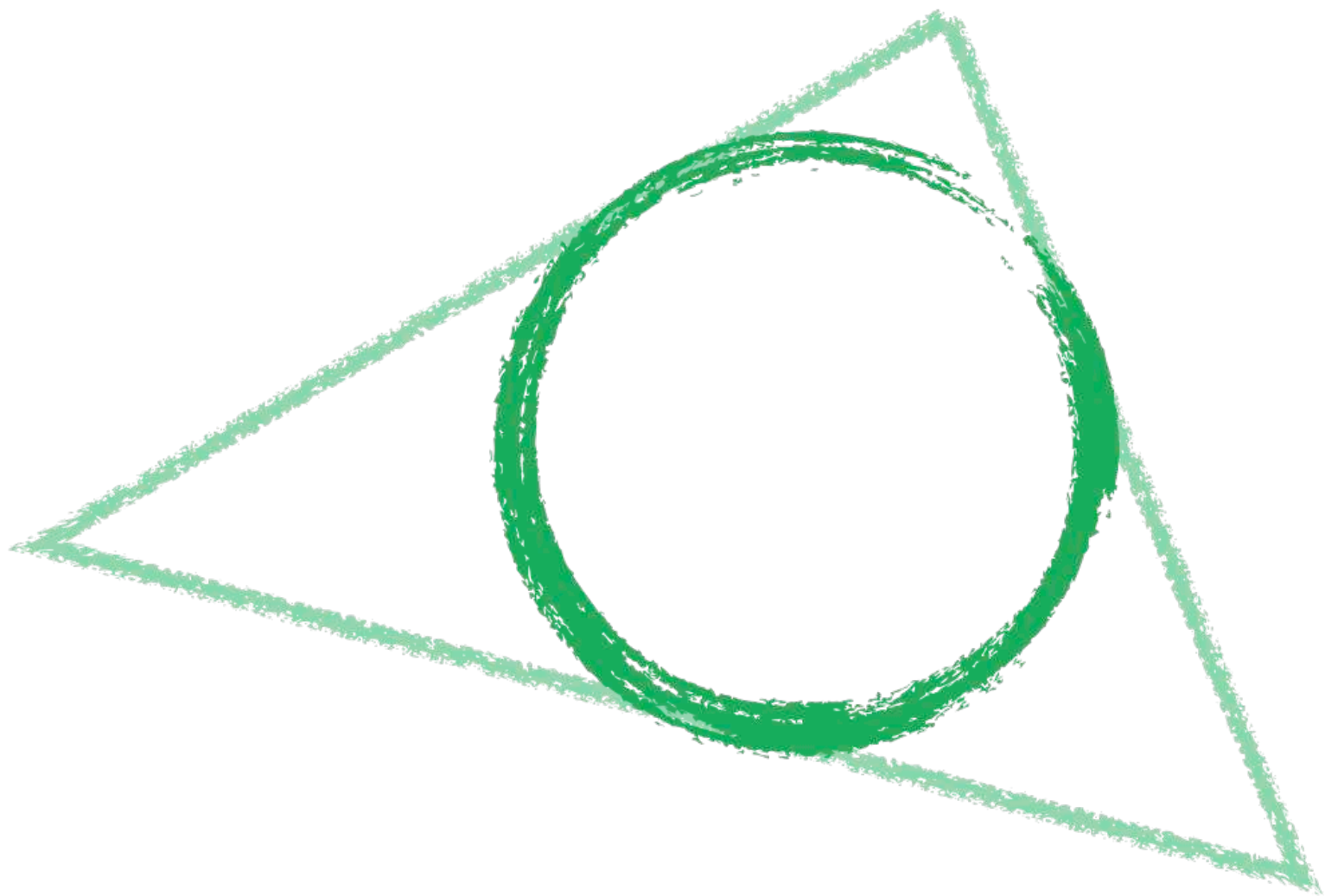
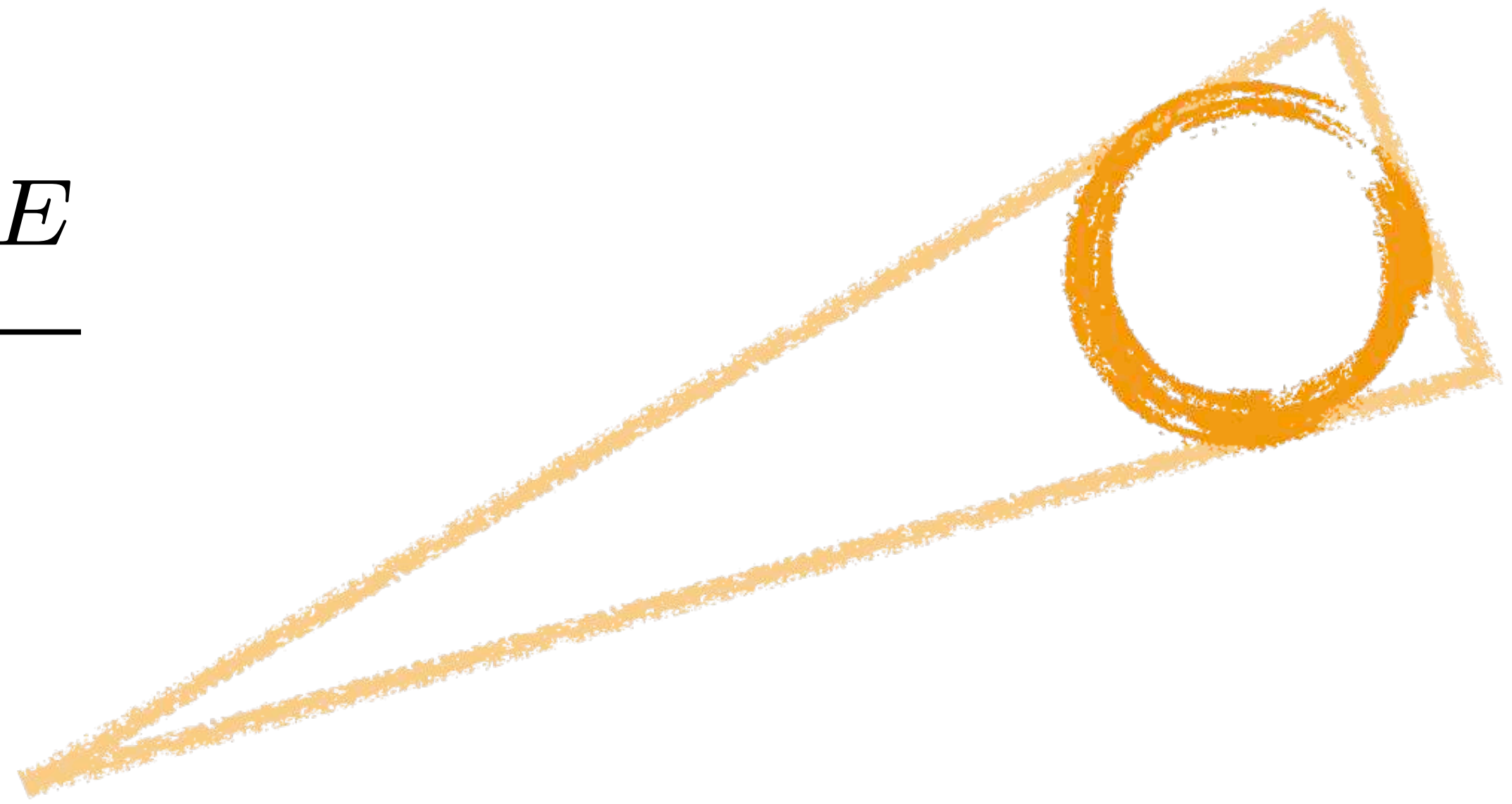
$$k = \frac{\ln \left( B \hat{h}^{\hat{k}+1} \frac{\sigma_E^2}{\hat{\sigma}^2} \right) - \ln h_E}{\ln h_E}$$





# Magic Formula

$$k = \frac{\ln \left( B \hat{h}^{\hat{k}+1} \frac{\sigma_E^2}{\hat{\sigma}^2} \right) - \ln h_E}{\ln h_E}$$

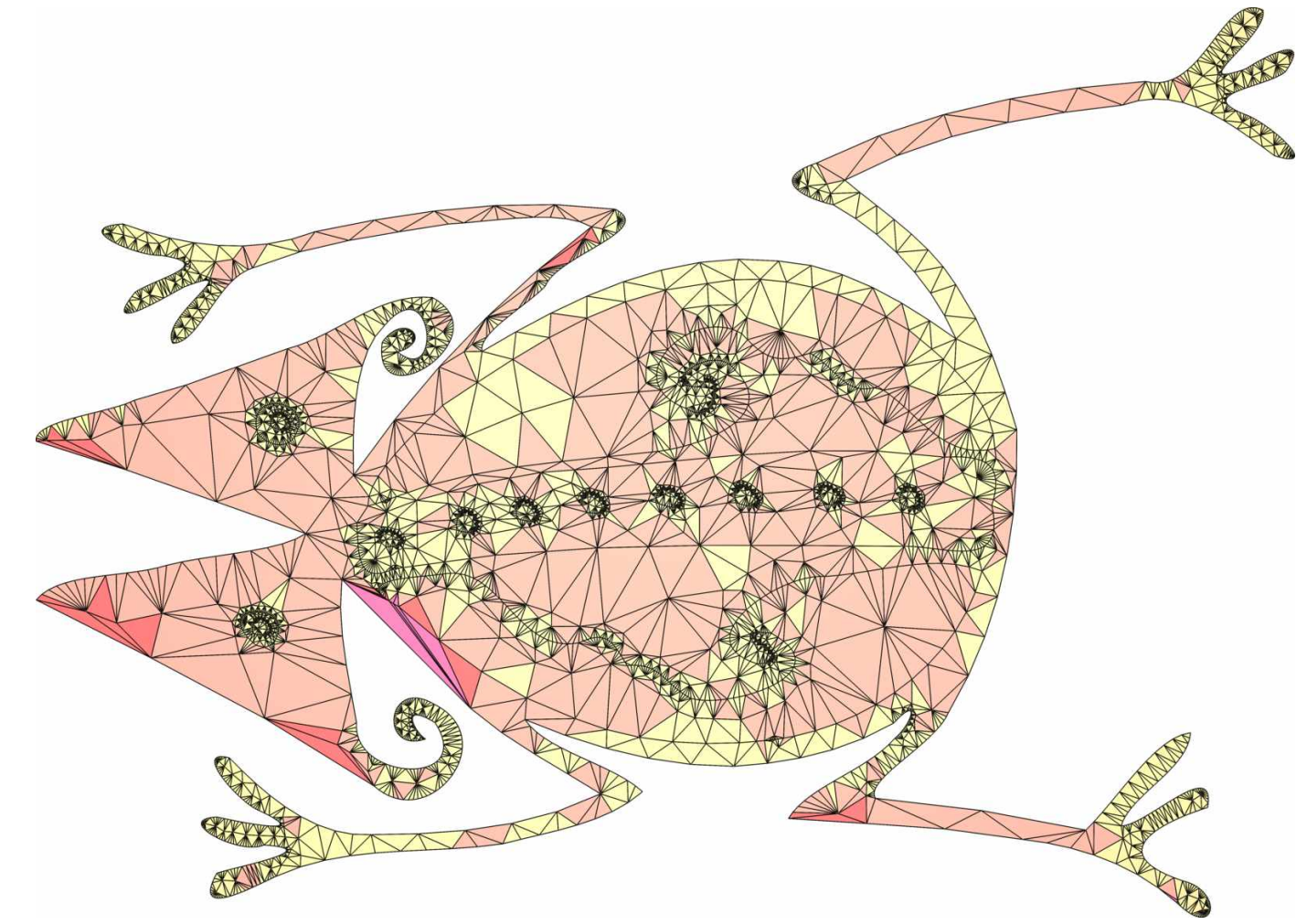




# Overview

$$k = \frac{\ln \left( B \hat{h}^{\hat{k}+1} \frac{\sigma_E^2}{\hat{\sigma}^2} \right) - \ln h_E}{\ln h_E}$$

1. Use formula





2. Propagate degrees




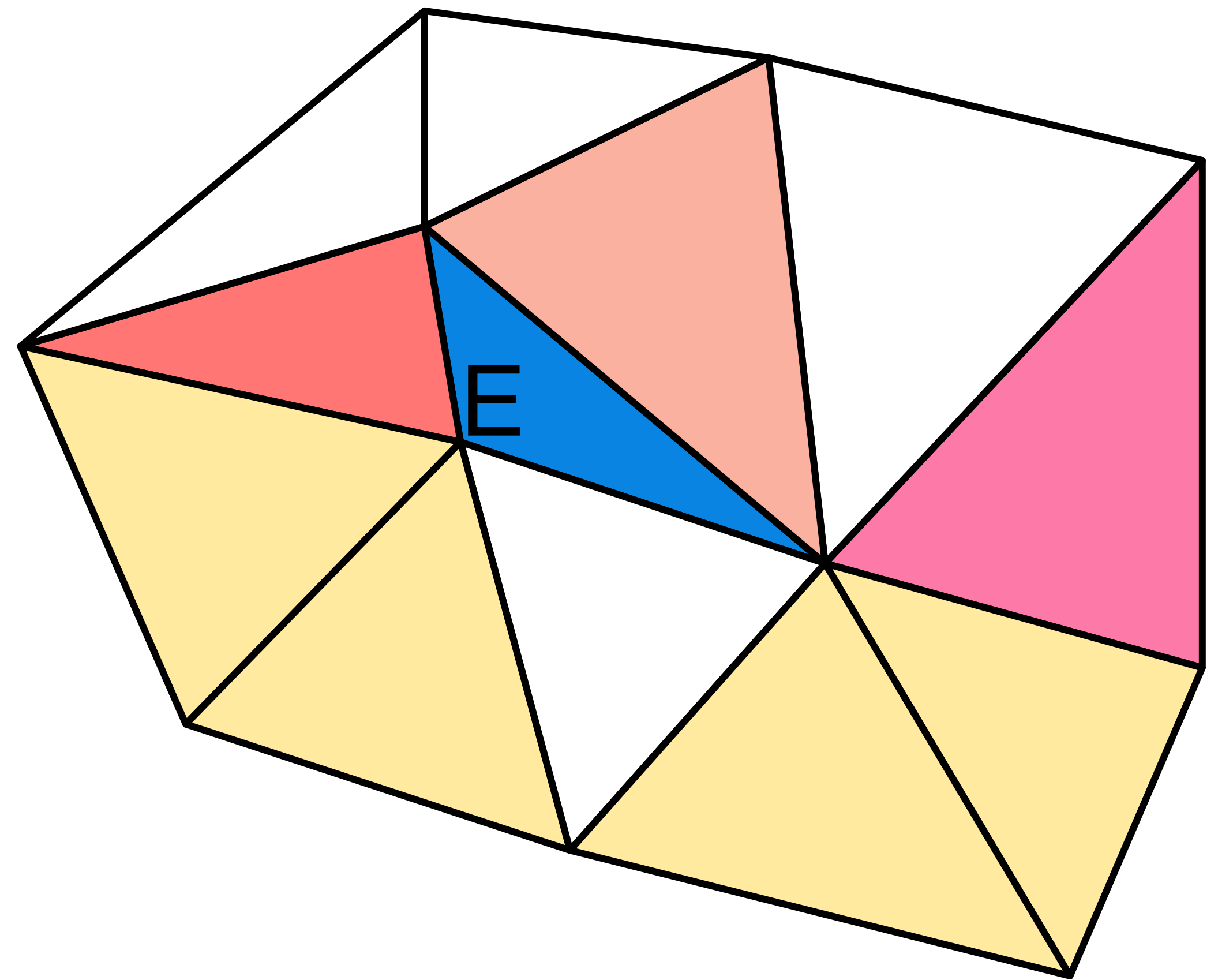
# Degree Propagation

  $P_1$

  $P_2$

  $P_3$

  $P_4$

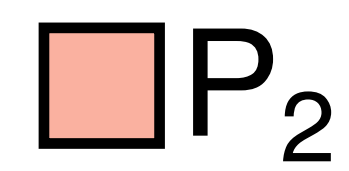




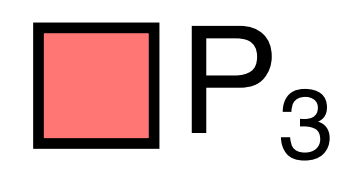
# Degree Propagation



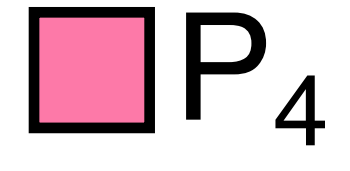
$P_1$



$P_2$

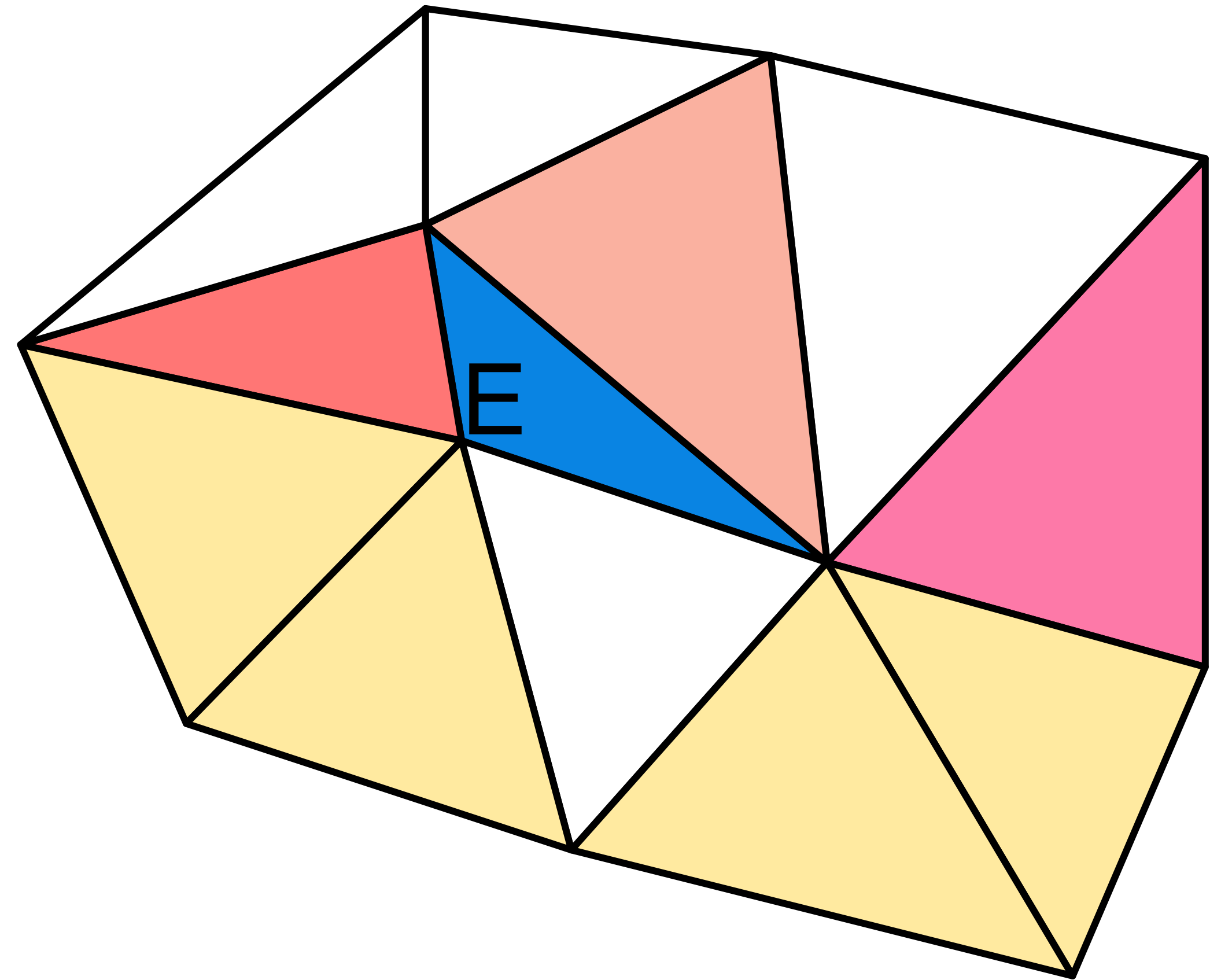


$P_3$



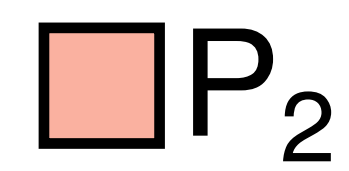
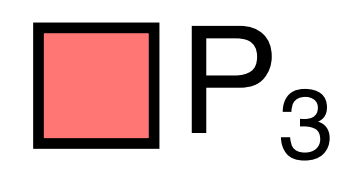
$P_4$

- For each element  $E$

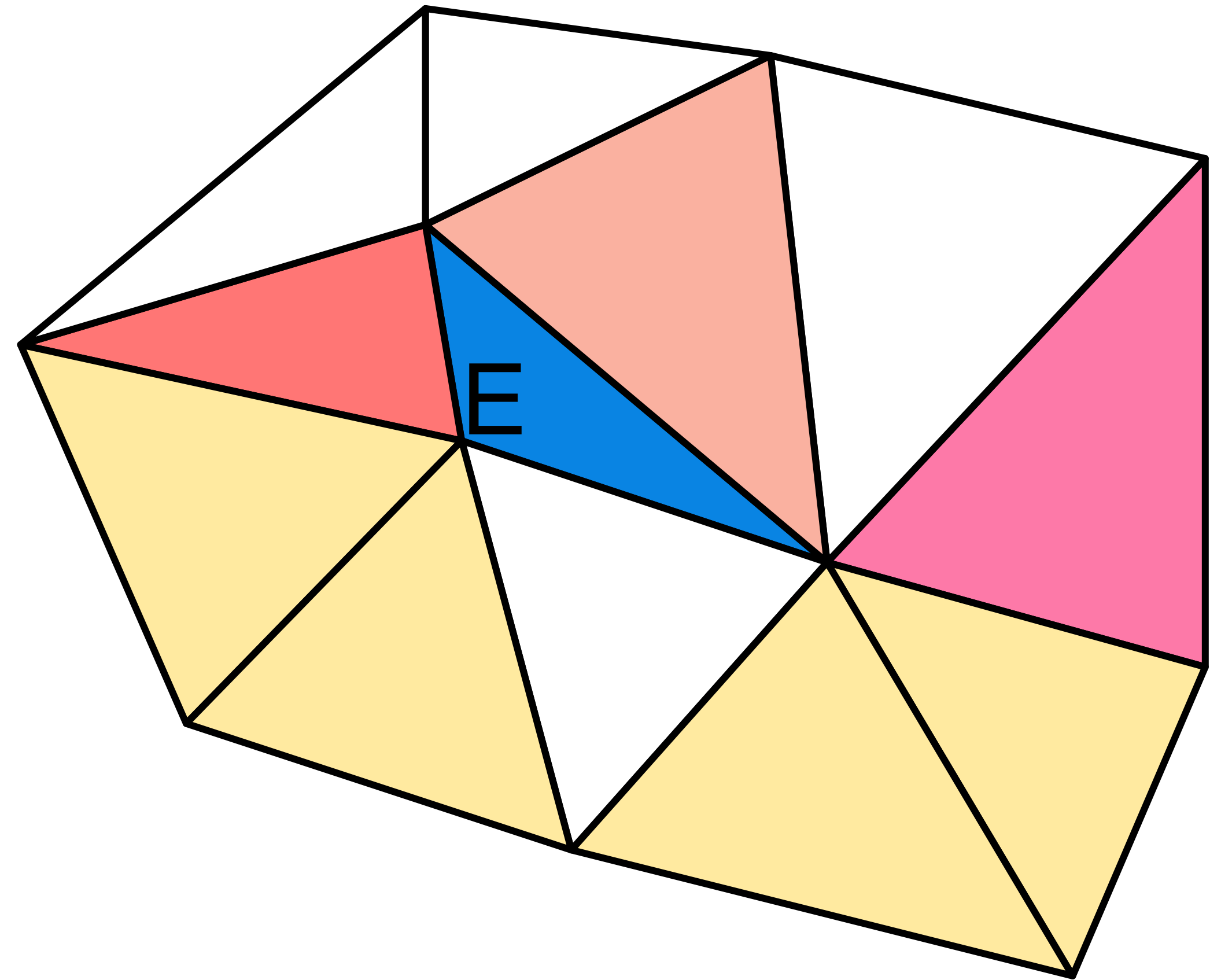




# Degree Propagation

 $P_1$  $P_2$  $P_3$  $P_4$ 

- For each element  $E$
- Compute  $k_E$  using formula





# Degree Propagation

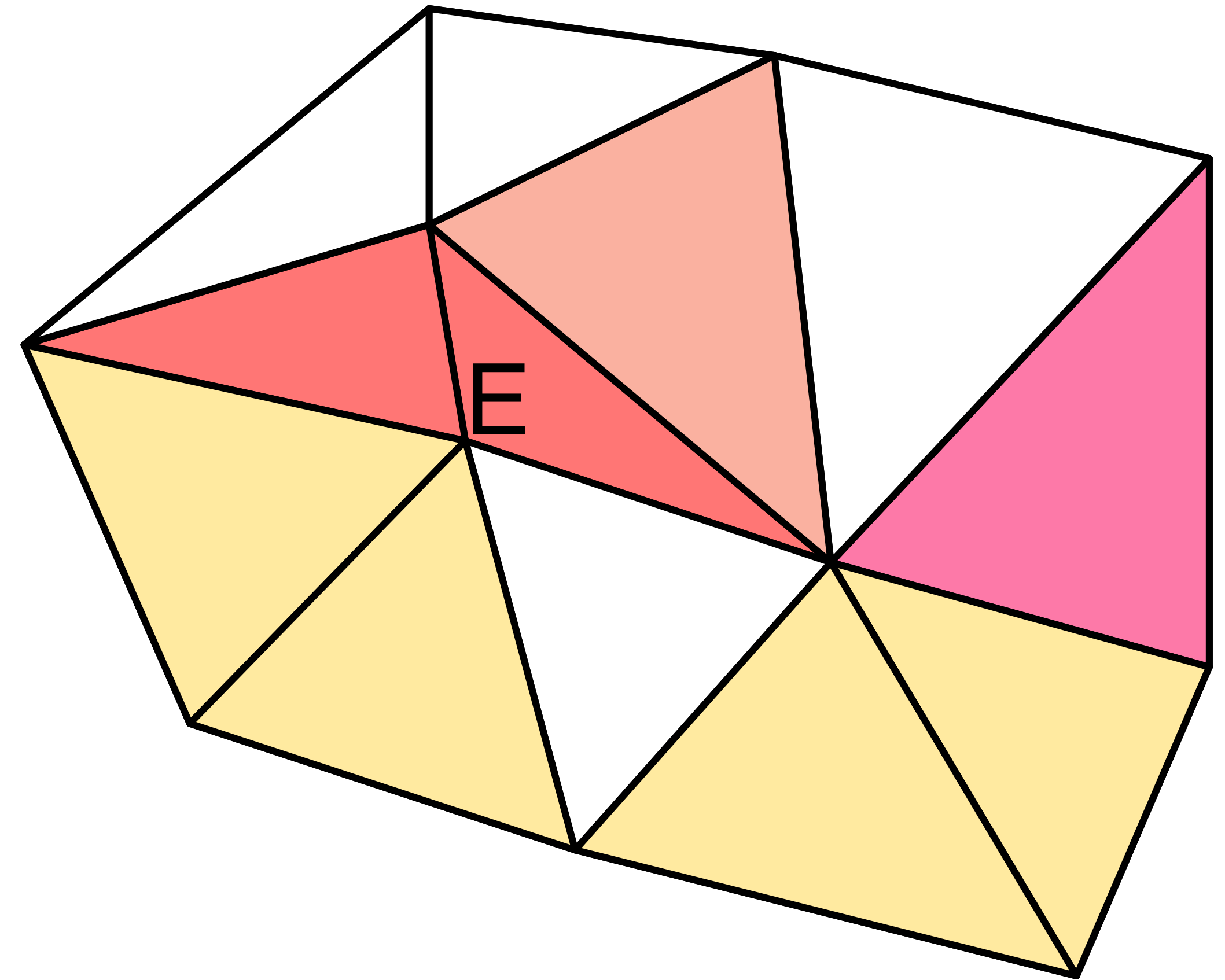
■  $P_1$

■  $P_2$

■  $P_3$

■  $P_4$

- For each element  $E$
- Compute  $k_E$  using formula
- Increase the order (if necessary) of:
  - The element  $E$





# Degree Propagation

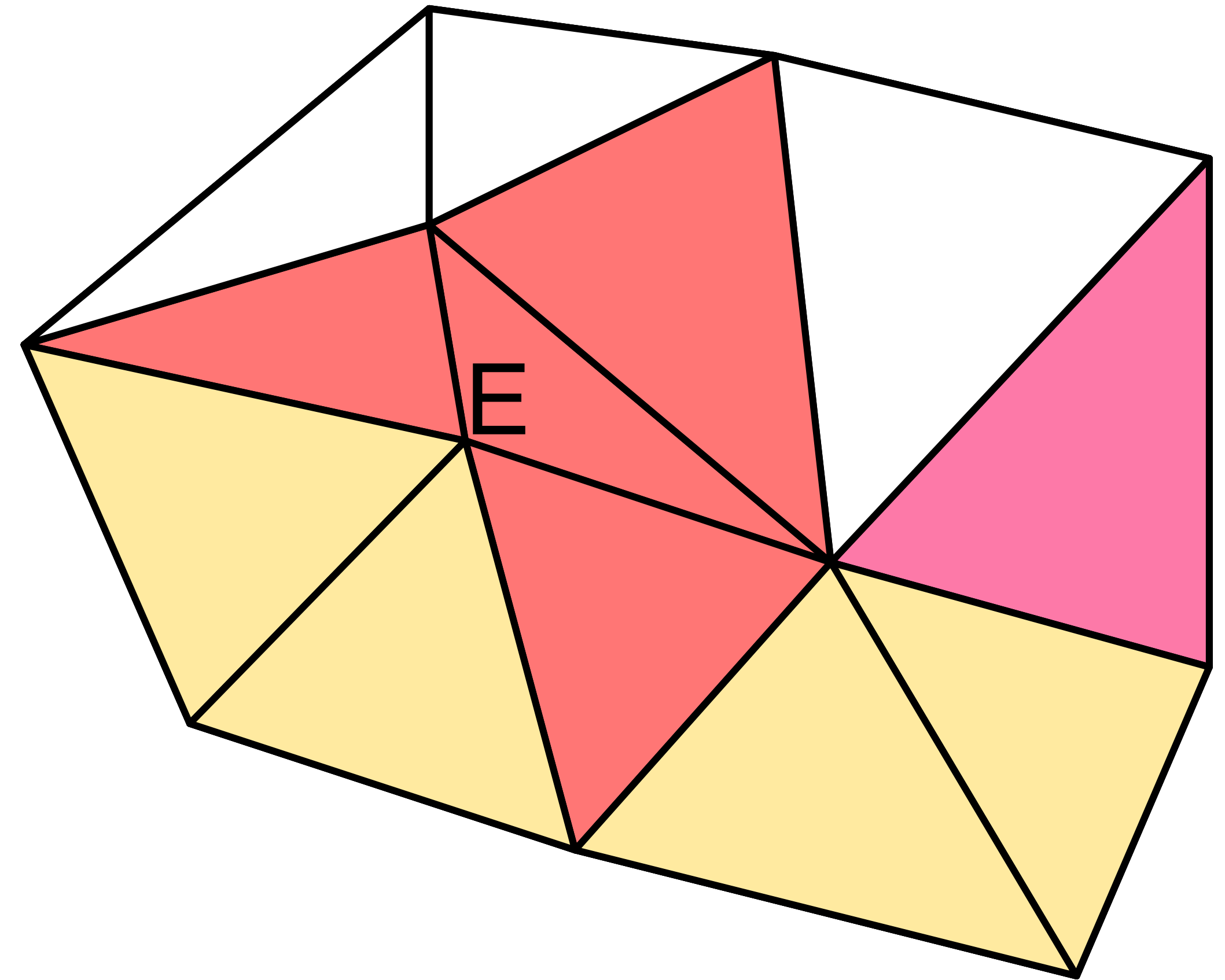
■  $P_1$

■  $P_2$

■  $P_3$

■  $P_4$

- For each element  $E$
- Compute  $k_E$  using formula
- Increase the order (if necessary) of:
  - The element  $E$
  - All edge/face neighbors





# Degree Propagation

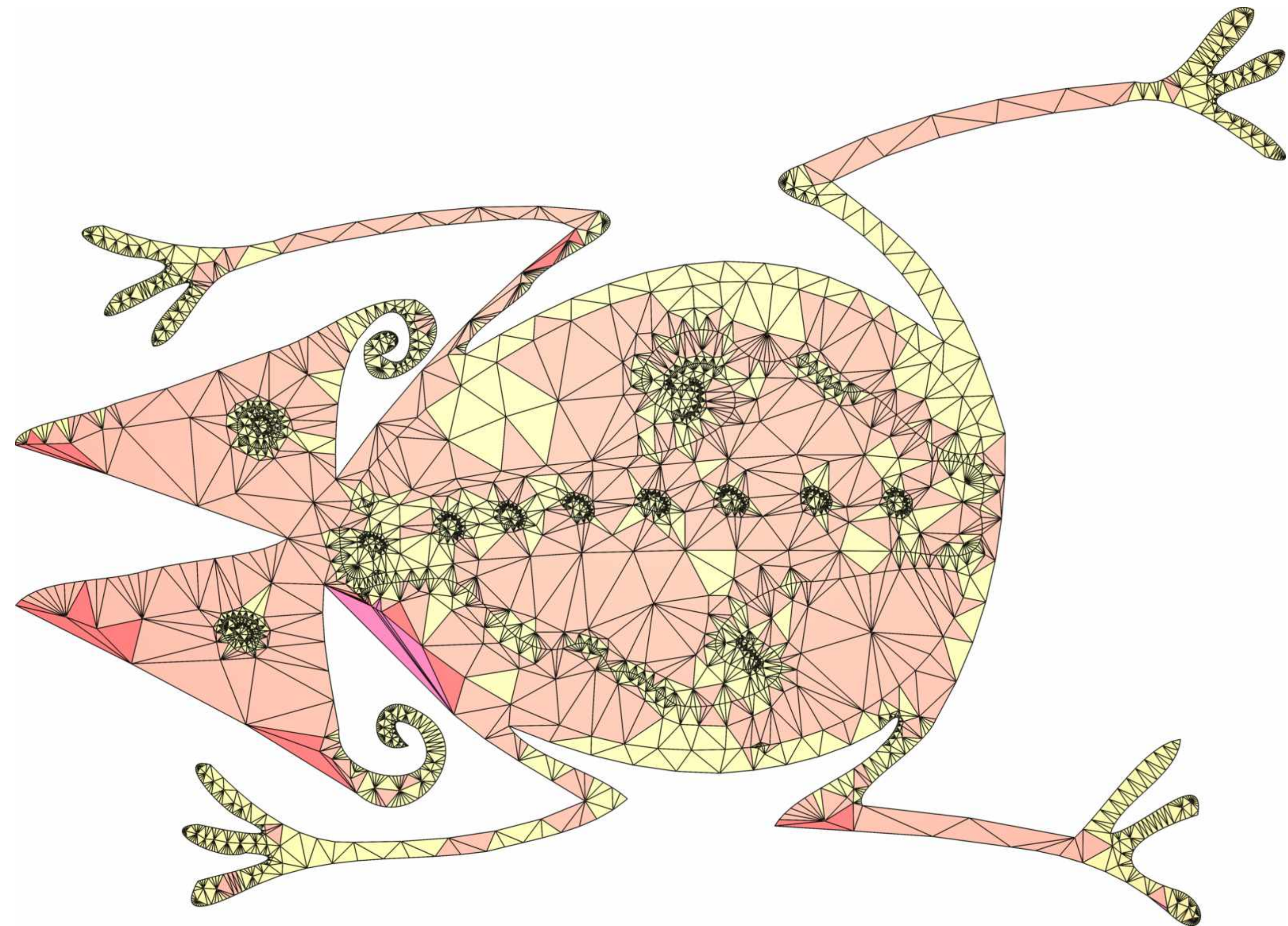
■  $P_1$

■  $P_2$

■  $P_3$

■  $P_4$

- For each element  $E$
- Compute  $k_E$  using formula
- Increase the order (if necessary) of:
  - The element  $E$
  - All edge/face neighbors

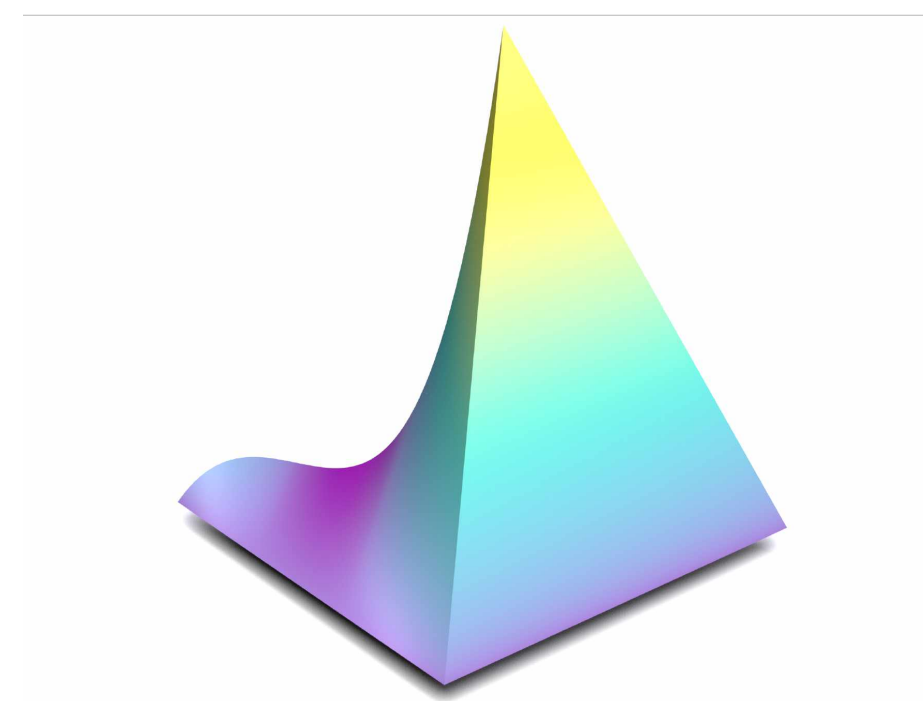




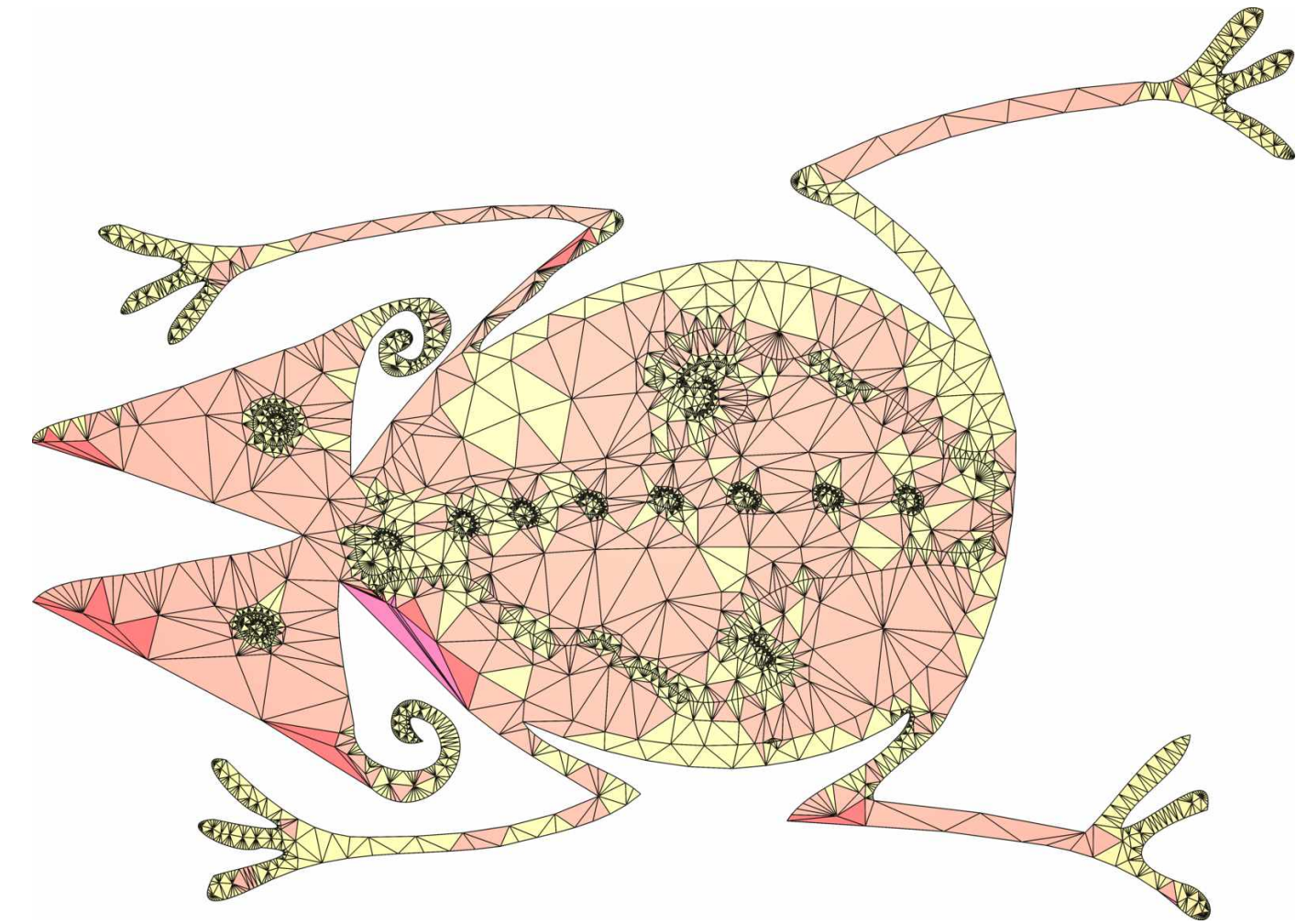
# Overview

$$k = \frac{\ln \left( B \hat{h}^{\hat{k}+1} \frac{\sigma_E^2}{\hat{\sigma}^2} \right) - \ln h_E}{\ln h_E}$$

1. Use formula



3. Construct  $C^0$  basis



2. Propagate degrees



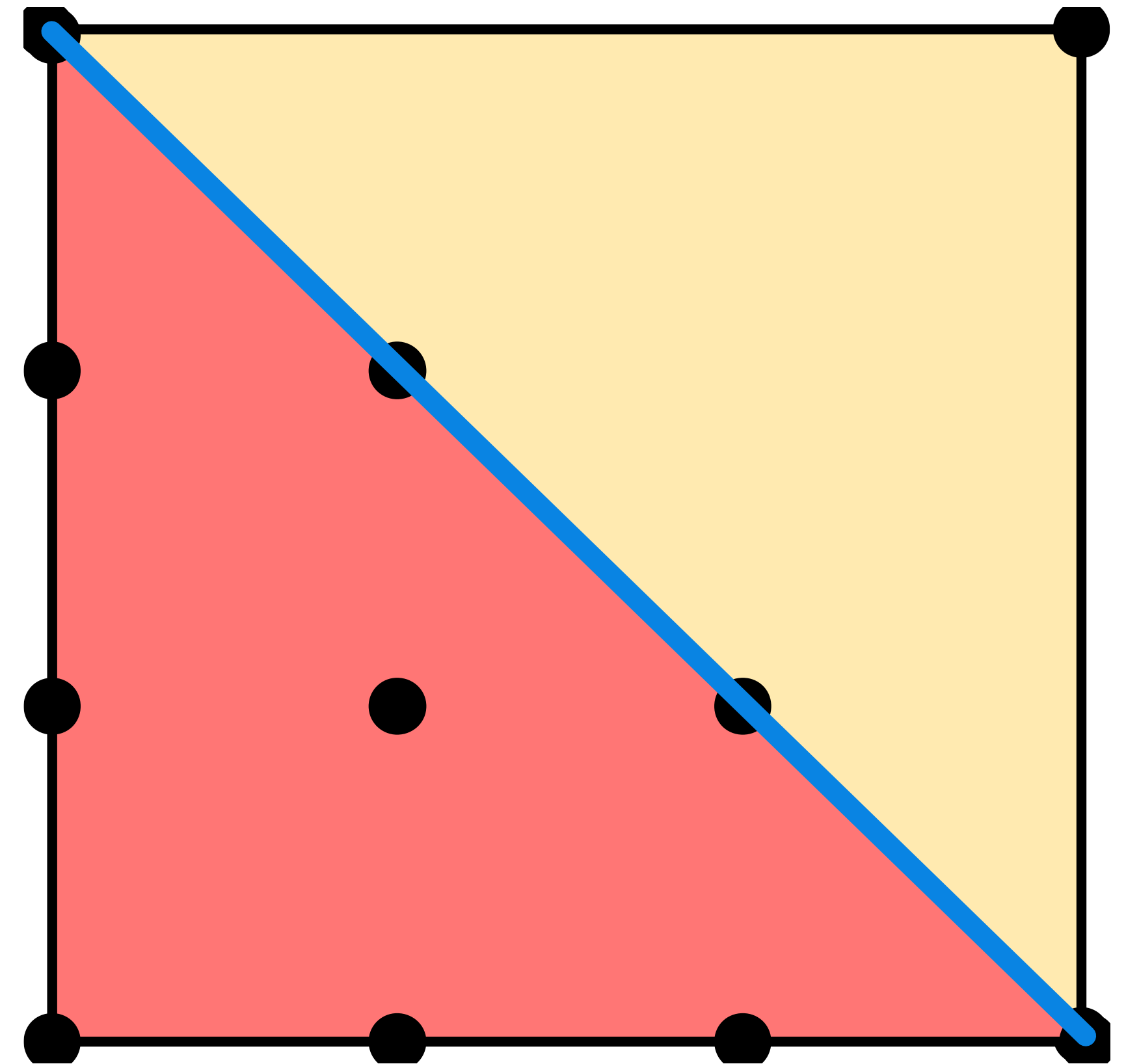
# Building Continuous Basis

 Linear

 Cubic

Linear

$$a + bt$$





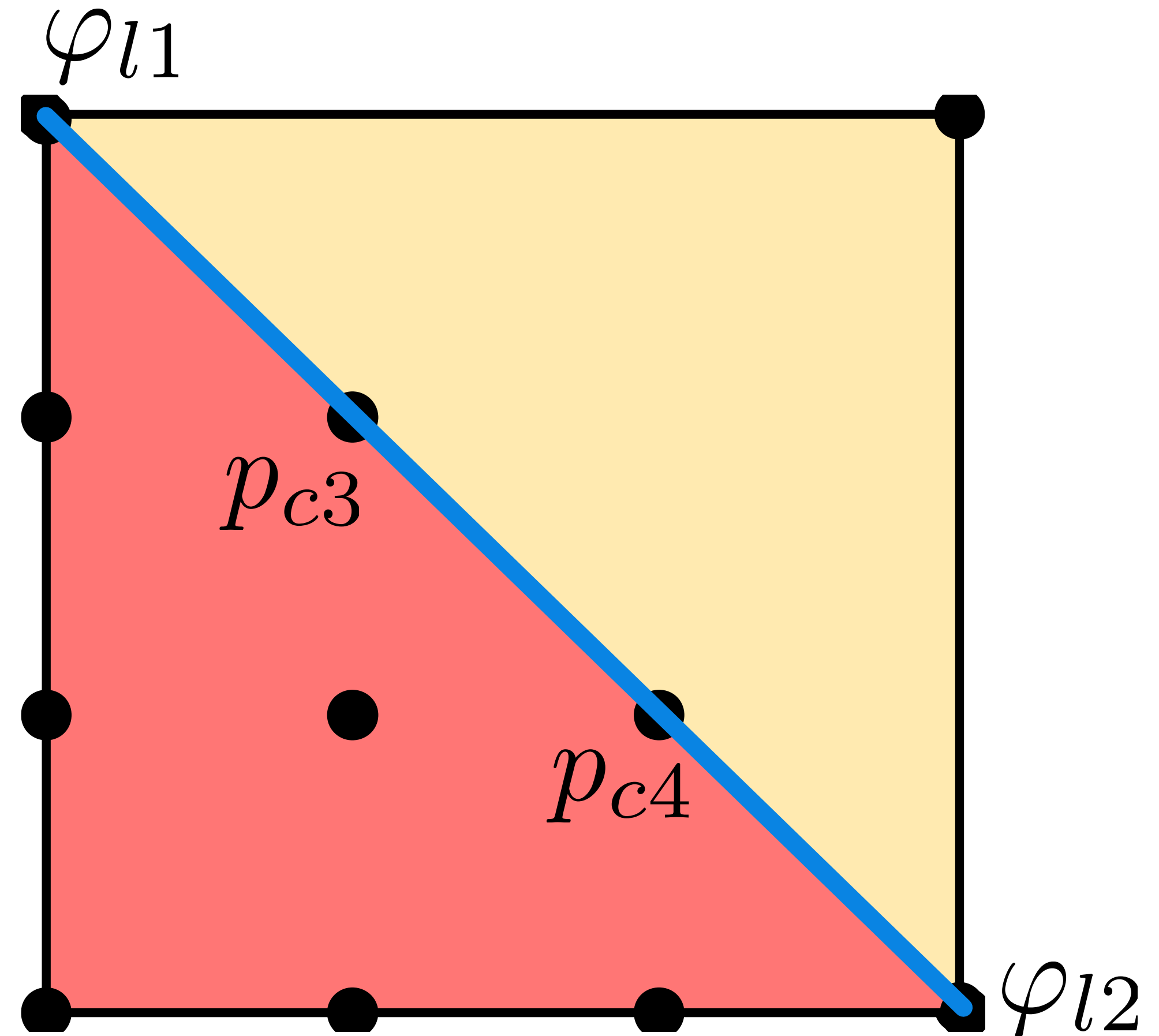
# Building Continuous Basis

 Linear

 Cubic

Linear

$$a + bt + 0t^2 + 0t^3$$





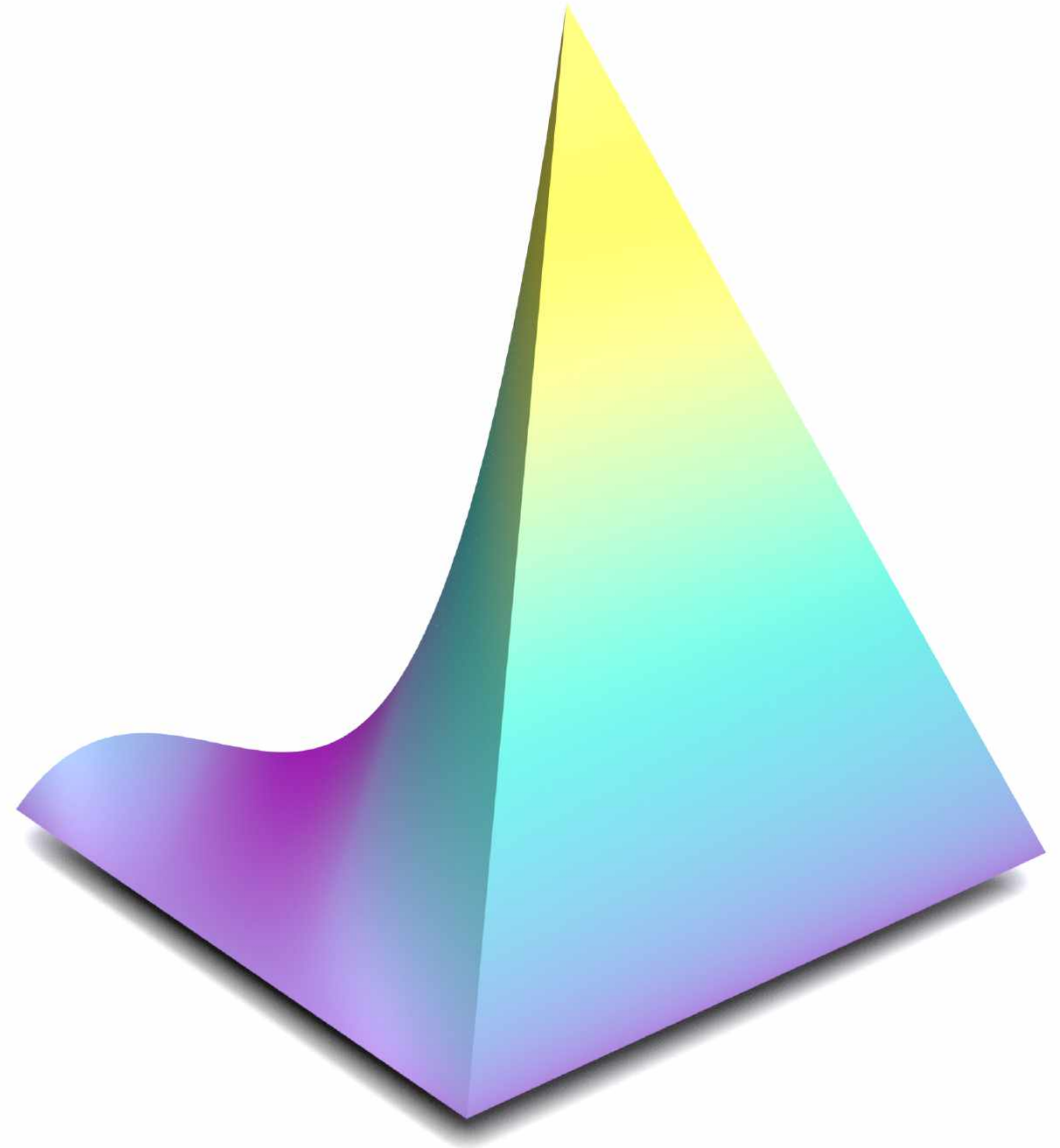
# Building Continuous Basis

Linear

Cubic

Linear

$$a + bt + 0t^2 + 0t^3$$

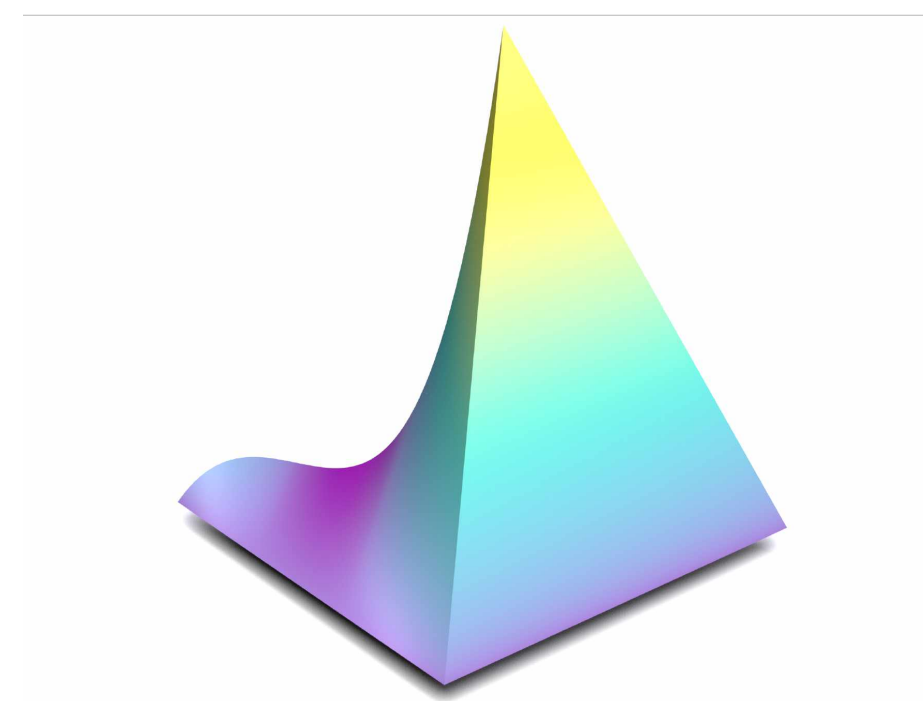




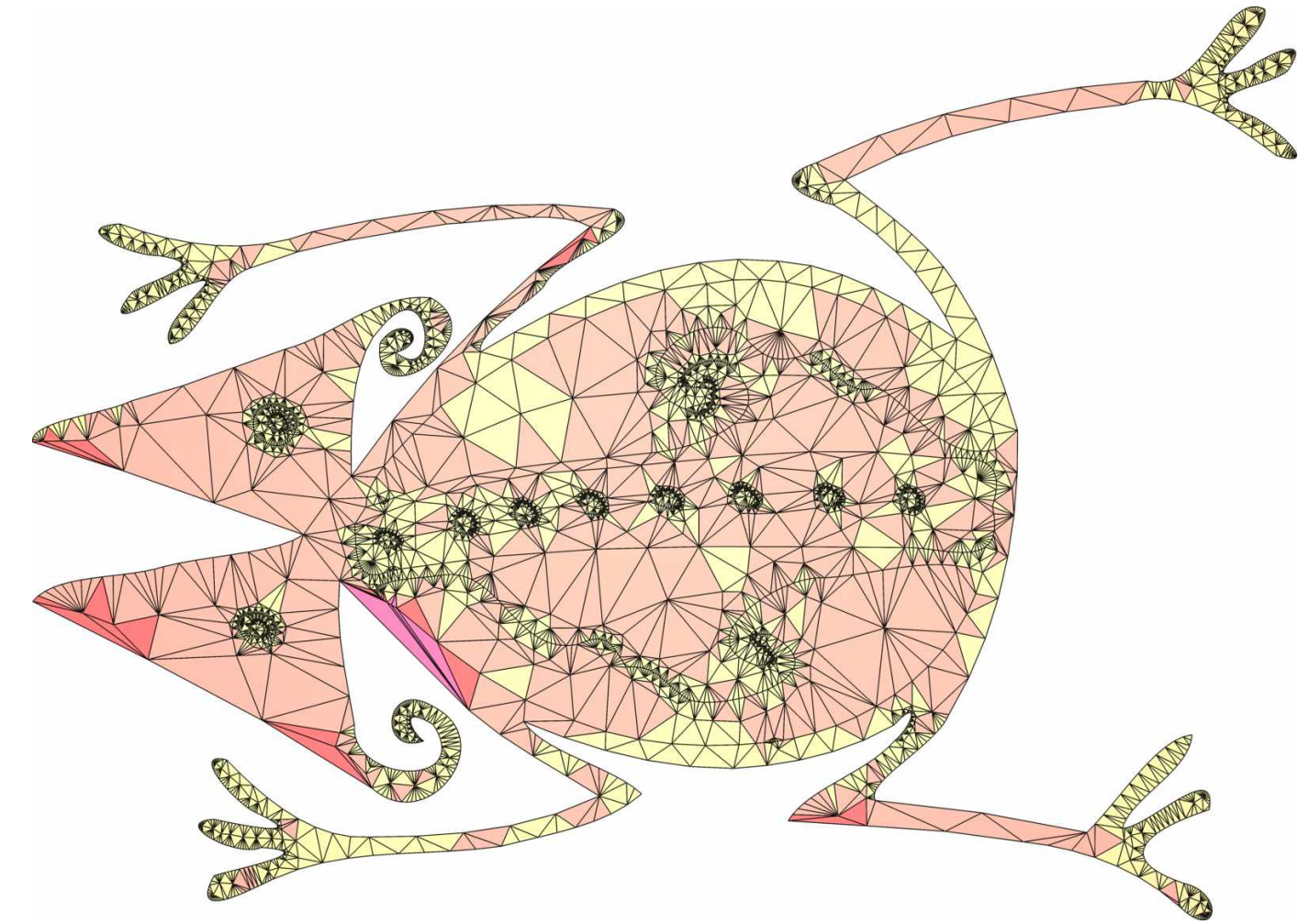
# Overview

$$k = \frac{\ln \left( B \hat{h}^{\hat{k}+1} \frac{\sigma_E^2}{\hat{\sigma}^2} \right) - \ln h_E}{\ln h_E}$$

1. Use formula



3. Construct  $C^0$  basis



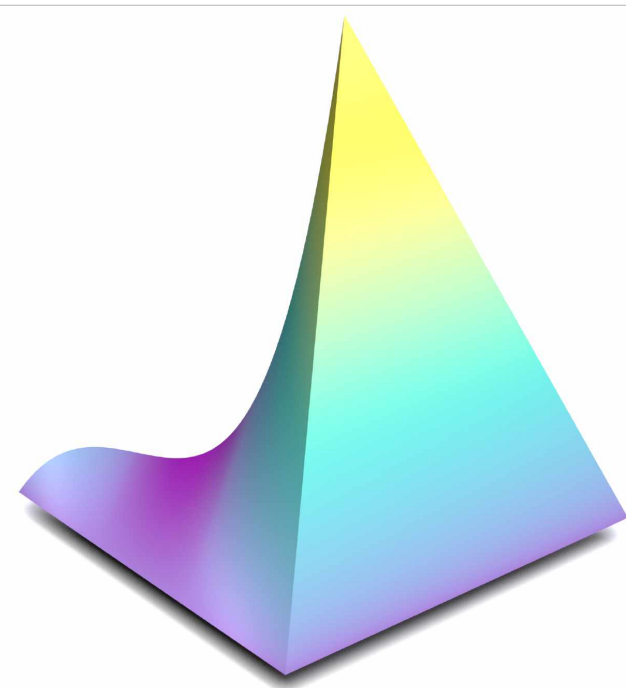
2. Propagate degrees



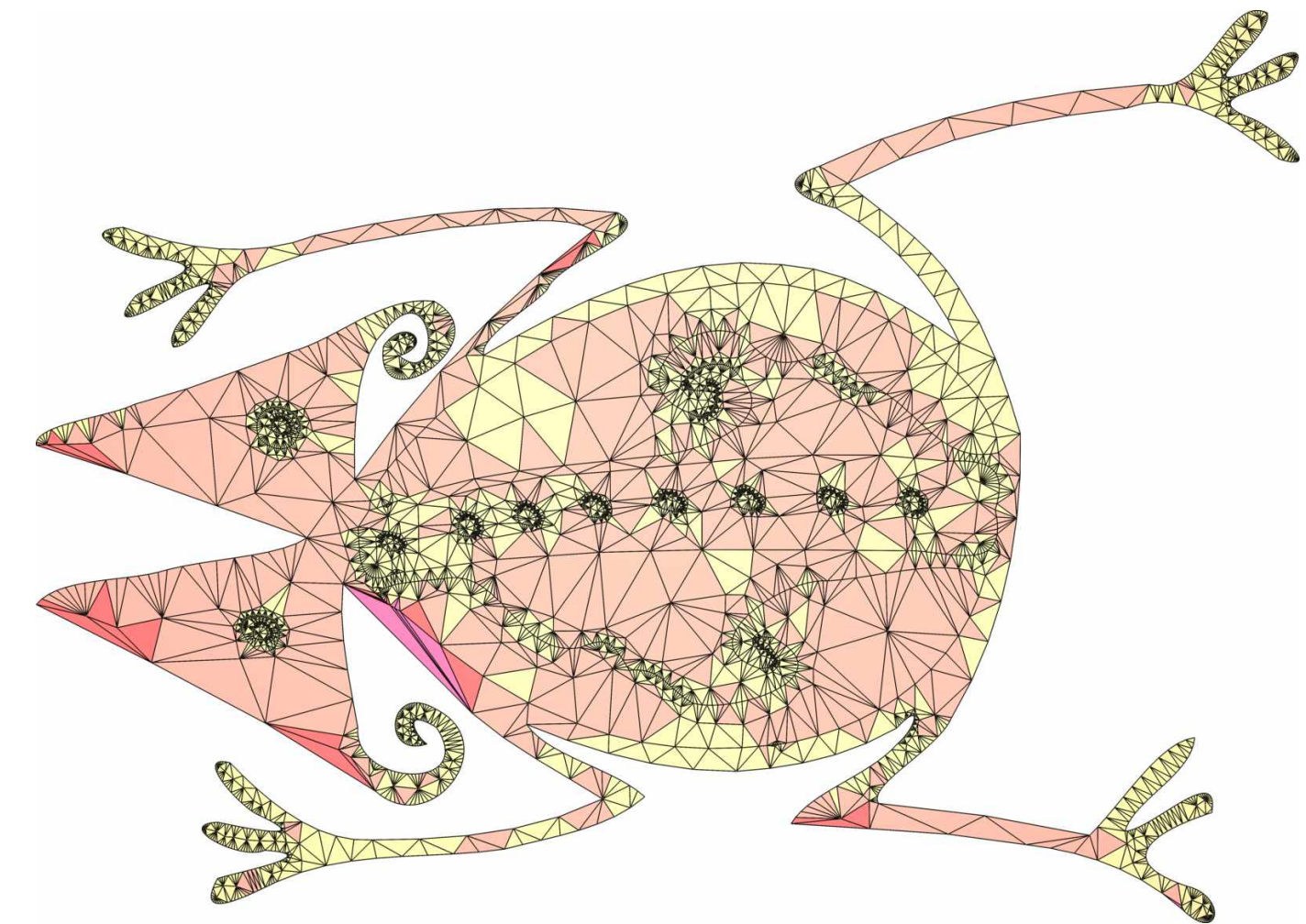
# Overview

$$k = \frac{\ln \left( B \hat{h}^{\hat{k}+1} \frac{\sigma_E^2}{\hat{\sigma}^2} \right) - \ln h_E}{\ln h_E}$$

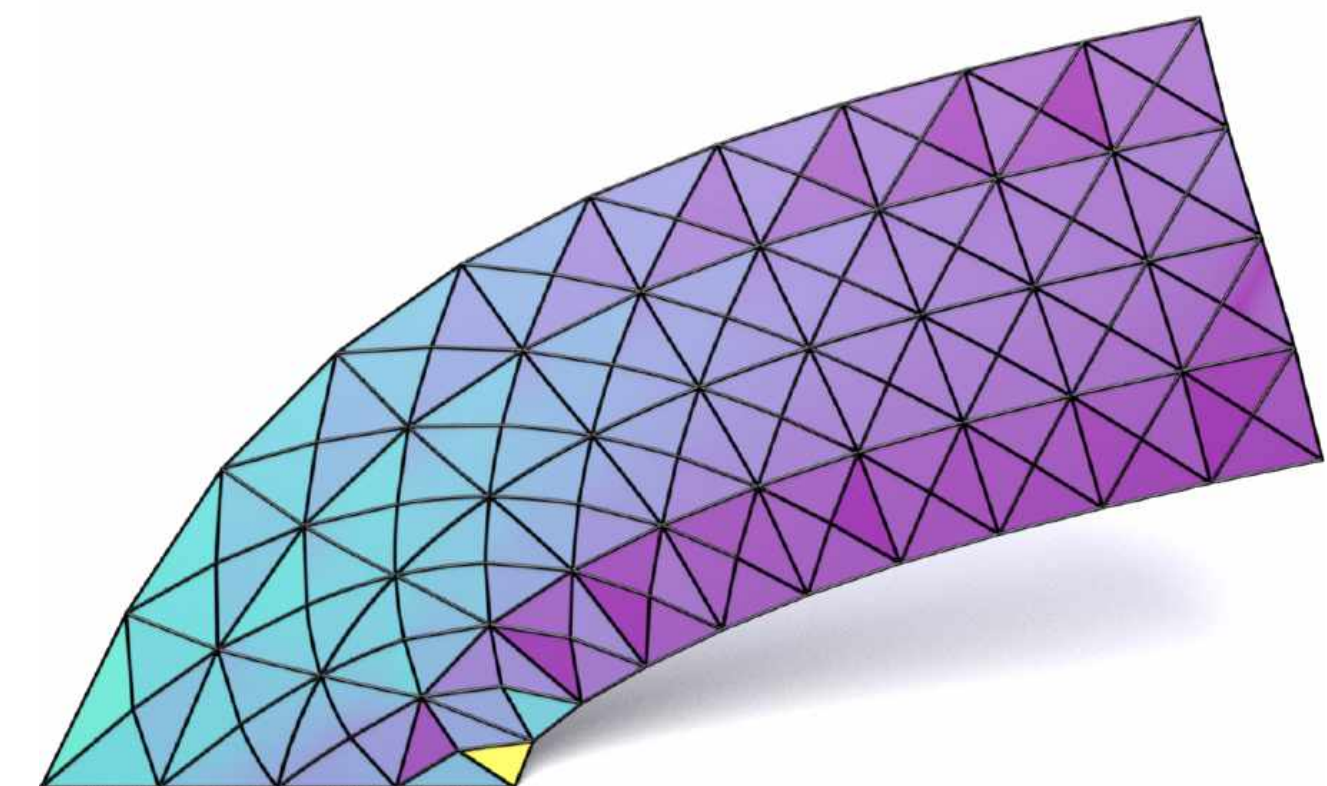
1. Use formula



3. Construct  $C^0$  basis



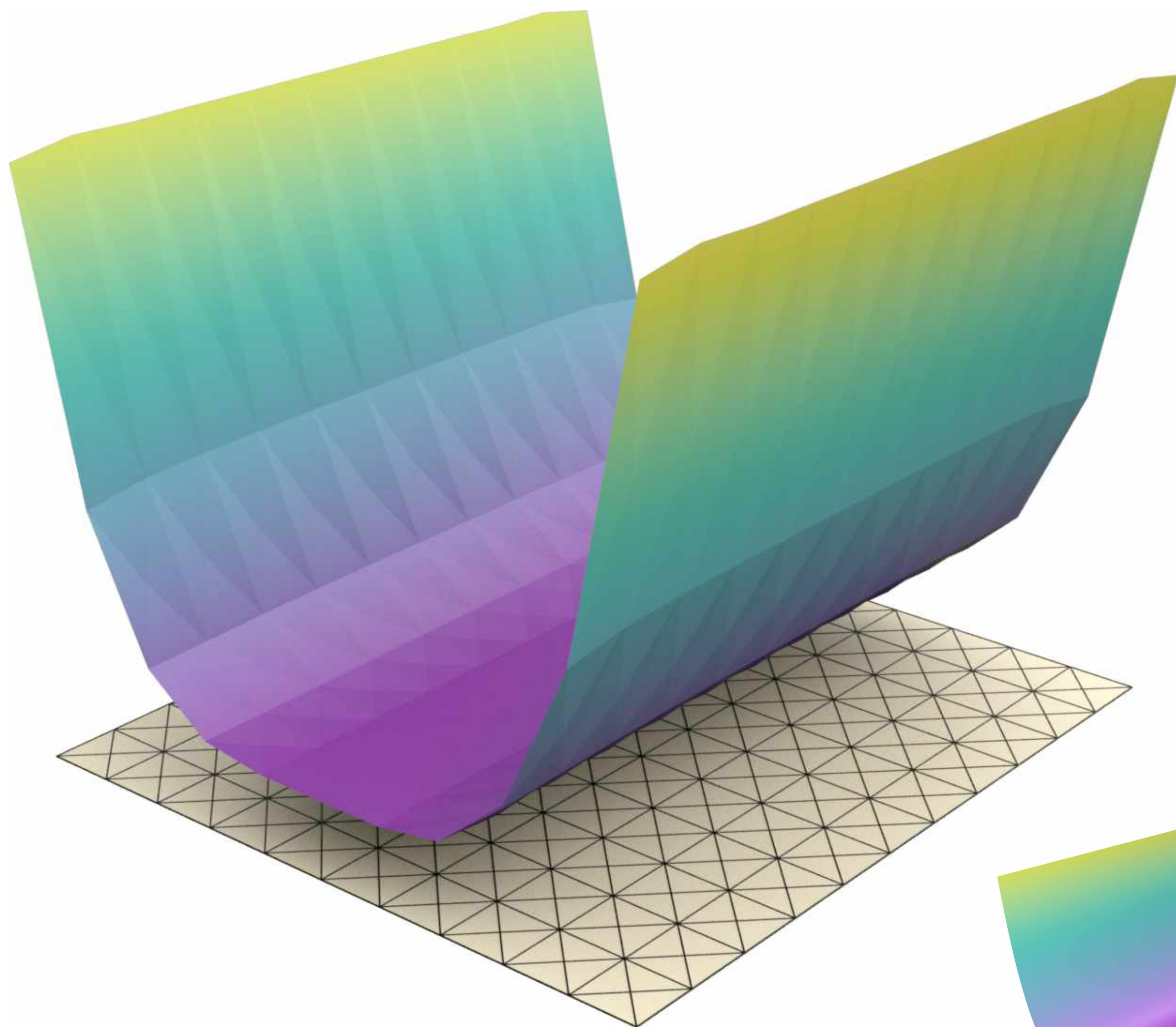
2. Propagate degrees



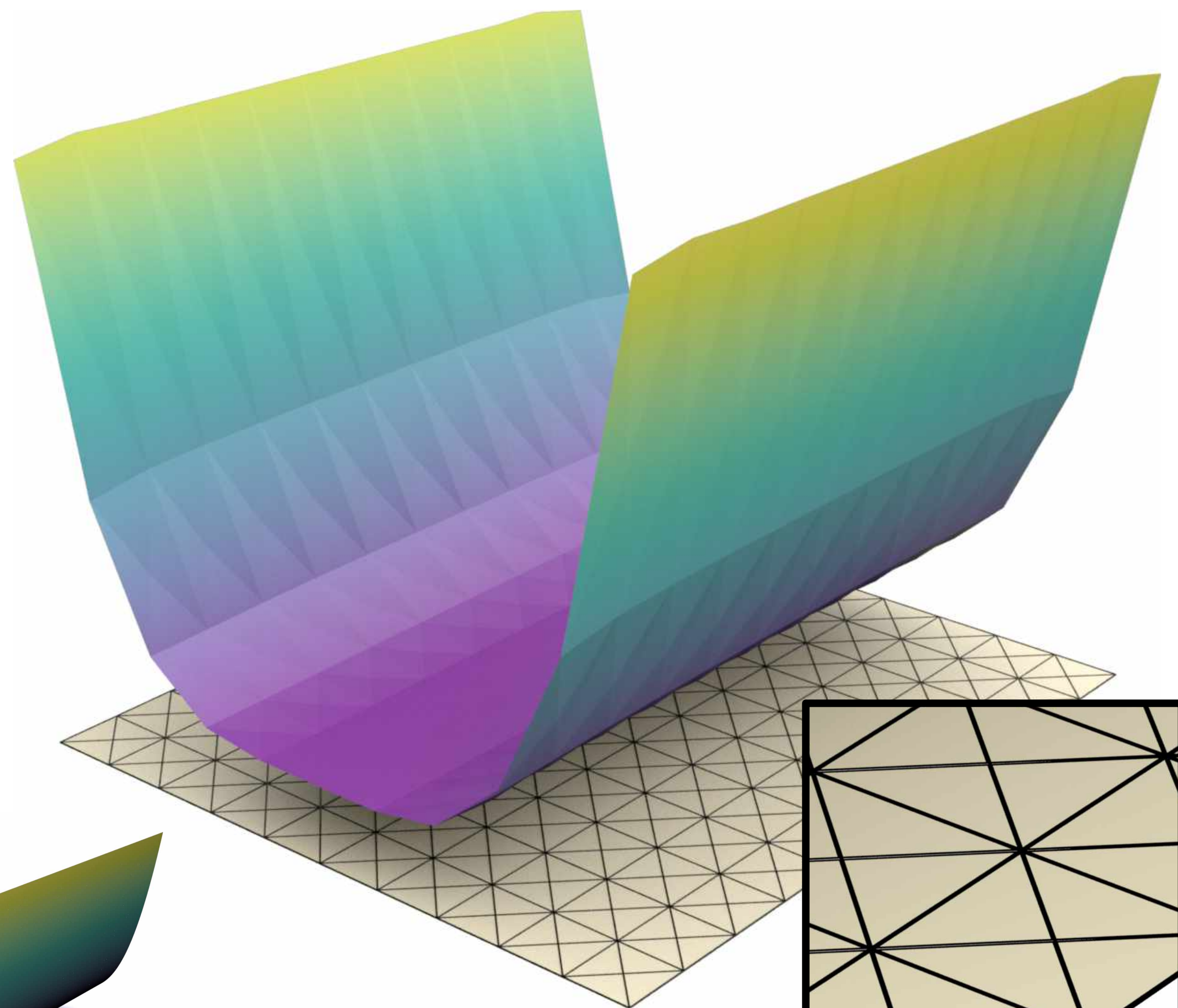
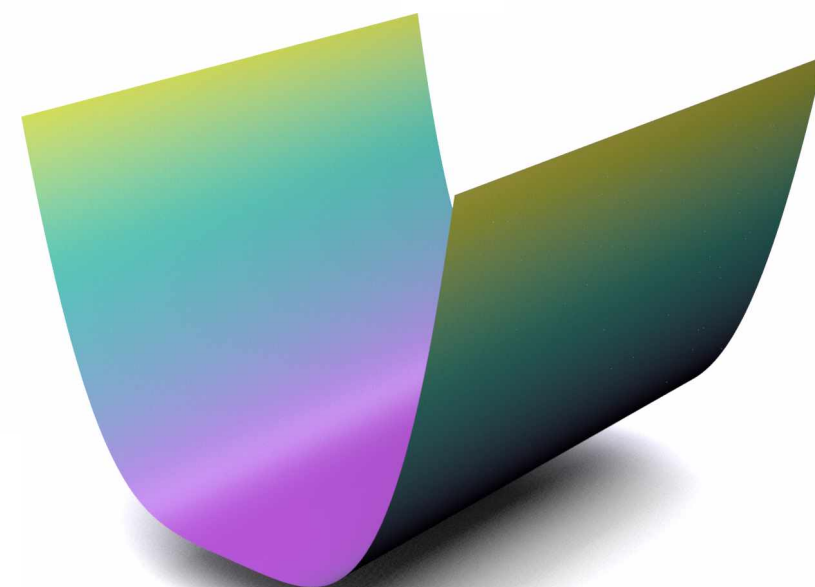
4. Simulate!



# Back to Laplace



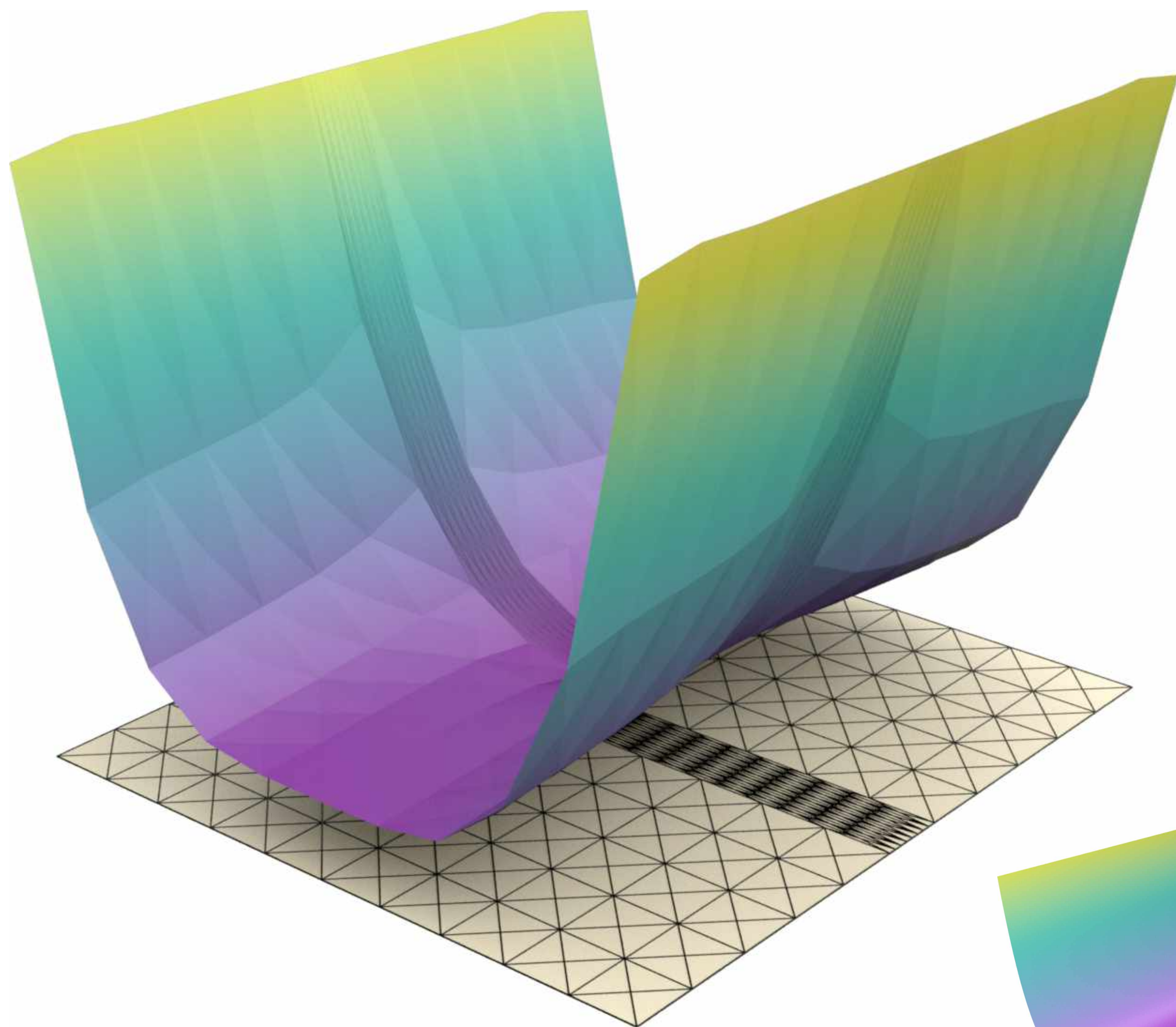
Standard



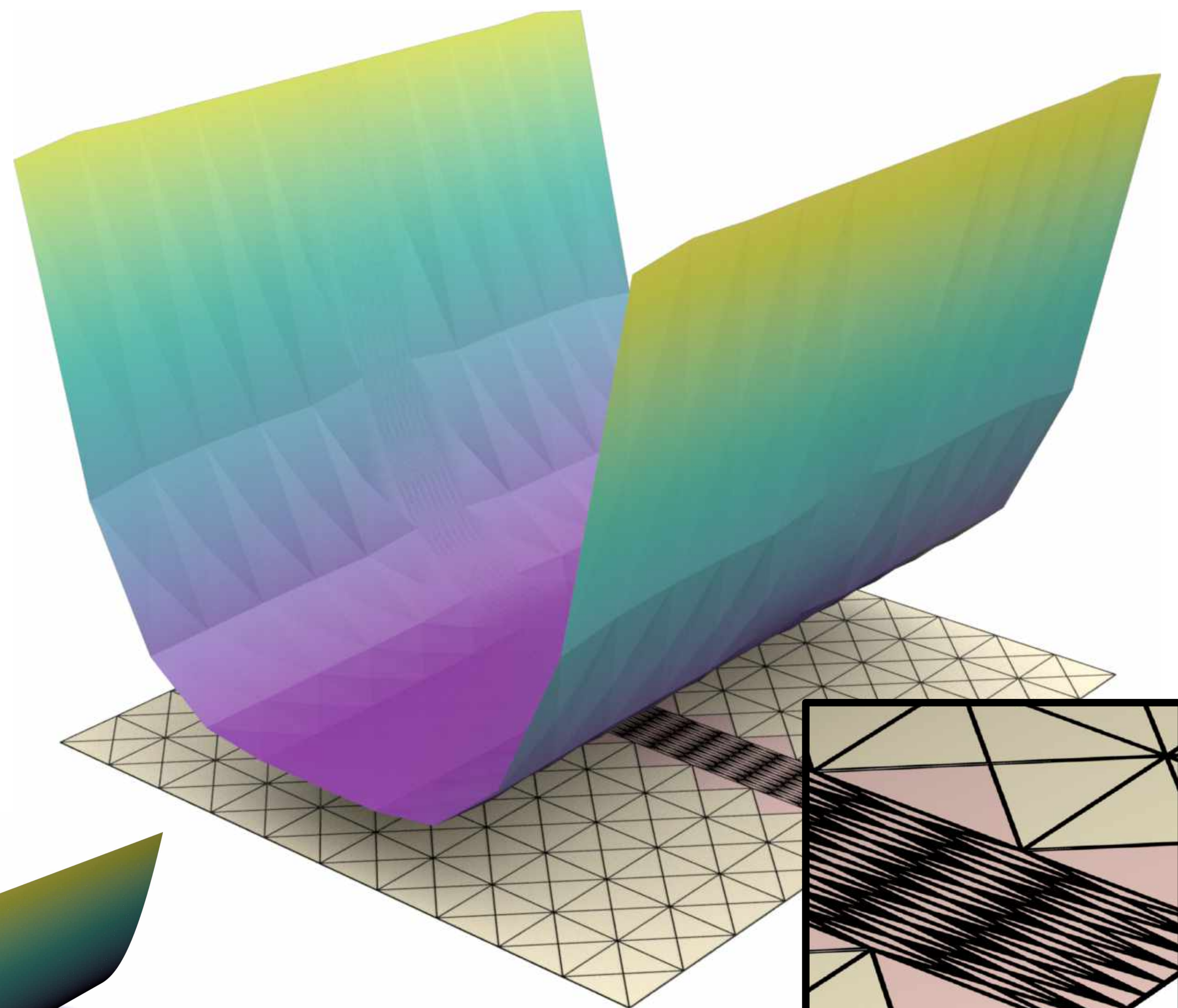
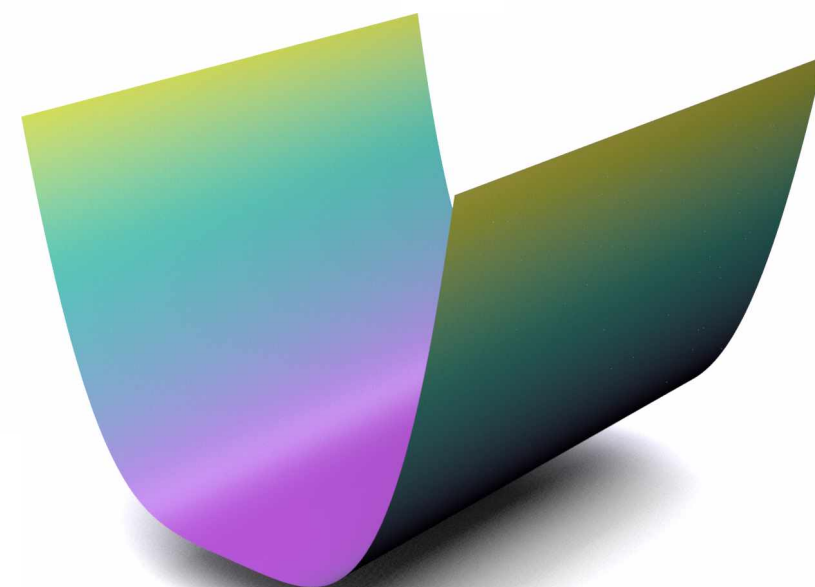
Our



# Back to Laplace



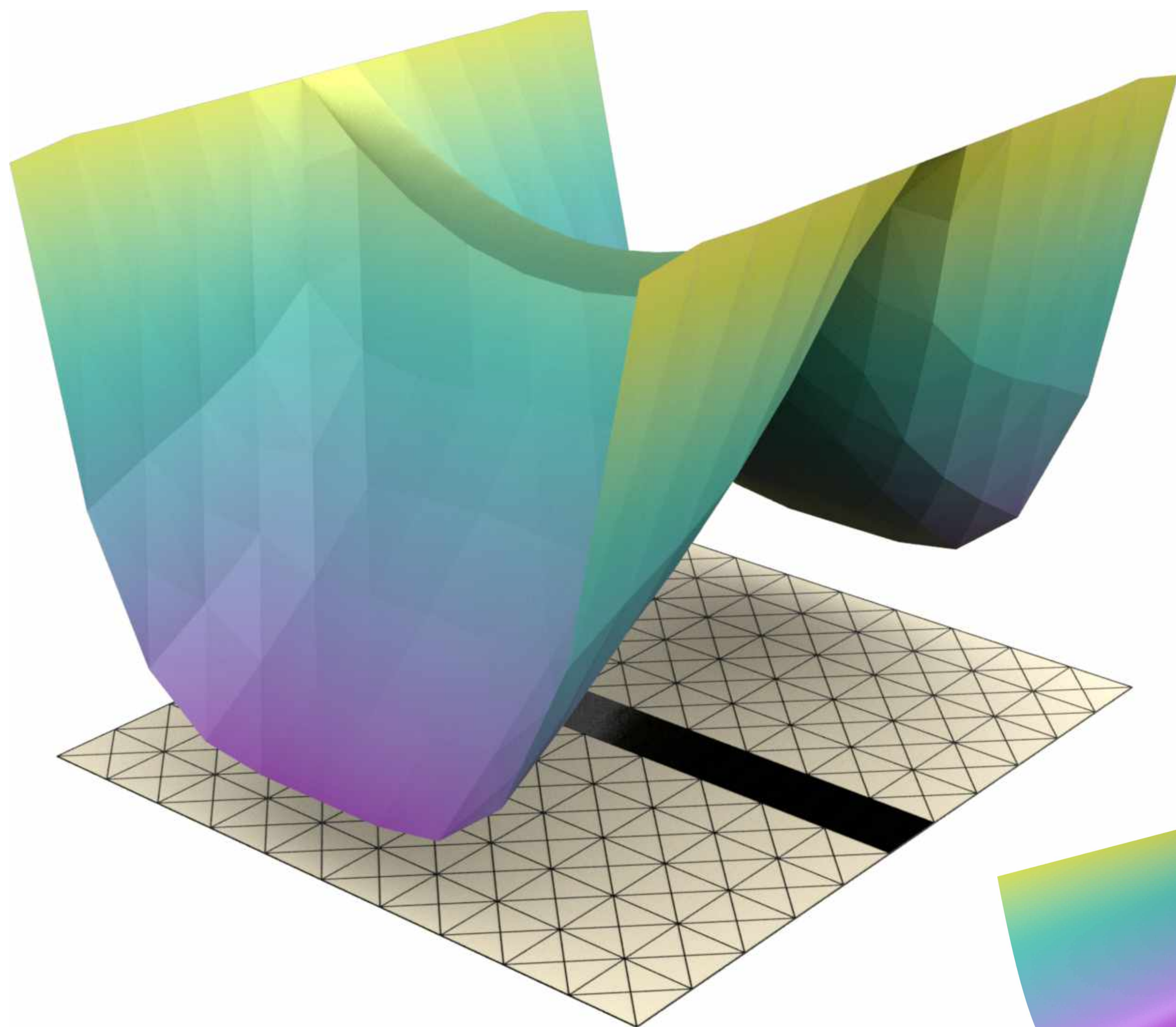
Standard



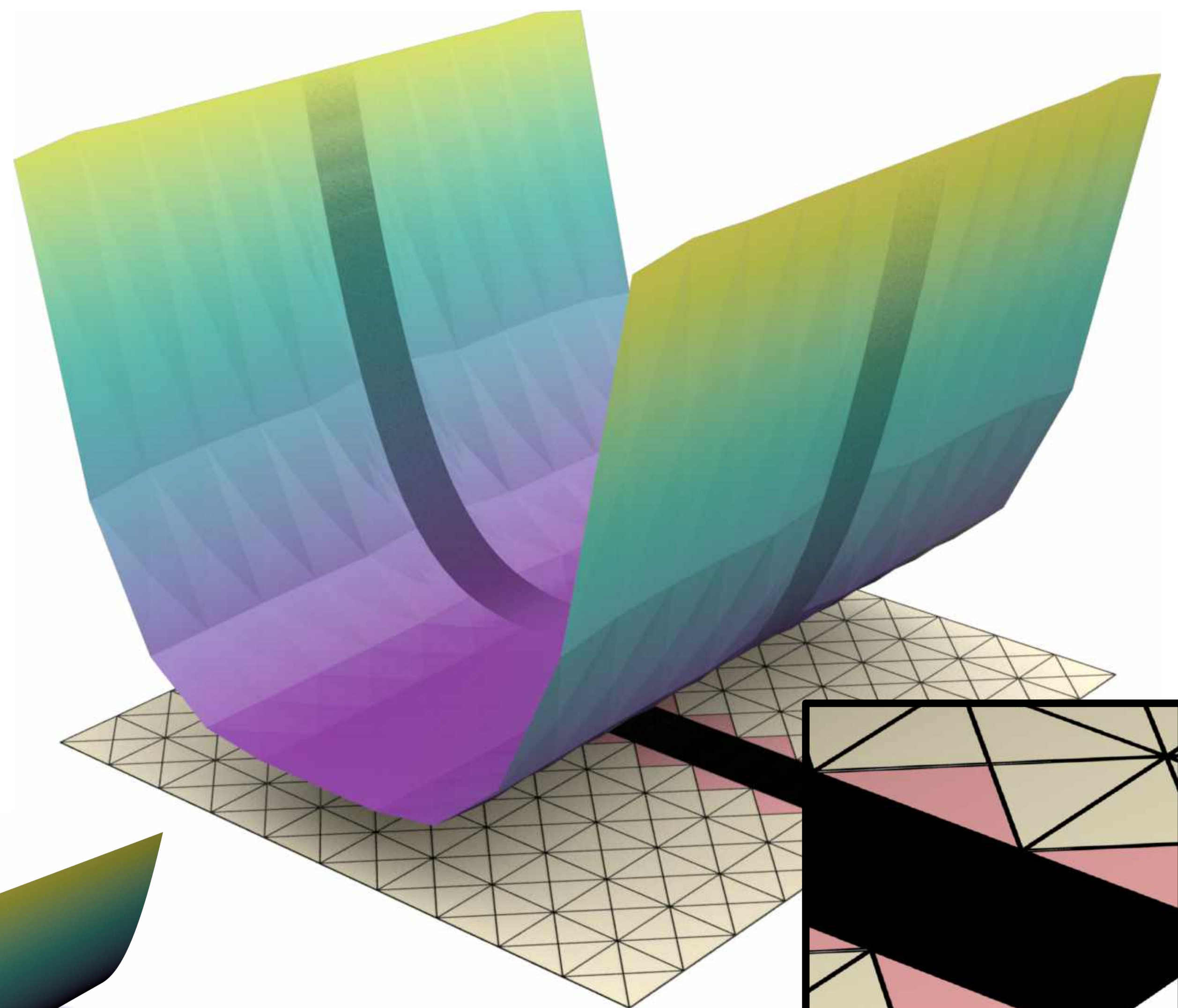
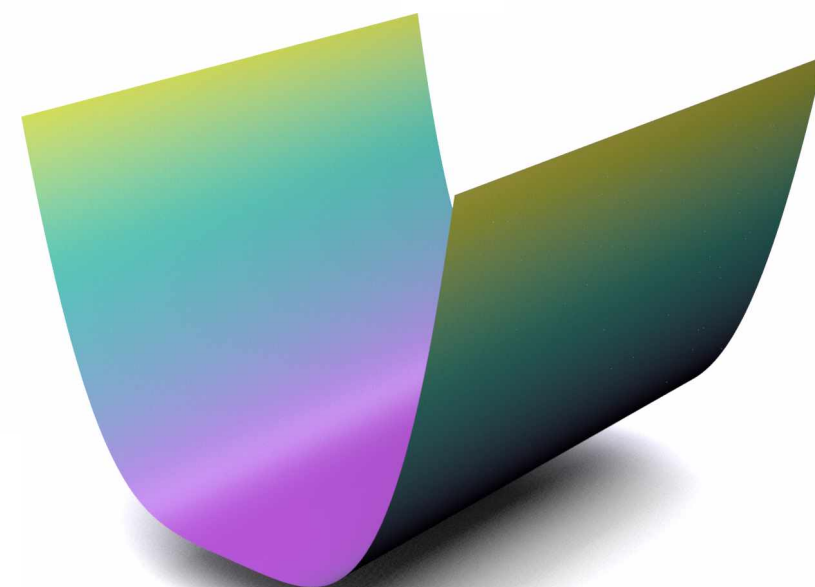
Our



# Back to Laplace



Standard

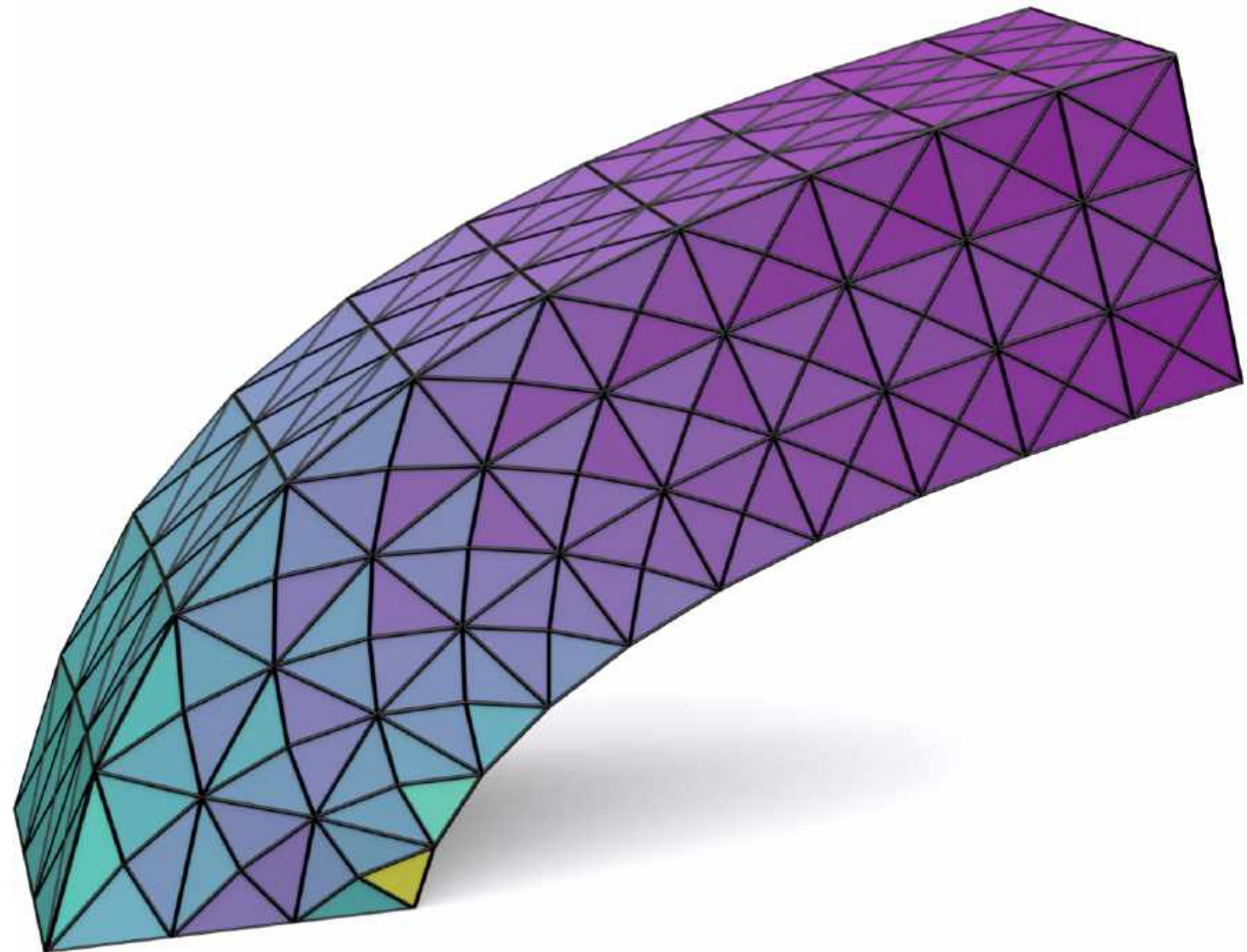
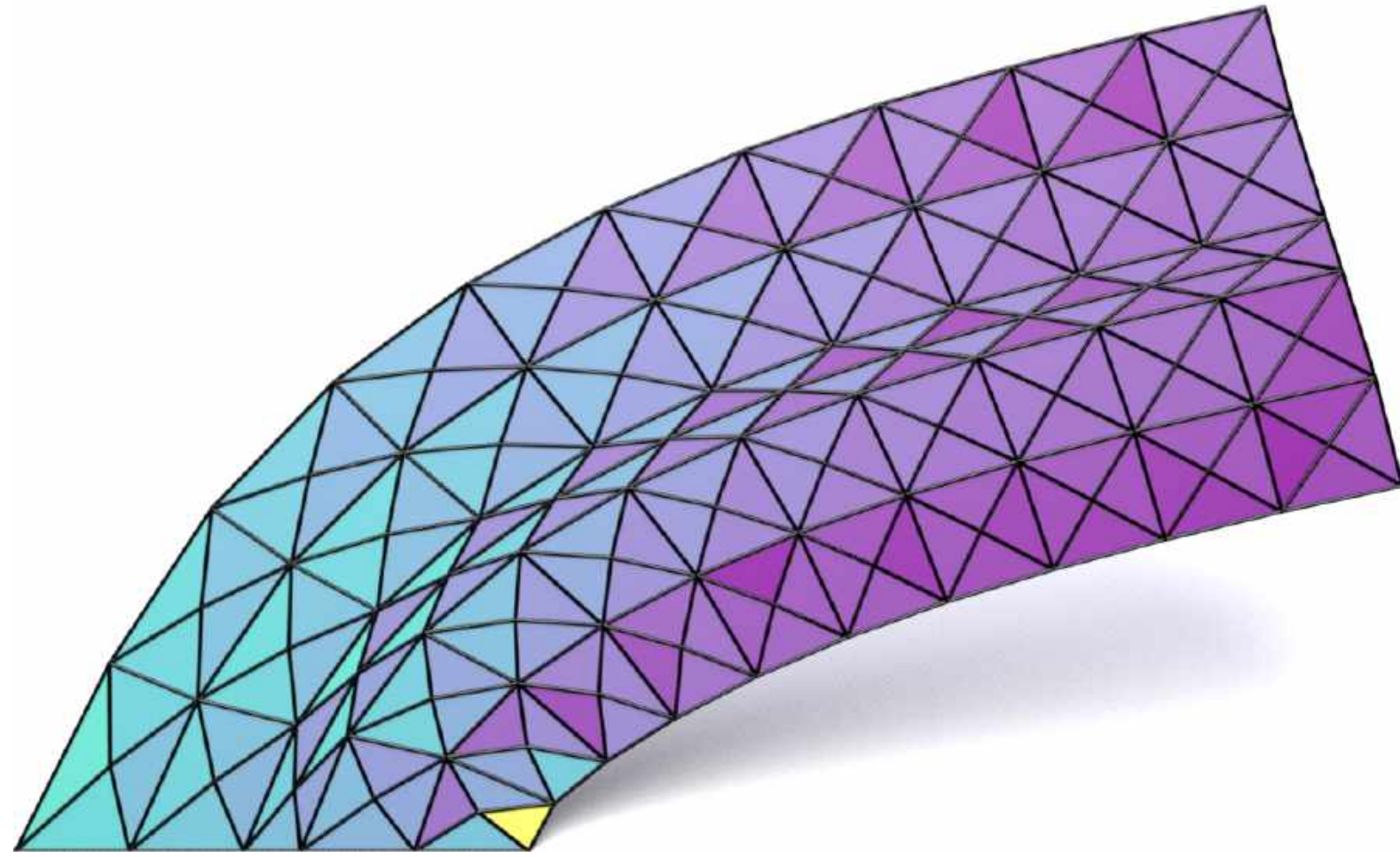


Our

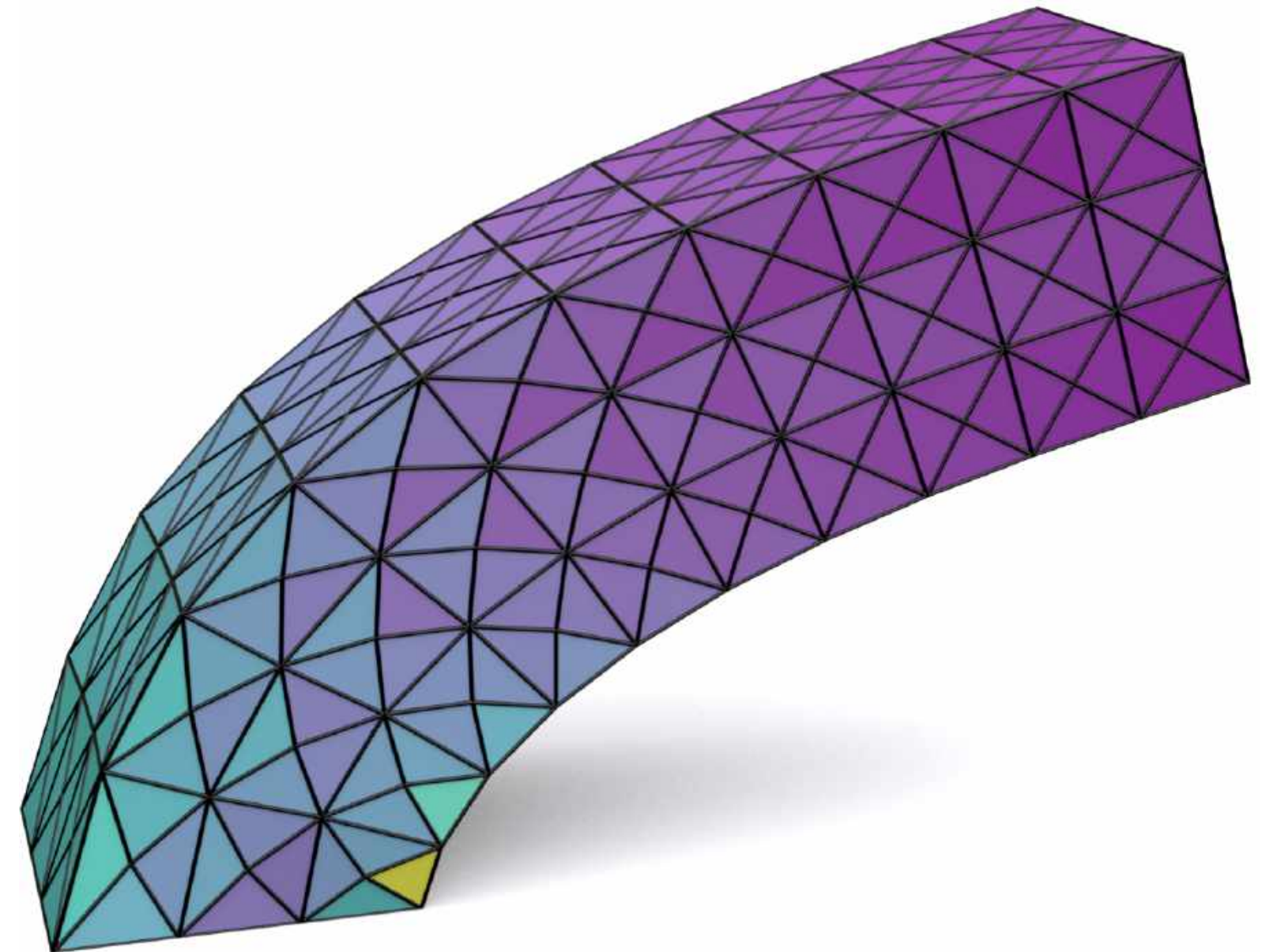
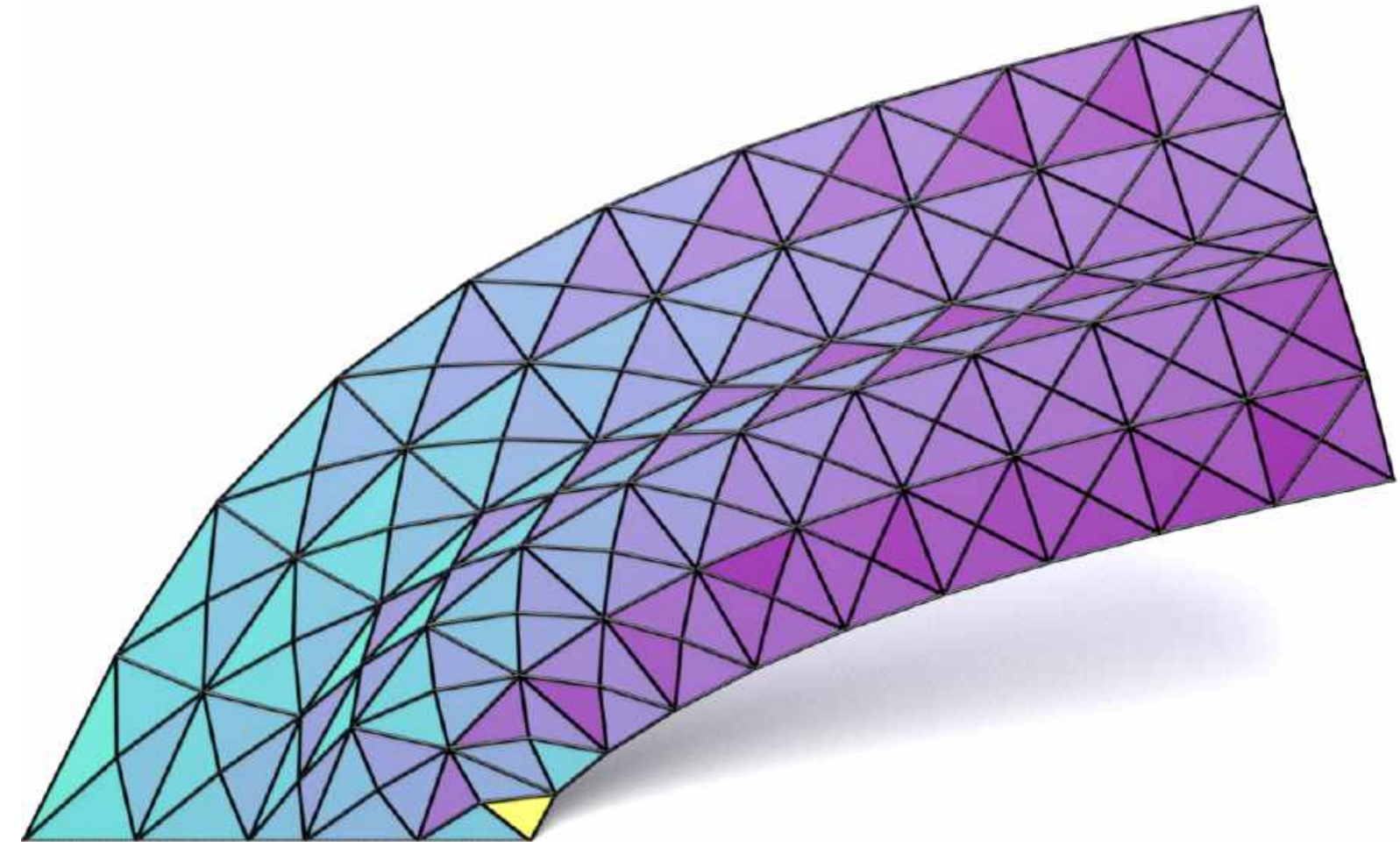


# Neo-Hookean Elasticity

Standard



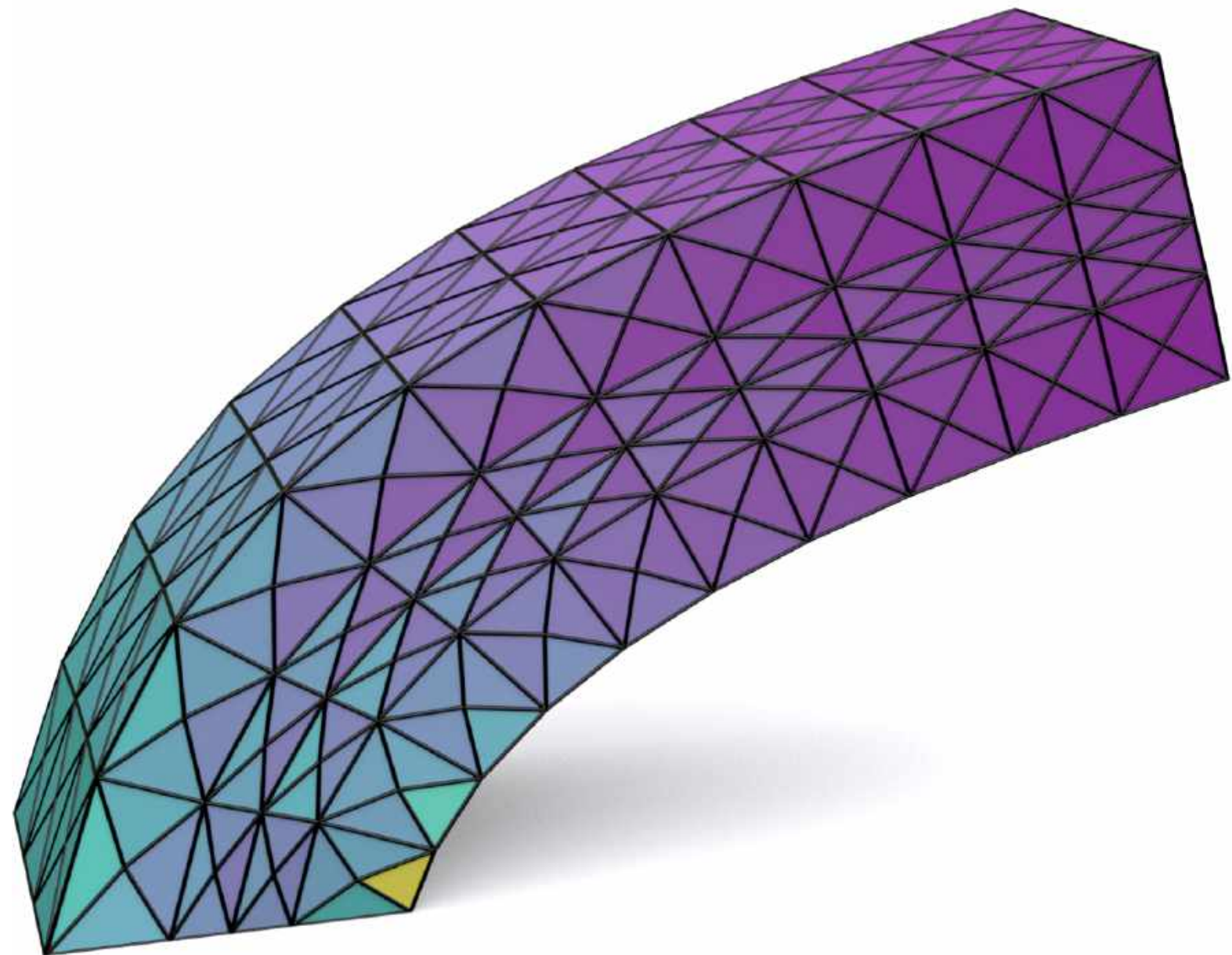
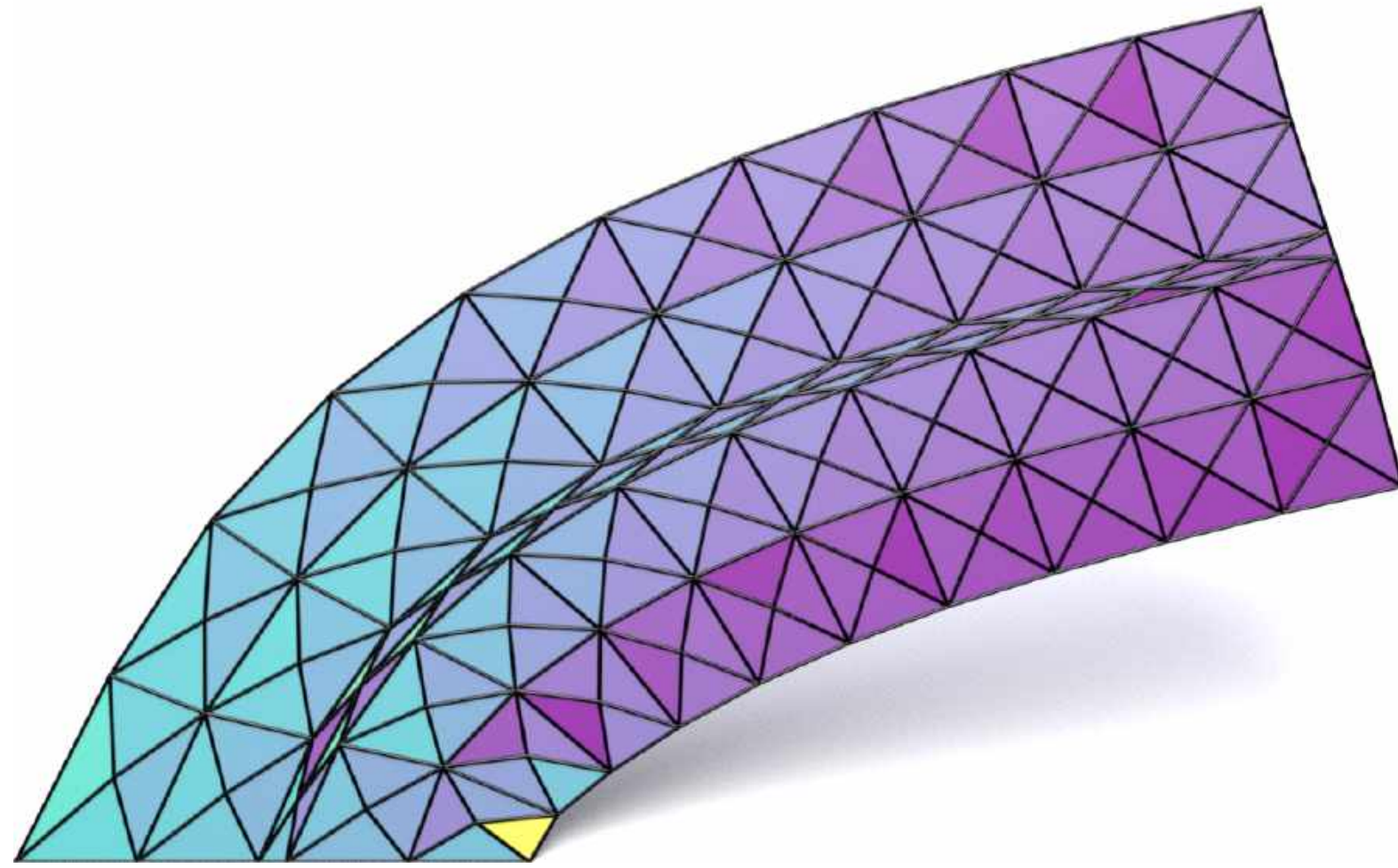
Our



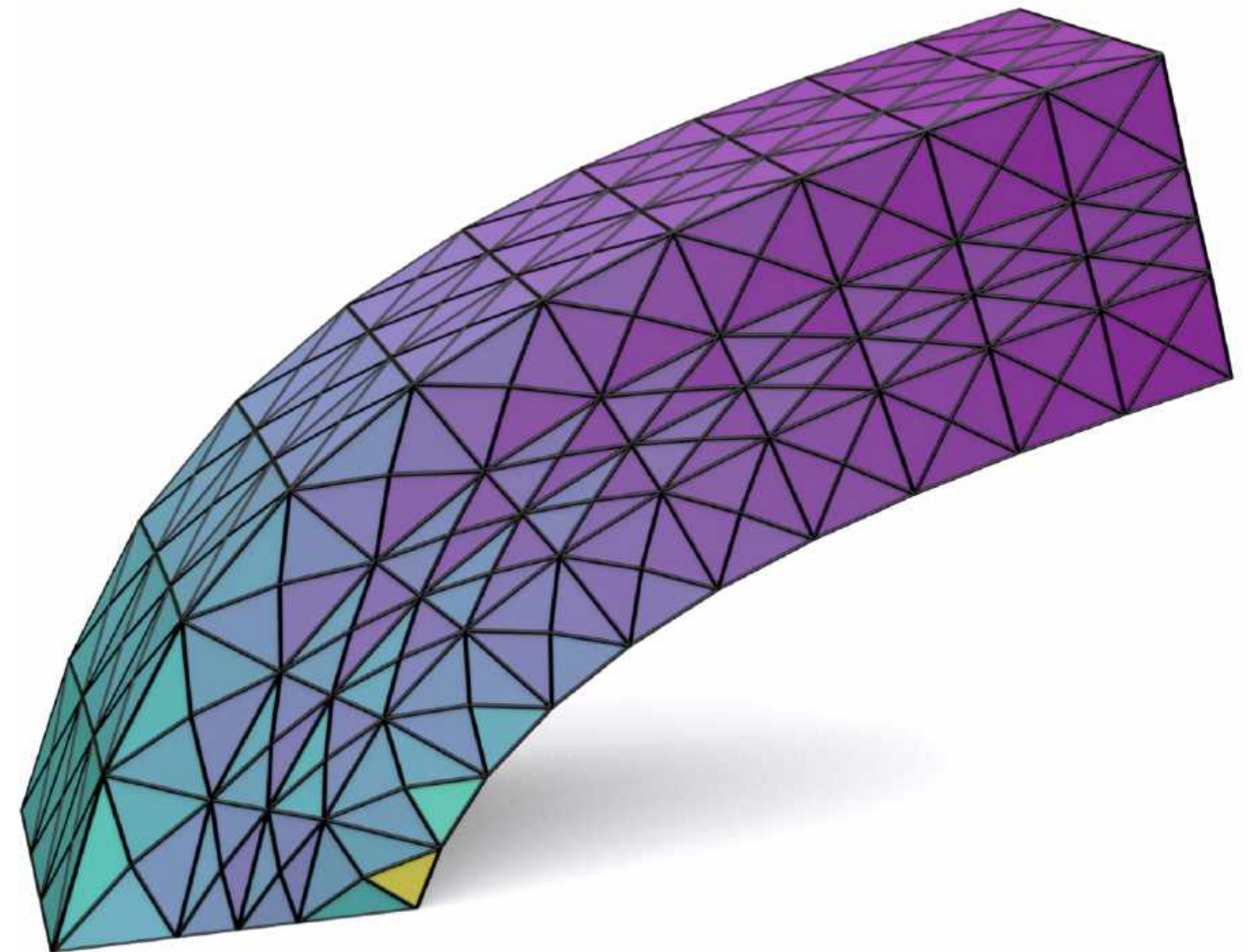
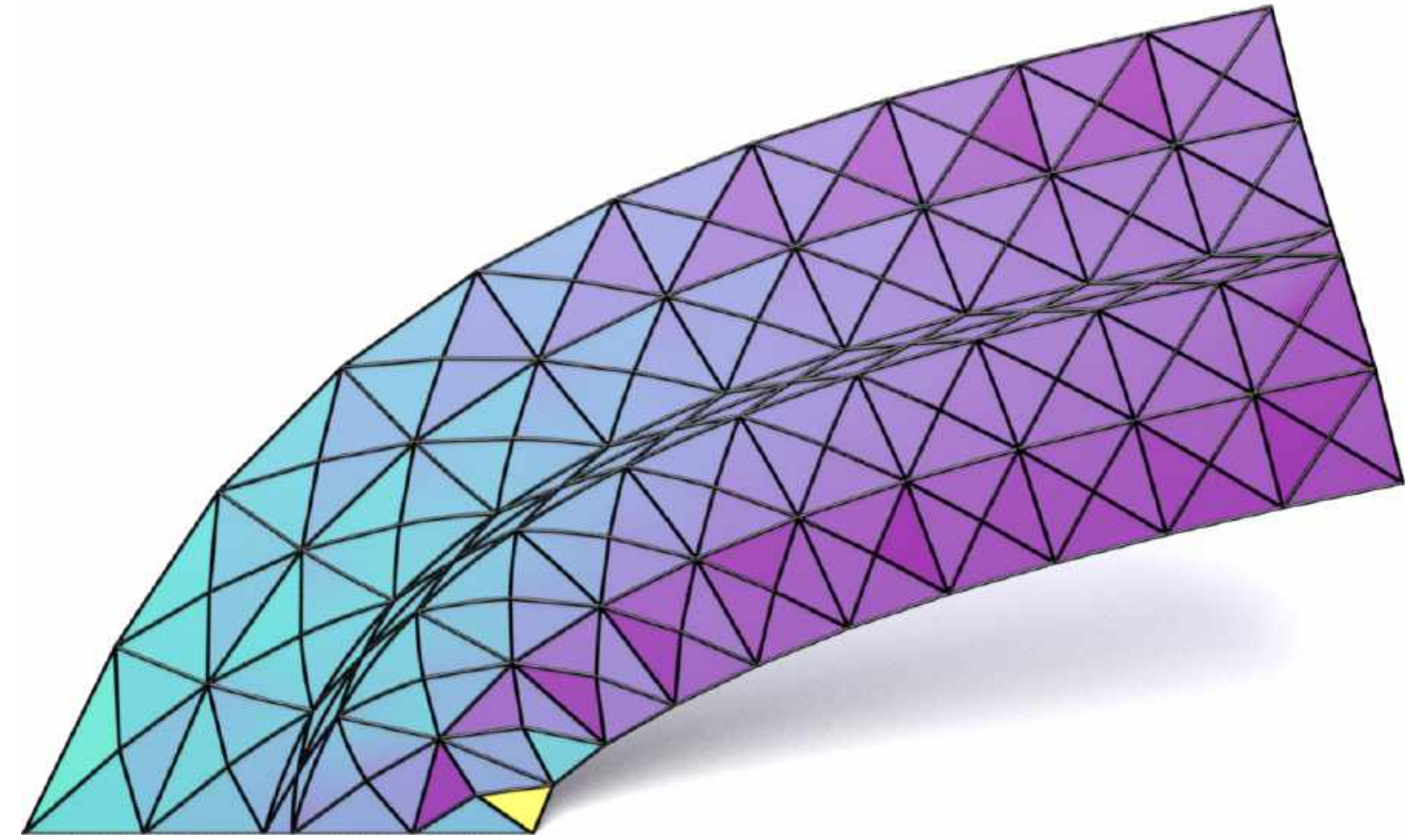


# Neo-Hookean Elasticity

Standard



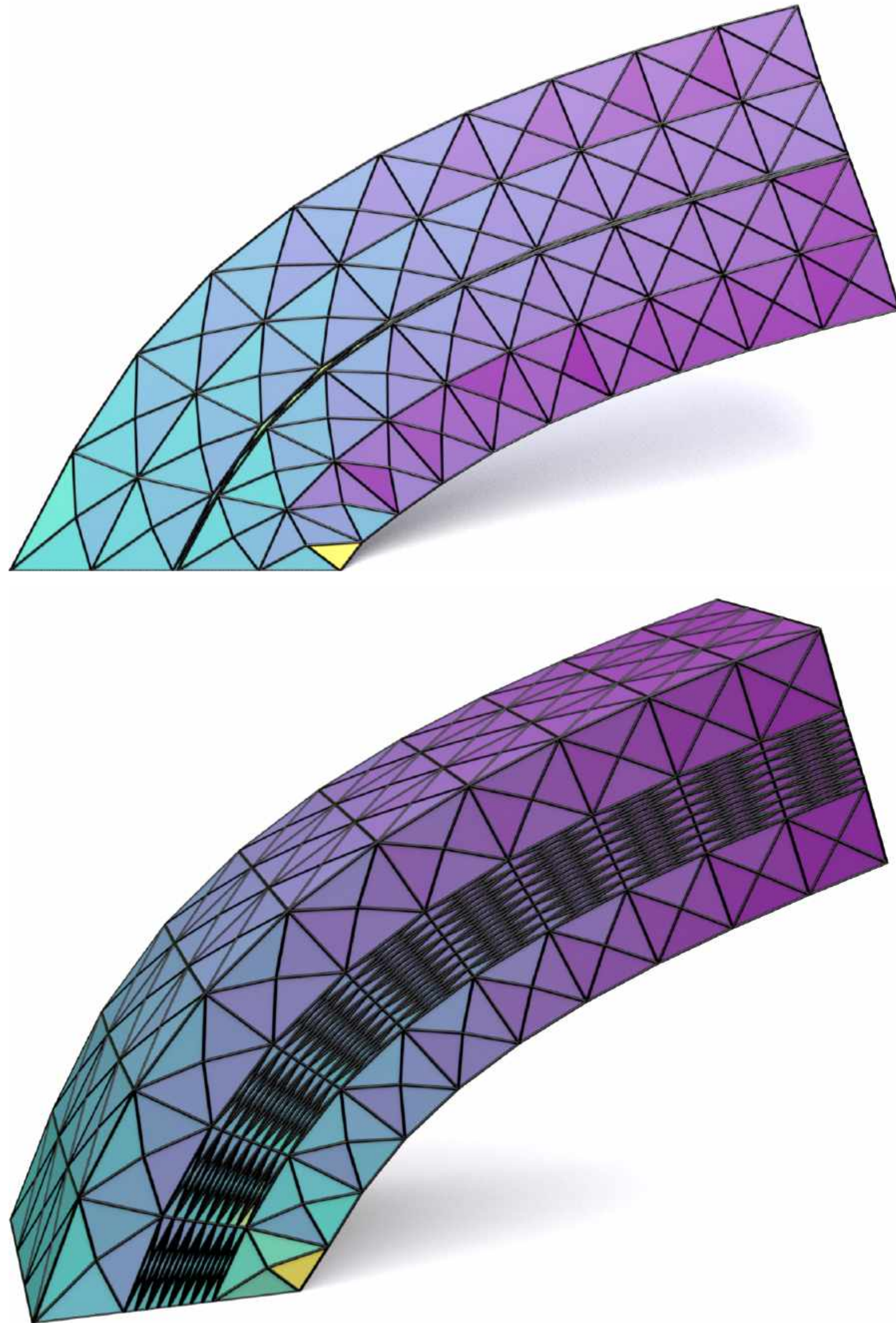
Our



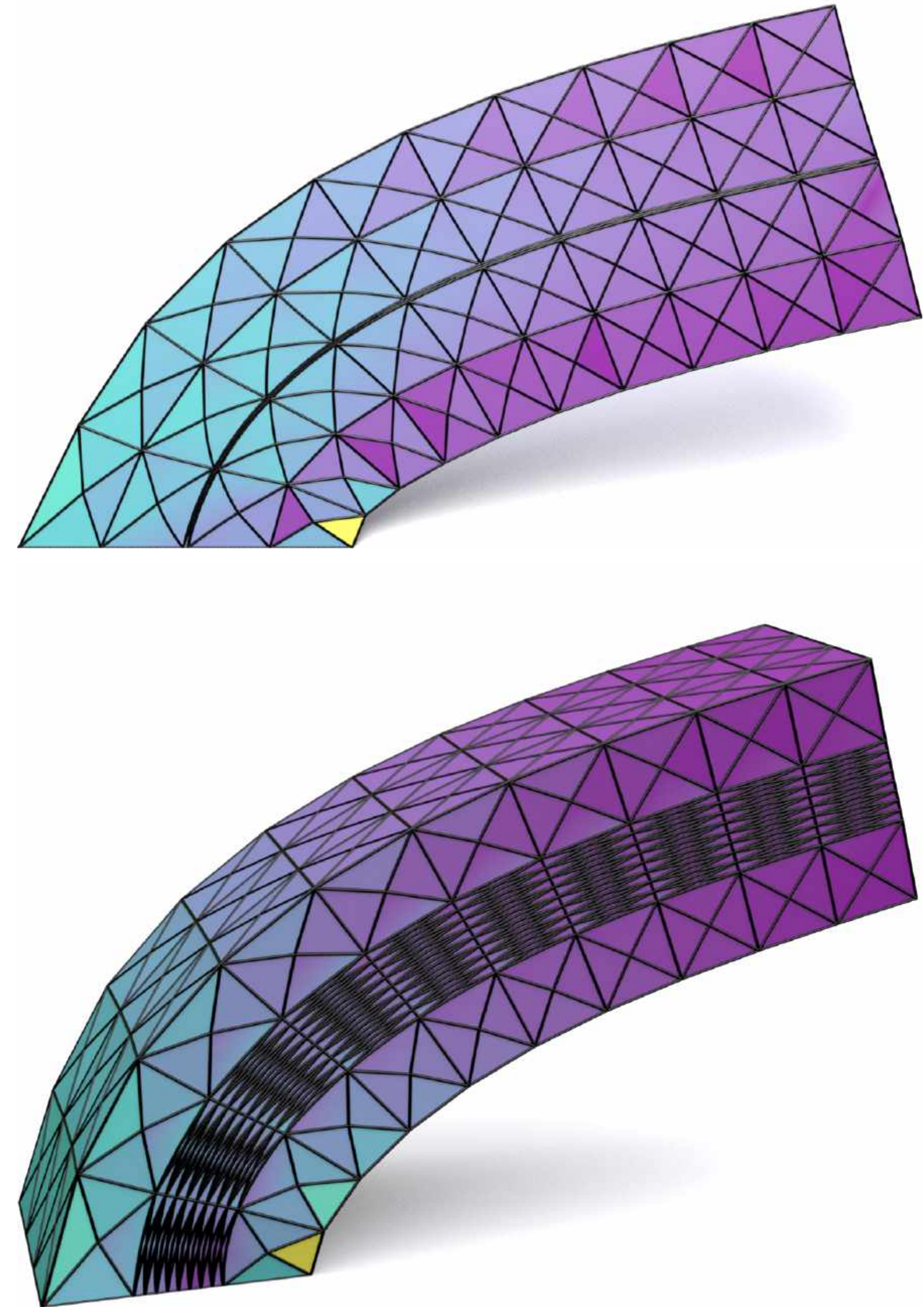


# Neo-Hookean Elasticity

Standard



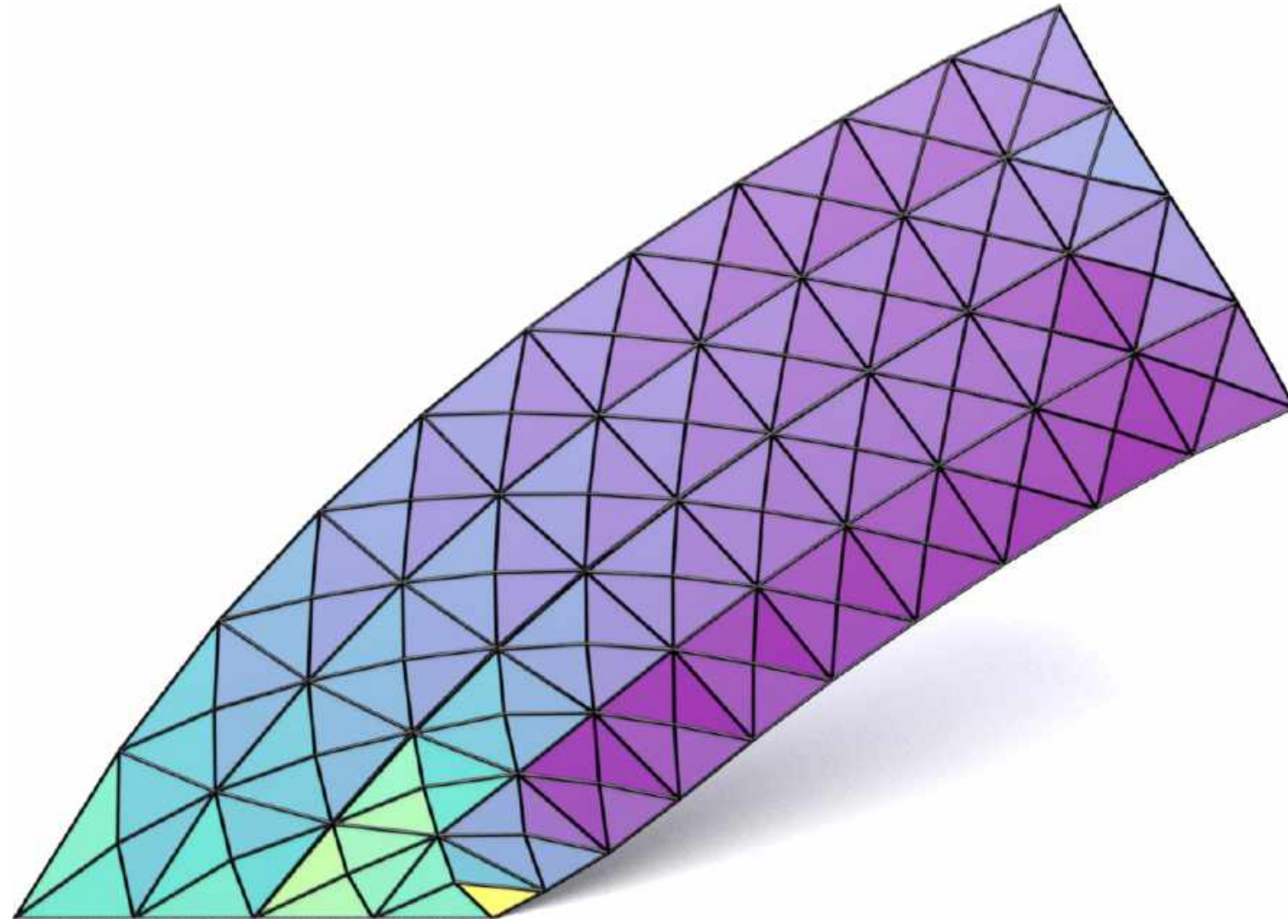
Our



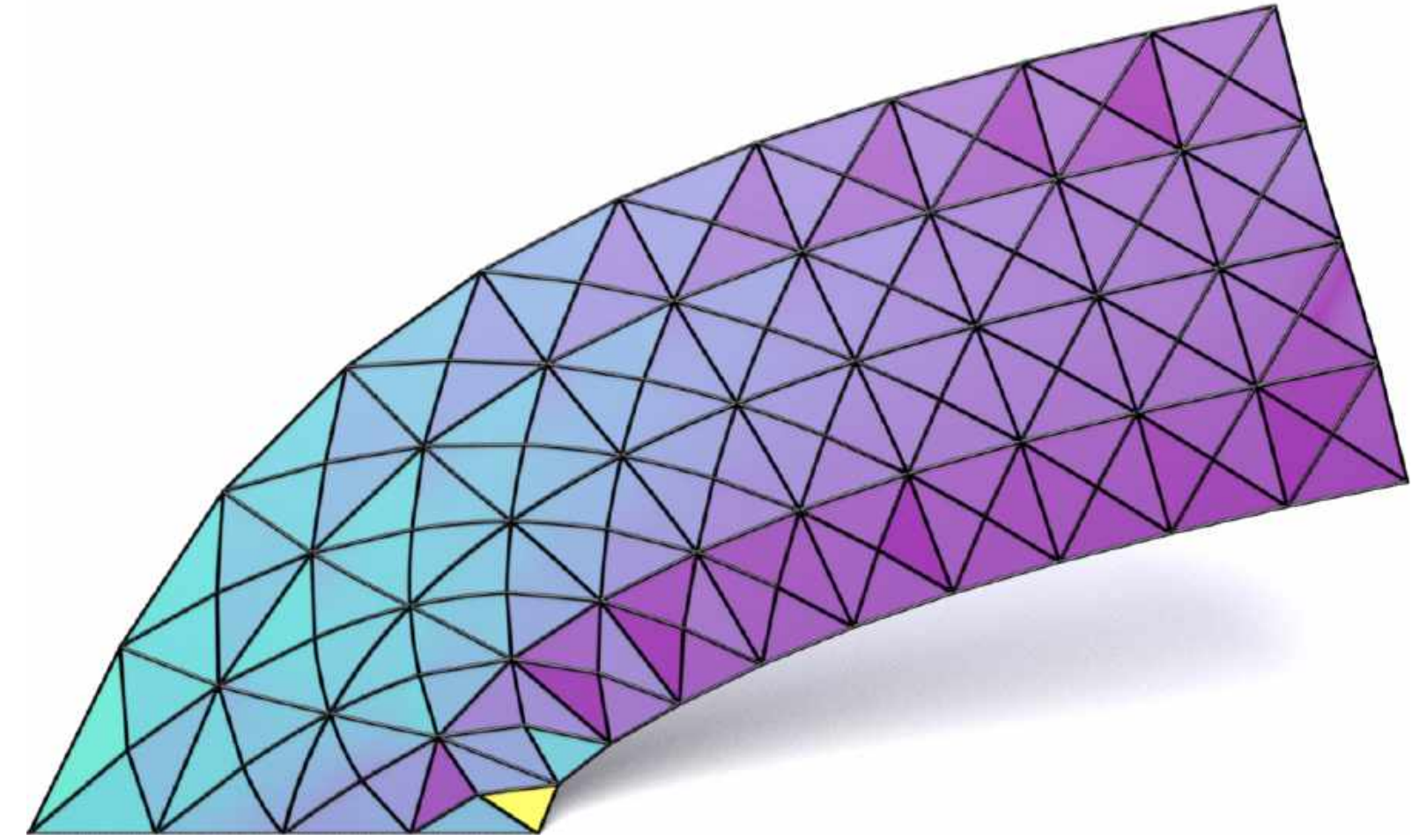


# Neo-Hookean Elasticity

Standard

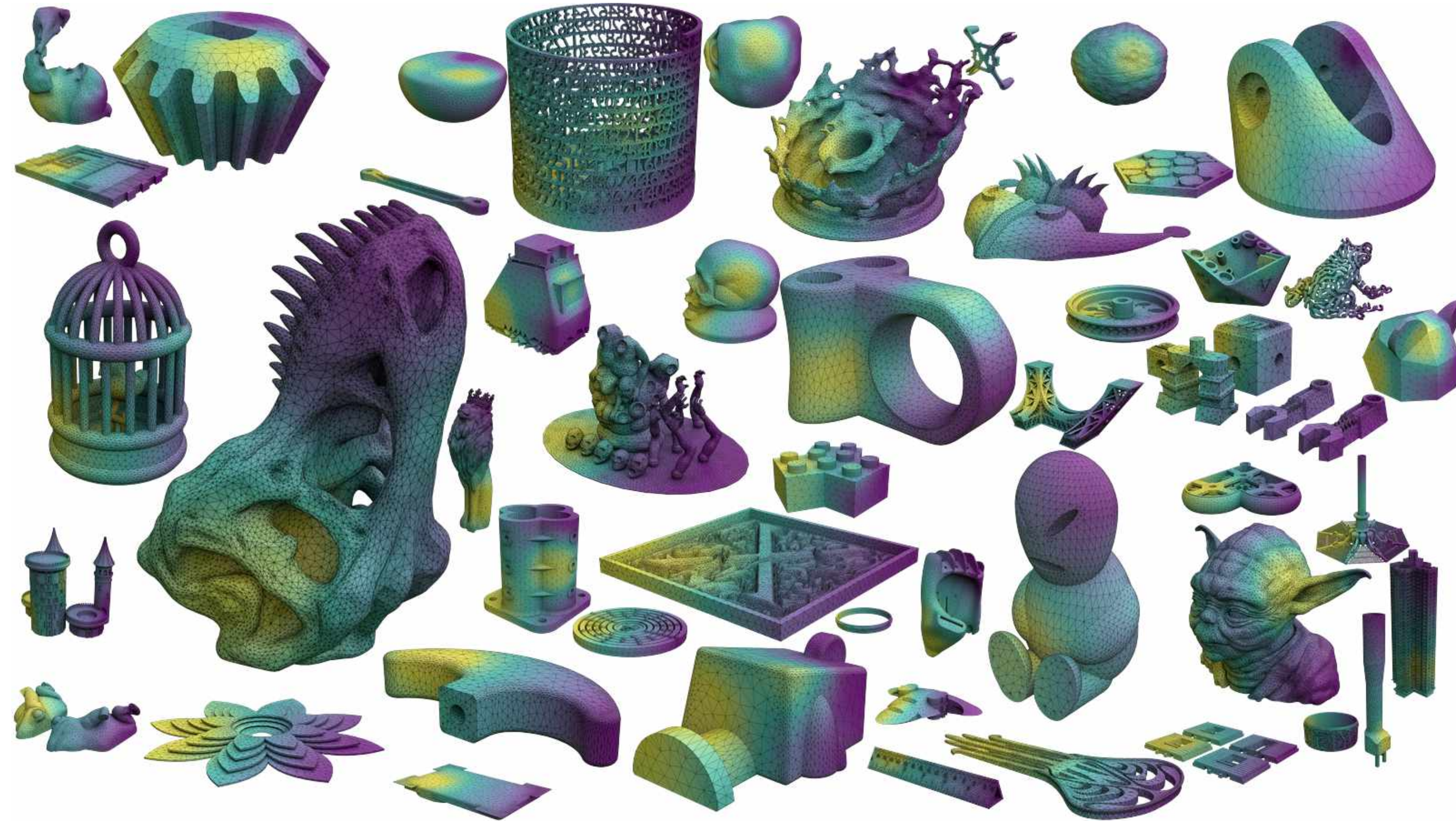


Our





# Large Dataset





## A large collection of 3D printed objects, including a dinosaur skull, a birdcage, a Yoda head, and various mechanical parts, all rendered with a rainbow color gradient. The objects are scattered across the image, showcasing a wide variety of shapes and sizes. Some objects are simple geometric shapes, while others are more complex, like the dinosaur skull and the Yoda head. The rainbow color gradient is applied to all objects, giving them a vibrant, multi-colored appearance. The objects are arranged in a way that fills the entire frame, with some objects overlapping others. The background is plain white, which makes the colorful objects stand out. The overall image is a collage of 3D printed items, demonstrating the versatility of 3D printing technology.

- Thingi10k  
[Zhou 17]











# How to Measure Errors?

$$e_h = \|u - u_h\|_0 \leq Ch^2 \|u\|_2$$

- Standard  $L_2$  error estimate for linear elements



# How to Measure Errors?

- Standard  $L_2$  error estimate for linear elements

$$e_h = \|u - u_h\|_0 \leq Ch^2 \|u\|_2$$

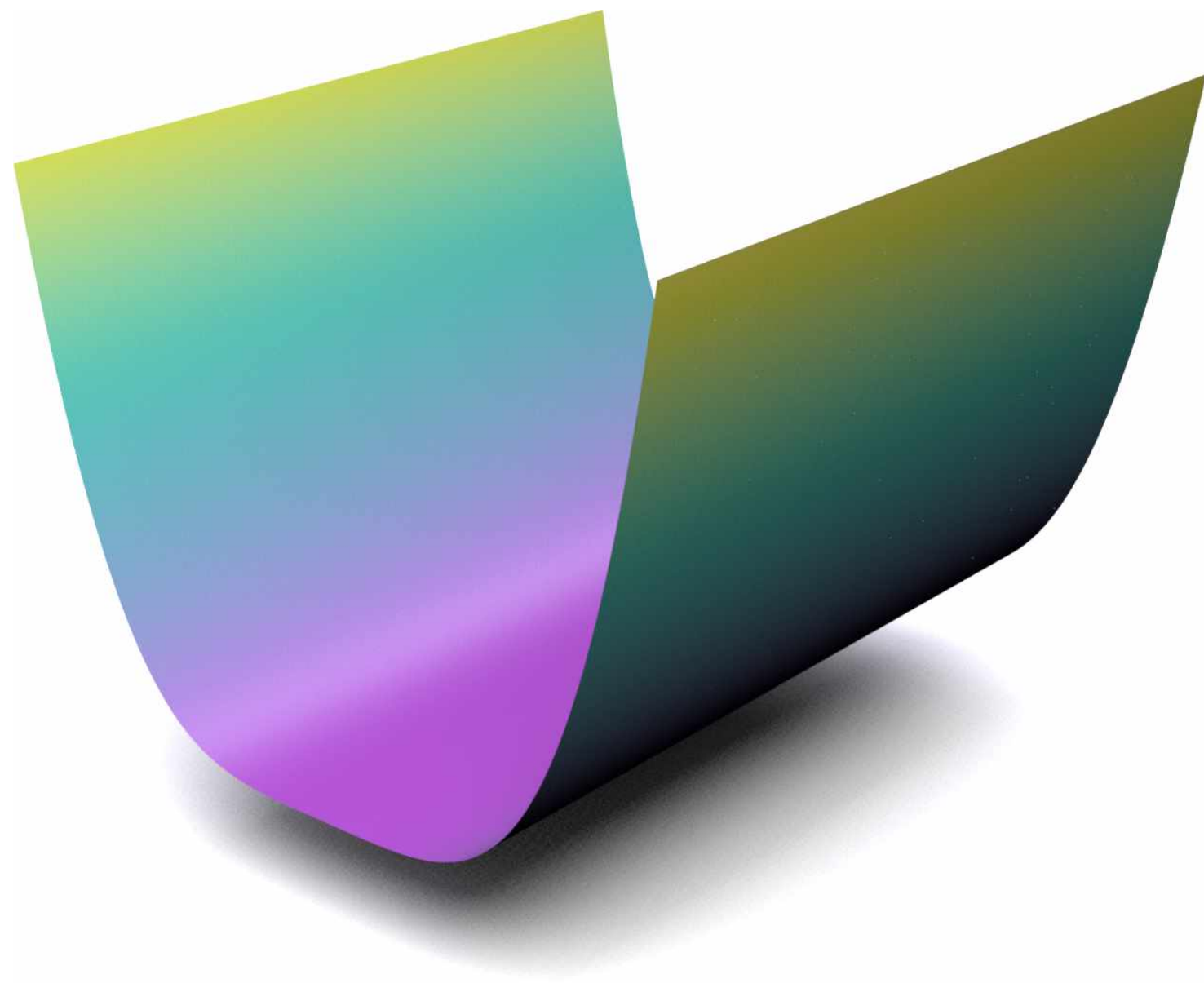
$L_2$  norm or average error



# FEM Error Estimate

- Standard  $L_2$  error estimate for linear elements

$$e_h = \|\underbrace{u}_{\text{Exact solution}} - u_h\|_0 \leq Ch^2 \|u\|_2$$



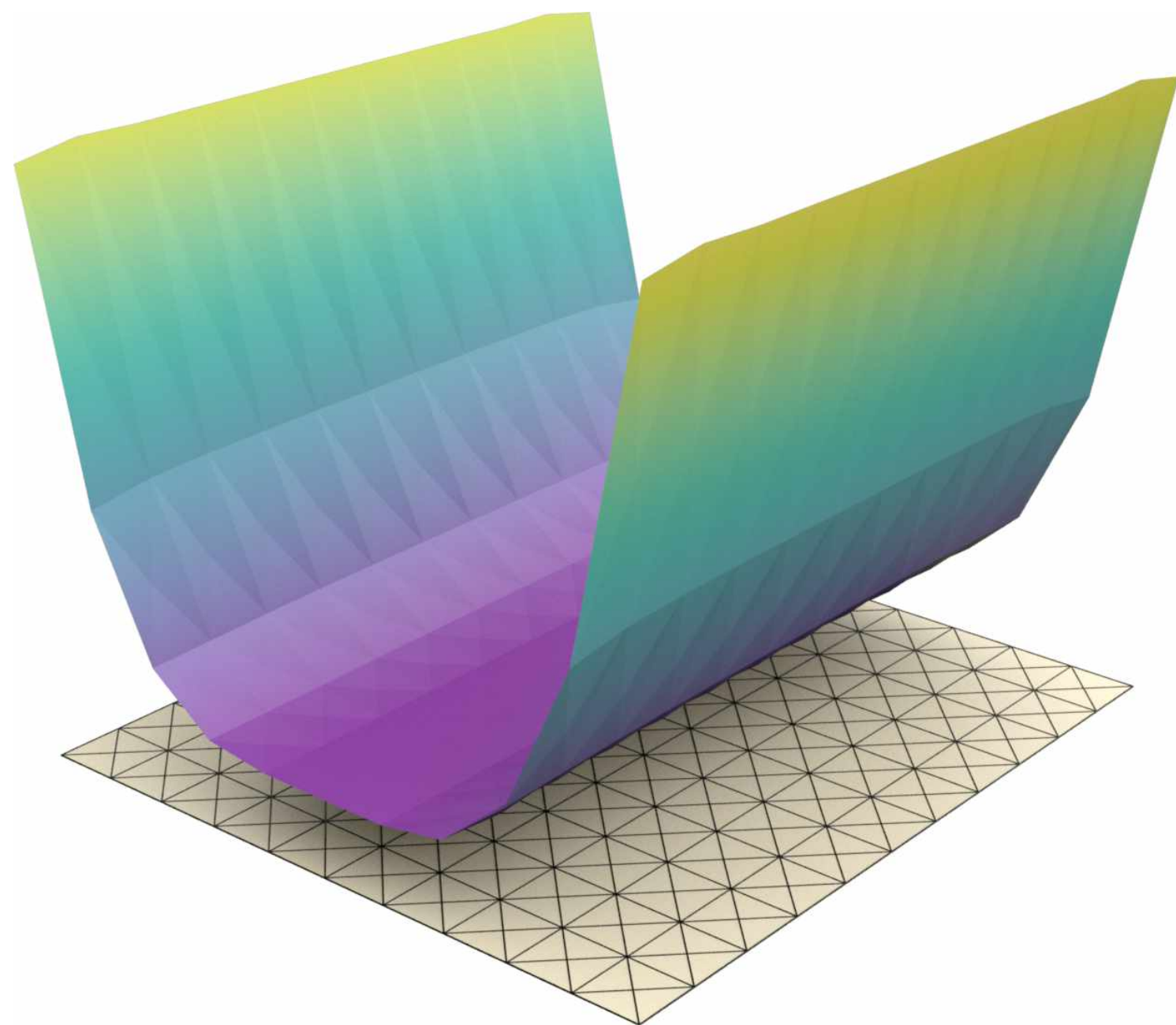


# FEM Error Estimate

- Standard  $L_2$  error estimate for linear elements

$$e_h = \|u - u_h\|_0 \leq Ch^2 \|u\|_2$$

Approximated solution





# How to Measure Errors?

- Standard  $L_2$  error estimate for linear elements

$$e_h = \|u - u_h\|_0 \leq C h^2 \|u\|_2$$

- Different  $h$  for every model!



# How to Measure Errors?

- Standard  $L_2$  error estimate for linear elements

$$e_h = \|u - u_h\|_0 \leq Ch^2 \|u\|_2$$

- Different  $h$  for every model!
- $L_2$  Efficiency

$$E_{L_2} = \frac{\|u - u_h\|_0}{h^2}$$



# How to Measure Errors?

- Standard  $L_2$  error estimate for linear elements

$$e_h = \|u - u_h\|_0 \leq Ch^2 \|u\|_2$$

- Different  $h$  for every model!
- $L_2$  Efficiency

$$E_{L_2} = \frac{\|u - u_h\|_0}{h^2}$$



# How to Measure Errors?

- Standard  $L_2$  error estimate for linear elements

$$e_h = \|u - u_h\|_0 \leq Ch^2 \|u\|_2$$

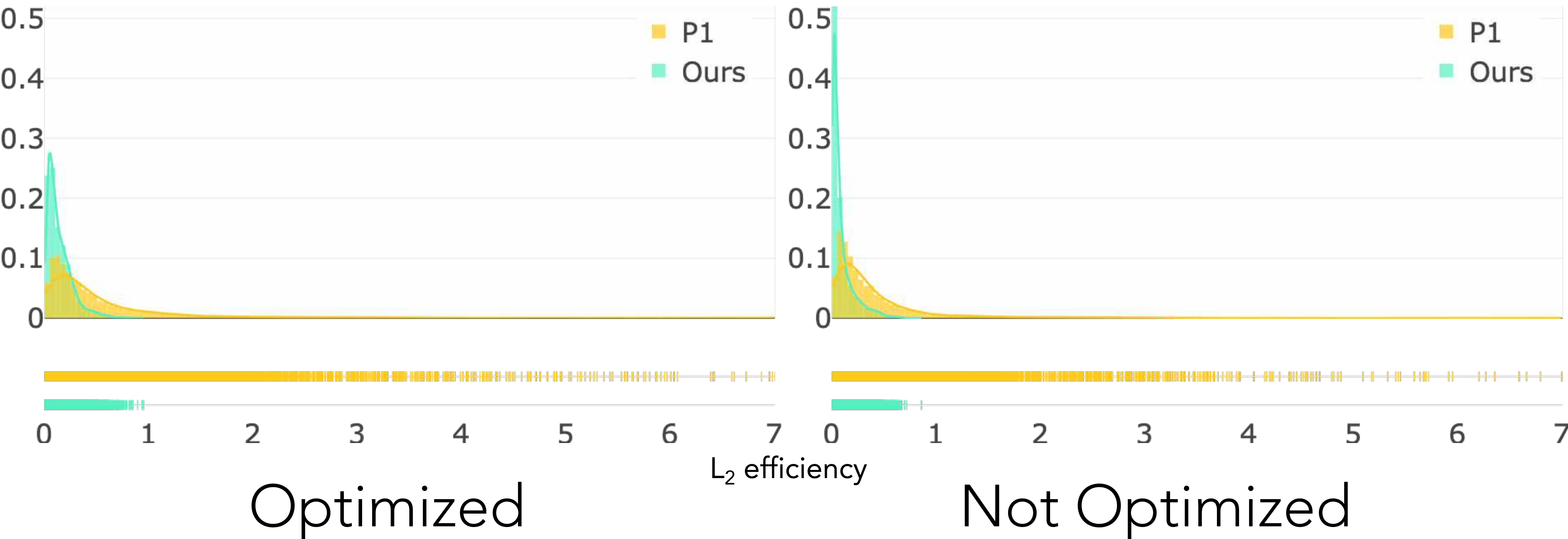
- Different  $h$  for every model!
- $L_2$  Efficiency

$$E_{L_2} = \frac{\|u - u_h\|_0}{h^2}$$

Small values are good!

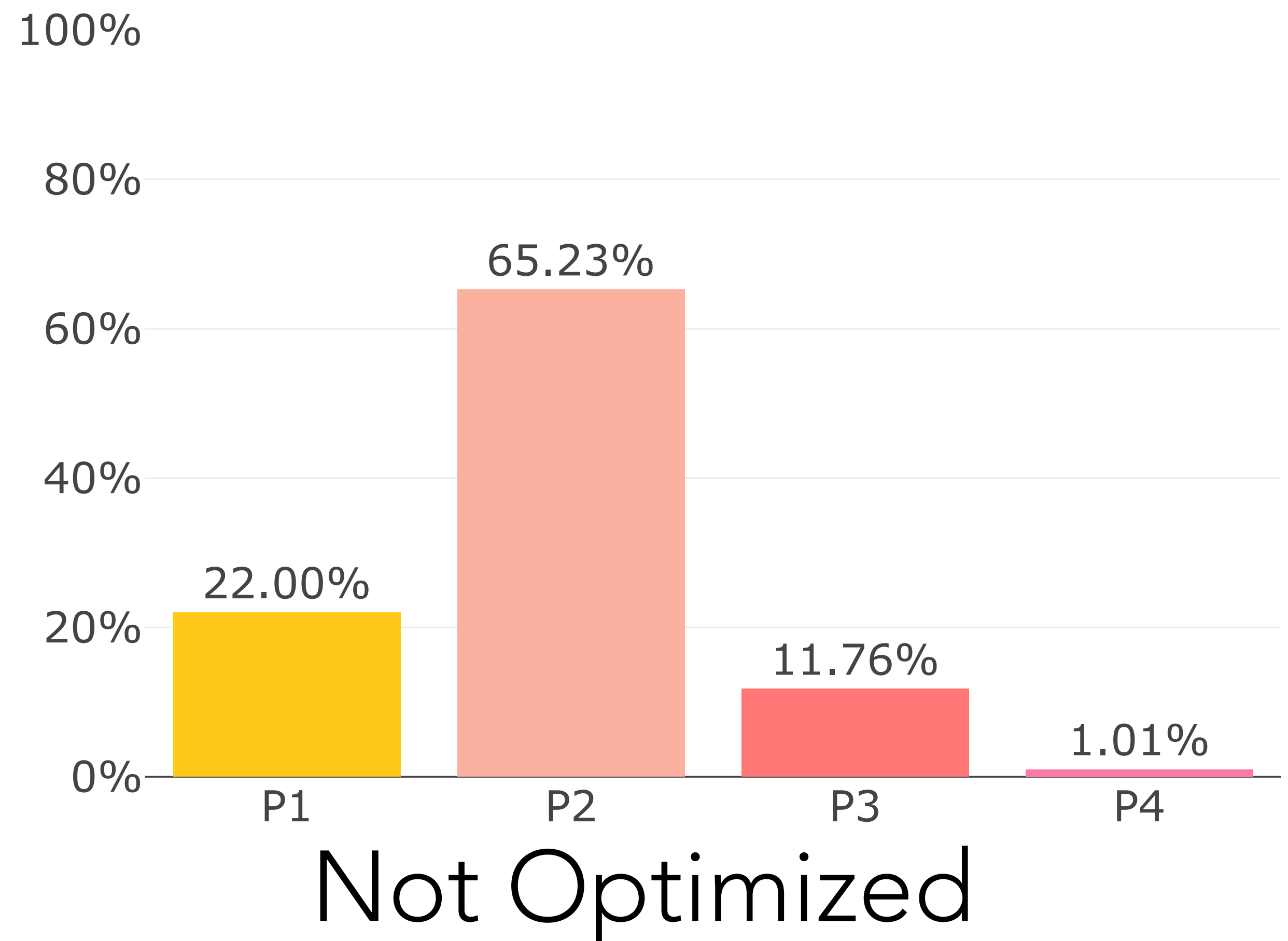
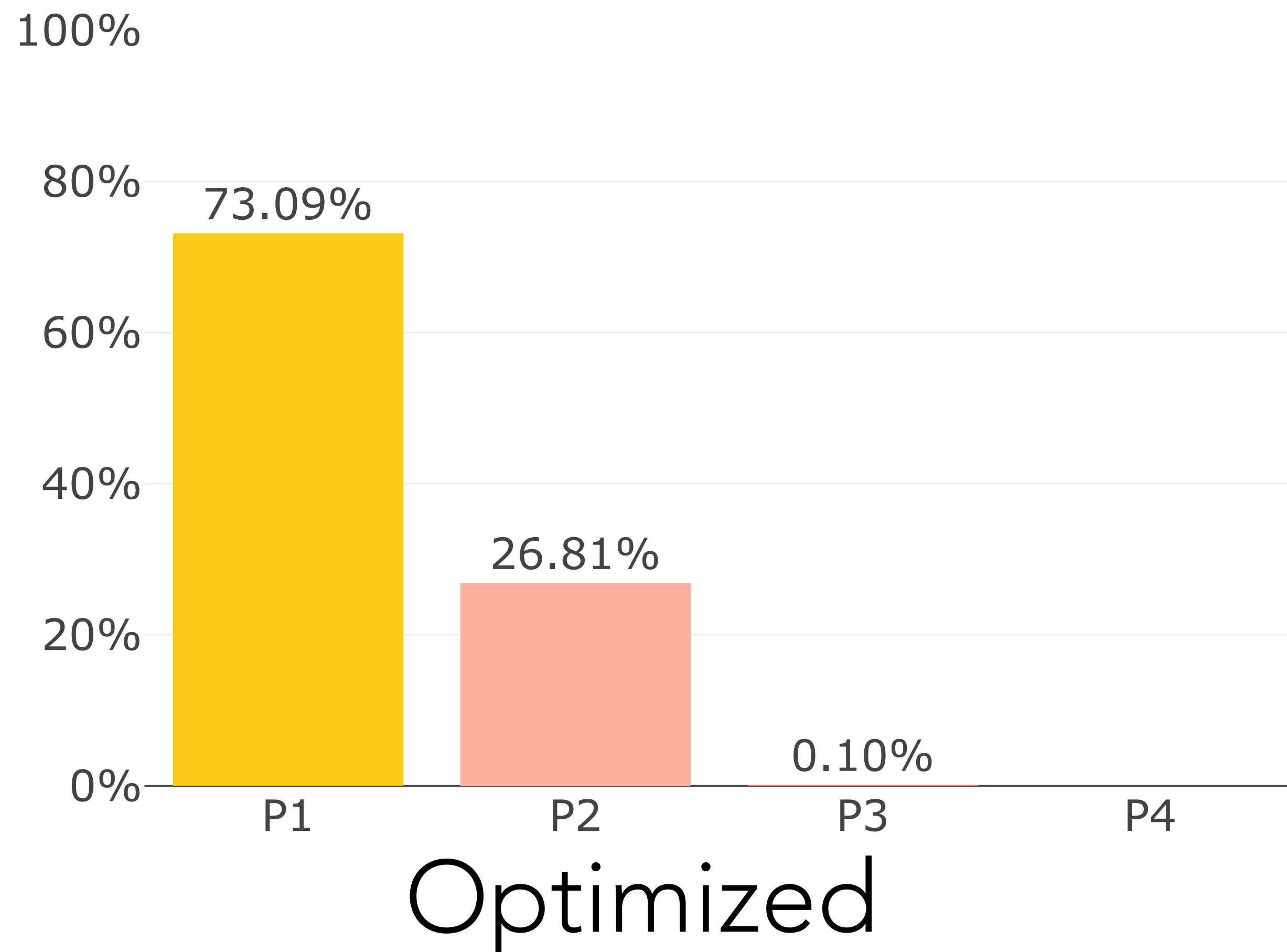


# Efficiency



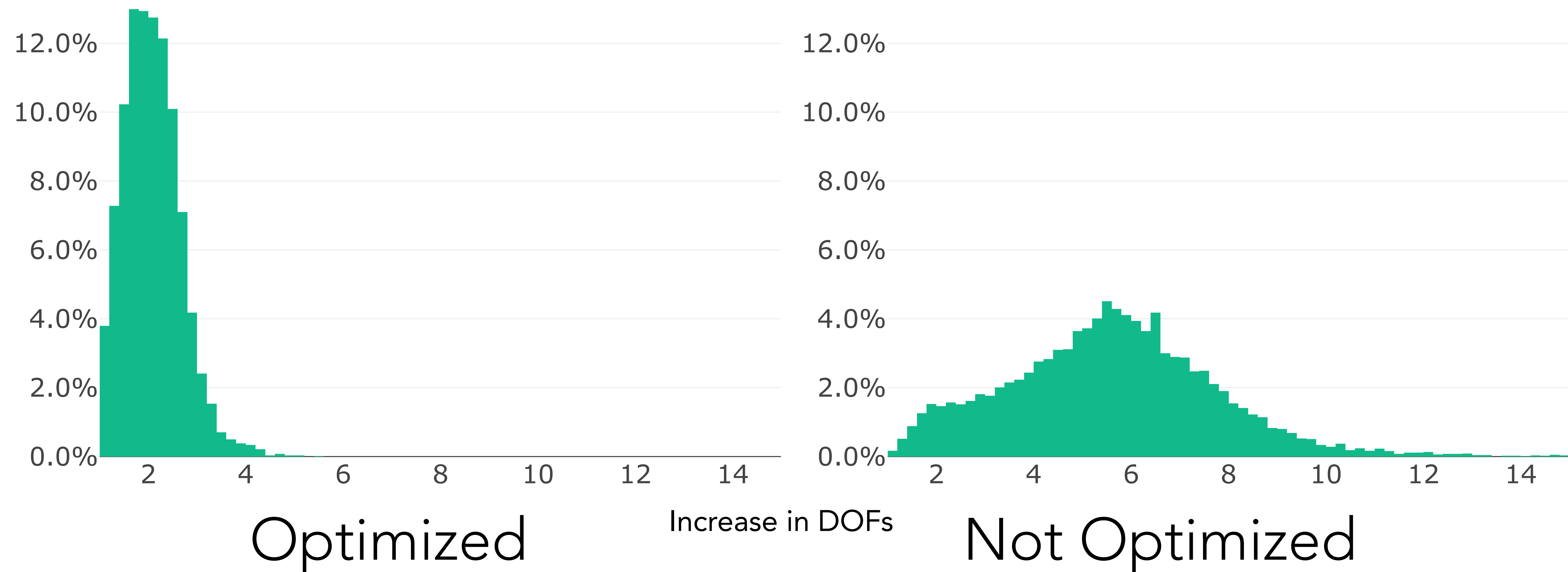


# Degree Distribution



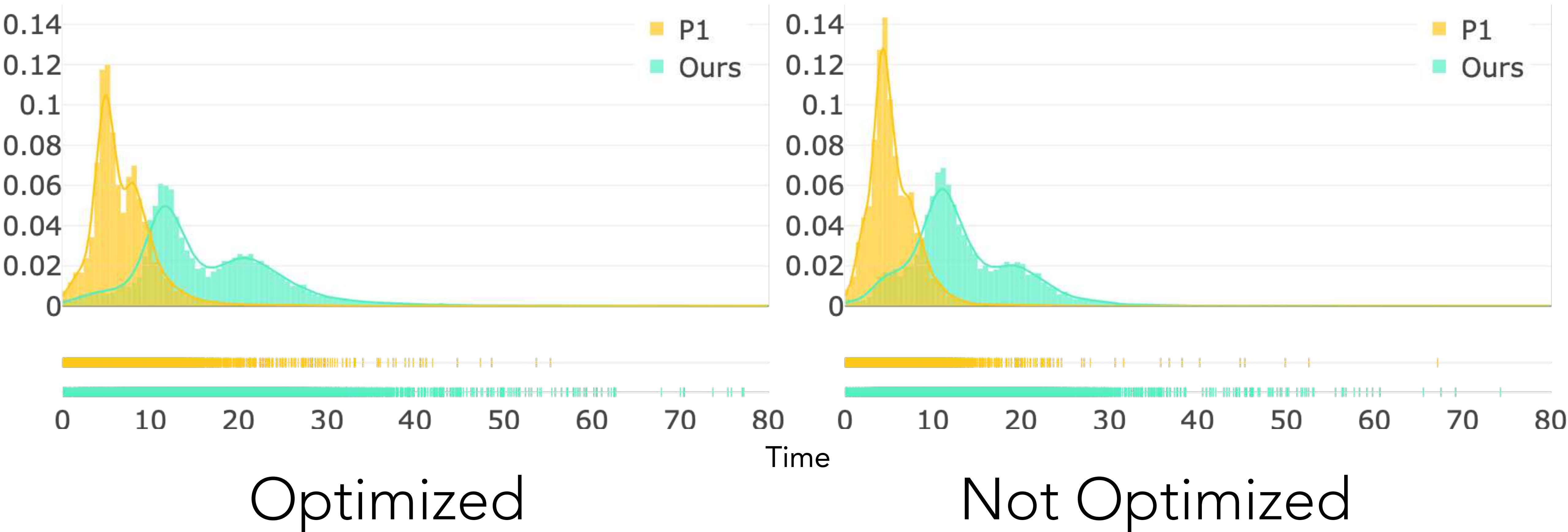


# Number of DOF



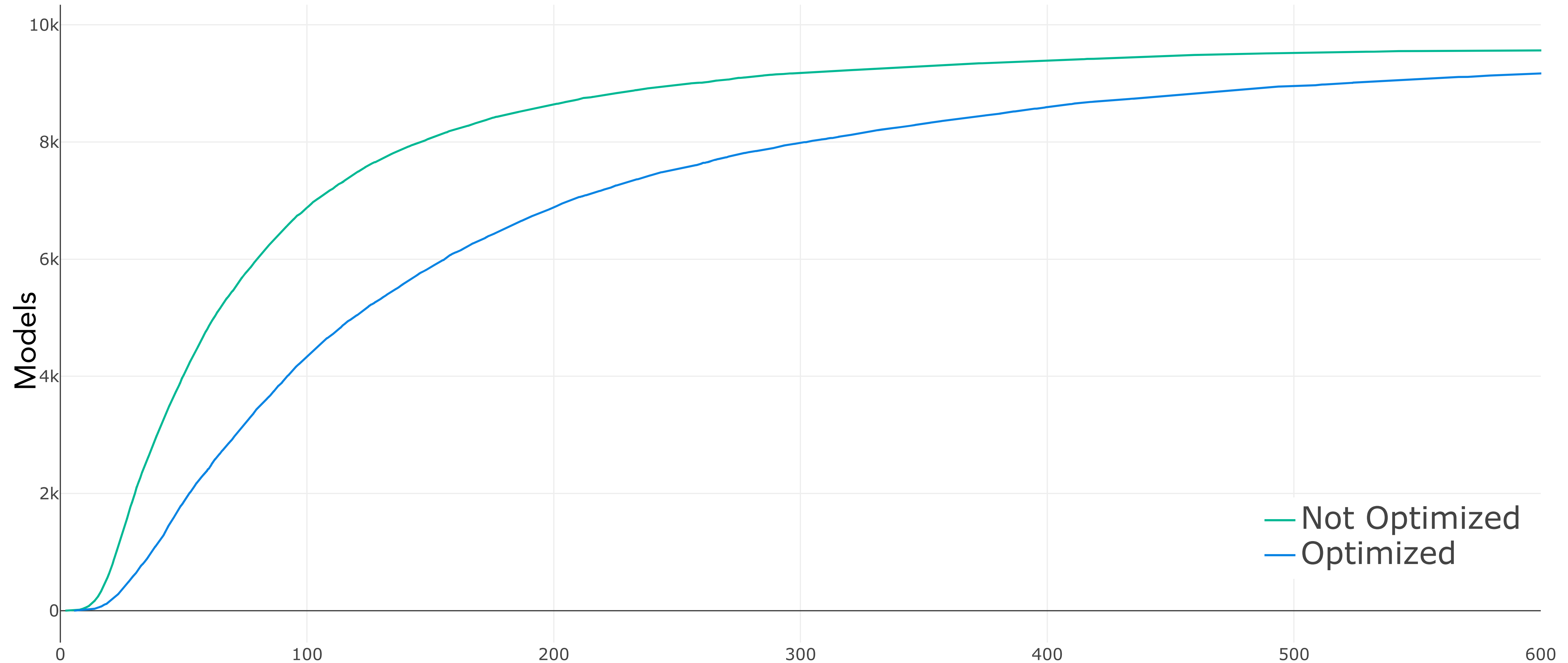


# Timings



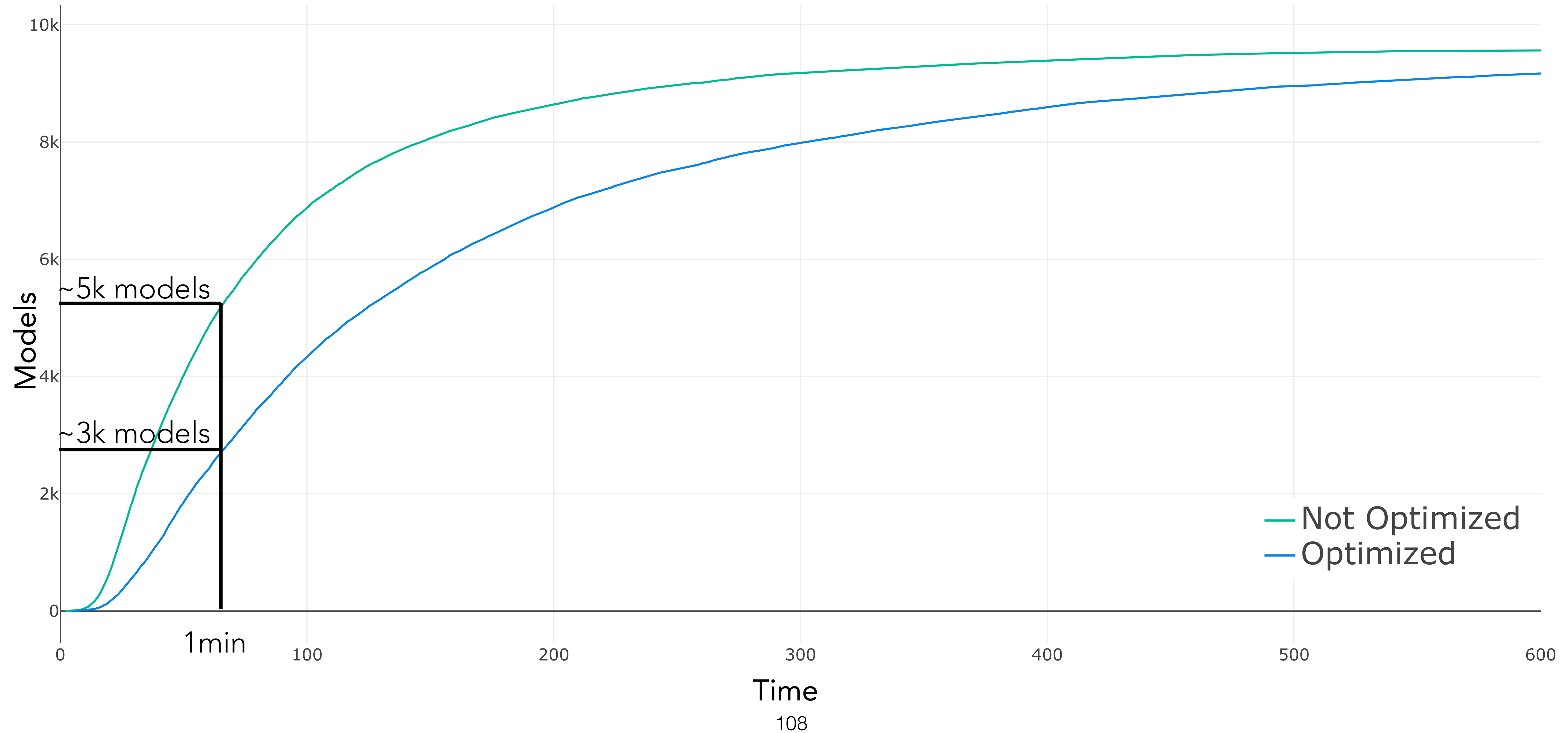


# Overall Time (Meshing + Simulation)





# Overall Time (Meshing + Simulation)





# Summary

$$k = \frac{\ln \left( B \hat{h}^{\hat{k}+1} \frac{\sigma_E^2}{\hat{\sigma}^2} \right) - \ln h_E}{\ln h_E}$$

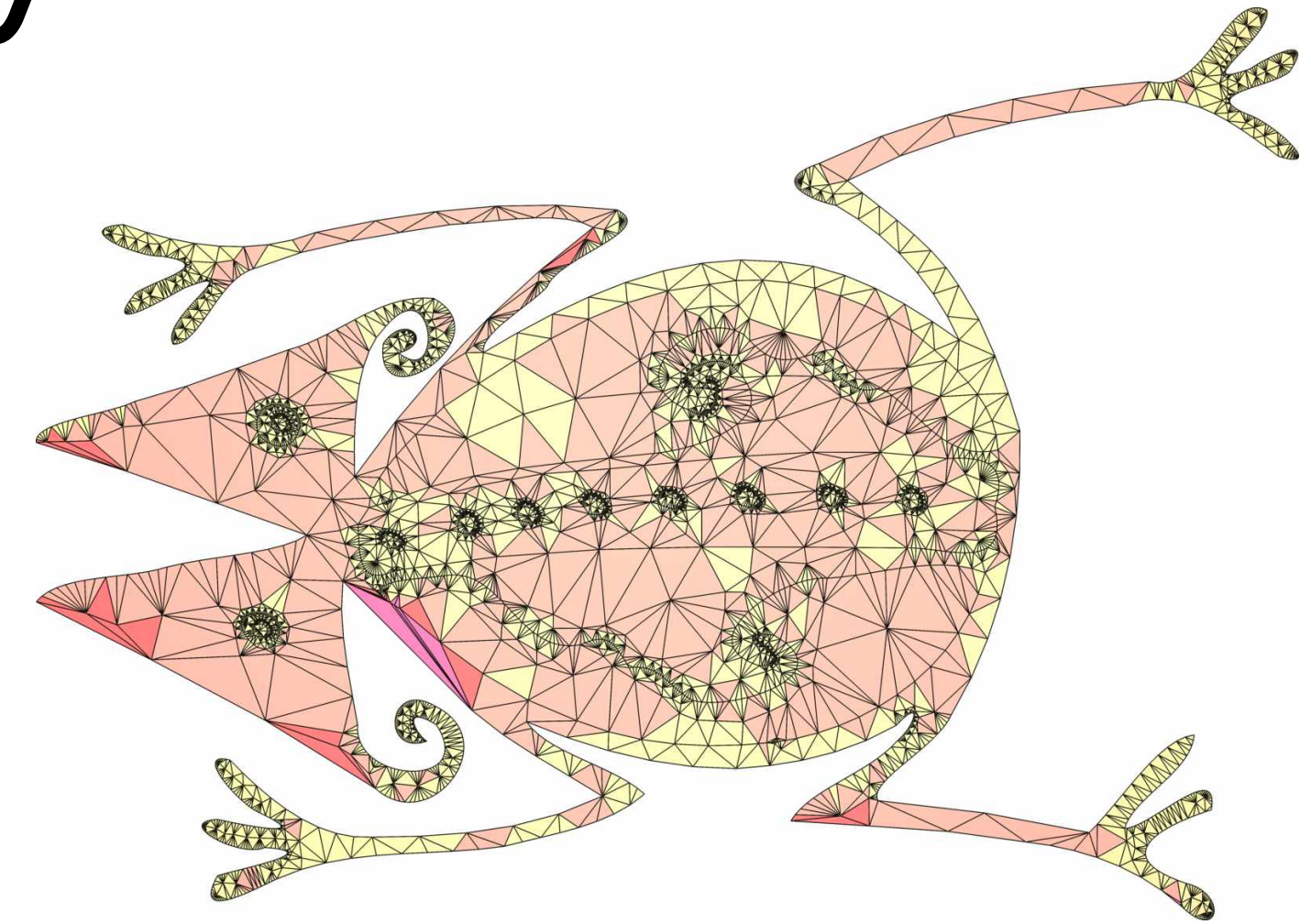
1. Use formula



# Summary

$$k = \frac{\ln \left( B \hat{h}^{\hat{k}+1} \frac{\sigma_E^2}{\hat{\sigma}^2} \right) - \ln h_E}{\ln h_E}$$

1. Use formula



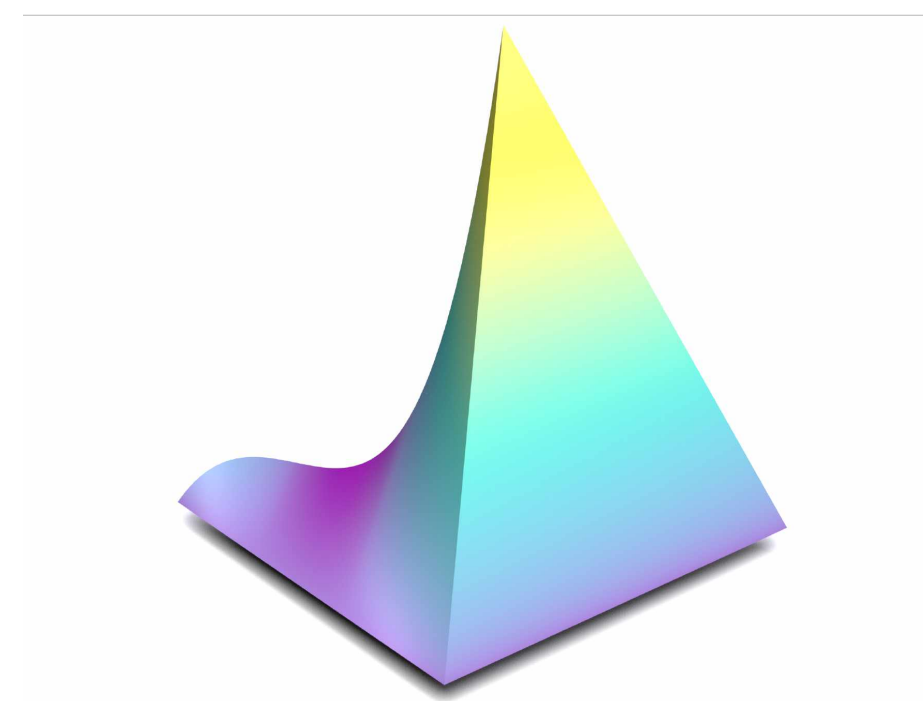
2. Propagate degrees



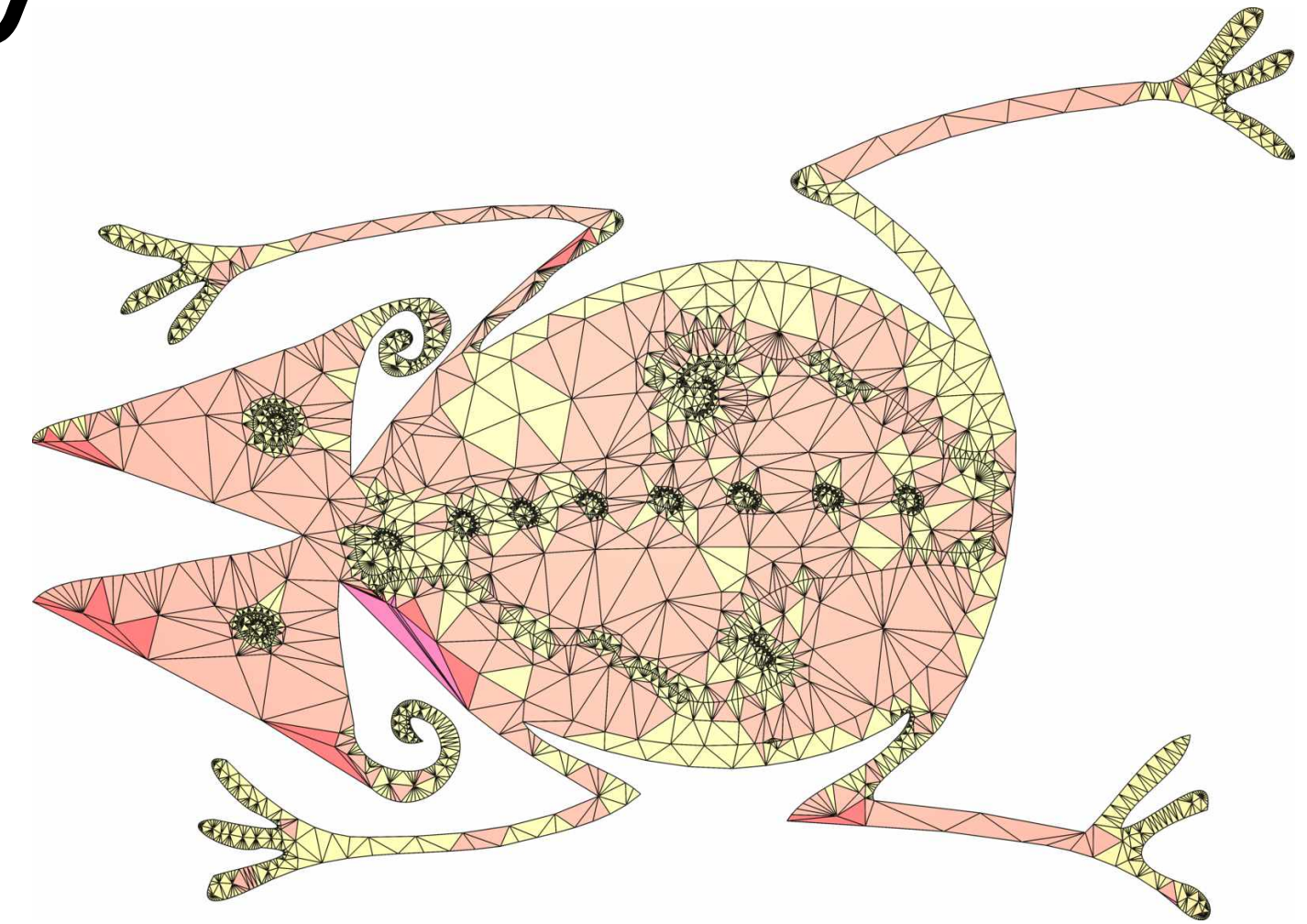
# Summary

$$k = \frac{\ln \left( B \hat{h}^{\hat{k}+1} \frac{\sigma_E^2}{\hat{\sigma}^2} \right) - \ln h_E}{\ln h_E}$$

1. Use formula



3. Construct  $C^0$  basis



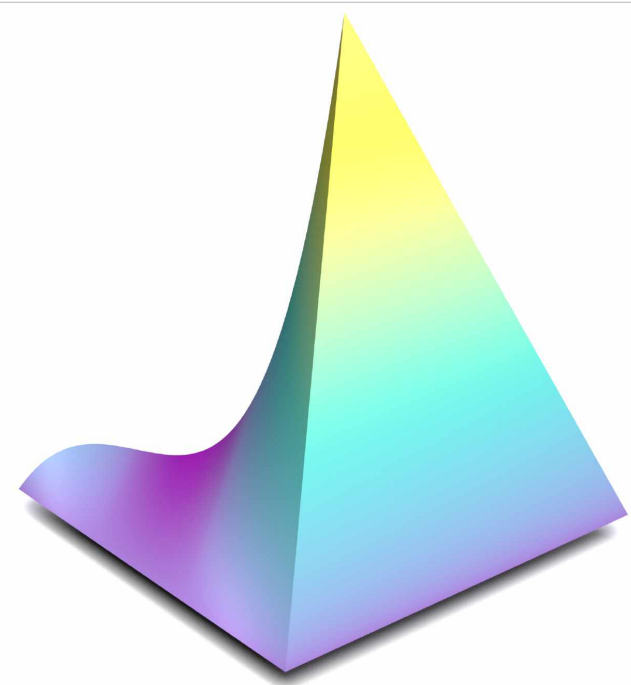
2. Propagate degrees



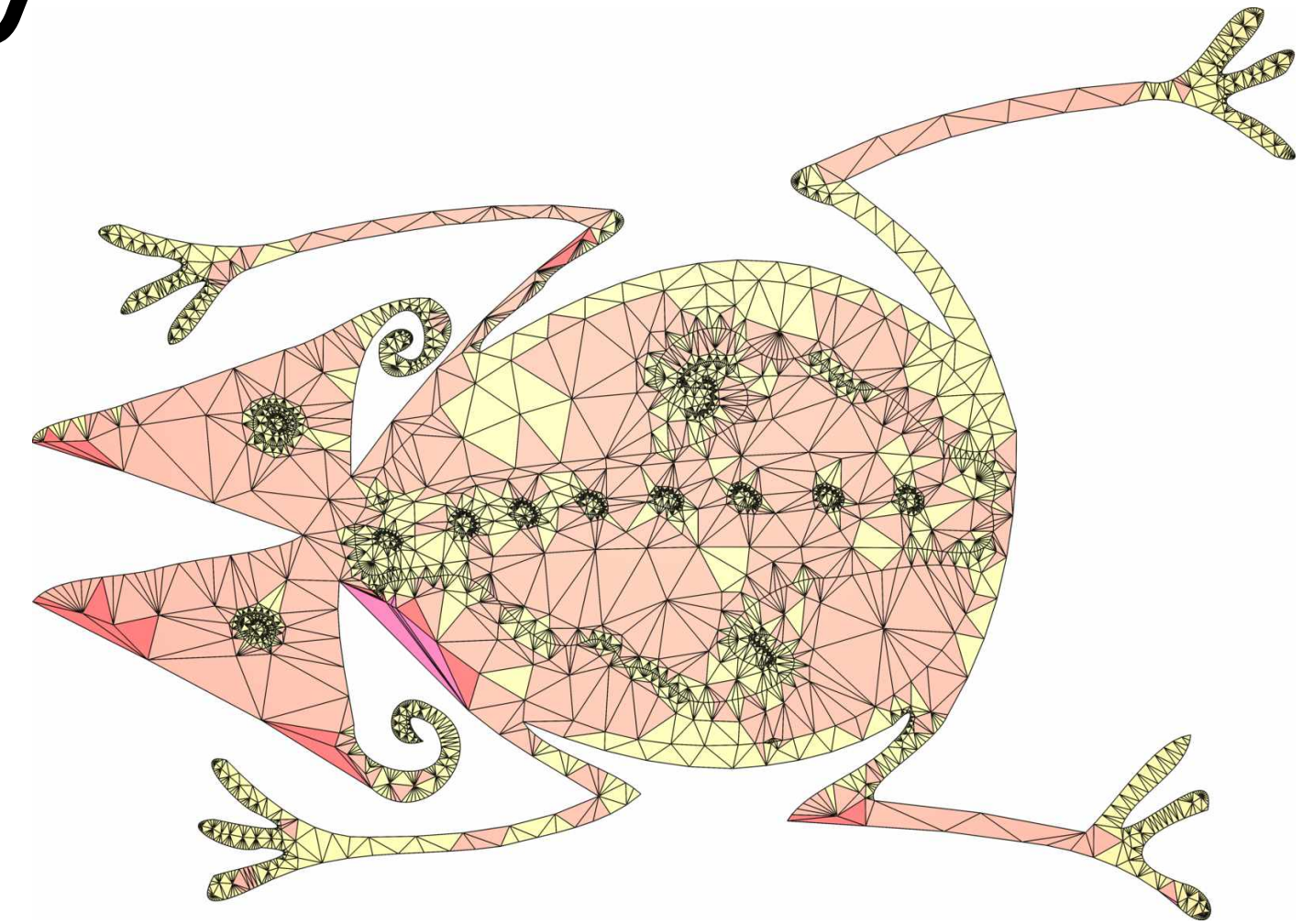
# Summary

$$k = \frac{\ln \left( B \hat{h}^{\hat{k}+1} \frac{\sigma_E^2}{\hat{\sigma}^2} \right) - \ln h_E}{\ln h_E}$$

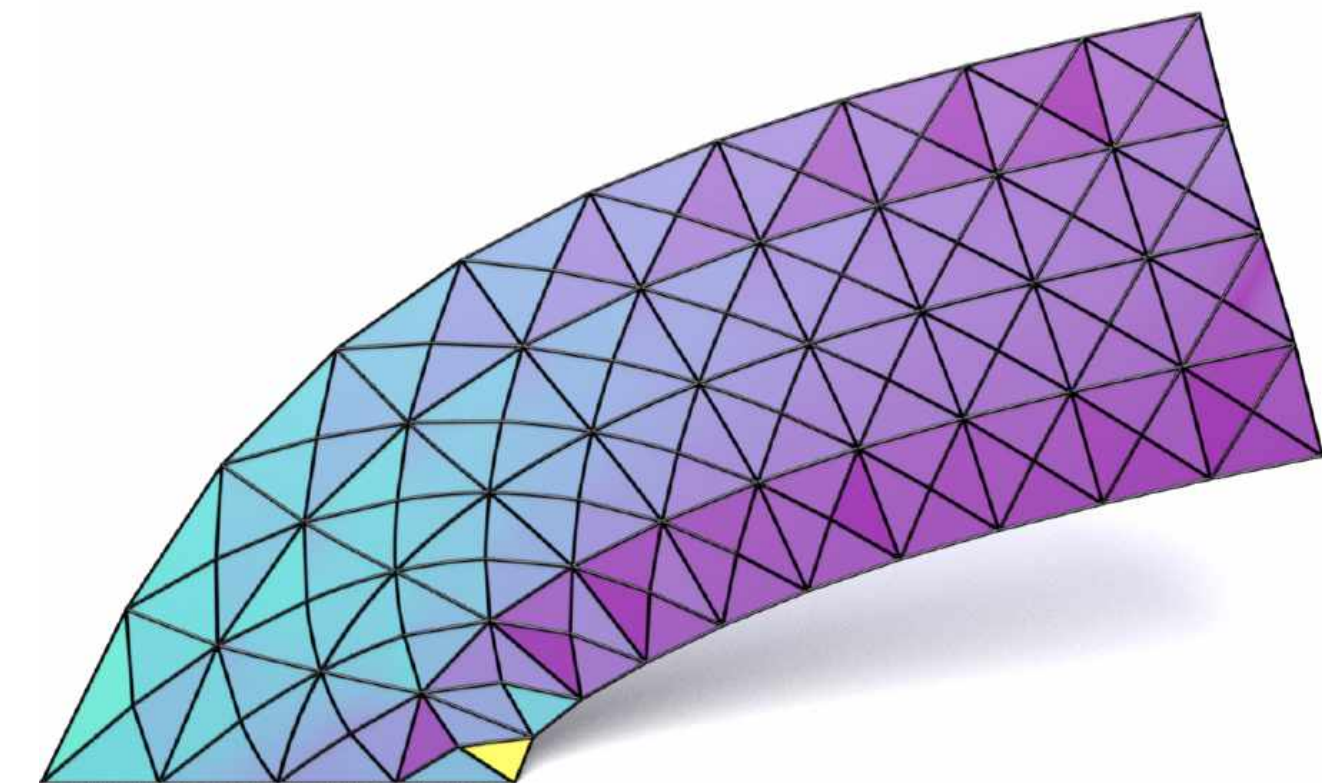
1. Use formula



3. Construct  $C^0$  basis



2. Propagate degrees



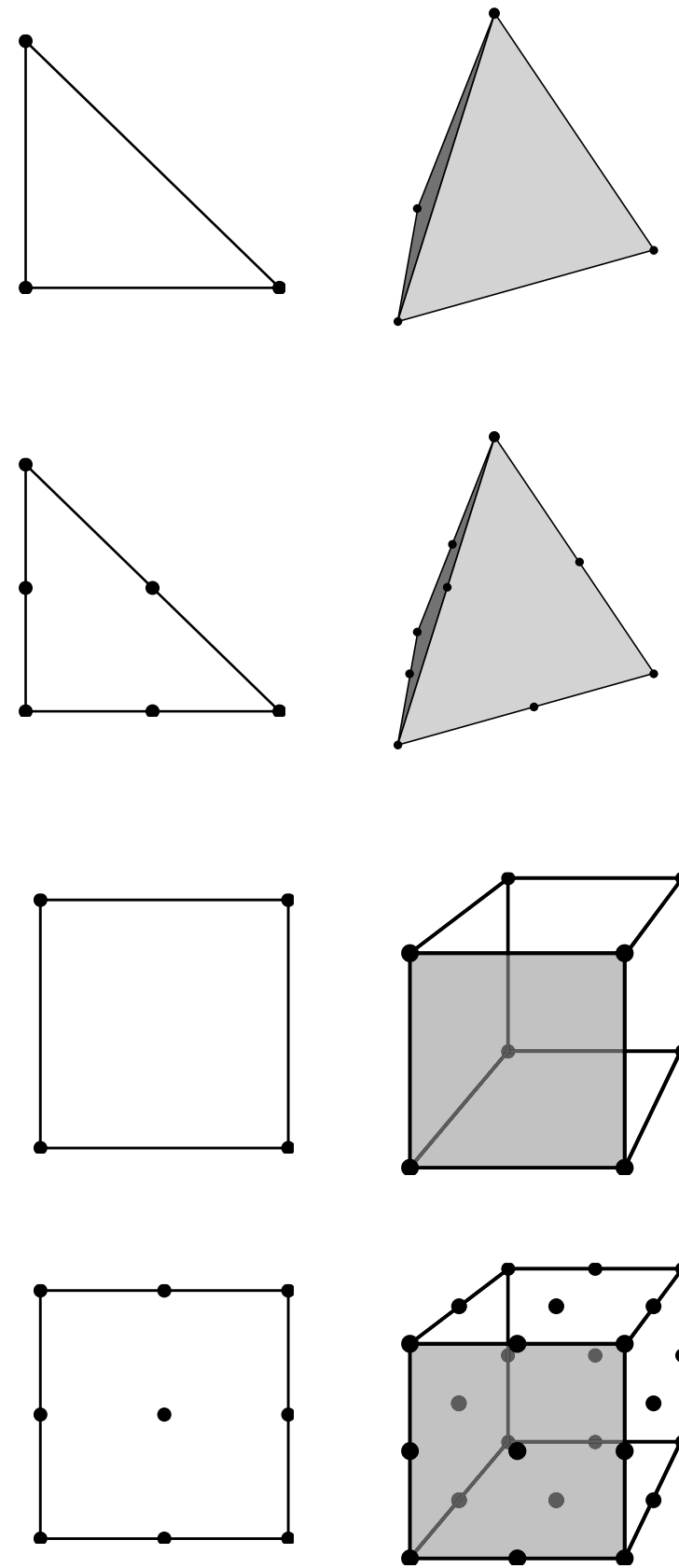
4. Simulate!



# Future Work



# Future Work

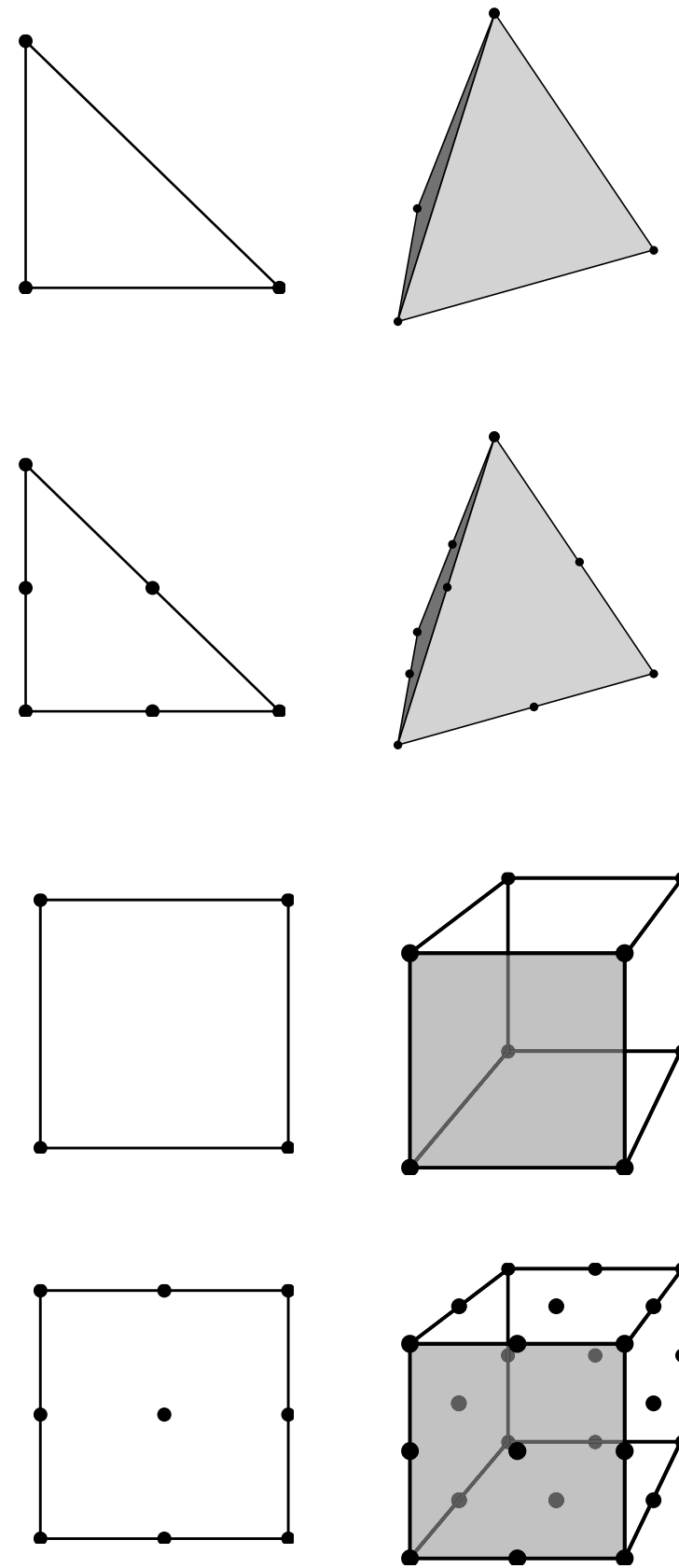


Analysis for elliptic PDEs only.

Does it make a difference for  
contacts or time-dependent problems?



# Future Work



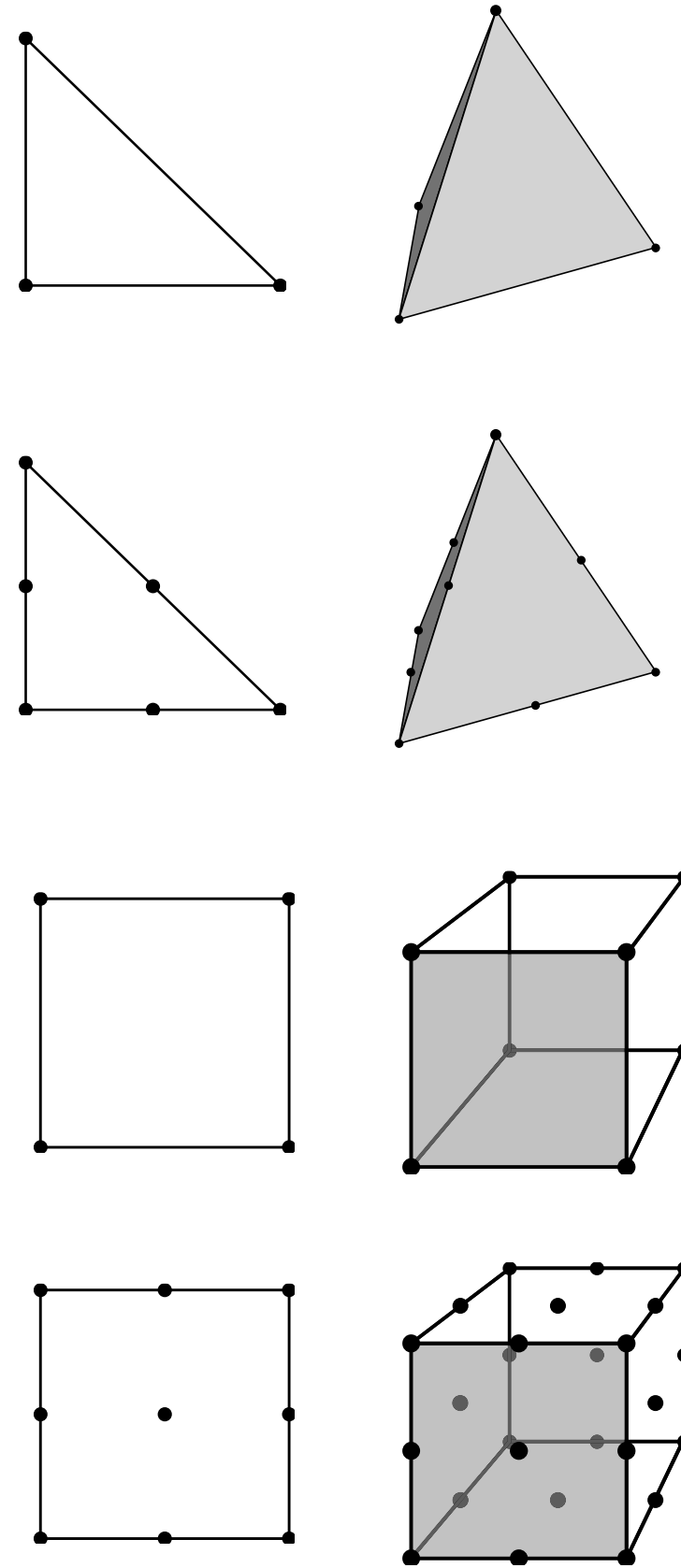
Analysis for elliptic PDEs only.

Does it make a difference for  
contacts or time-dependent problems?

**Maybe**



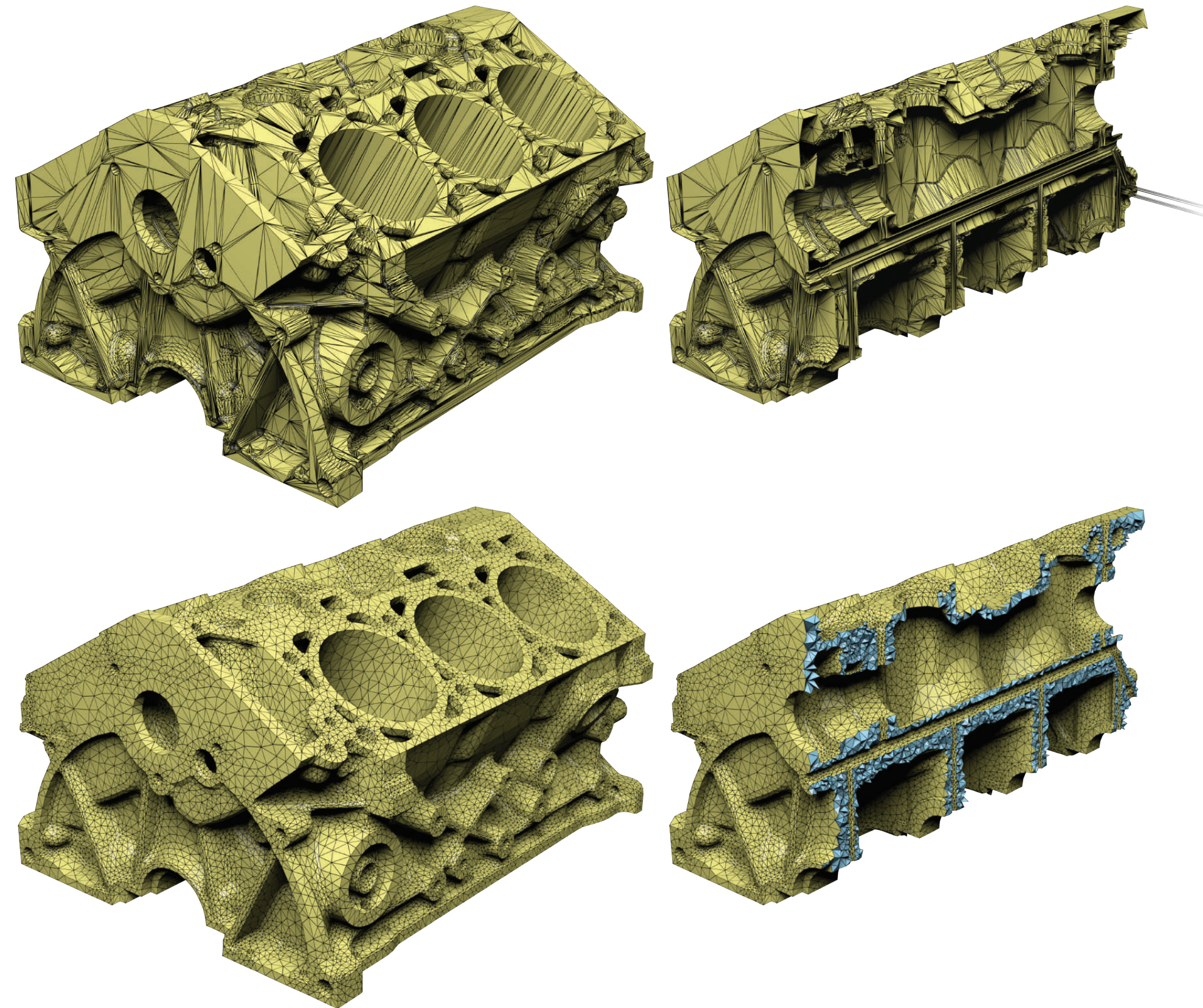
# Future Work



Analysis for elliptic PDEs only.

Does it make a difference for  
contacts or time-dependent problems?

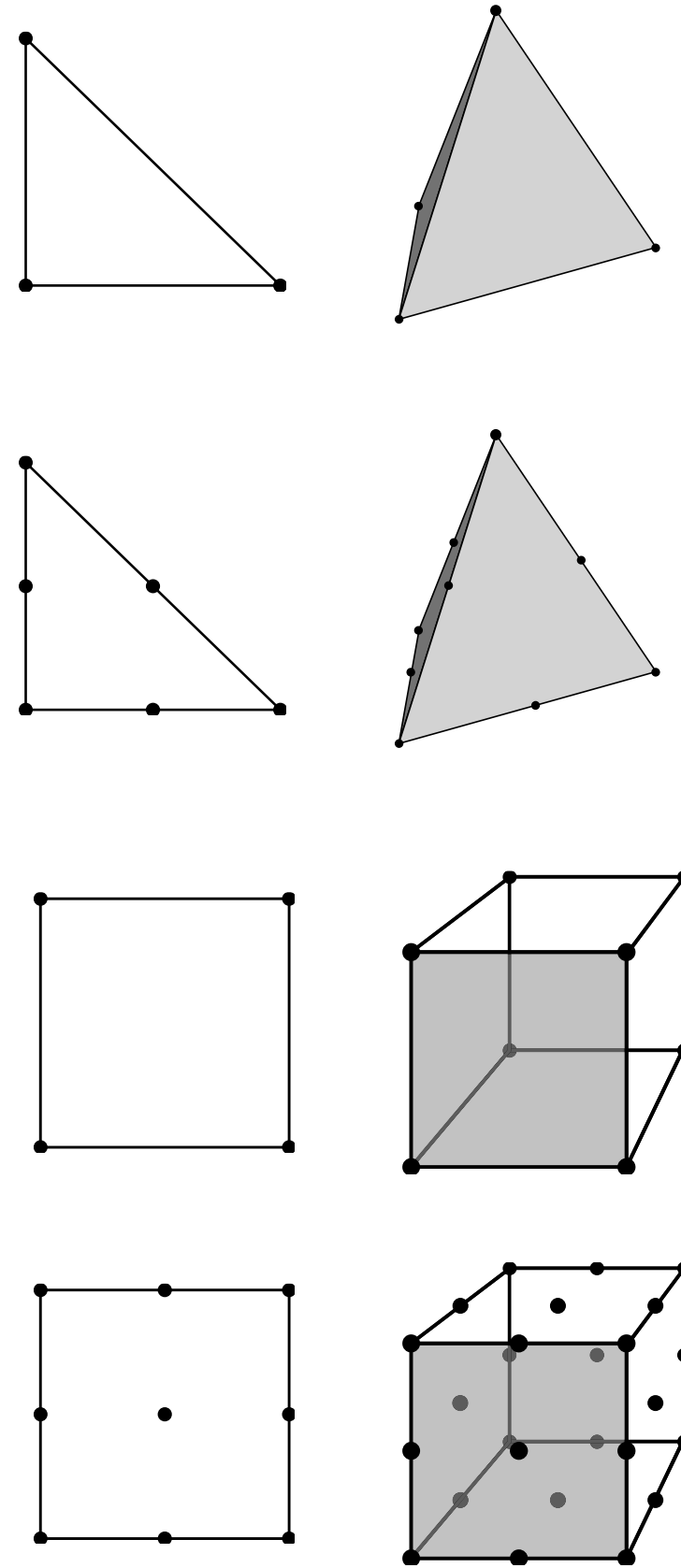
**Maybe**



Meshing still takes way longer than  
the FEM solve. Can we make it real-time?

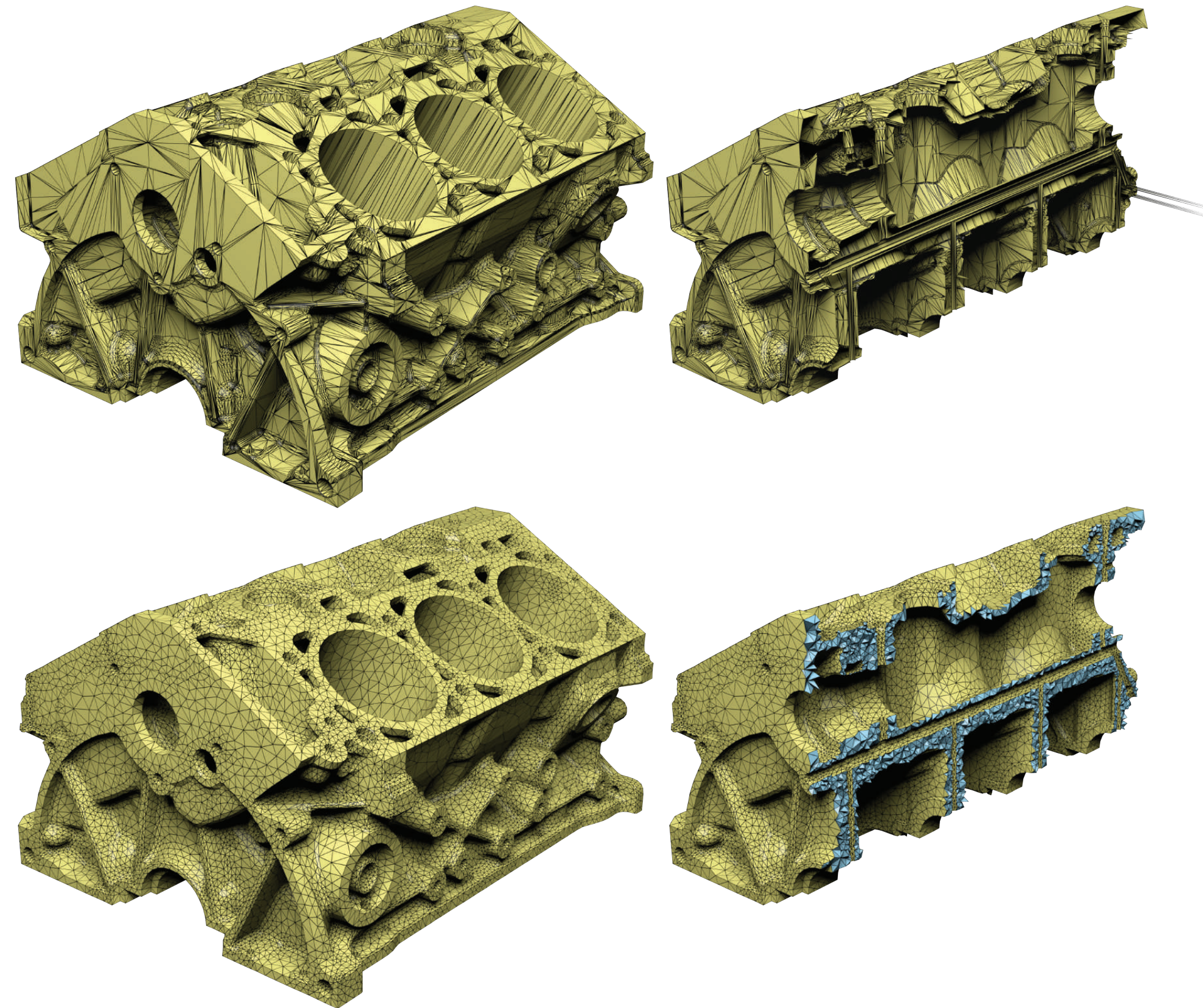


# Future Work



Analysis for elliptic PDEs only.  
Does it make a difference for  
contacts or time-dependent problems?

Maybe

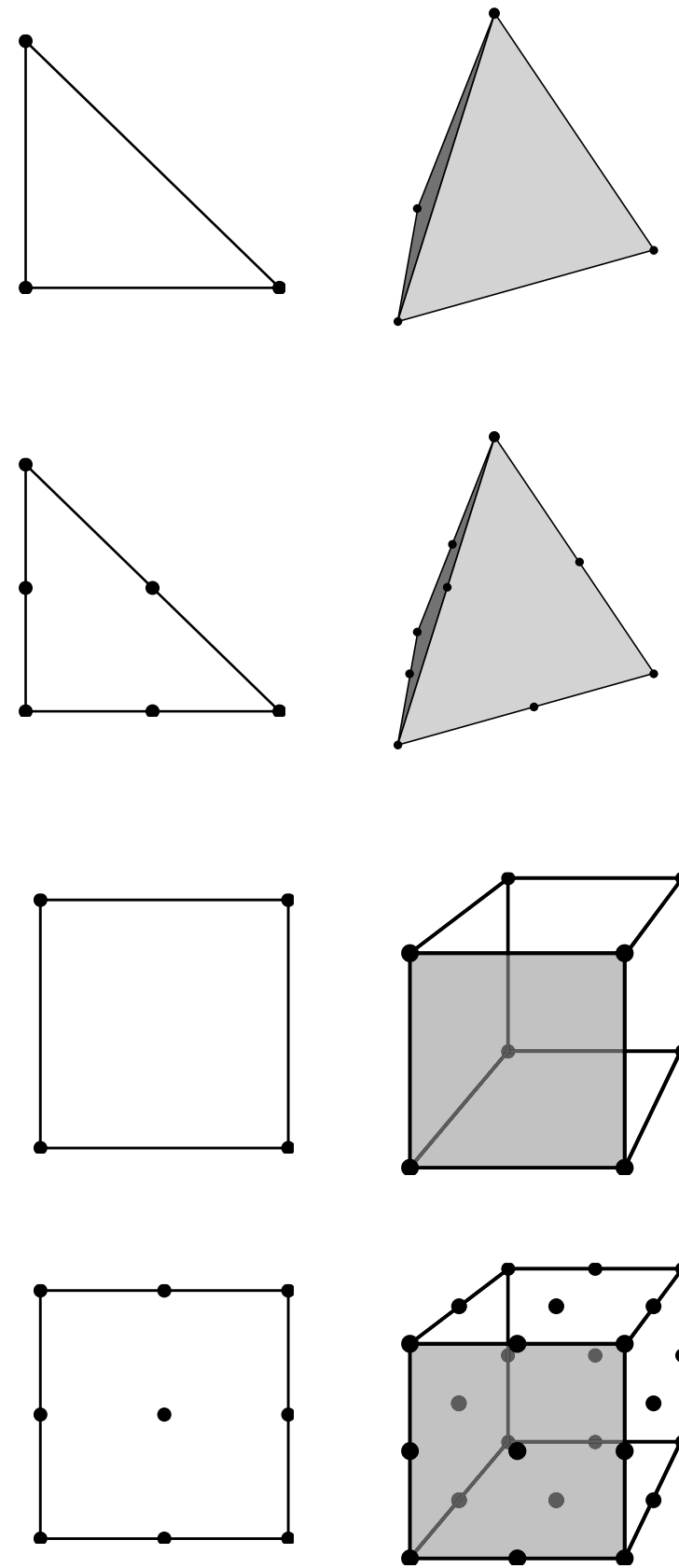


Meshing still takes way longer than  
the FEM solve. Can we make it real-time?

Maybe

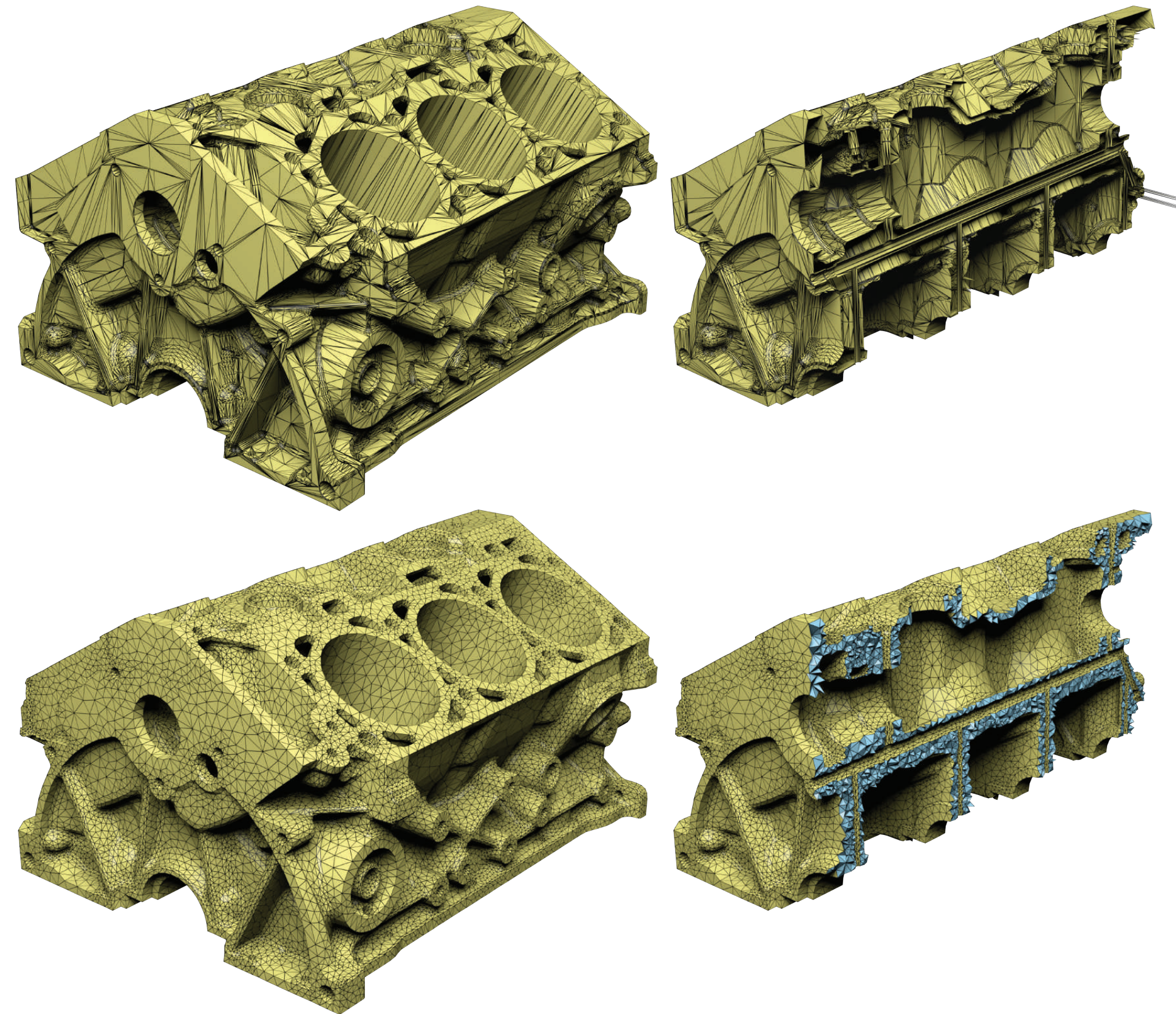


# Future Work



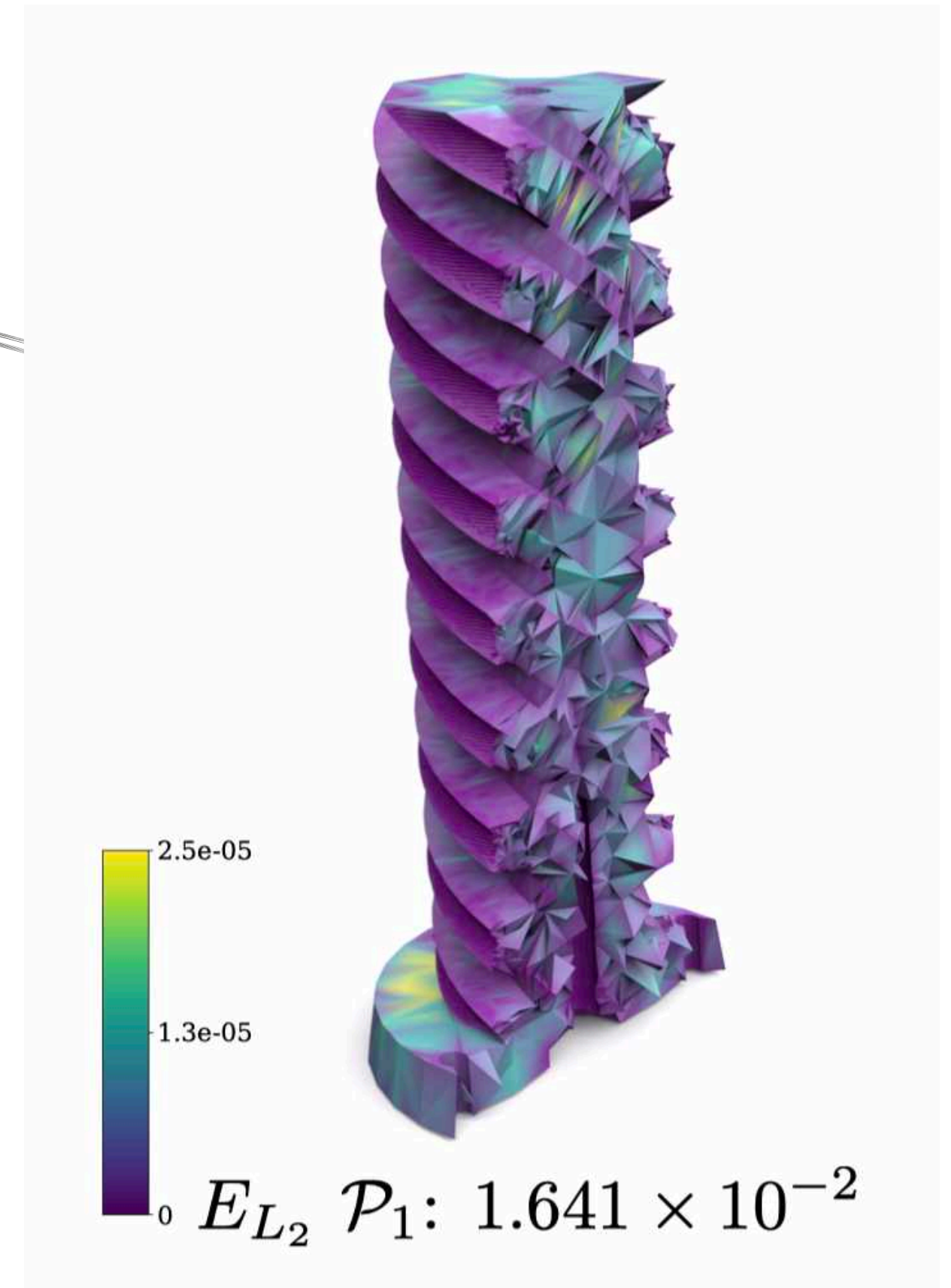
Analysis for elliptic PDEs only.  
Does it make a difference for  
contacts or time-dependent problems?

Maybe



Meshing still takes way longer than  
the FEM solve. Can we make it real-time?

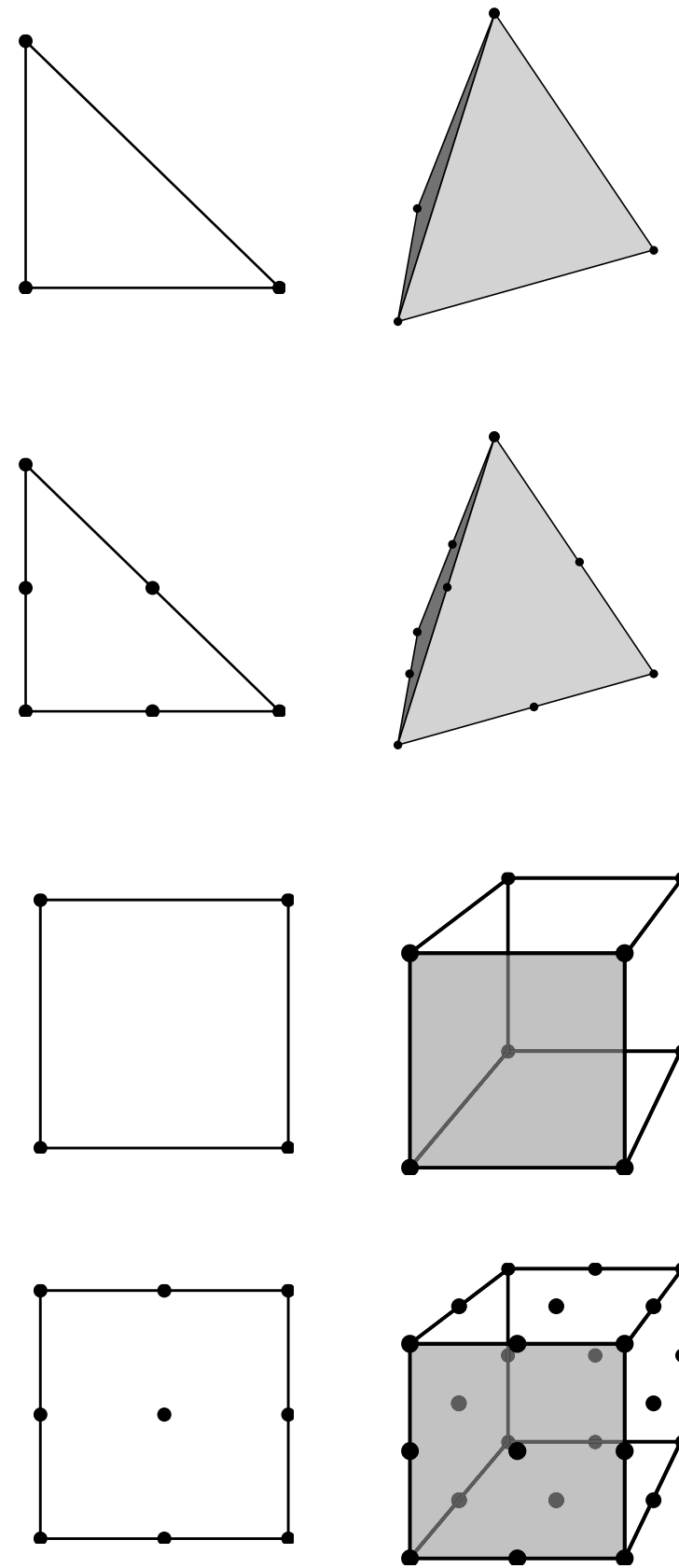
Maybe



Can we use a similar strategy  
to limit/avoid remeshing in  
dynamic simulations?

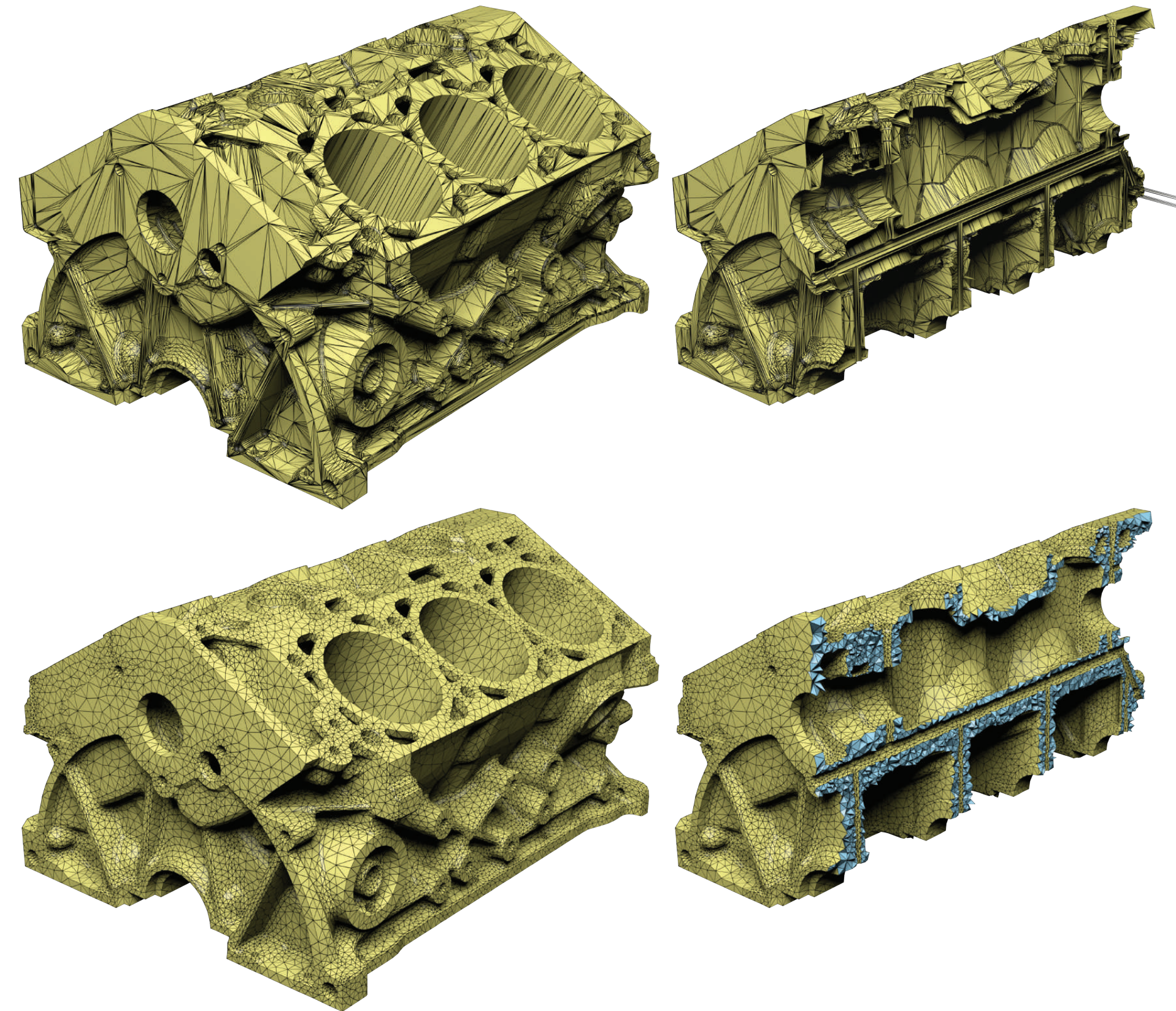


# Future Work



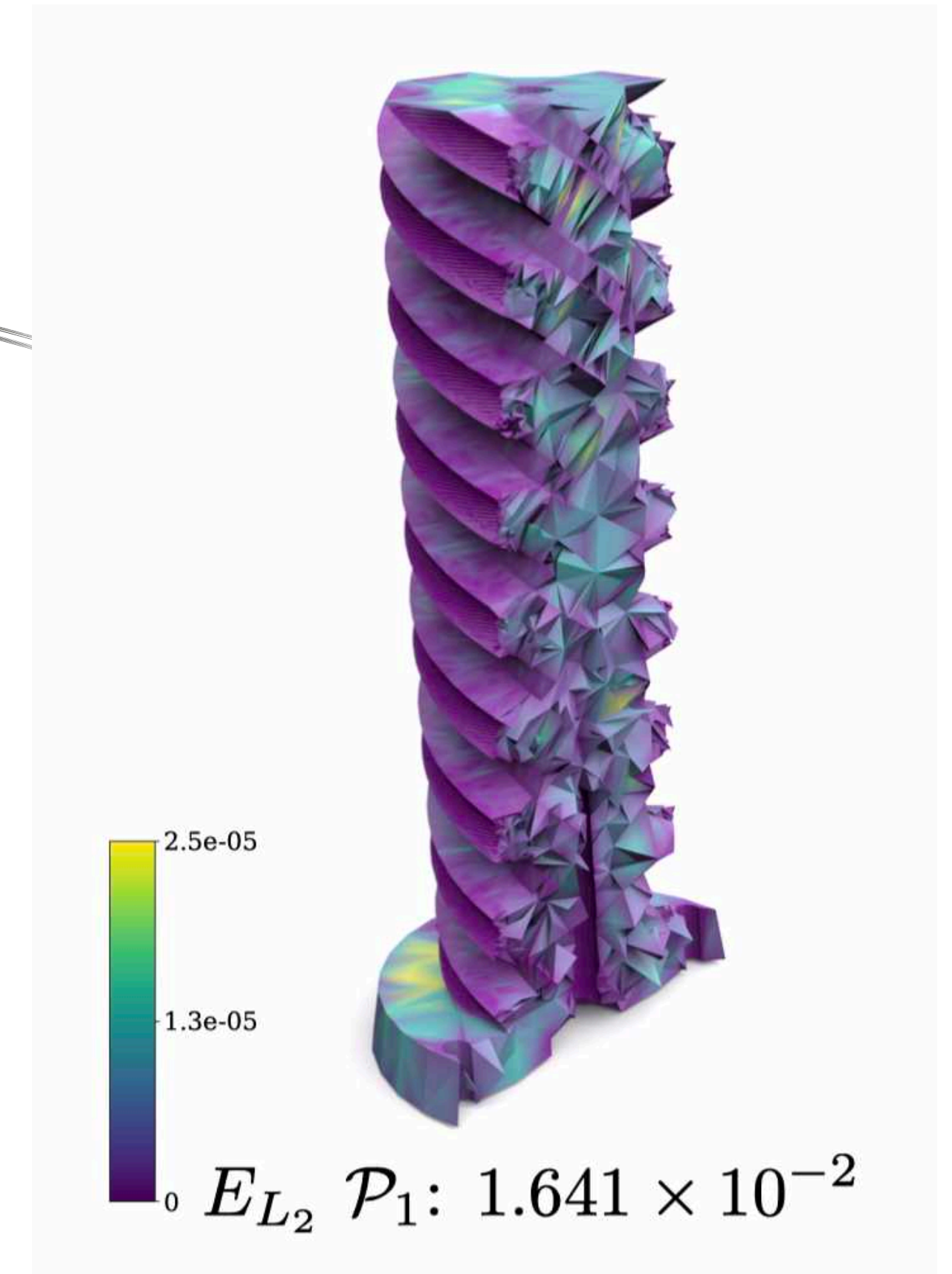
Analysis for elliptic PDEs only.  
Does it make a difference for  
contacts or time-dependent problems?

Maybe



Meshing still takes way longer than  
the FEM solve. Can we make it real-time?

Maybe

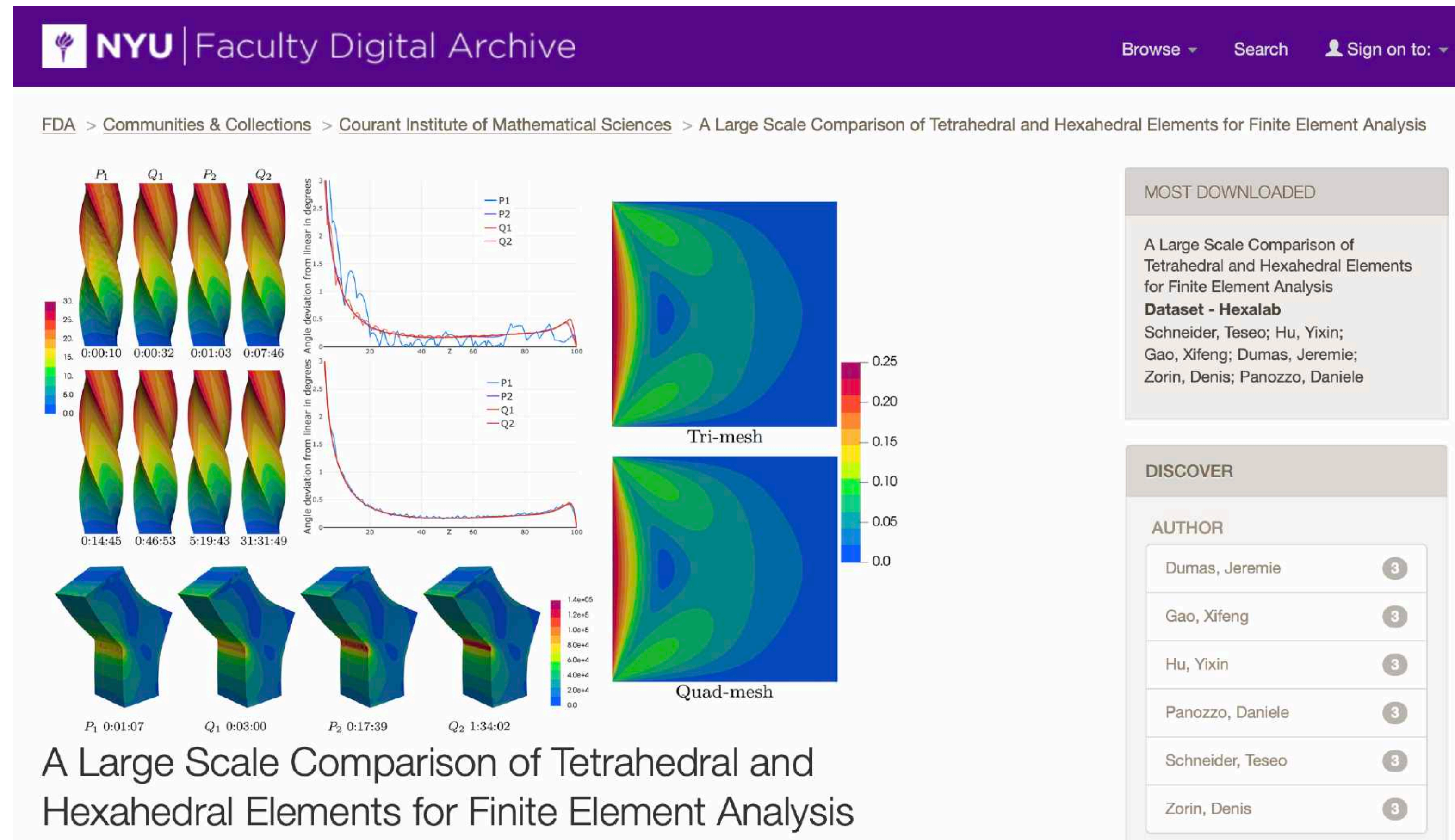


Can we use a similar strategy  
to limit/avoid remeshing in  
dynamic simulations?

Why not?



# Large Scale Comparison



<https://archive.nyu.edu/handle/2451/44221>

MIT License





# MeshPlot

```
mp.plot(v, f)
```

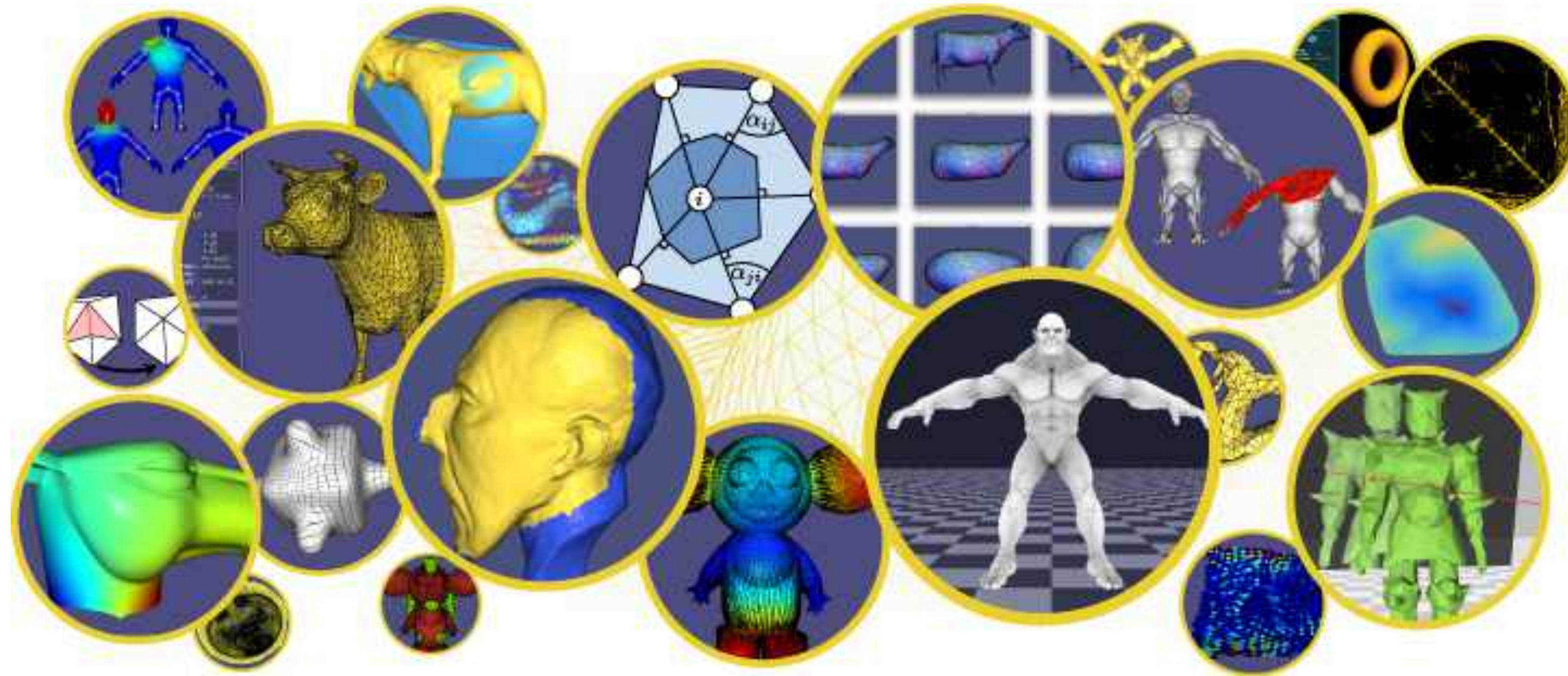


<https://skoch9.github.io/meshplot/>





# Interactive Geometry Library (libigl)



<https://libigl.github.io>



# Wild Meshing (TetWild)

Yixin-Hu / TetWild

Unwatch 14 Unstar 174 Fork 25

Code Issues 14 Pull requests 0 Projects 0 Wiki Insights

Robust Tetrahedral Meshing in the Wild. <https://dl.acm.org/citation.cfm?id=32...>

geometry-processing tetrahedral-meshing surface-repair 3d-triangulation

122 commits 2 branches 0 releases 6 contributors View license

Branch: master New pull request Create new file Upload files Find File Clone or download

<https://wildmeshing.github.io>





# PolyFEM

[Home](#)



 [polyfem/polyfem](#)  
22 Stars · 3 Forks

**polyfem**  
[Home](#)  
[Tutorial](#)  
[Documentation](#)  
[Python \[alpha\]](#)  
[Jupyter examples](#)  
[Python docs](#)



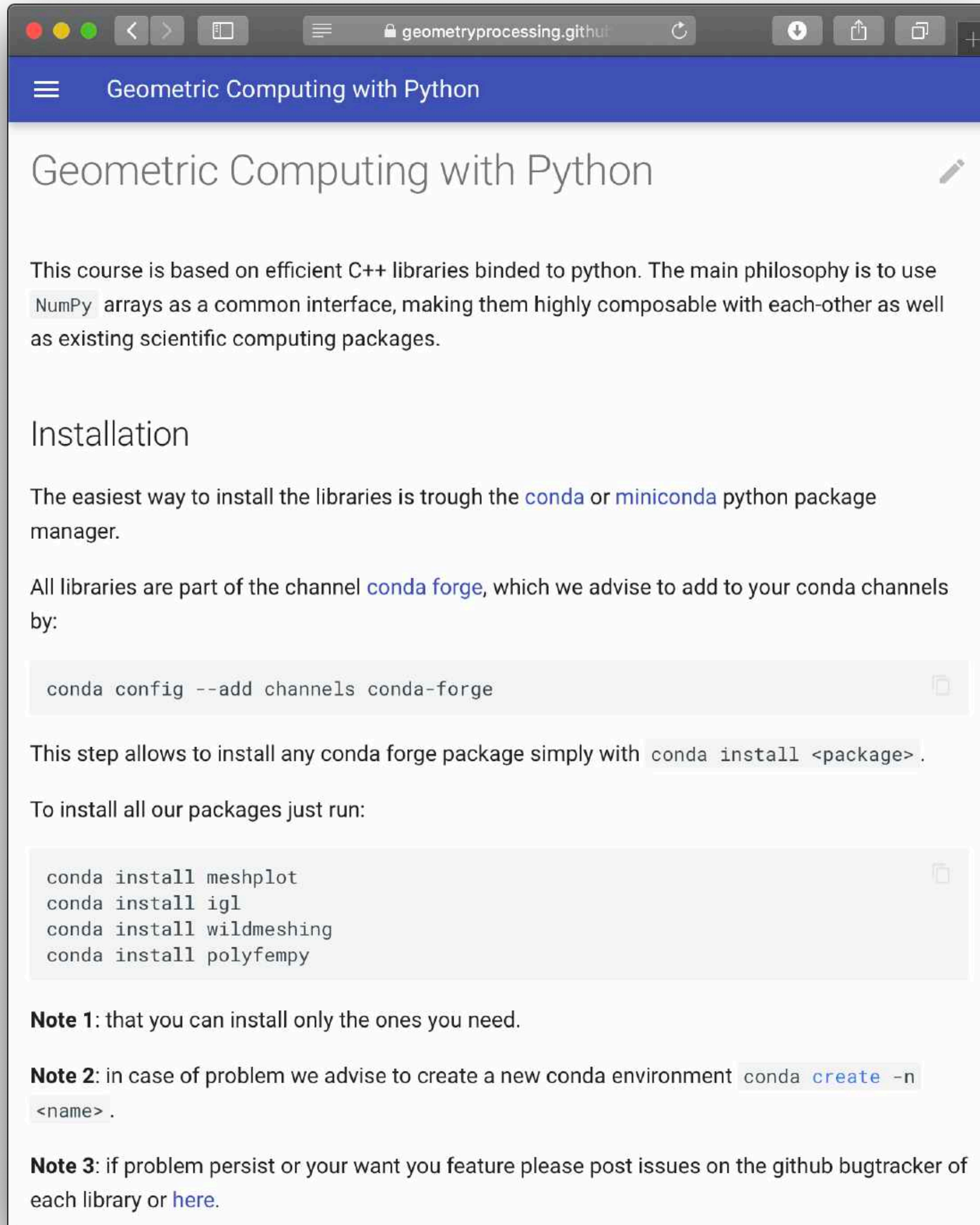
**Table of contents**  
[Compilation](#)  
[Optional](#)  
[Usage](#)  
[License](#)  
[Citation](#)  
[Acknowledgements & Funding](#)

<https://polyfem.github.io>





# Geometric Computing in Python



Geometric Computing with Python

This course is based on efficient C++ libraries binded to python. The main philosophy is to use `NumPy` arrays as a common interface, making them highly composable with each-other as well as existing scientific computing packages.

## Installation

The easiest way to install the libraries is trough the [conda](#) or [miniconda](#) python package manager.

All libraries are part of the channel [conda forge](#), which we advise to add to your conda channels by:

```
conda config --add channels conda-forge
```

This step allows to install any conda forge package simply with `conda install <package>`.

To install all our packages just run:

```
conda install meshplot
conda install igl
conda install wildmeshing
conda install polyfempy
```

**Note 1:** that you can install only the ones you need.

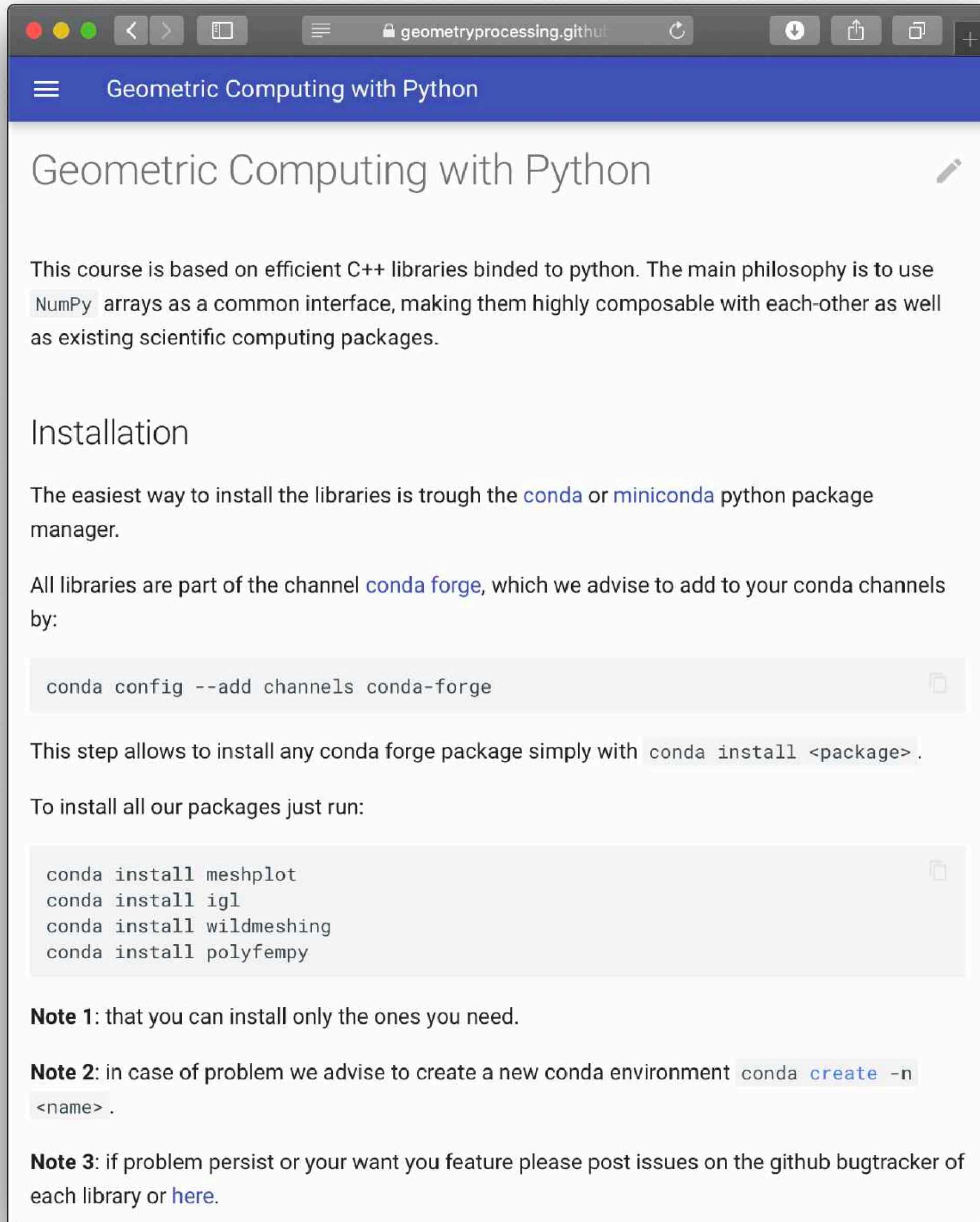
**Note 2:** in case of problem we advise to create a new conda environment `conda create -n <name>`.

**Note 3:** if problem persist or your want you feature please post issues on the github bugtracker of each library or [here](#).





# Geometric Computing in Python



Geometric Computing with Python

This course is based on efficient C++ libraries binded to python. The main philosophy is to use NumPy arrays as a common interface, making them highly composable with each-other as well as existing scientific computing packages.

## Installation

The easiest way to install the libraries is trough the [conda](#) or [miniconda](#) python package manager.

All libraries are part of the channel [conda forge](#), which we advise to add to your conda channels by:

```
conda config --add channels conda-forge
```

This step allows to install any conda forge package simply with `conda install <package>`.

To install all our packages just run:

```
conda install meshplot
conda install igl
conda install wildmeshing
conda install polyfempy
```

**Note 1:** that you can install only the ones you need.

**Note 2:** in case of problem we advise to create a new conda environment `conda create -n <name>`.

**Note 3:** if problem persist or your want you feature please post issues on the github bugtracker of each library or [here](#).

- Easy to use, integrates meshing, geometry processing (libigl), and FEM analysis
- All on Conda, works out of the box on linux, mac, windows
- Easy to edit, pure C++
- Based on numpy/scipy







# Black-Box Analysis

Daniele Panozzo



117



NYU

COURANT INSTITUTE OF  
MATHEMATICAL SCIENCES