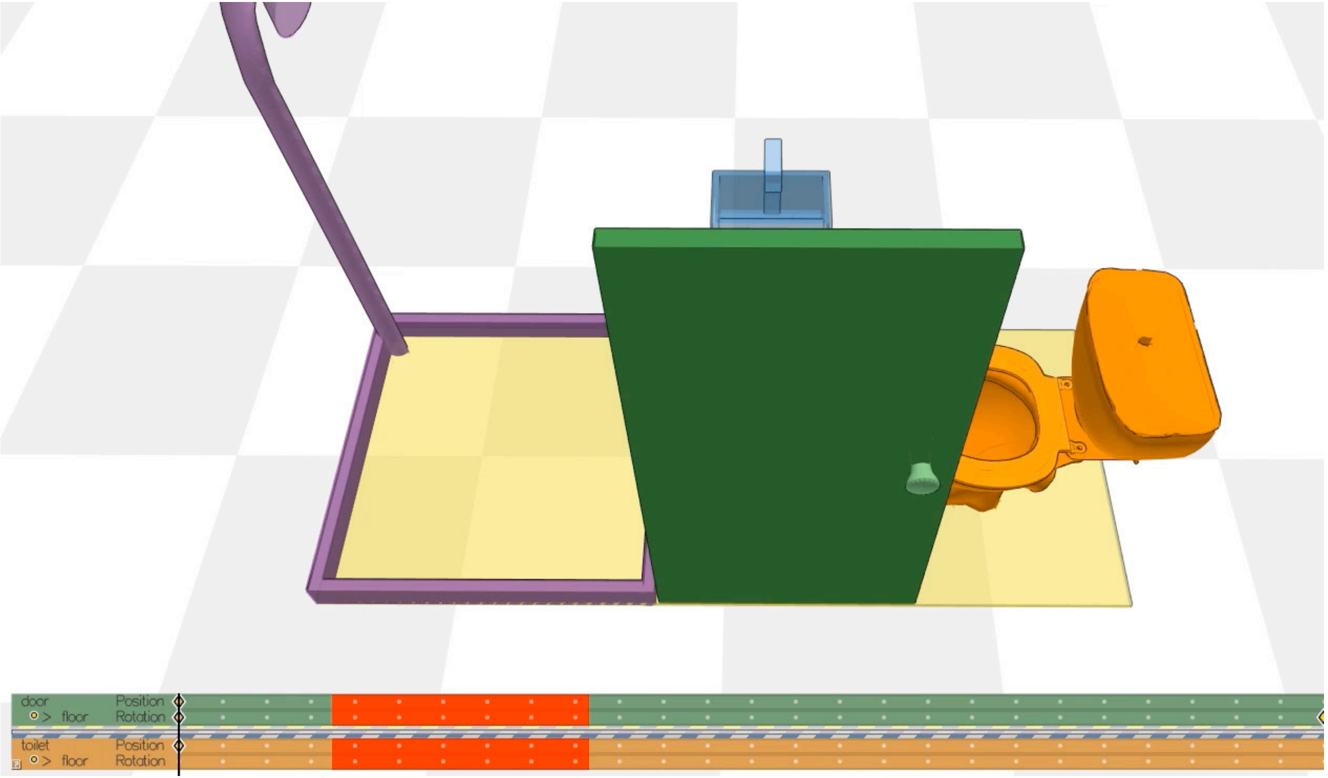
Generalized Matryoshka
Computational Design of Nesting Objects

Alec Jacobson

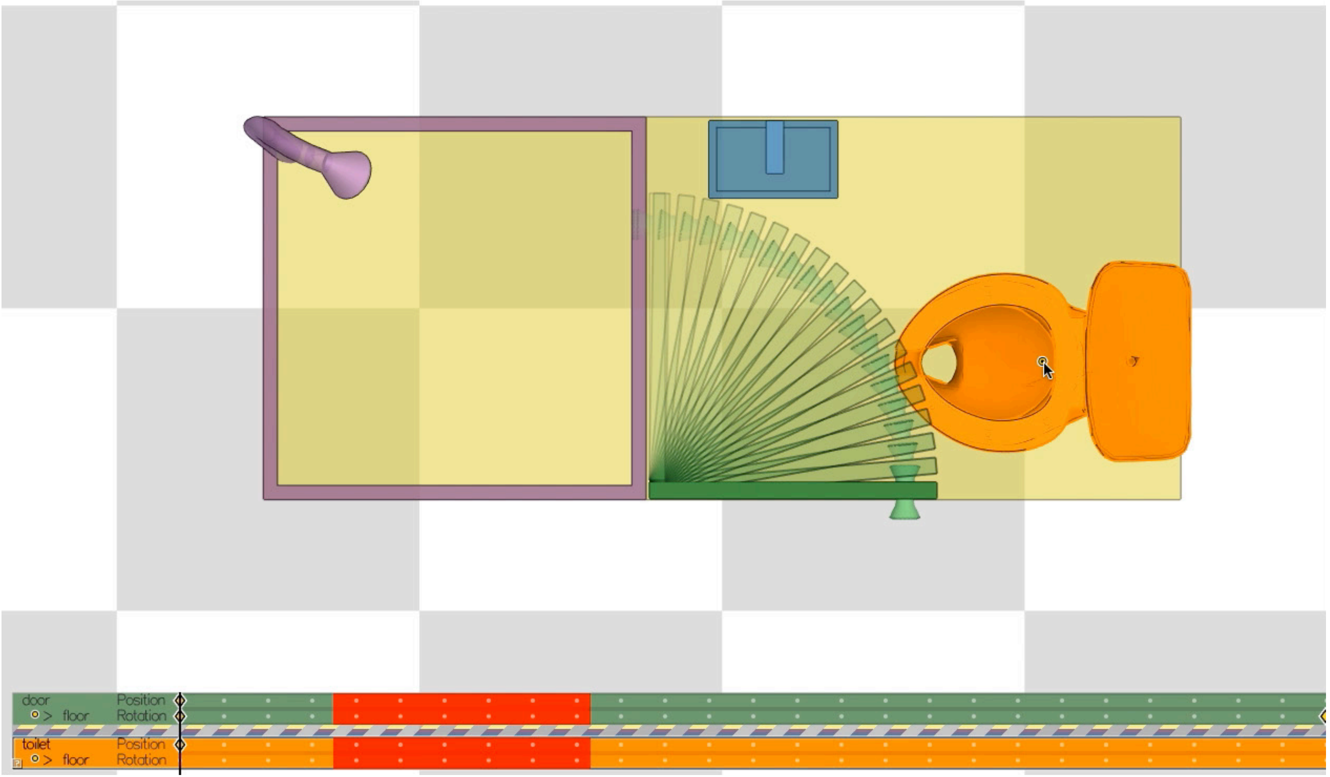
University of Toronto

Previous work enables computational design of *reconfigurables*



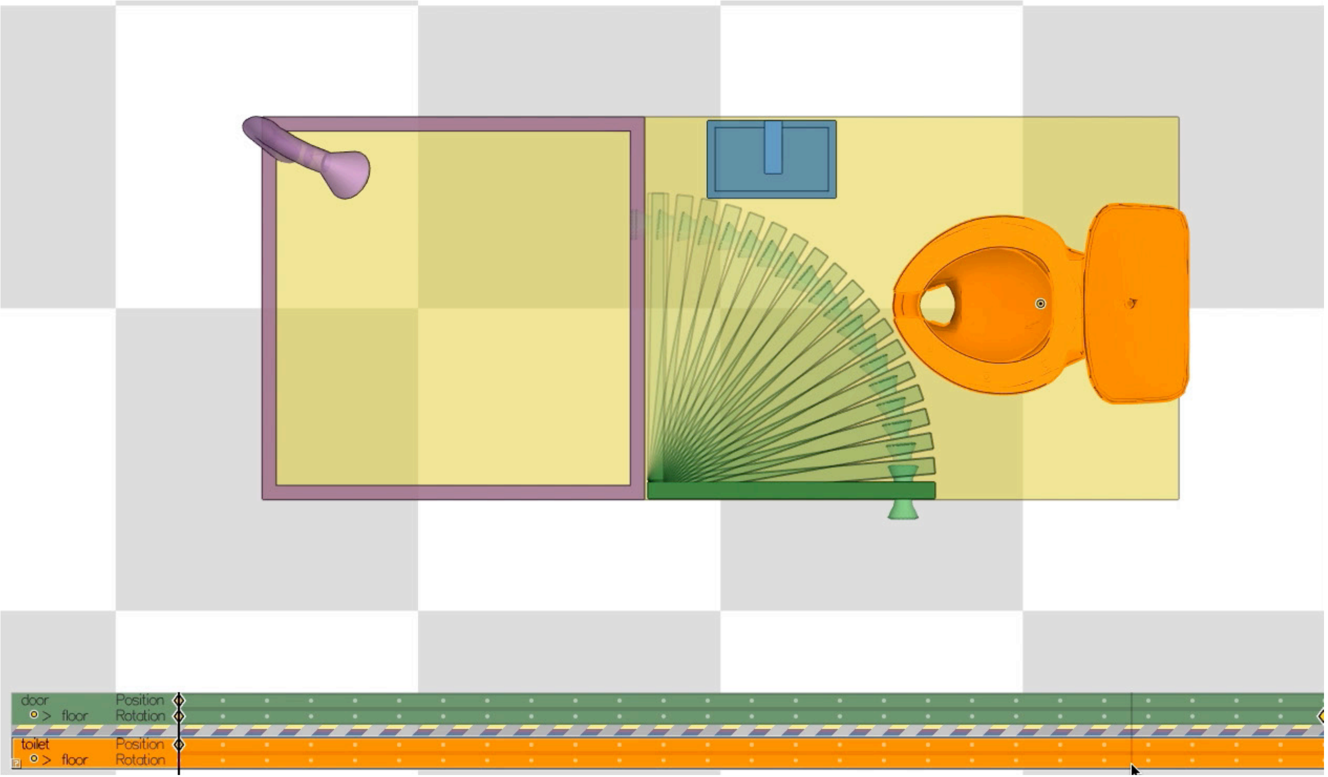
[Garg et al. 2016]

Previous work enables computational design of *reconfigurables*



[Garg et al. 2016]

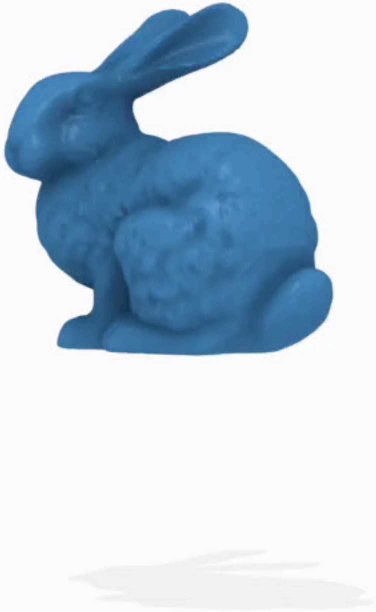
Previous work enables computational design of *reconfigurables*



[Garg et al. 2016]

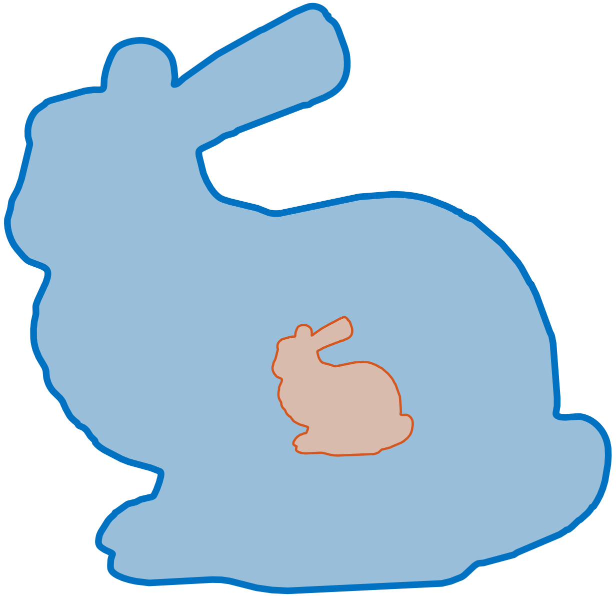
[Zvyozdochkin & Malyutin 1890]

We present a method to generalize Matryoshka to arbitrary shapes



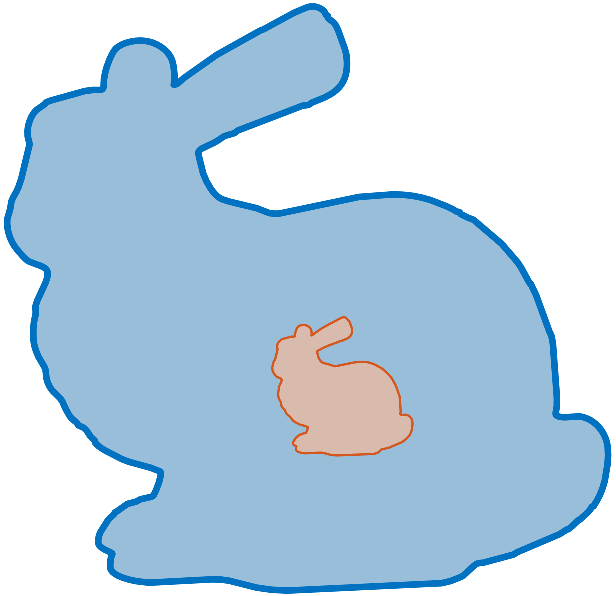
We present a method to generalize Matryoshka to arbitrary shapes

Nesting requires strict enclosure…

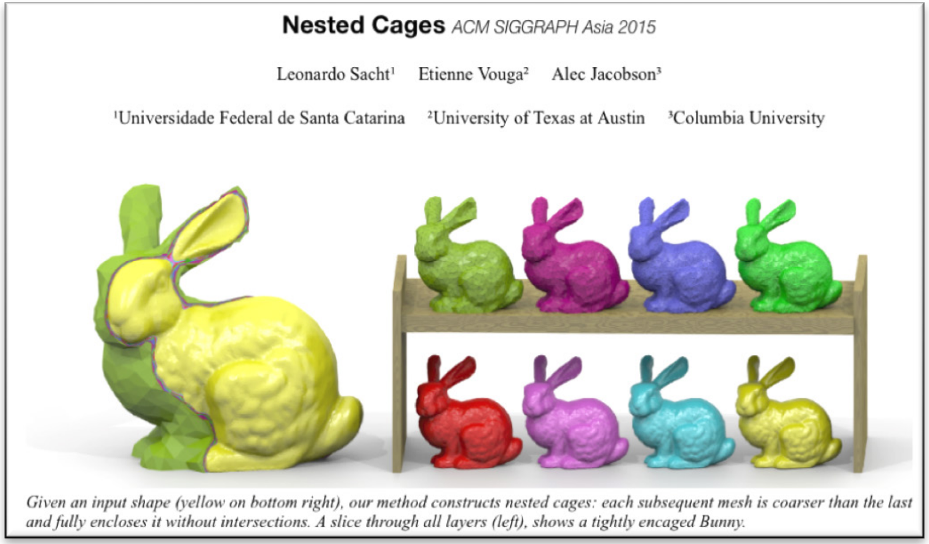


loose enclosure

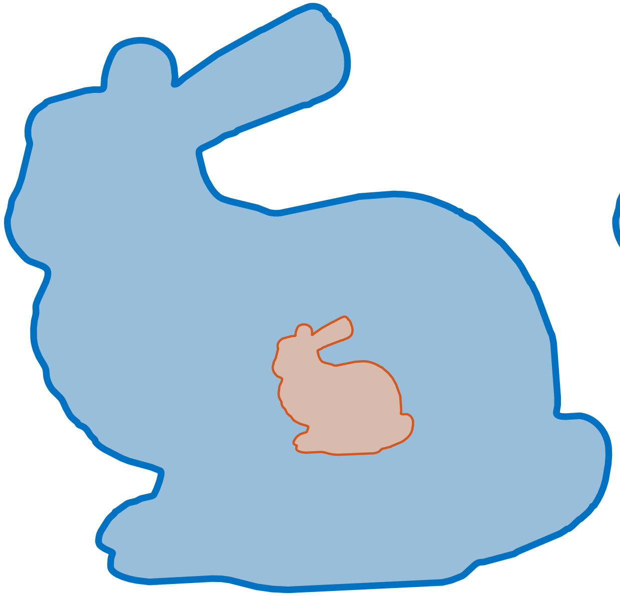
Nesting requires strict enclosure…



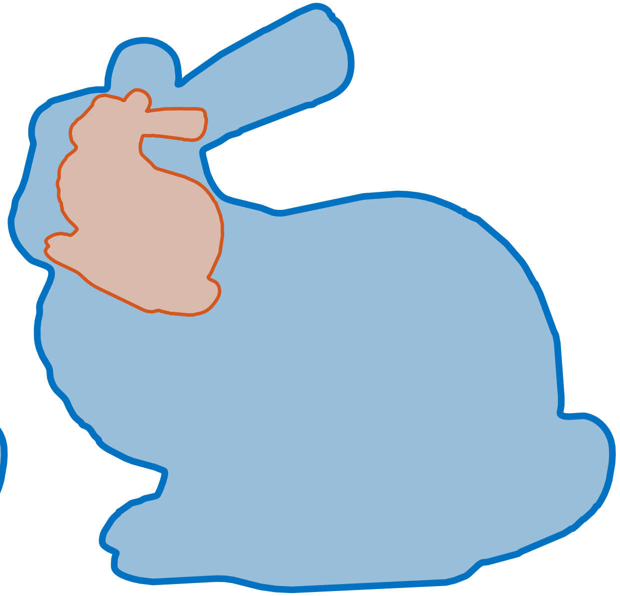
enclosure



Nesting also requires *removal*

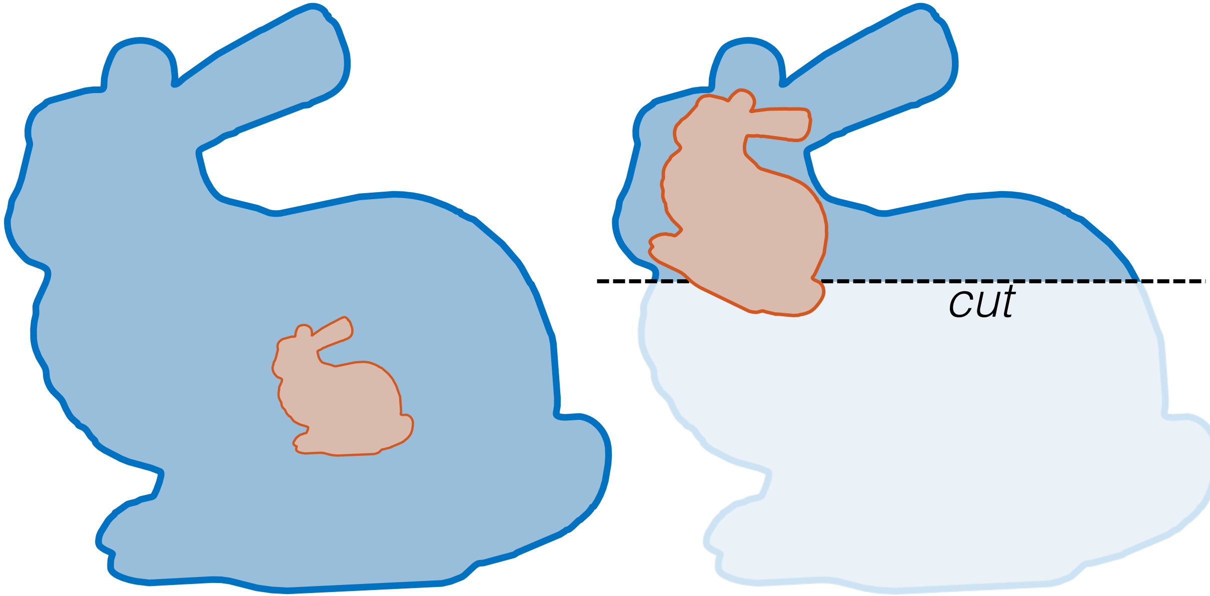


loose enclosure



enclosed, but not removable

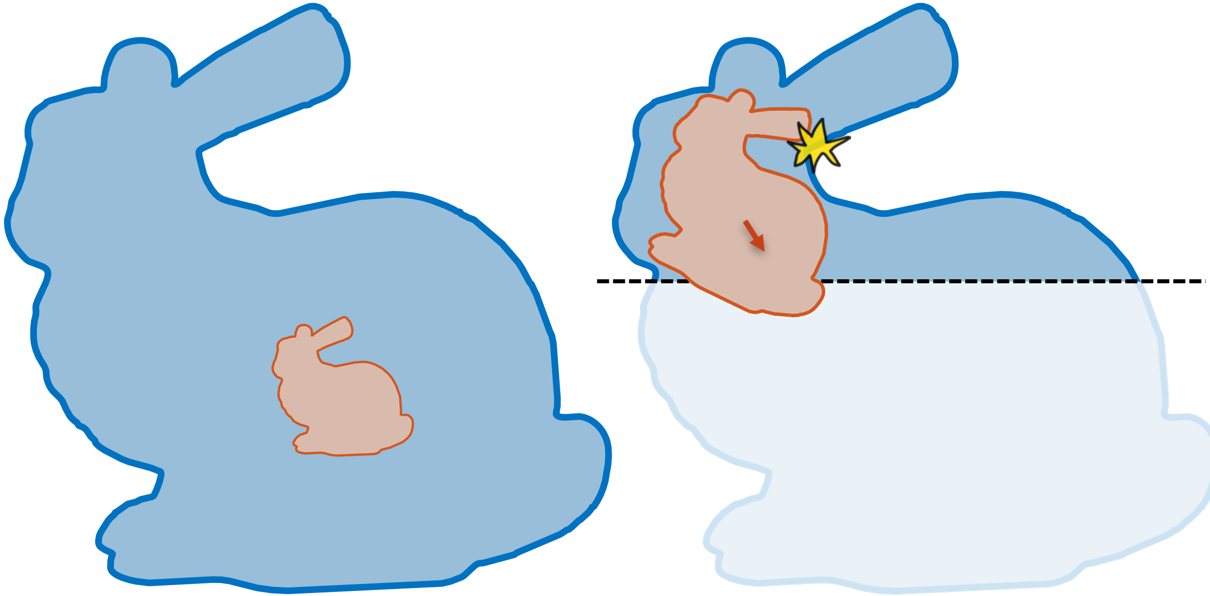
Nesting also requires *removal*



loose enclosure

enclosed, but not removable

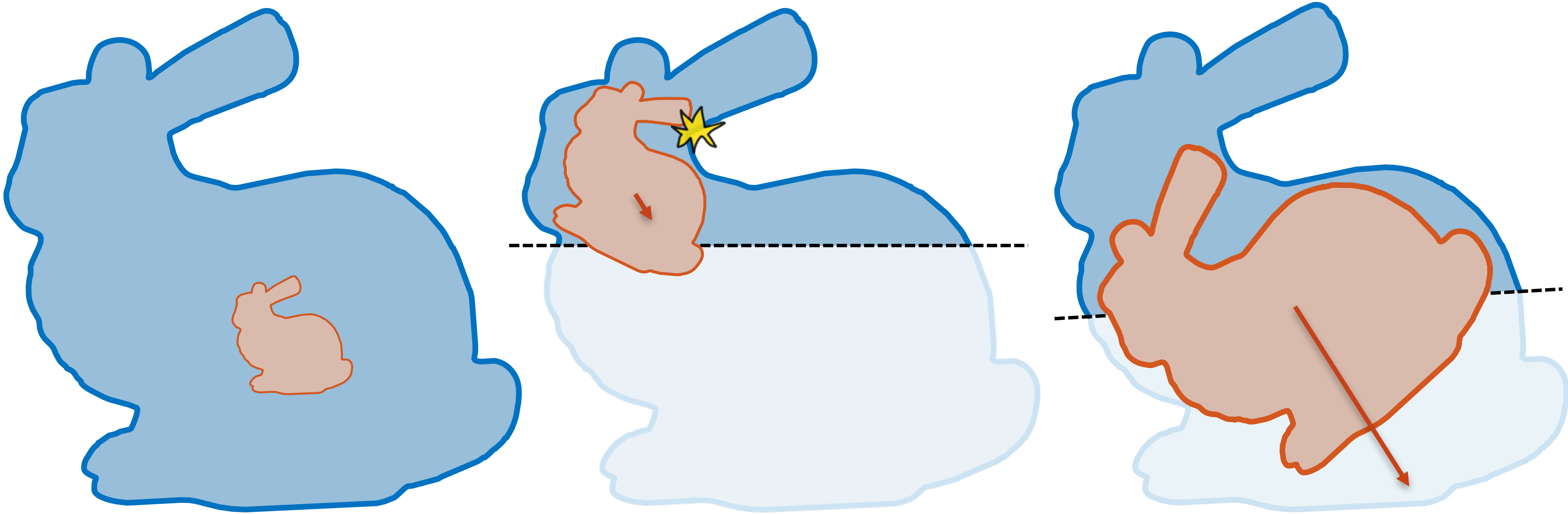
Nesting also requires *removal*



loose enclosure

enclosed, but not removable

Nesting also requires *removal*



loose enclosure | enclosed, but not removable | enclosed and removable

We present highly parallelizable methods to…

- determine feasibility of nesting,

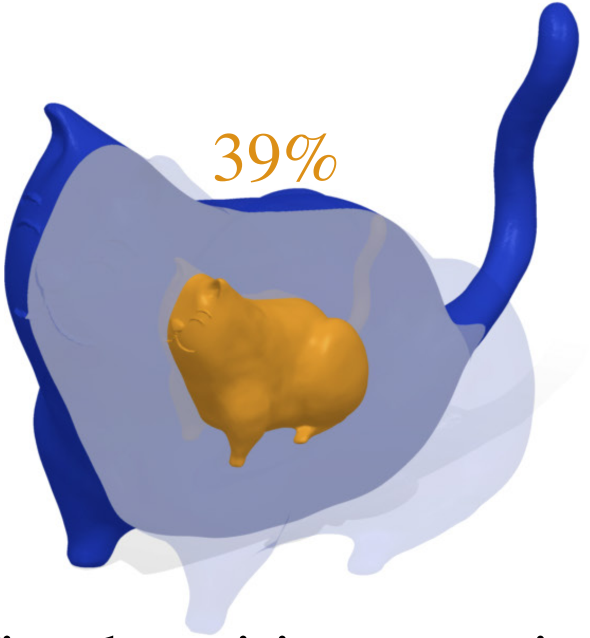
We present highly parallelizable methods to…

- determine feasibility of nesting,
- find maximum scale,

We present highly parallelizable methods to…

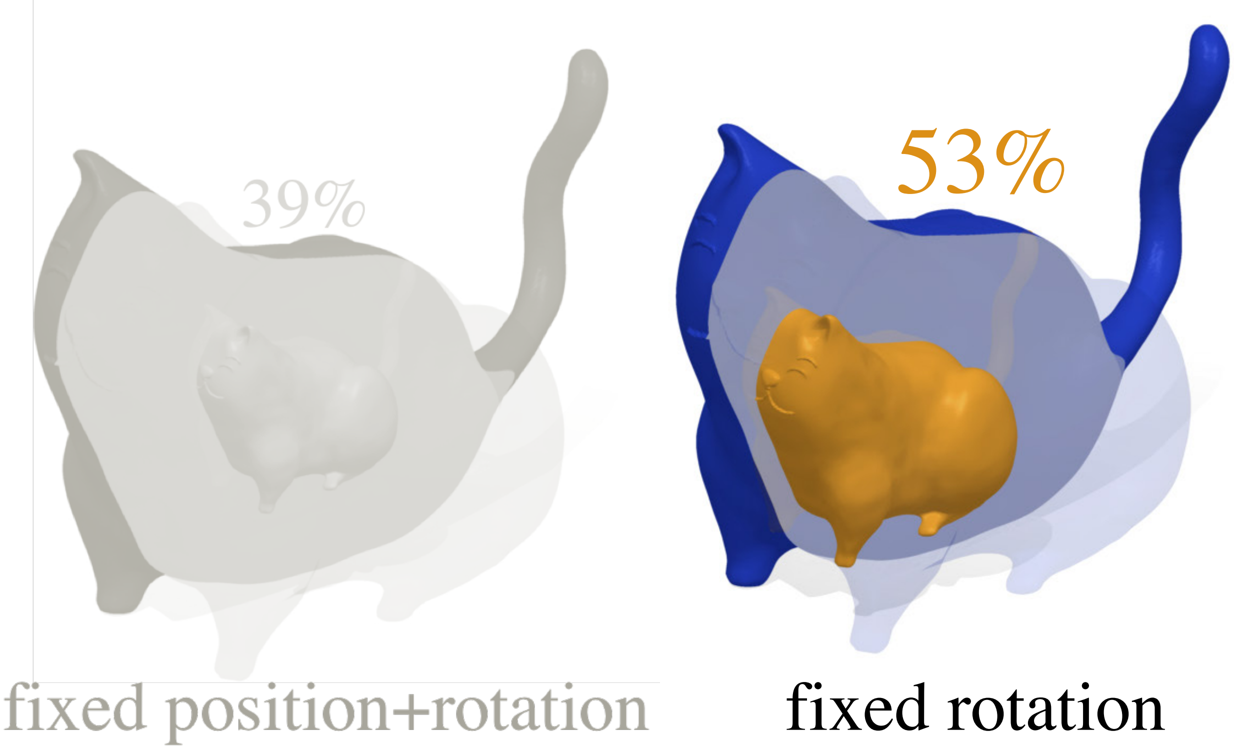
- determine feasibility of nesting,
- find maximum scale, and
- optimize nesting scale
  over some or all parameters

Our optimization utilizes rigid motion for tighter nesting

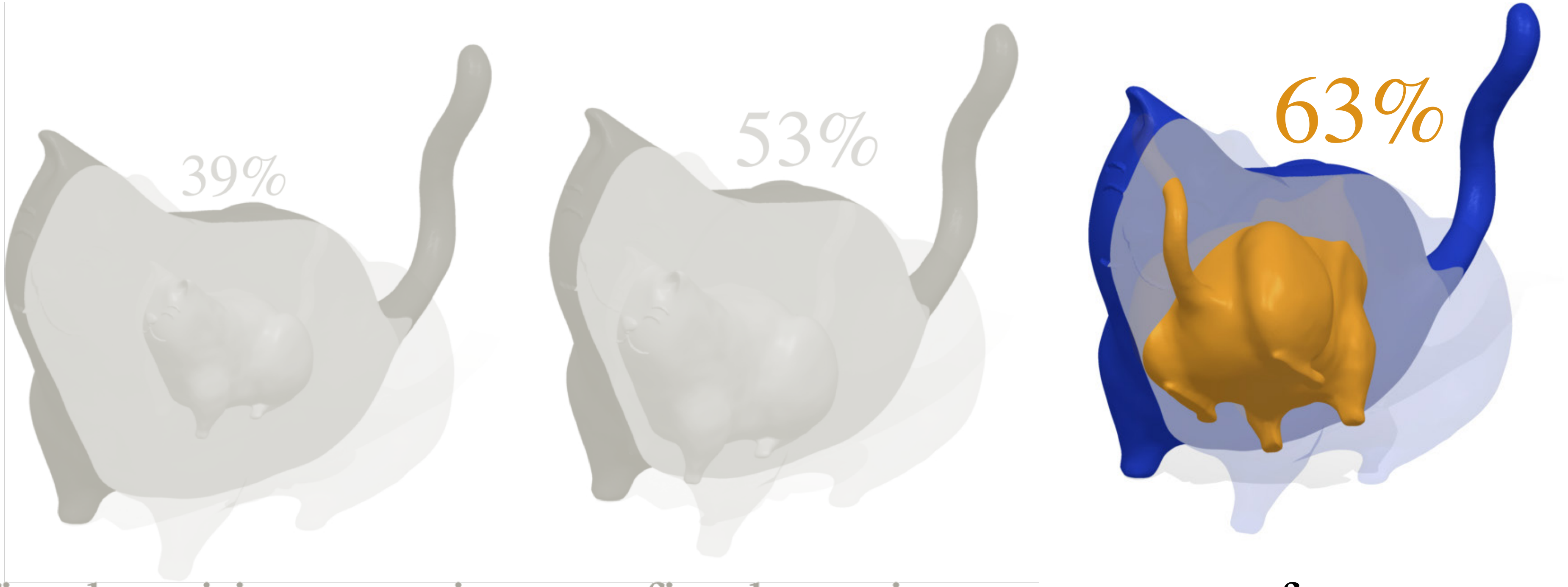


fixed position+rotation

Our optimization utilizes rigid motion for tighter nesting



Our optimization utilizes rigid motion for tighter nesting

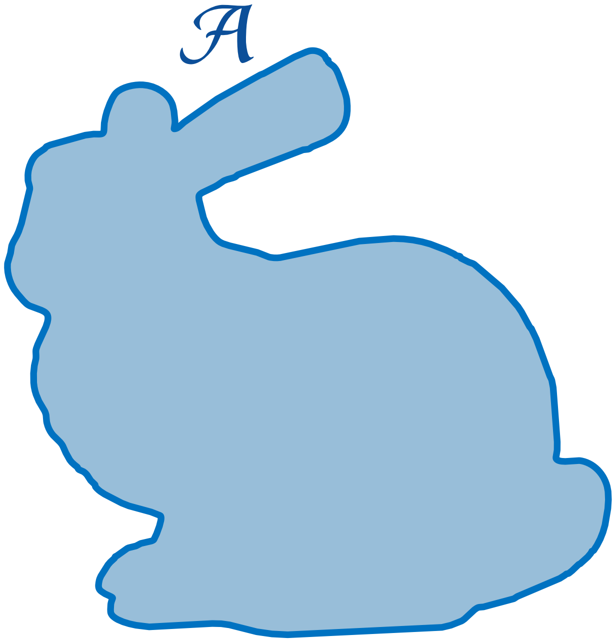


fixed position+rotation fixed rotation free

We define *valid self-nesting*

Given:

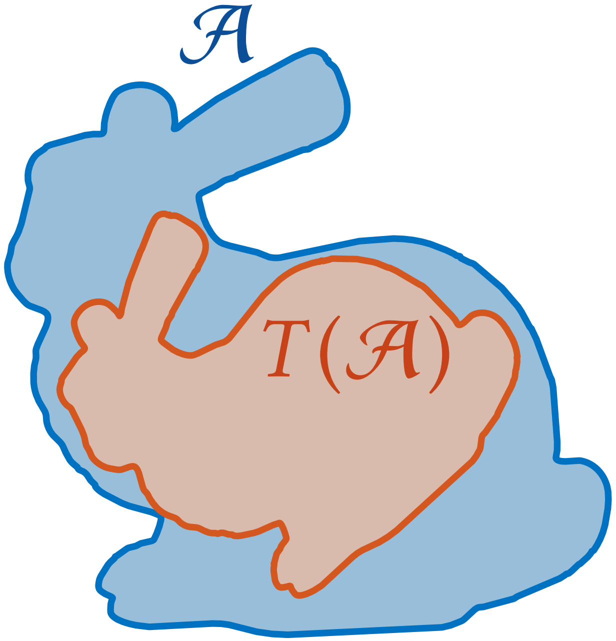
1. shape $\mathcal{A}$,



We define *valid self-nesting*

Given:

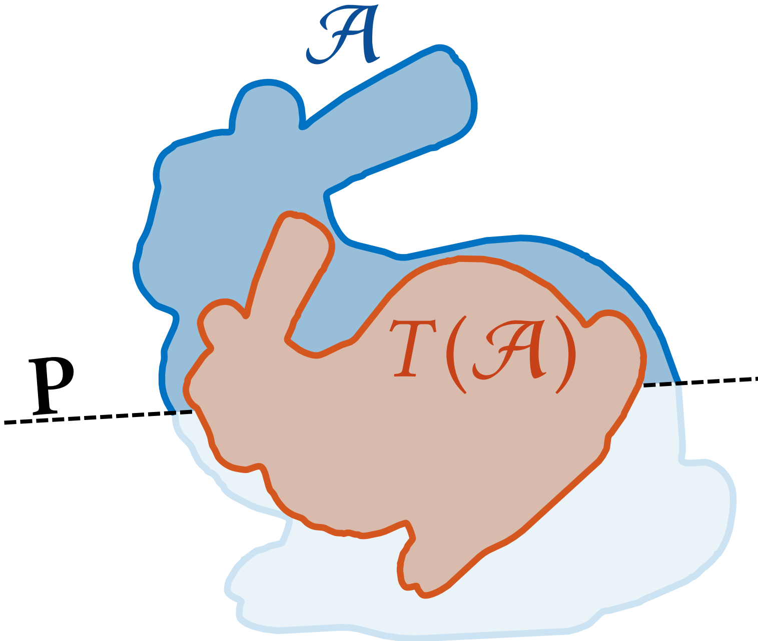
1. shape $\mathcal{A}$,
2. similarity transform T,



We define *valid self-nesting*

Given:

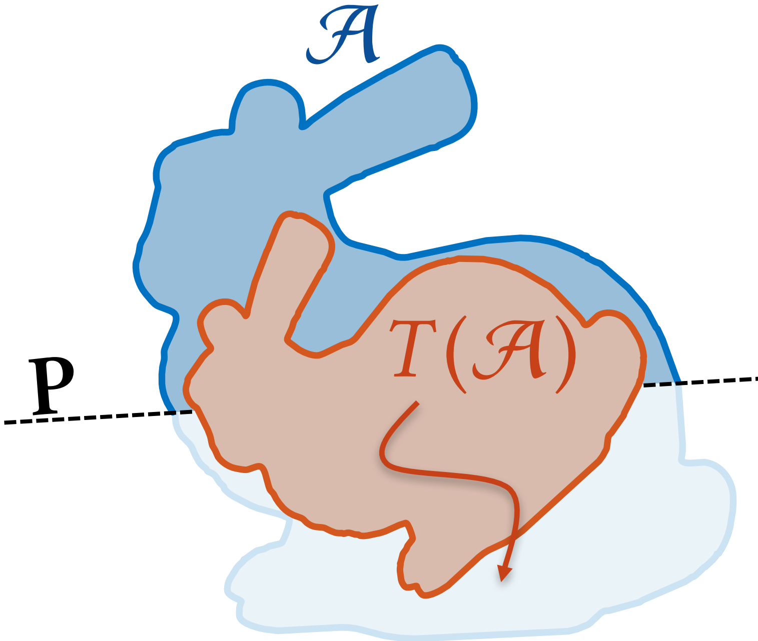
1. shape $\mathcal{A}$,
2. similarity transform T,
3. cut plane $\mathbf{P}$, and



We define *valid self-nesting*

Given:

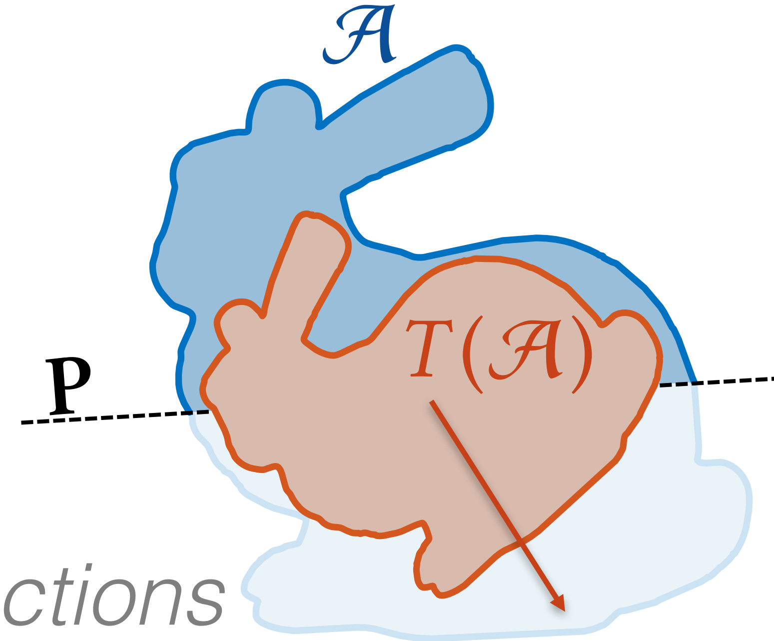
1. shape $\mathcal{A}$,
2. similarity transform T,
3. cut plane $\mathbf{P}$, and
4. removal *trajectories*



We define *valid self-nesting*

Given:

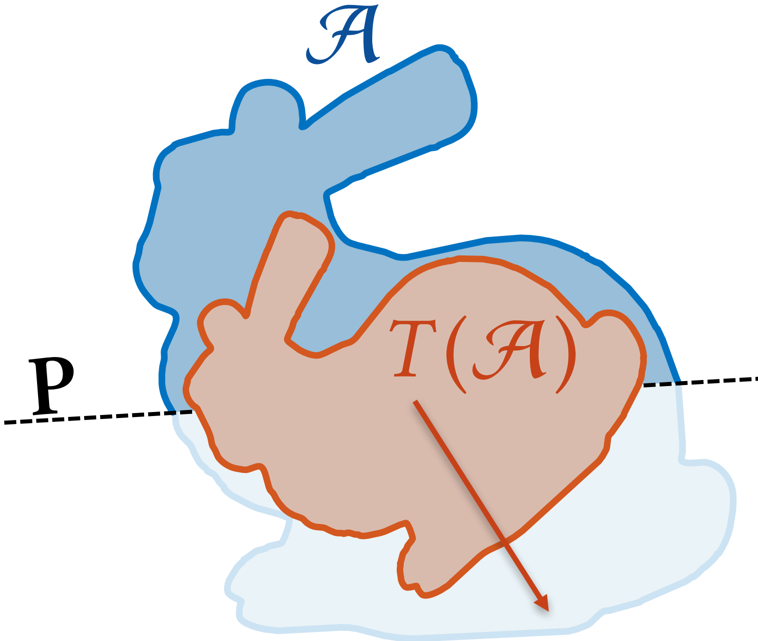
1. shape $\mathcal{A}$,
2. similarity transform T,
3. cut plane $\mathbf{P}$, and
4. removal ~~*trajectories*~~ *directions*



We define *valid self-nesting*

Must have:

1. $T(\mathcal{A}) \subset \mathcal{A}$, and
2. no collisions along either direction after cutting $\mathcal{A}$ by $\mathbf{P}$



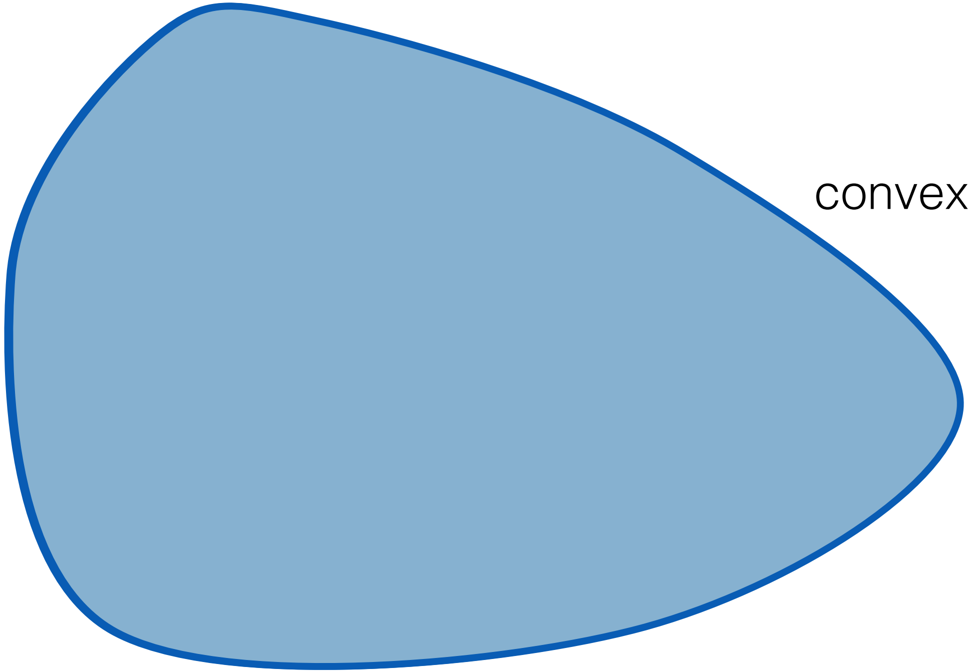
We define *valid self-nesting*

Must have:

1. $T(\mathcal{A}) \subset \mathcal{A}$, and
2. no collisions along either direction after cutting $\mathcal{A}$ by $\mathbf{P}$

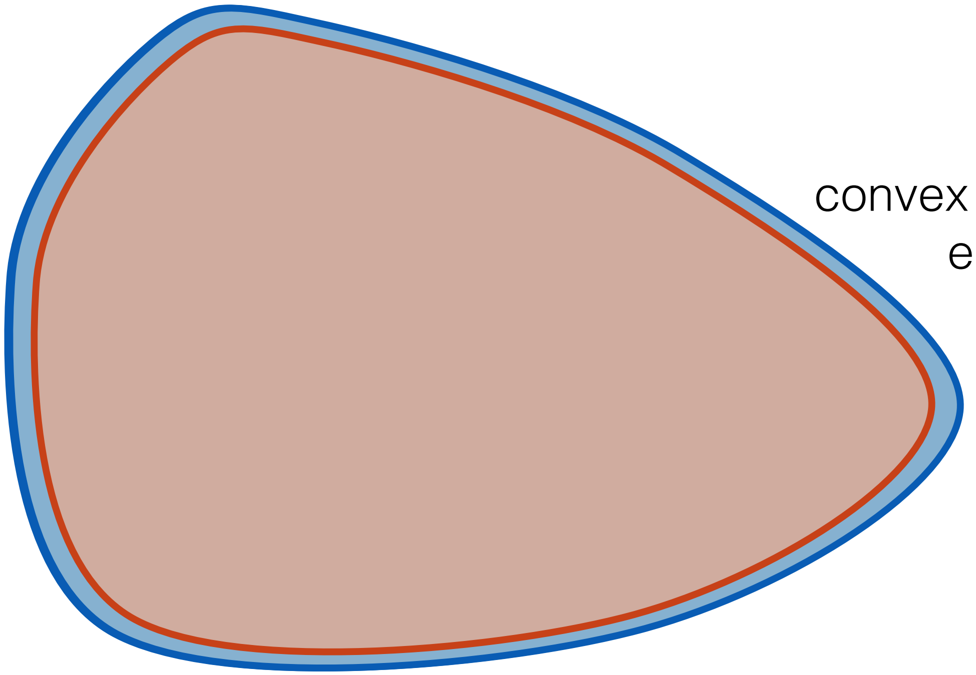
Definition depends on choice of cut plane and removal directions.

Some configurations admit *perfect self-nesting*



convex shapes?

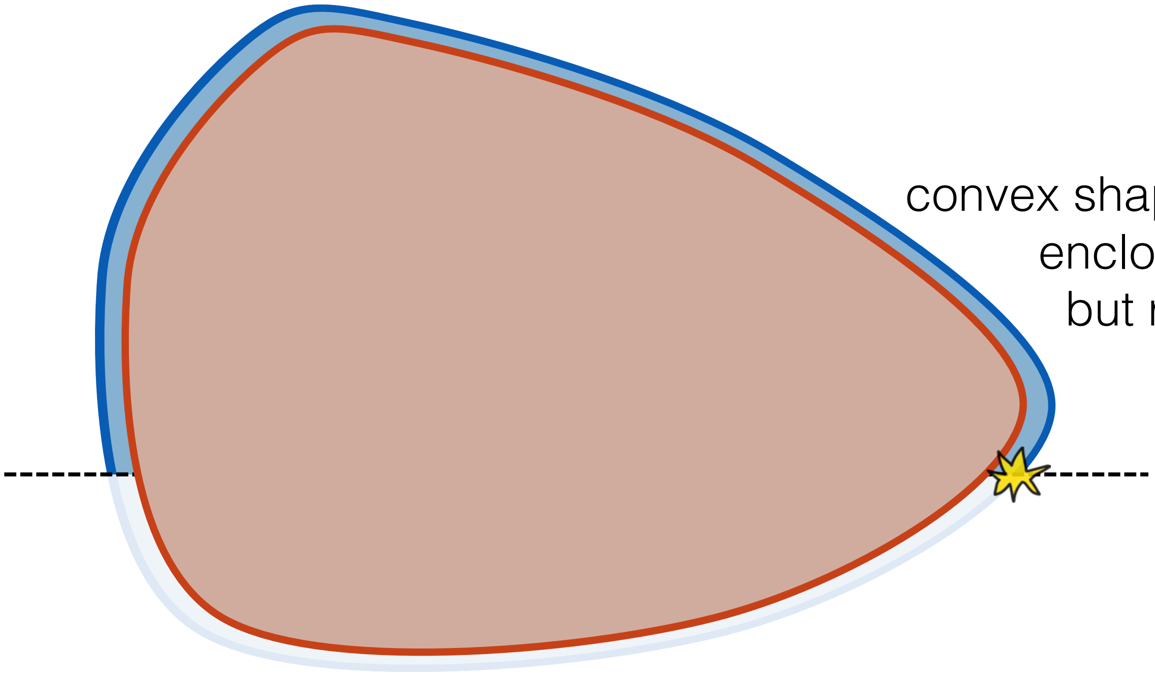
Some configurations admit *perfect self-nesting*



convex shapes?

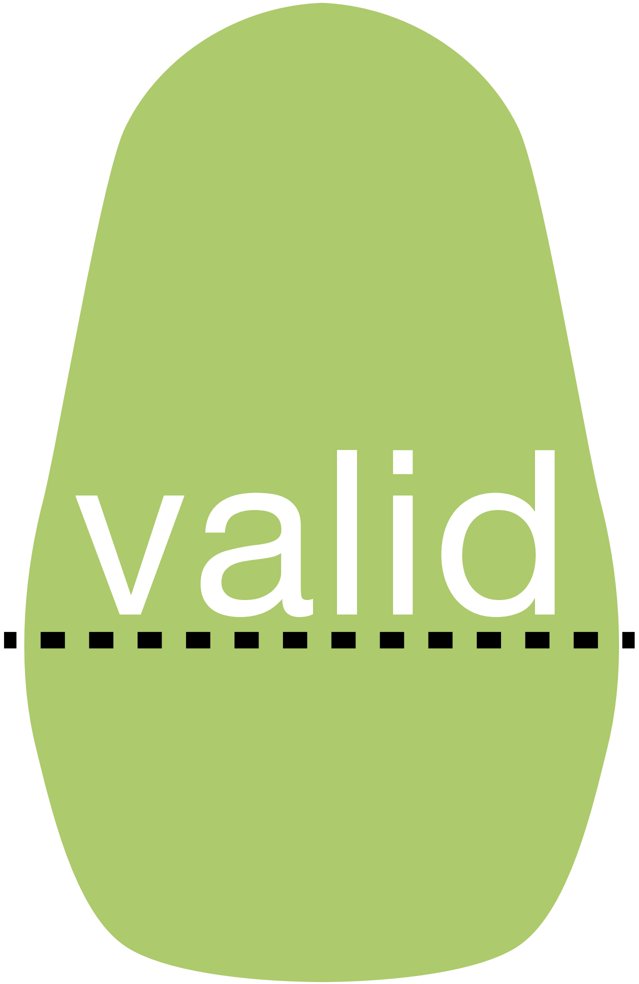
enclosure is easy ….

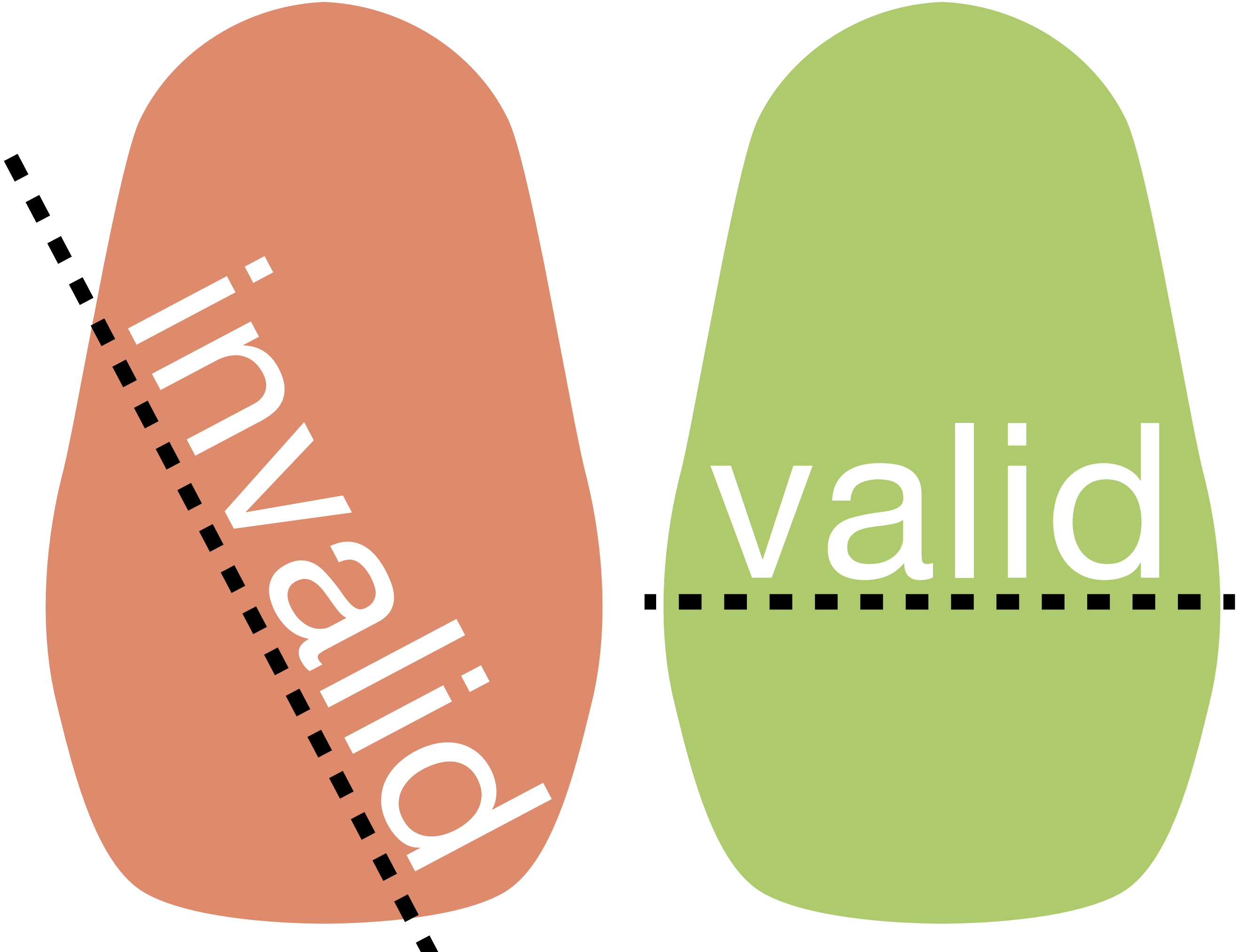
Some configurations admit *perfect self-nesting*

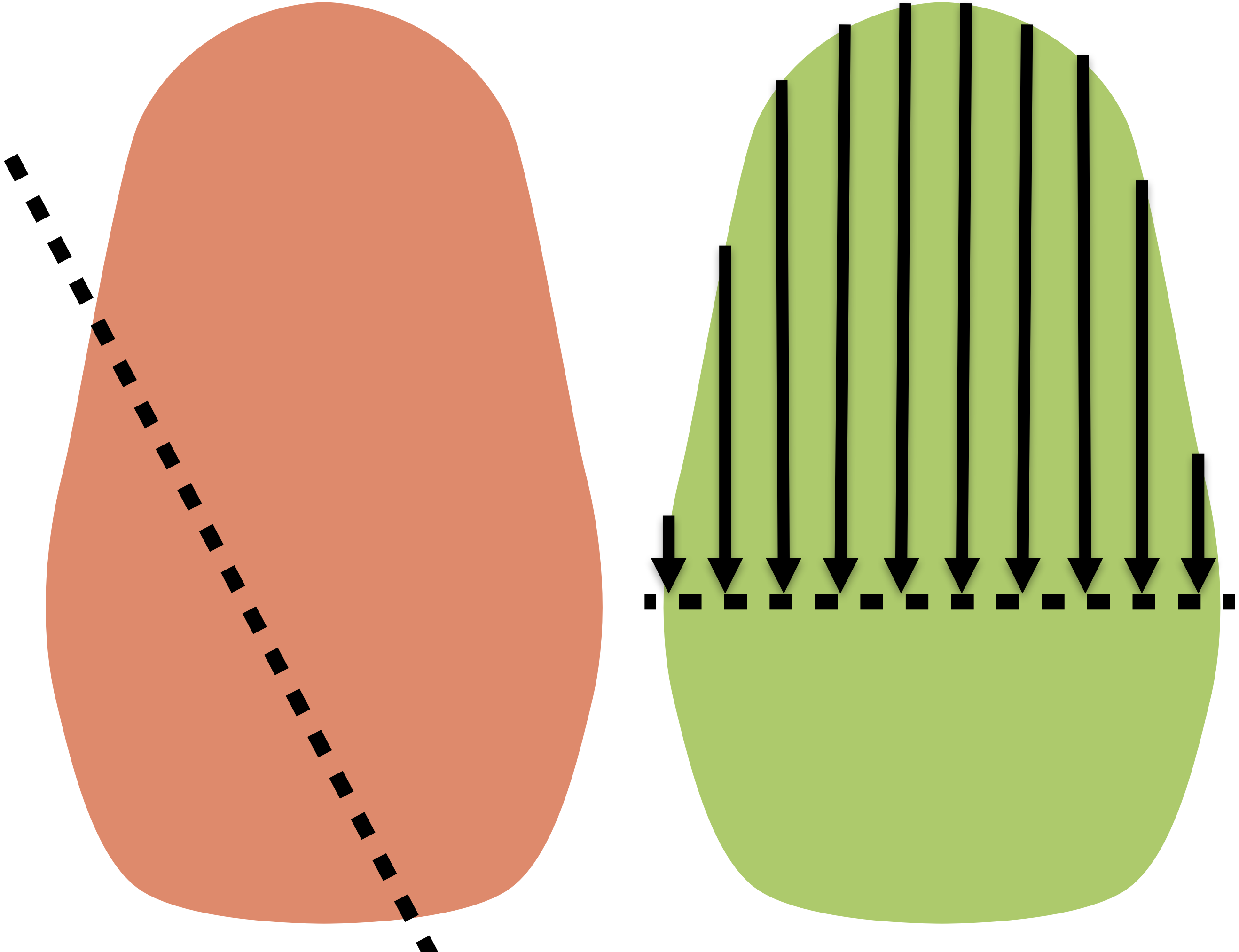


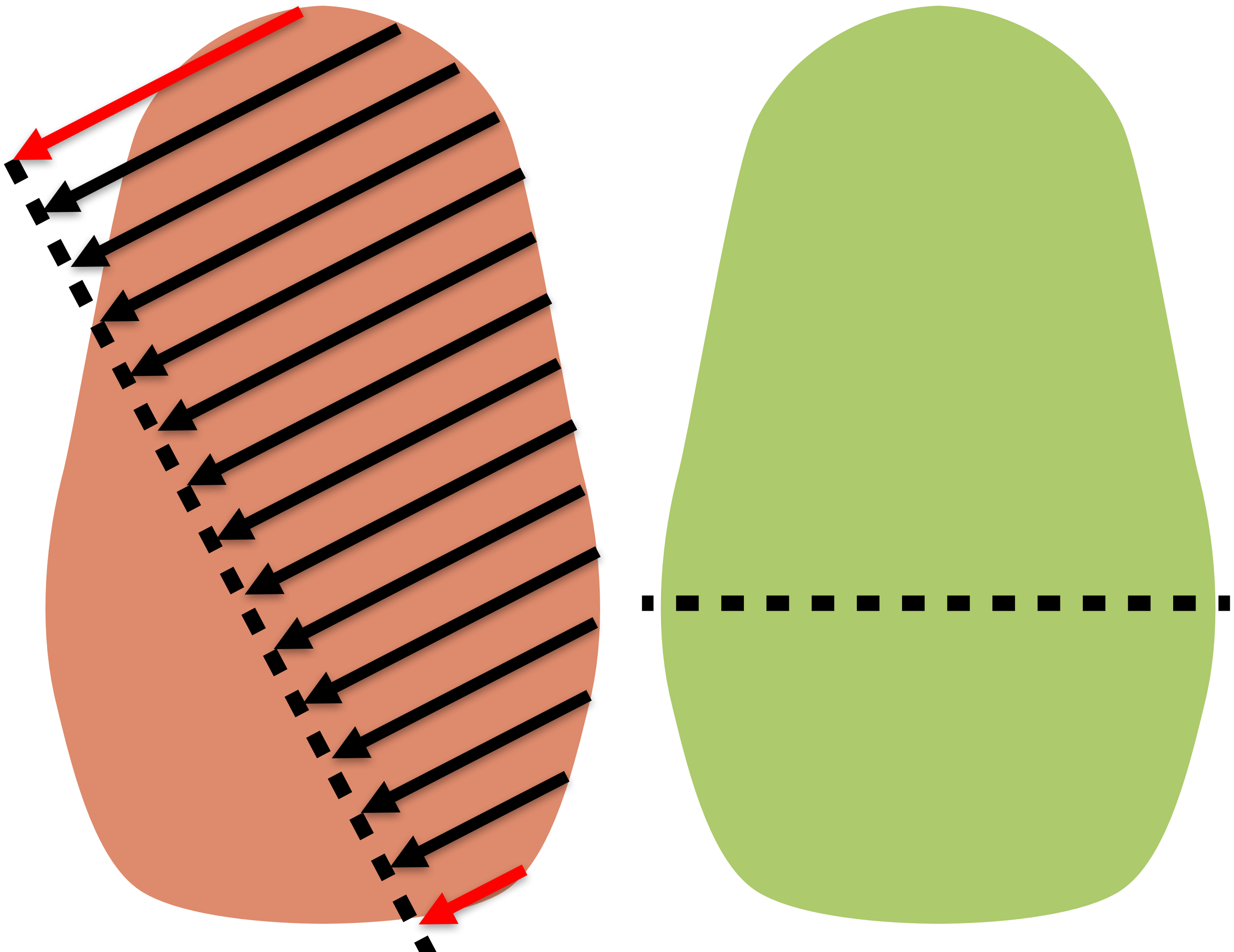
convex shapes?
enclosure is easy ….
but removal depends on cut plane!

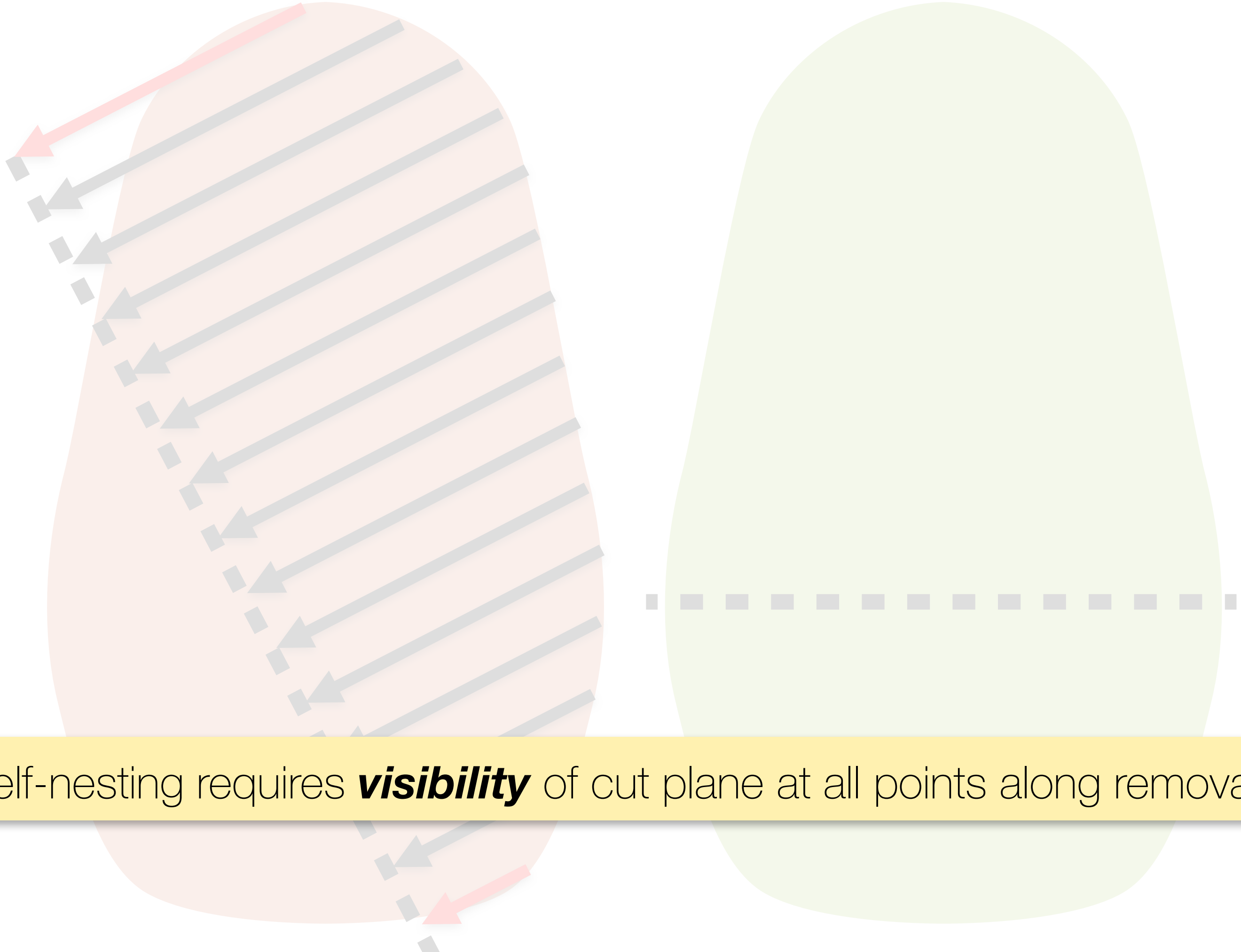
[Zvyozdochkin & Malyutin 1890]


valid


invalid
valid

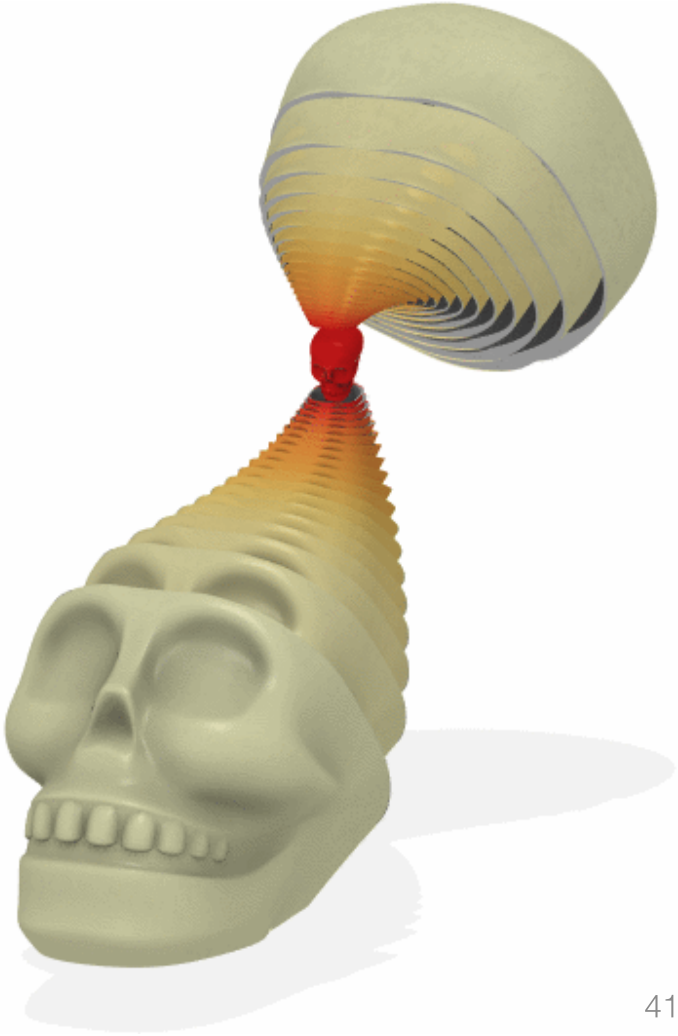




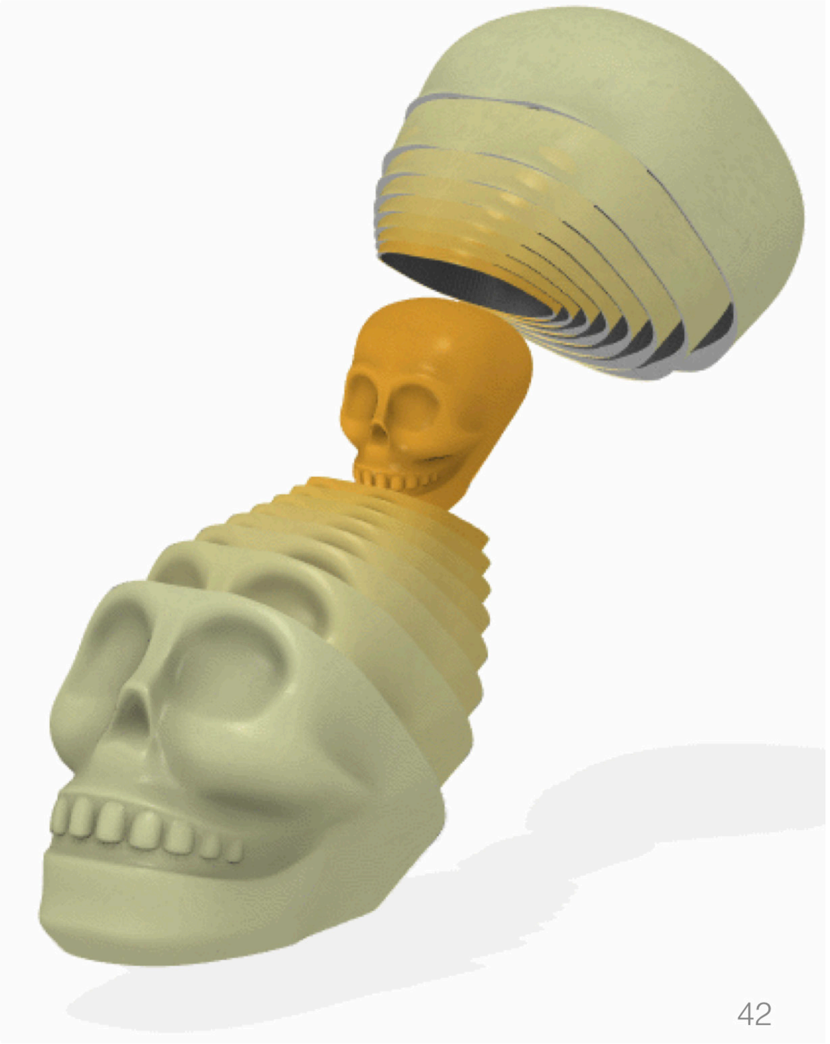


Perfect self-nesting requires ***visibility*** of cut plane at all points along removal directions

Our tool explores nesting of *arbitrary* solid 3D shapes

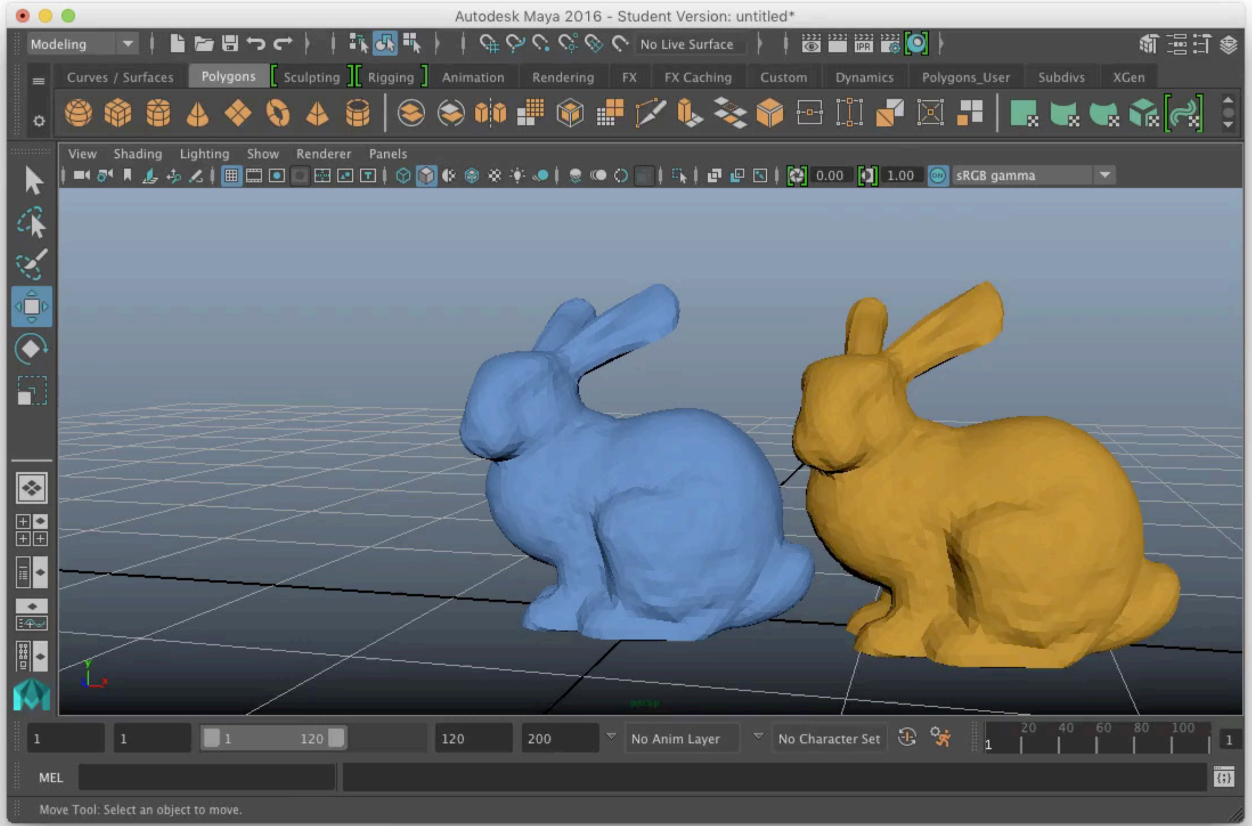


Our tool explores nesting of *arbitrary* solid 3D shapes



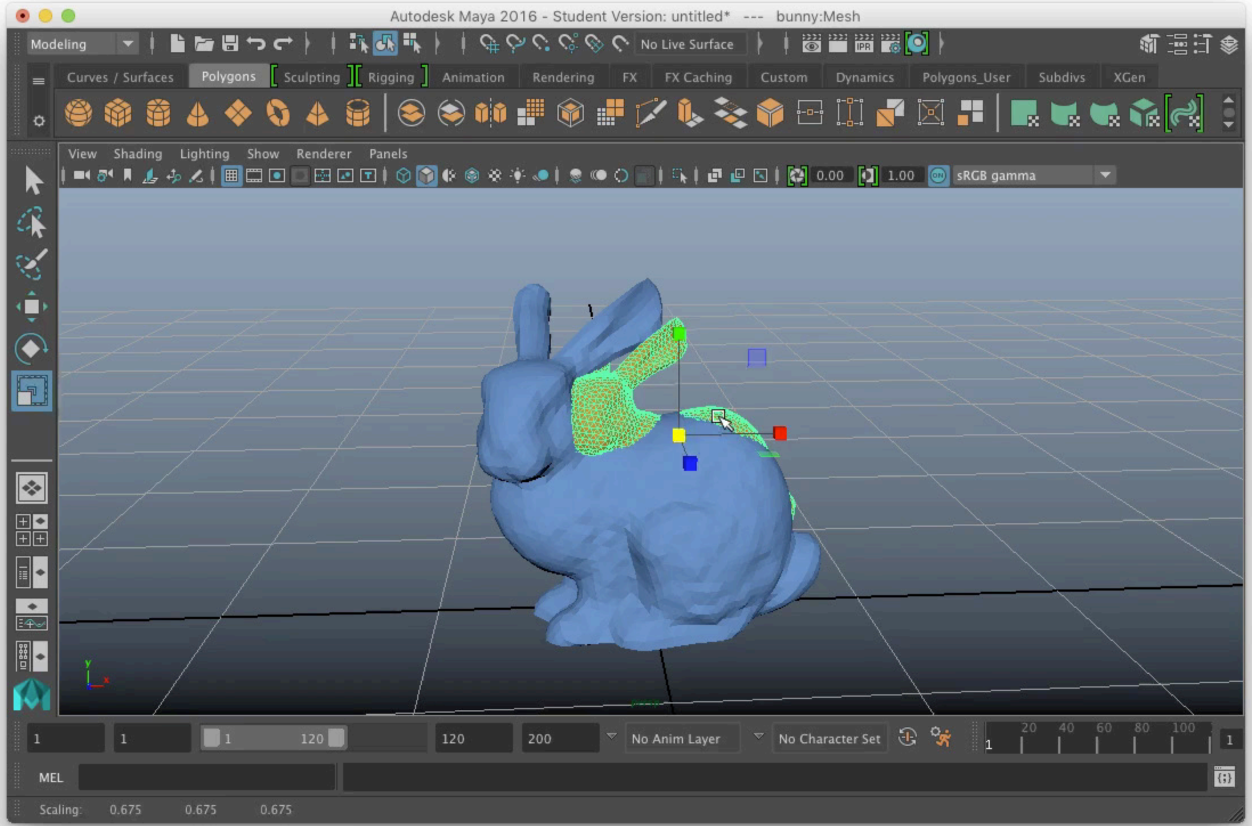
Our tool explores nesting of *arbitrary* solid 3D shapes

We cast this as a *computational design* problem



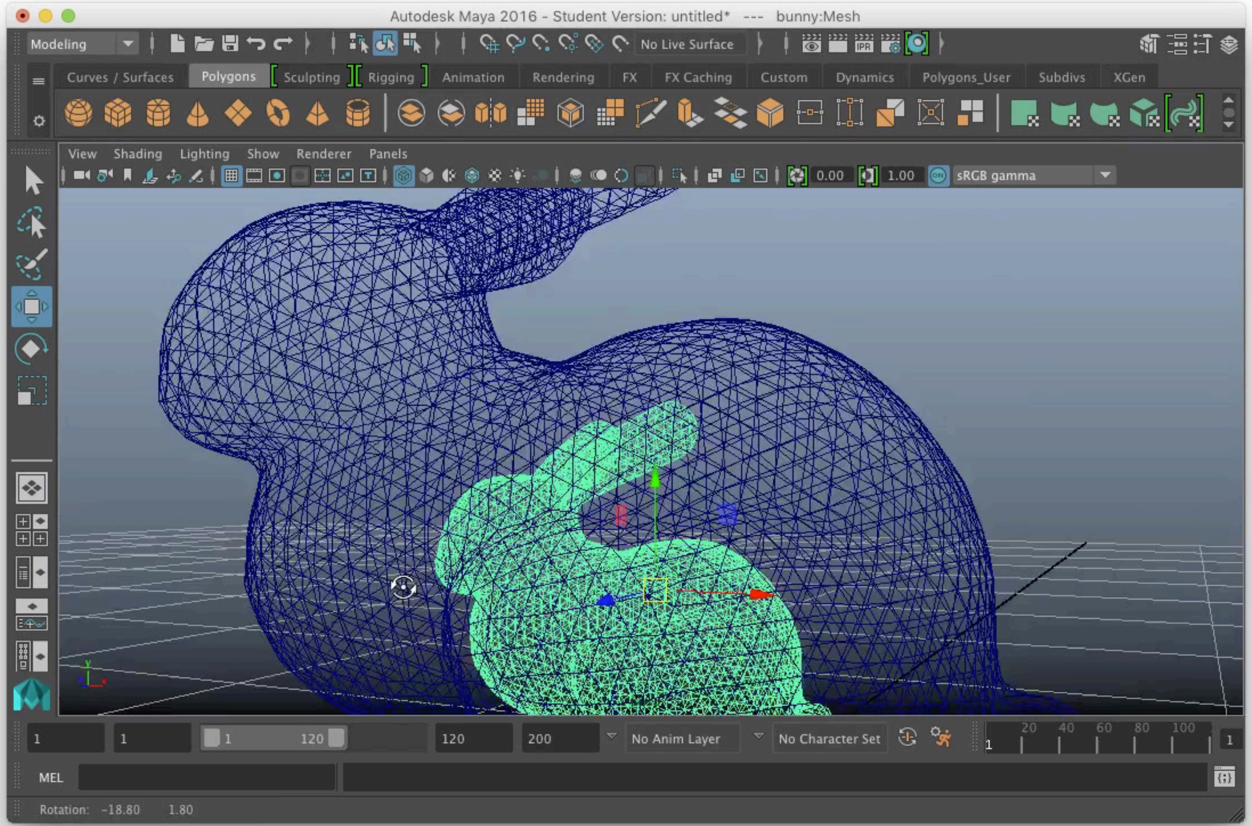
Manual design with traditional tools would be tortuous

We cast this as a *computational design* problem



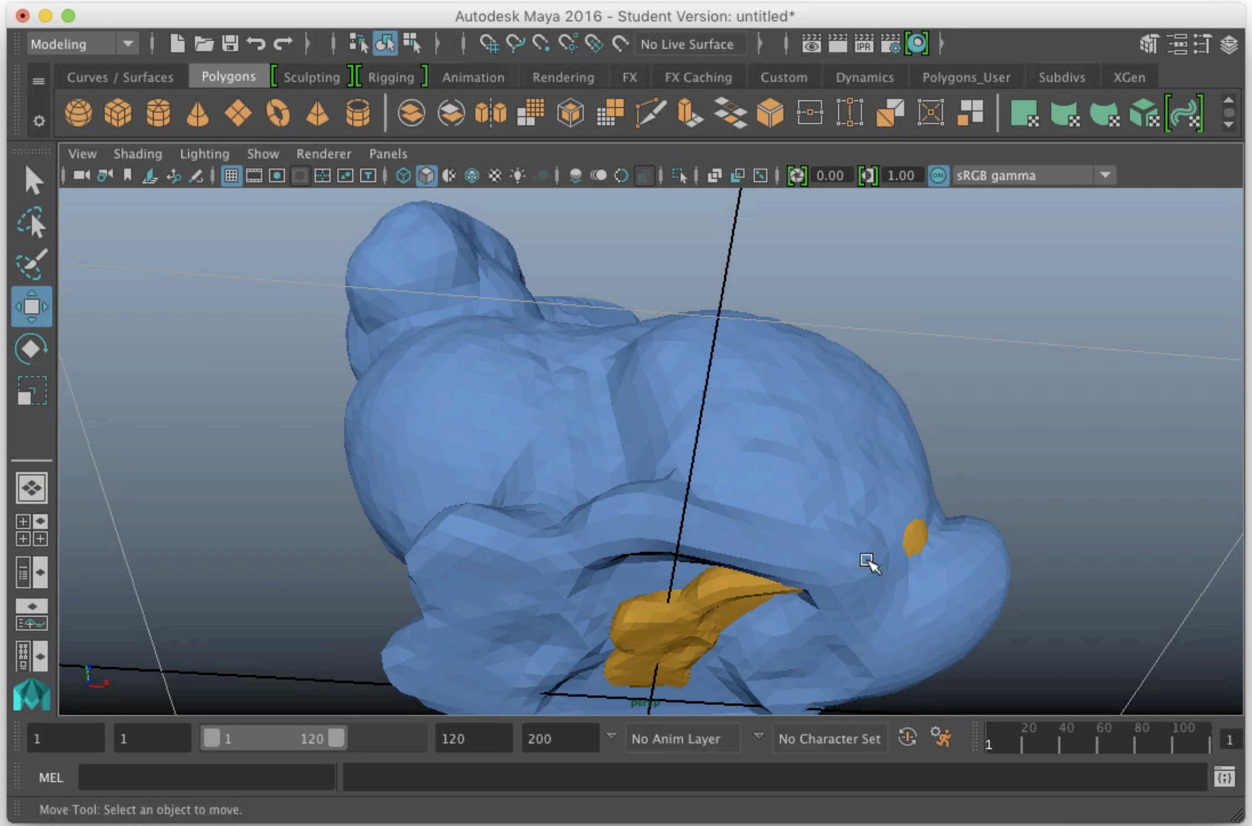
Manual design with traditional tools would be tortuous

We cast this as a *computational design* problem



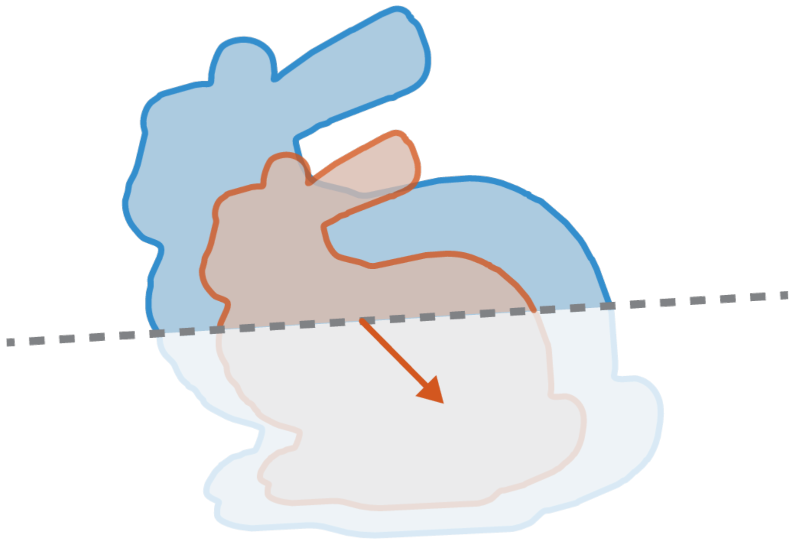
Manual design with traditional tools would be tortuous

We cast this as a *computational design* problem

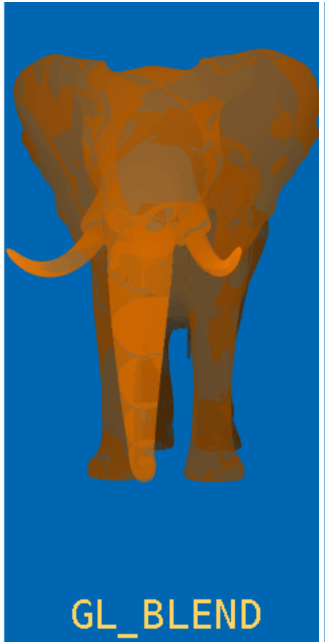


Manual design with traditional tools would be tortuous

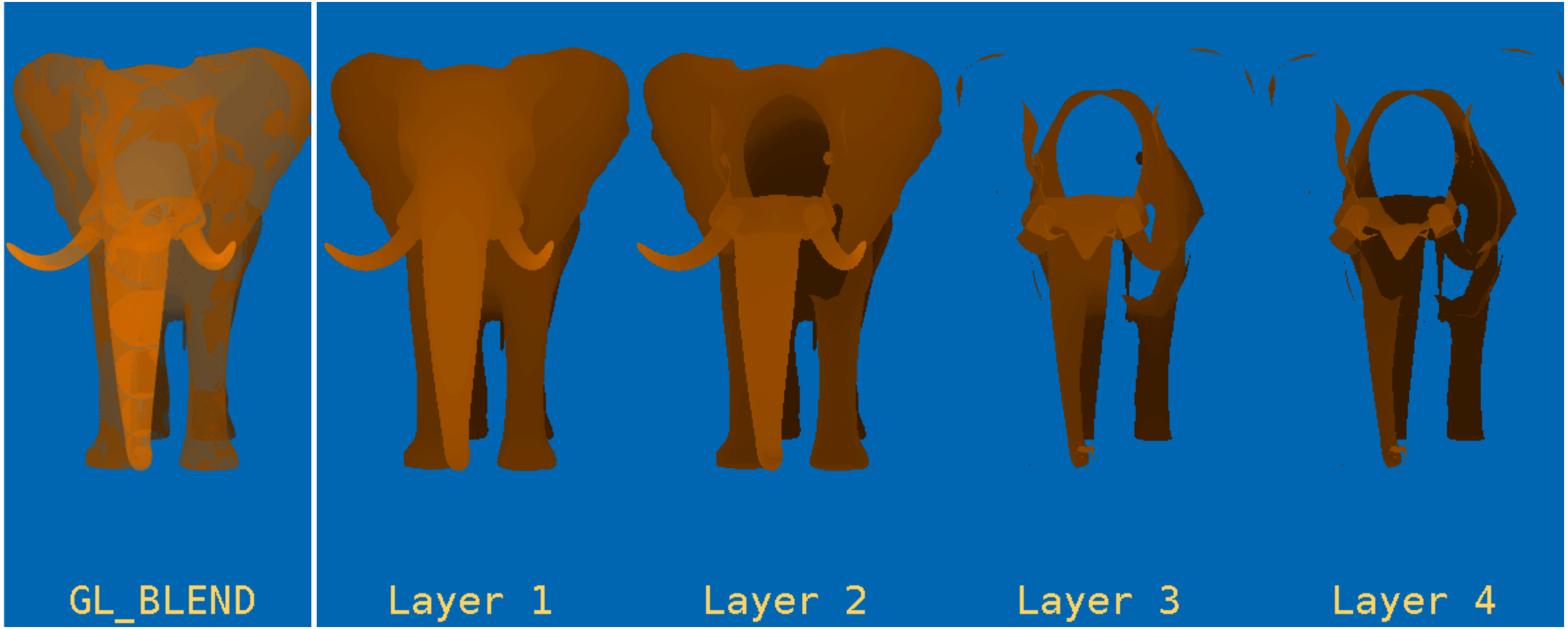
Step 1: we determine feasibility in real-time by exploiting orthographic rendering



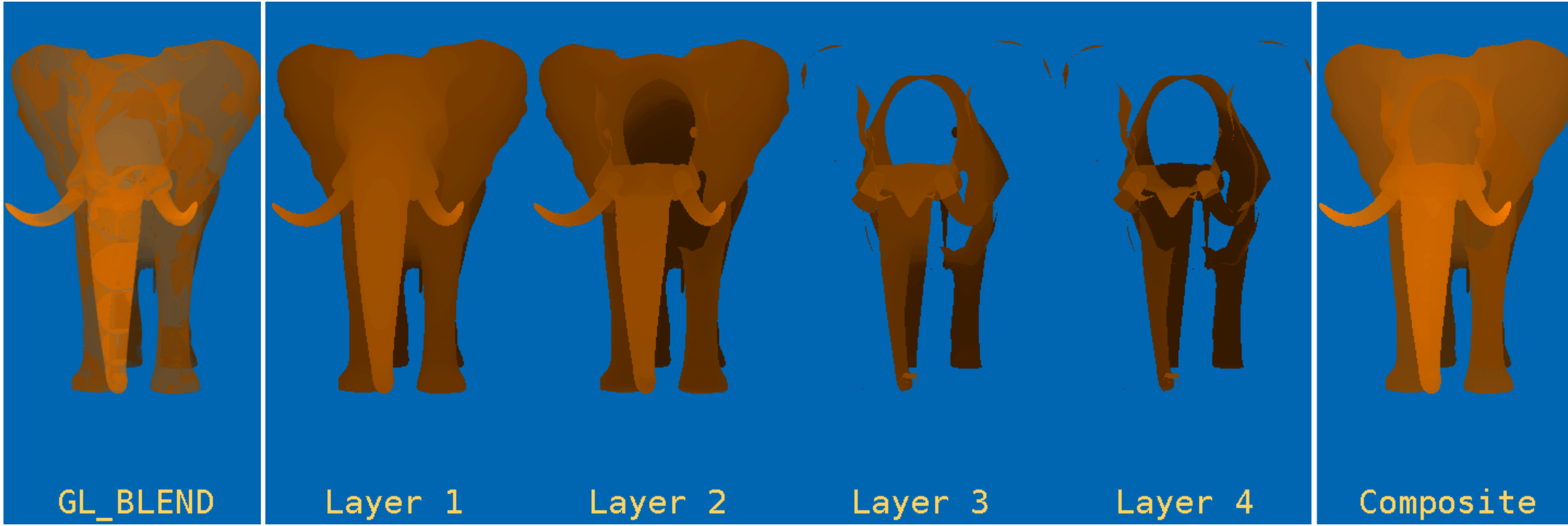
Take a clue from order-independent transparency by "depth peeling"



Take a clue from order-independent transparency by “depth peeling”



Take a clue from order-independent transparency by "depth peeling"



Take a clue from order-independent transparency by "depth peeling"

a.k.a. K-Buffer, Layered Depth Images

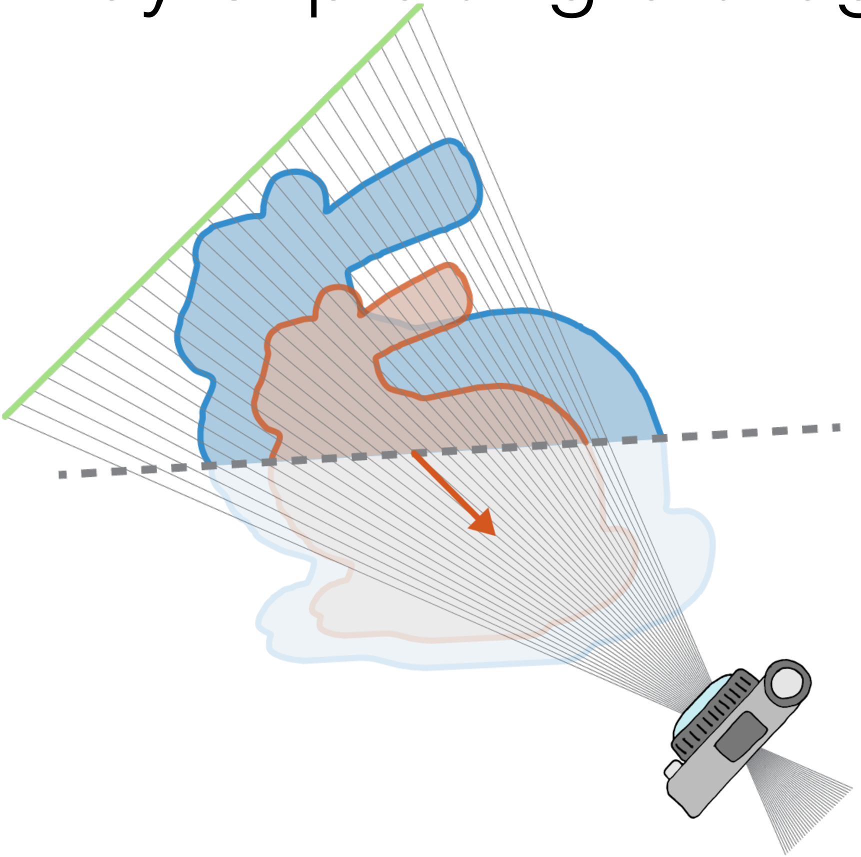
transparency	[Everitt 2001, Bavoil et al. 2007]
shape diameter	[Baldacci et al. 2016]
image-based rendering	[Shade et al. 1998]
CNC milling	[Inui & Ohta 2007]
intersection volume	[Faure et al. 2008]
swept volumes	[Kim et al. 2002]
collision detection	[Myszkowski et al.1995, Knott & Pai 2003, Heidelberger et al. 2004]
CSG operations	[Goldfeather et al. 1986, Kelley et al. 1994, Hable & Rossignac 2005]

Take a clue from order-independent transparency by “depth peeling”

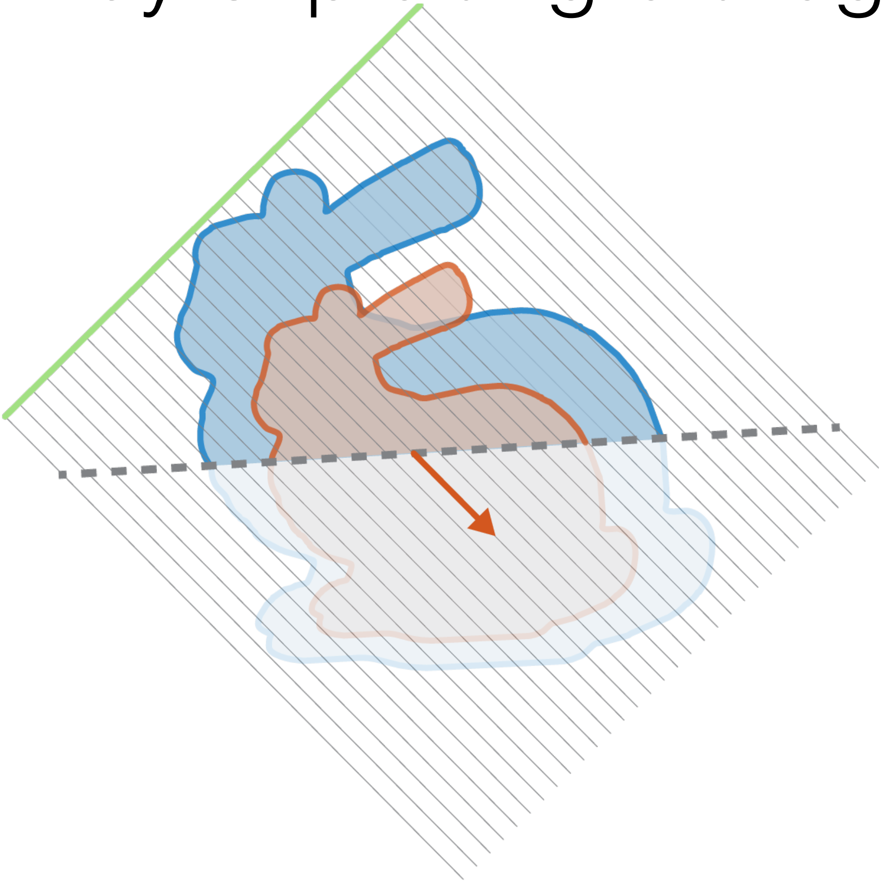
a.k.a. K-Buffer, Layered Depth Images

transparency	[Everitt 2001, Bavoil et al. 2007]
shape diameter	[Baldacci et al. 2016]
image-based rendering	[Shade et al. 1998]
CNC milling	[Inui & Ohta 2007]
intersection volume	[Faure et al. 2008]
swept volumes	[Kim et al. 2002]
collision detection	[Myszkowski et al.1995, Knott & Pai 2003, Heidelberger et al. 2004]
CSG operations	[**Goldfeather et al. 1986**, Kelley et al. 1994, Hable & Rossignac 2005]

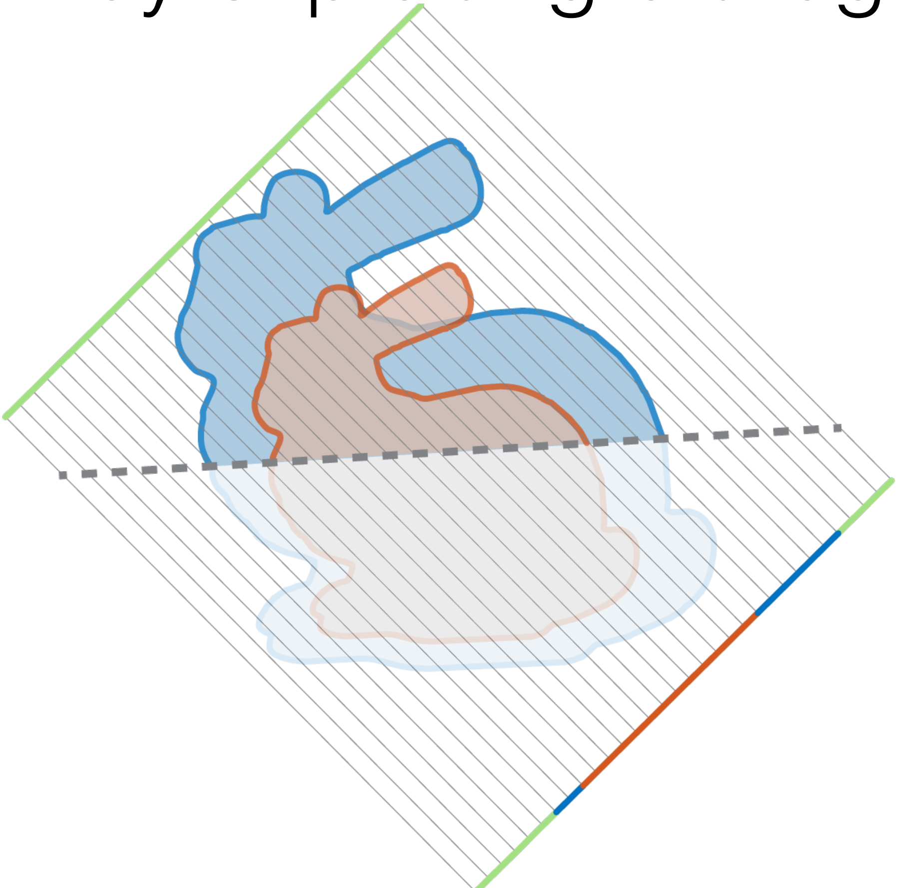
Step 1: we determine feasibility in real-time by exploiting orthographic rendering



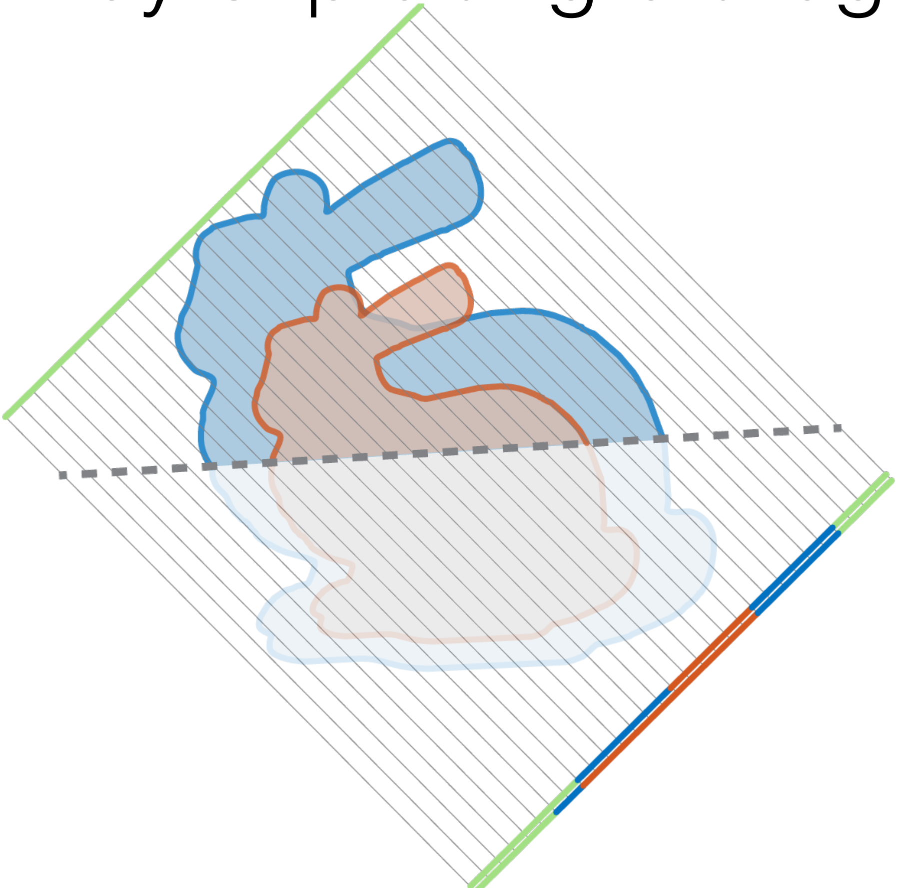
Step 1: we determine feasibility in real-time by exploiting orthographic rendering



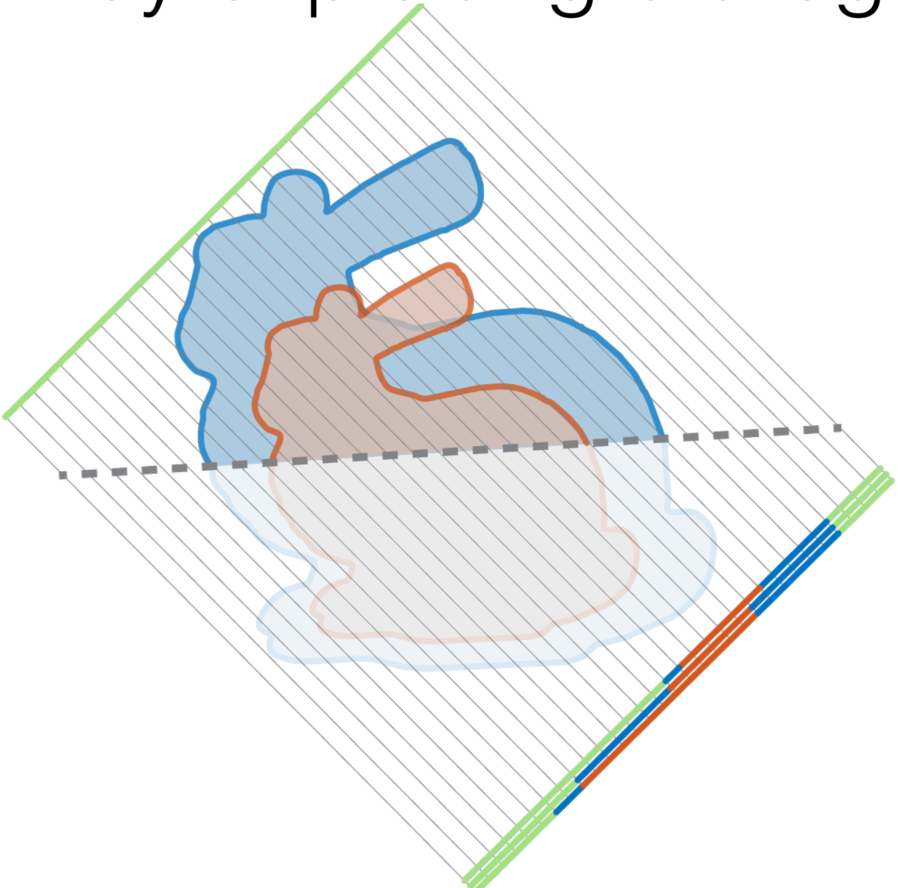
Step 1: we determine feasibility in real-time by exploiting orthographic rendering



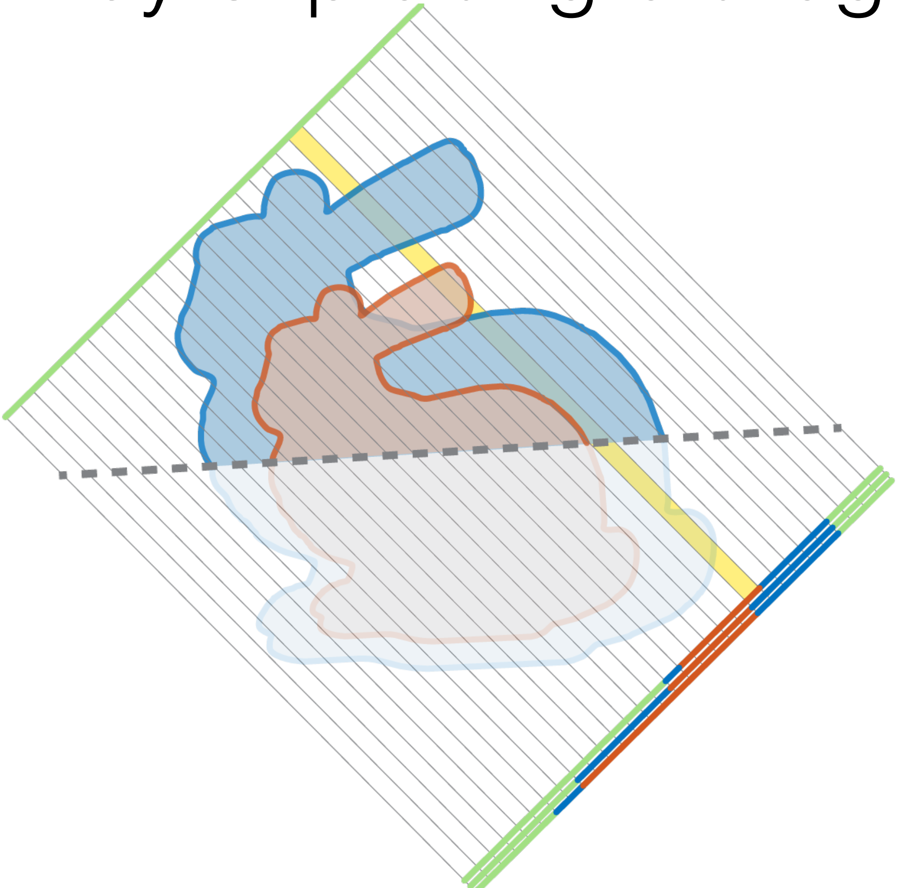
Step 1: we determine feasibility in real-time by exploiting orthographic rendering



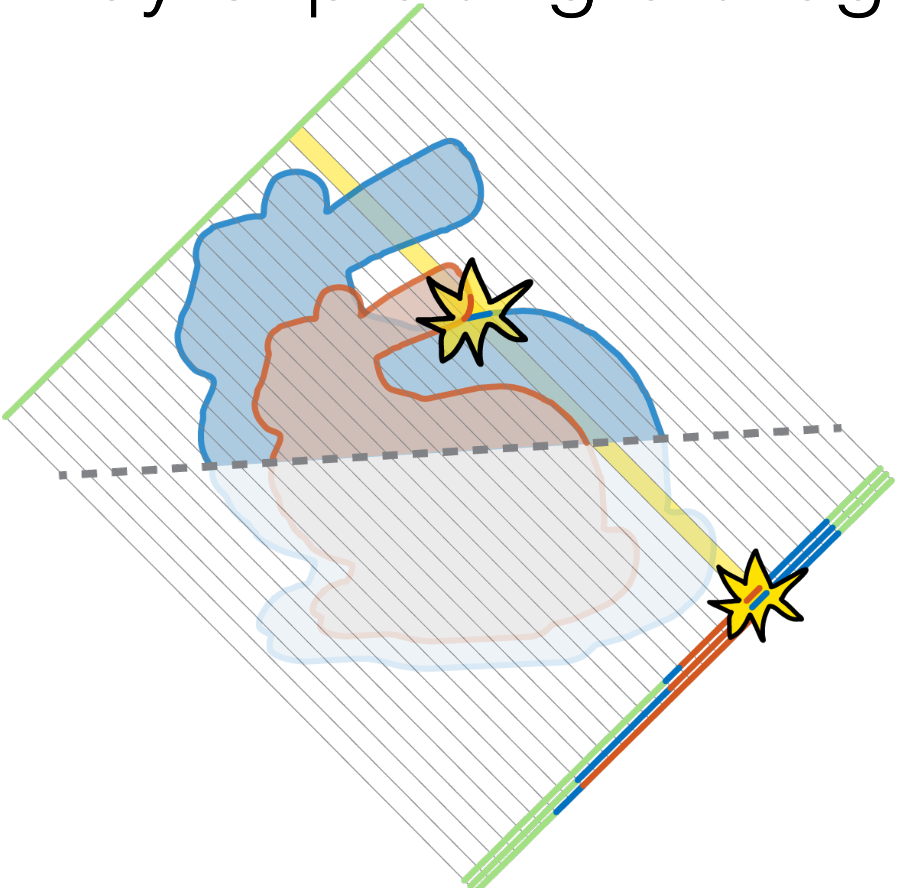
Step 1: we determine feasibility in real-time by exploiting orthographic rendering



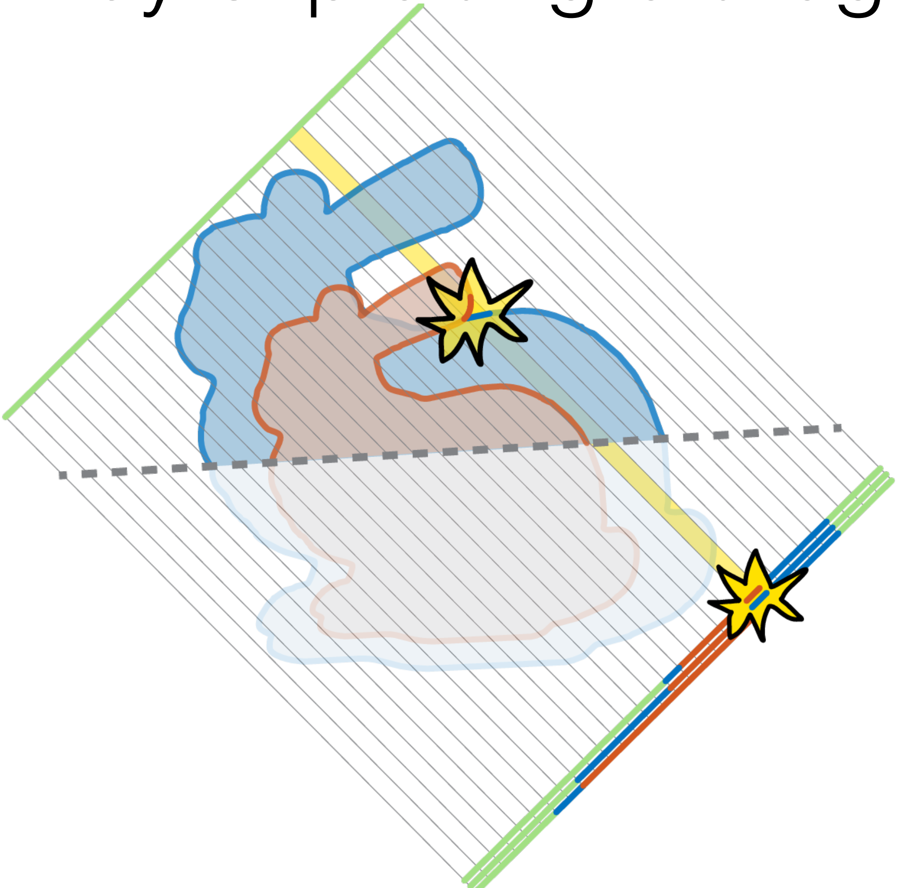
Step 1: we determine feasibility in real-time by exploiting orthographic rendering



Step 1: we determine feasibility in real-time by exploiting orthographic rendering



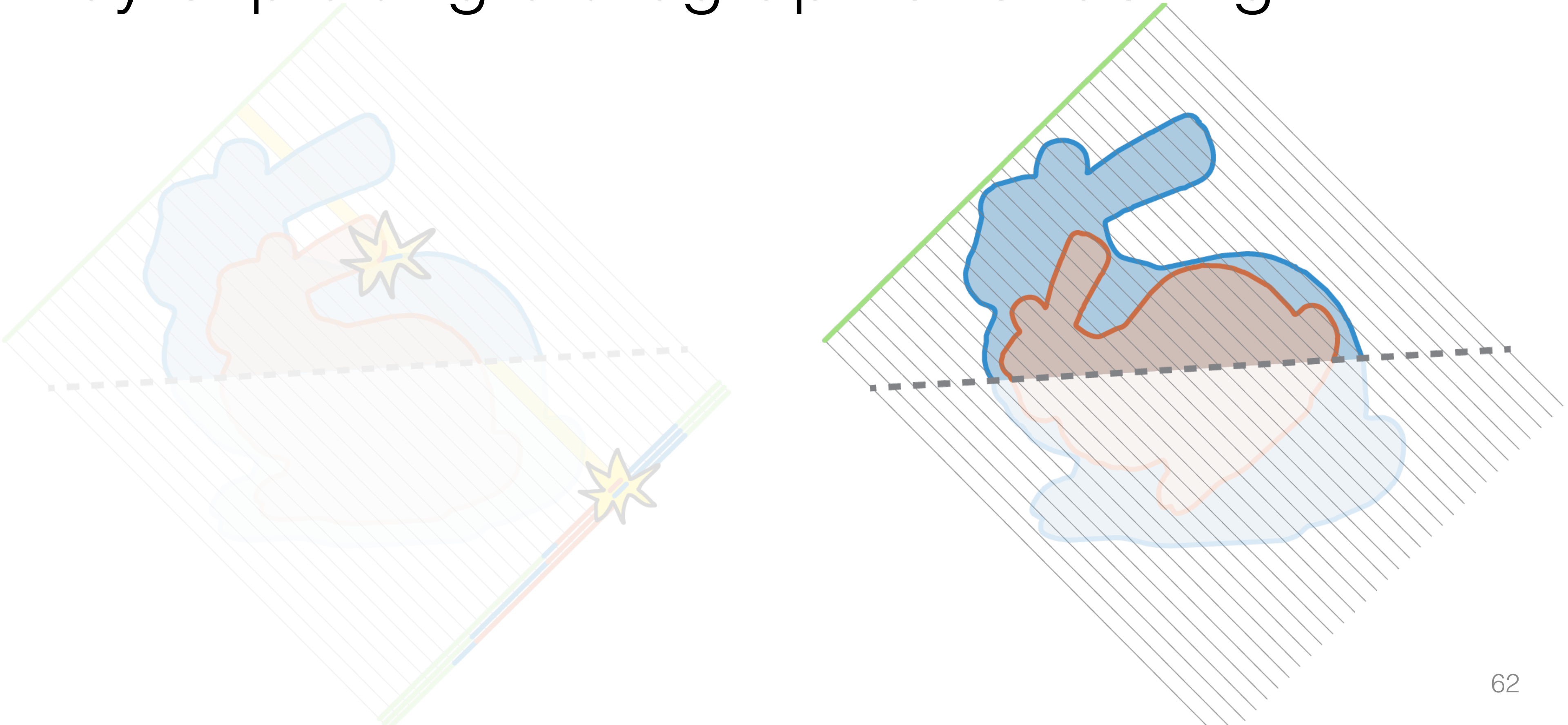
Step 1: we determine feasibility in real-time by exploiting orthographic rendering



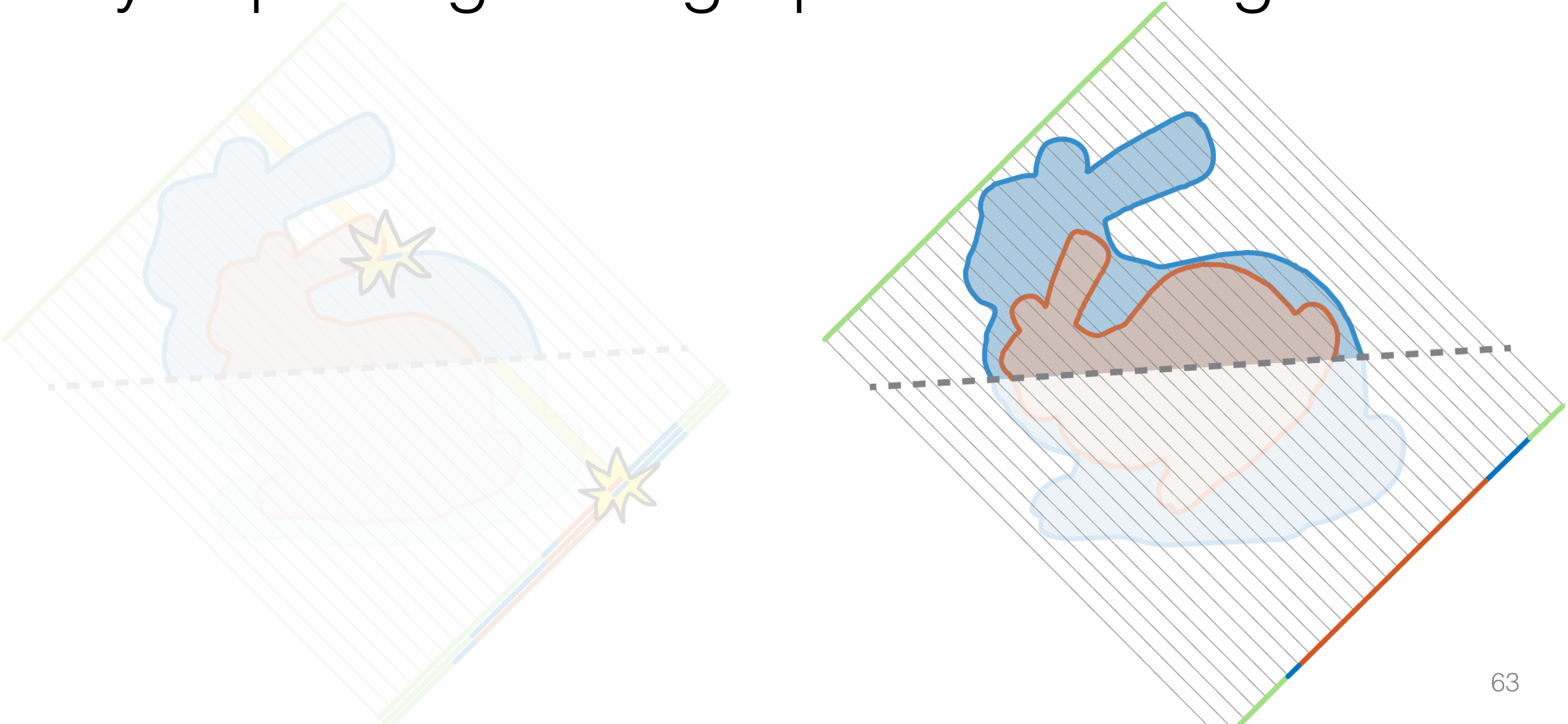
Bad “codes”:

- **blue** before **orange**
- **orange** before **green**
- **orange** before front-facing **blue**

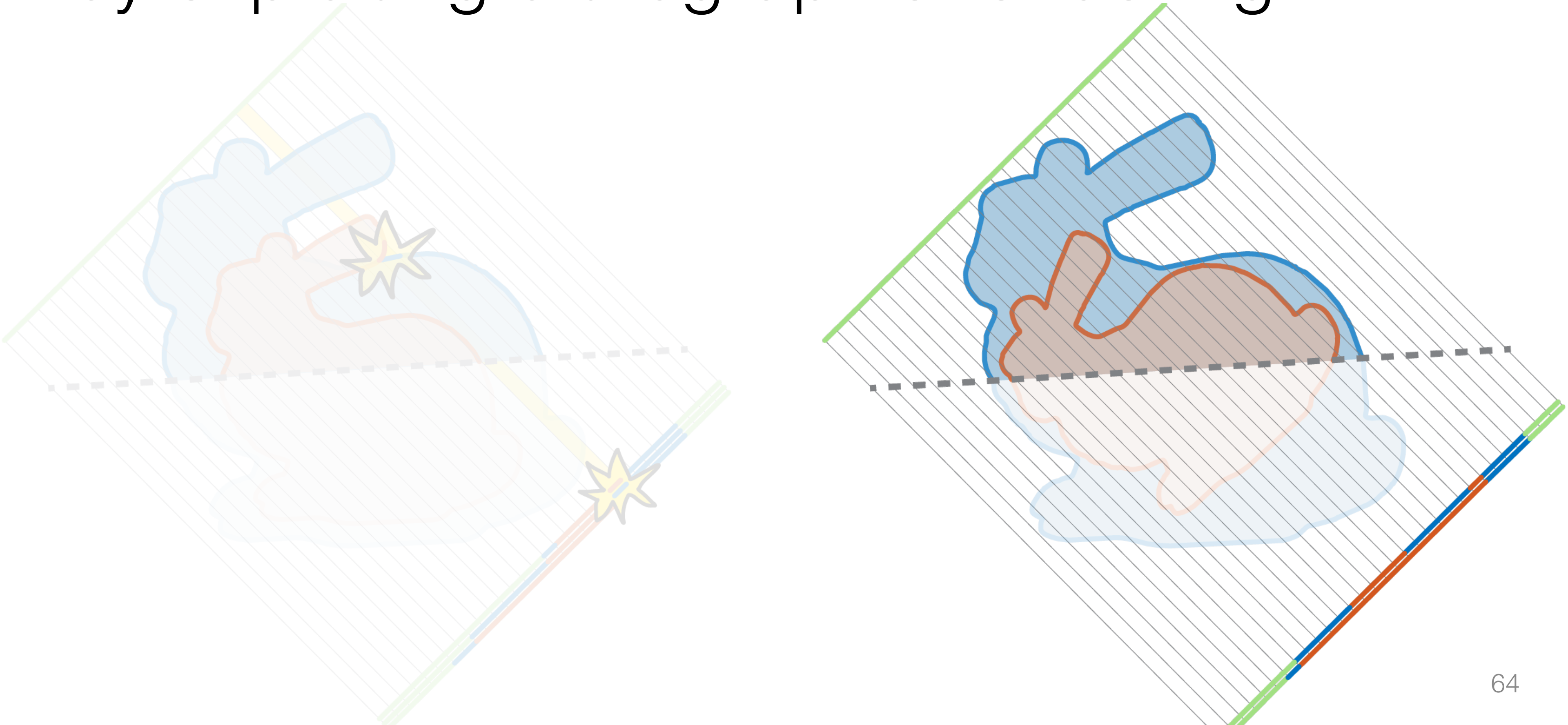
Step 1: we determine feasibility in real-time by exploiting orthographic rendering



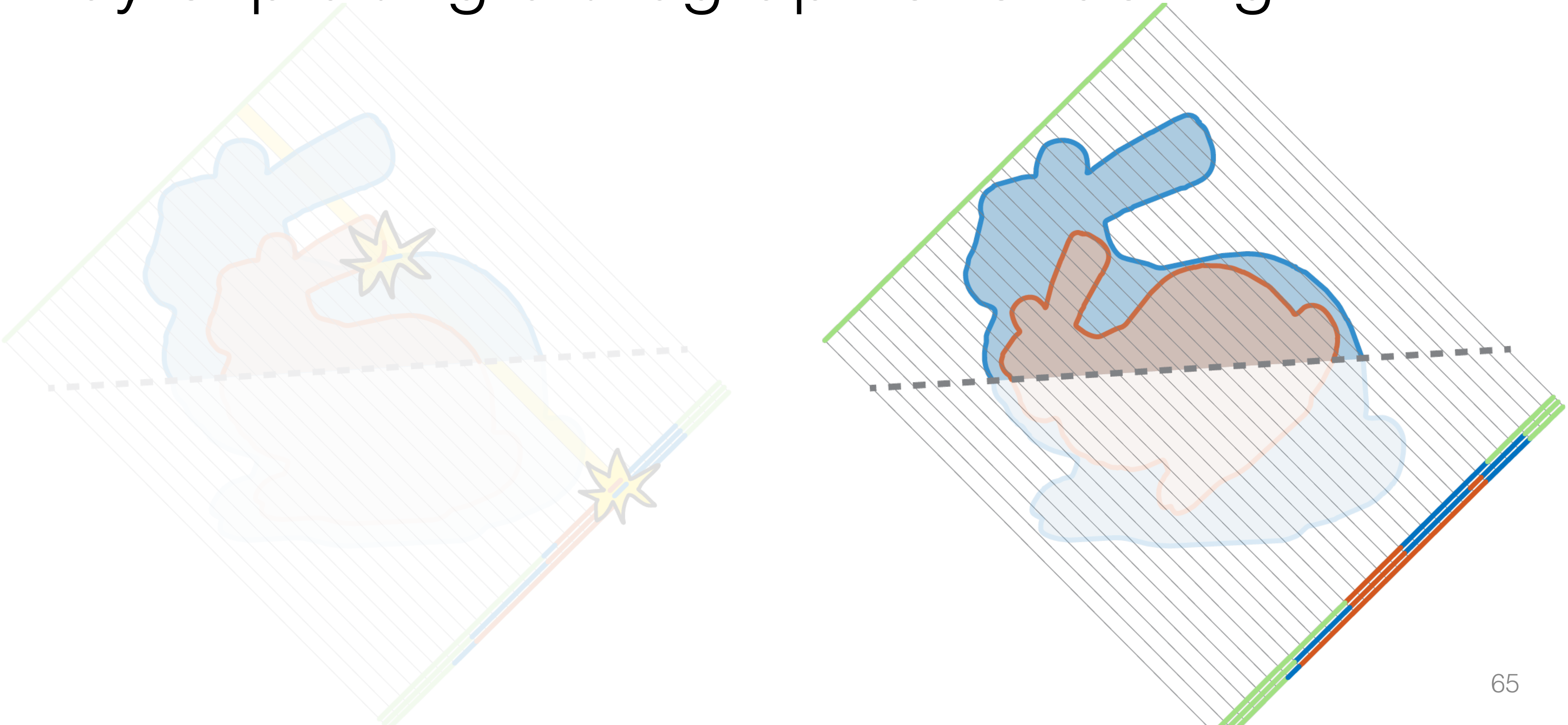
Step 1: we determine feasibility in real-time by exploiting orthographic rendering



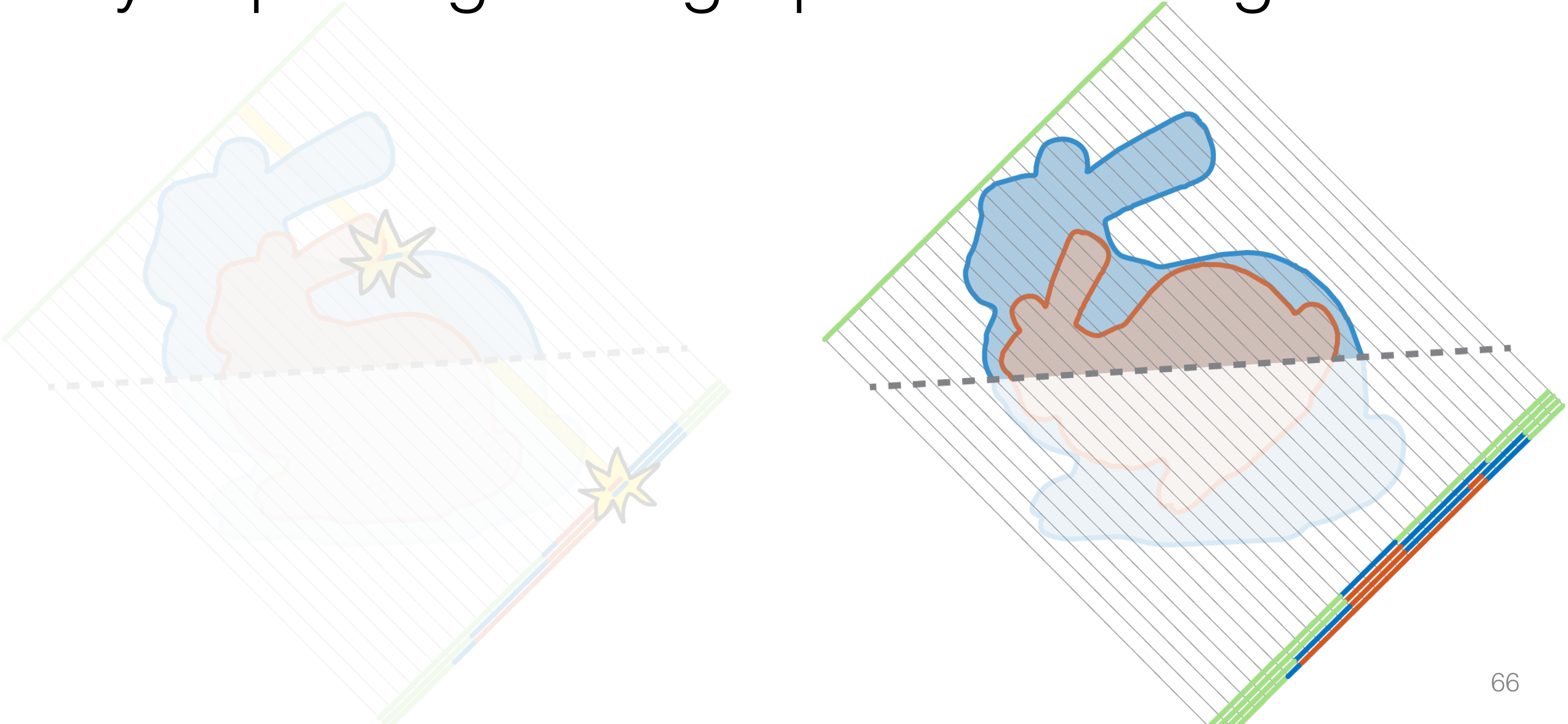
Step 1: we determine feasibility in real-time by exploiting orthographic rendering



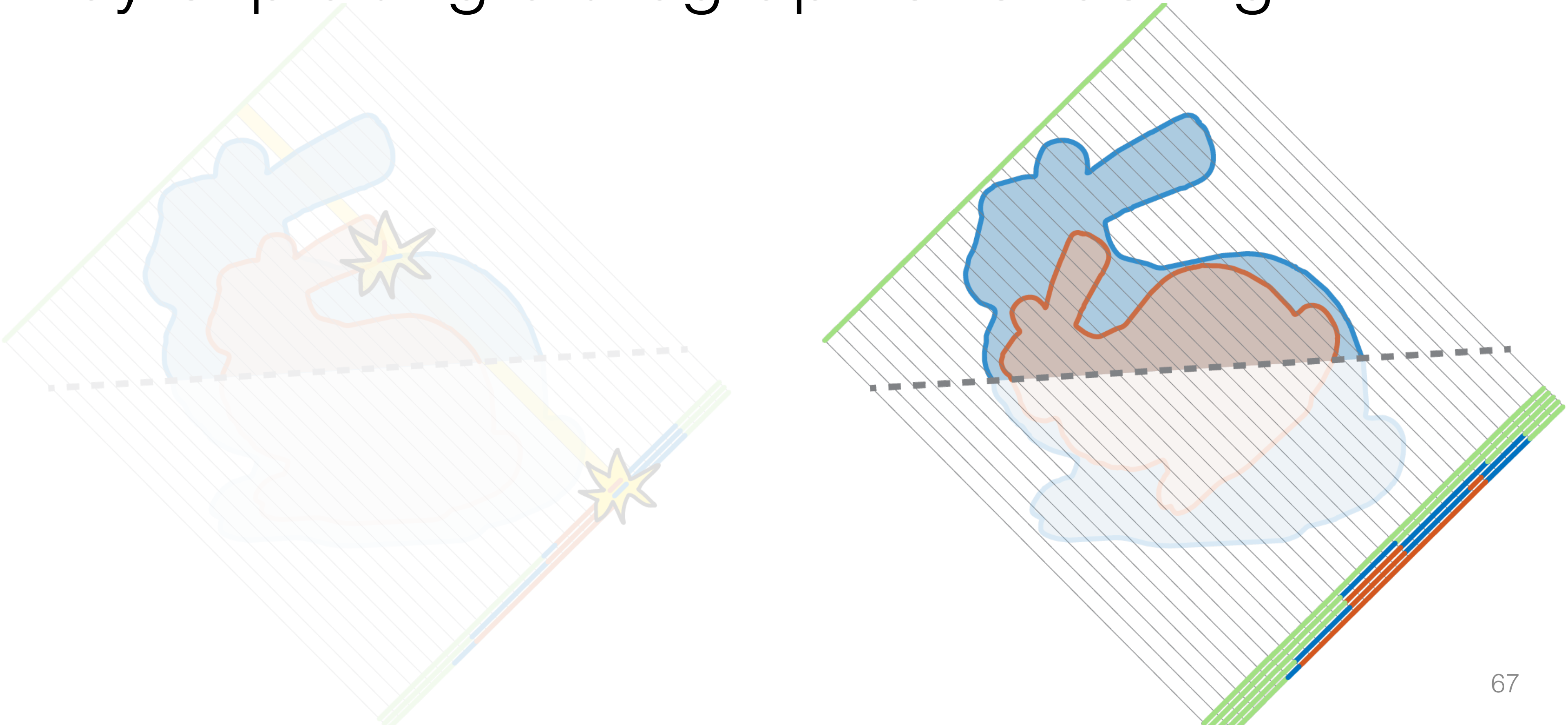
Step 1: we determine feasibility in real-time by exploiting orthographic rendering



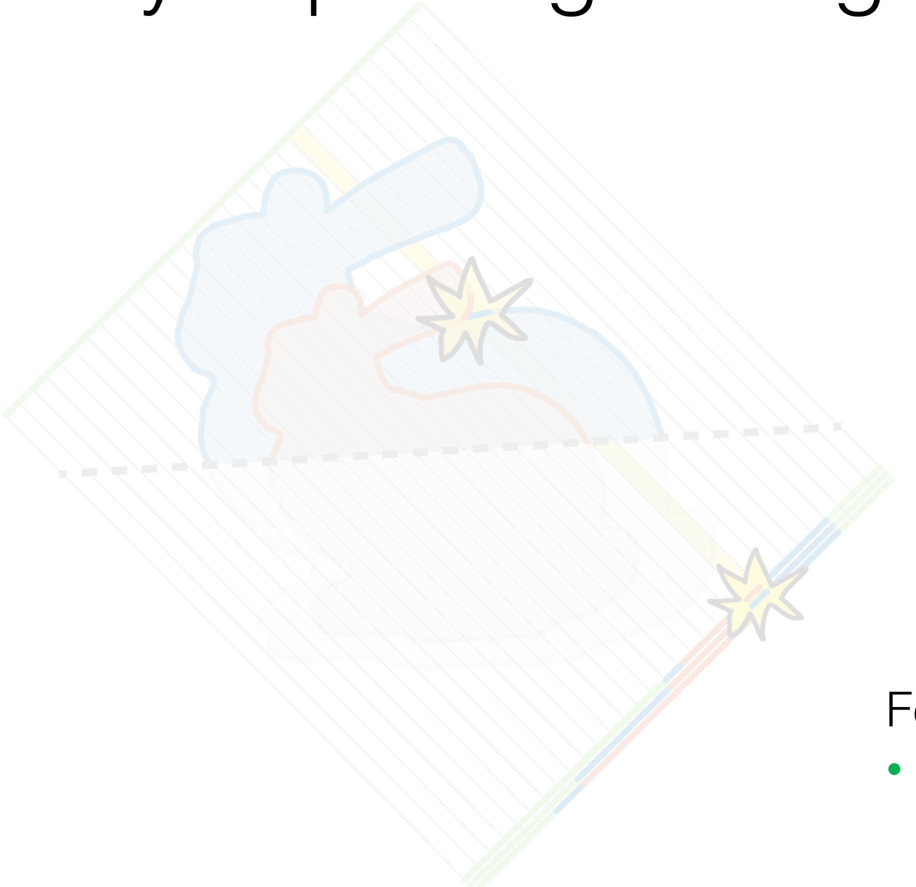
Step 1: we determine feasibility in real-time by exploiting orthographic rendering

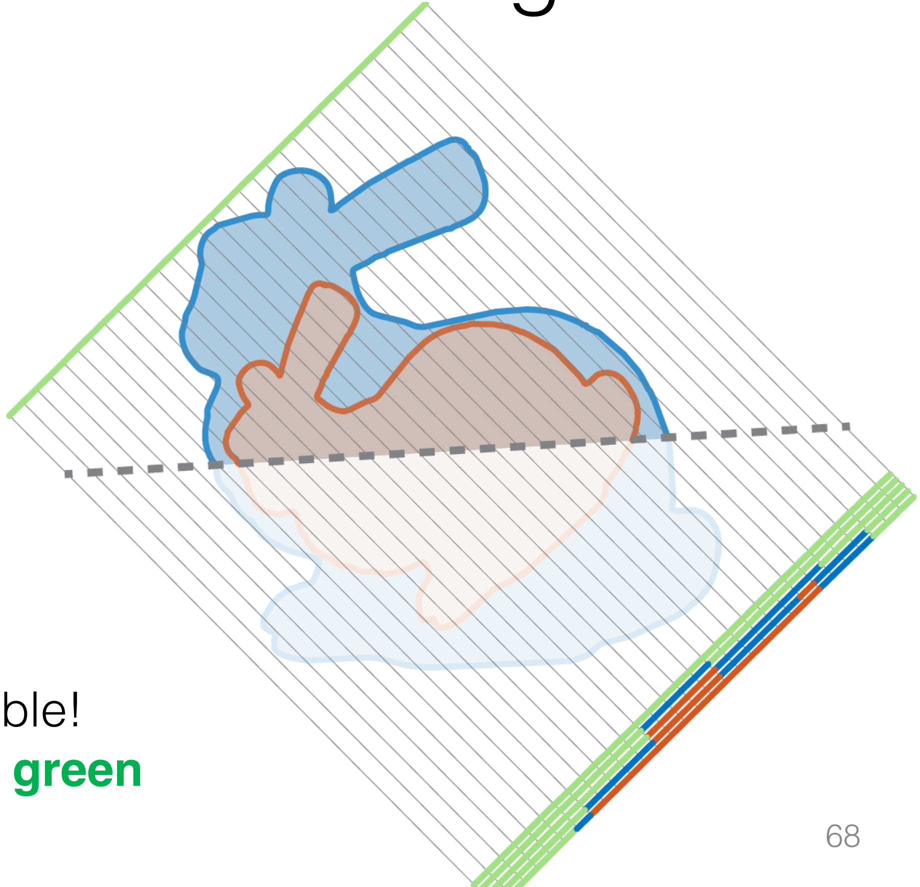


Step 1: we determine feasibility in real-time by exploiting orthographic rendering



Step 1: we determine feasibility in real-time by exploiting orthographic rendering

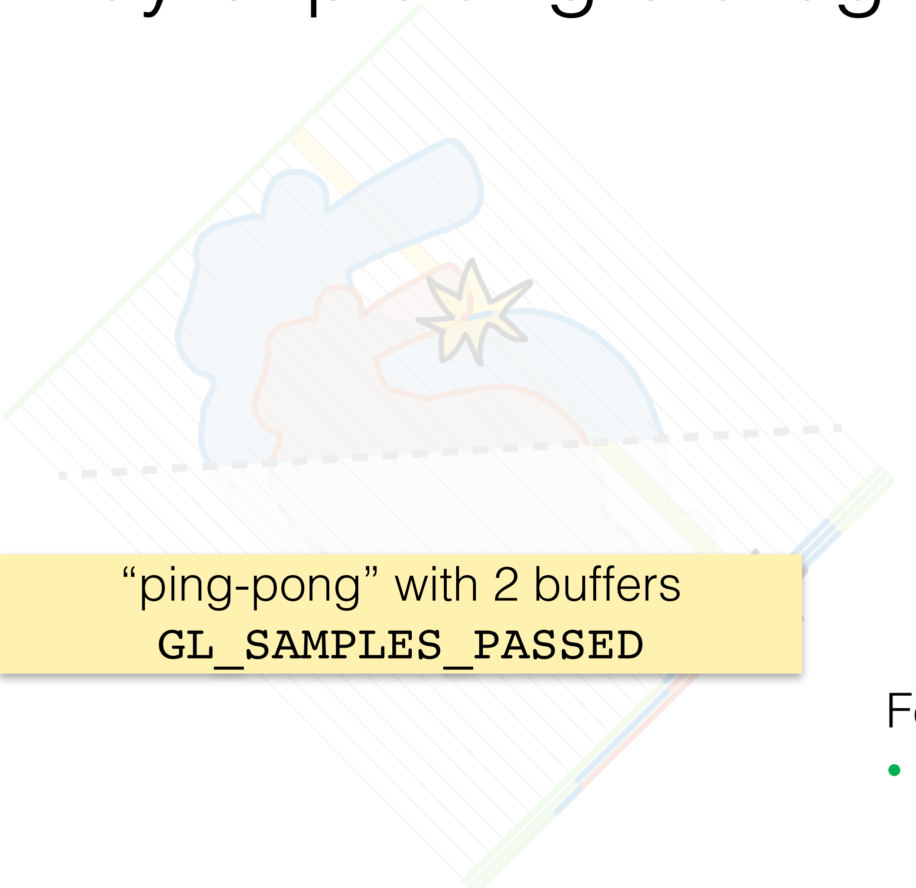




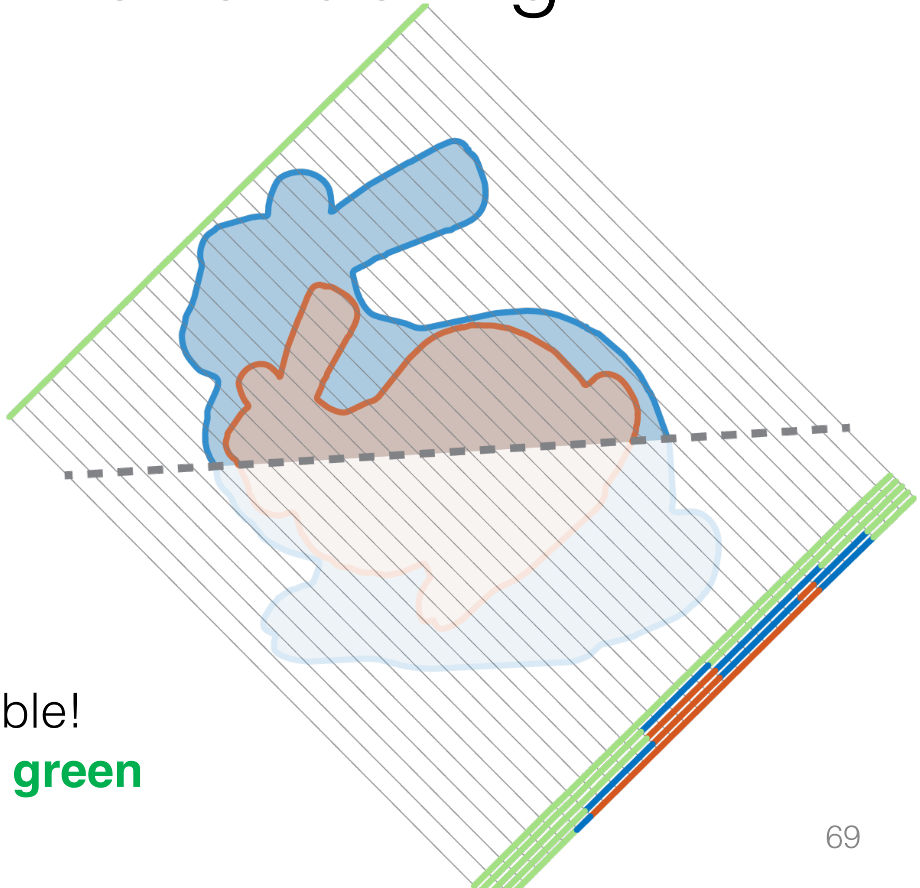
Feasible!

- **all green**

Step 1: we determine feasibility in real-time by exploiting orthographic rendering



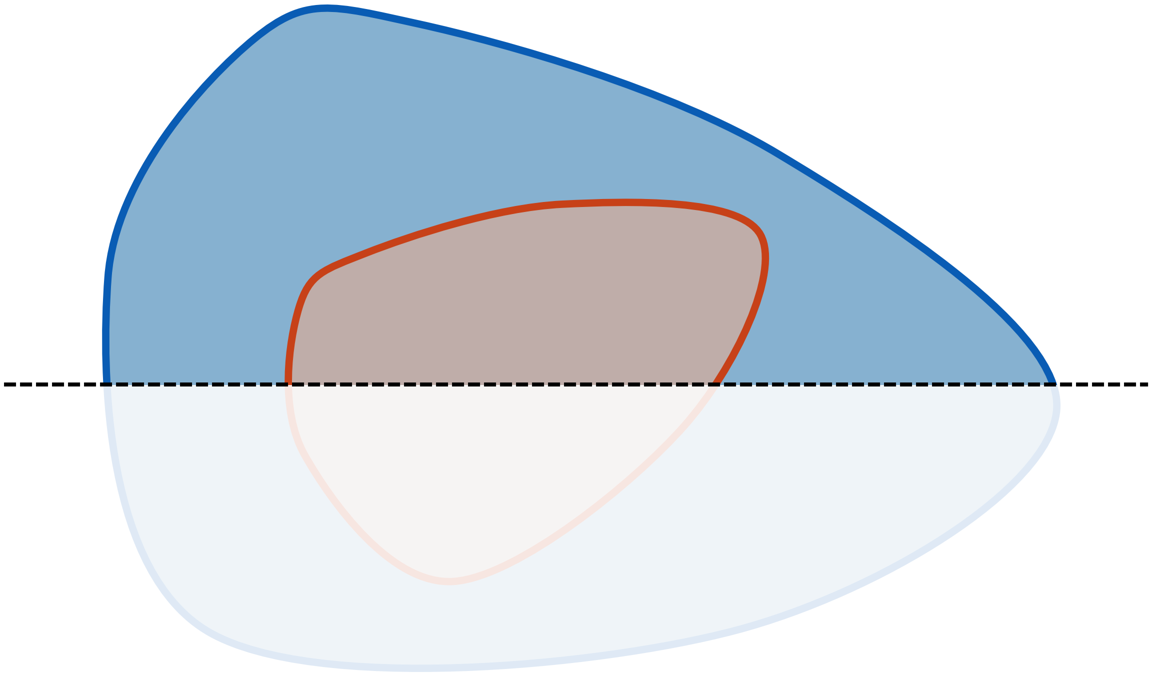
“ping-pong” with 2 buffers
`GL_SAMPLES_PASSED`



Feasible!

- **all green**

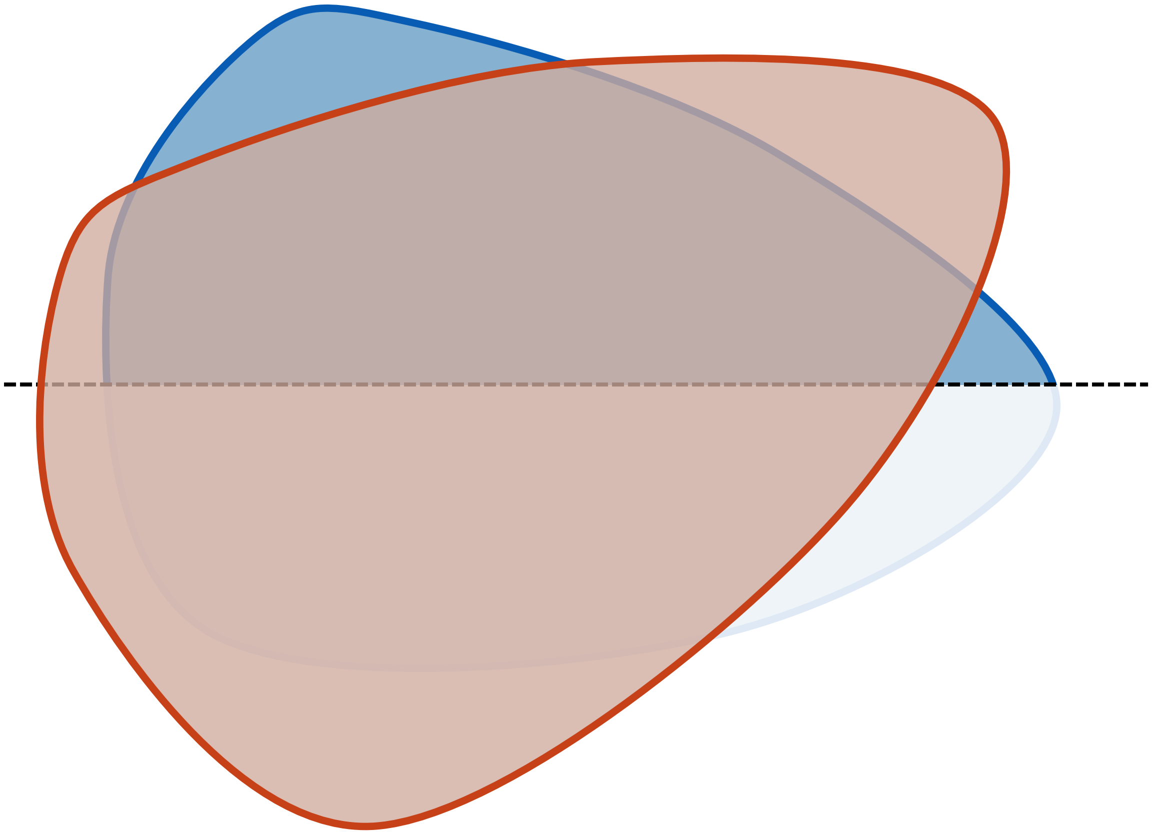
Step 2: binary search to maximize scale



Assume *momentarily* that shape is convex

Fix cut plane, center of mass, rotation

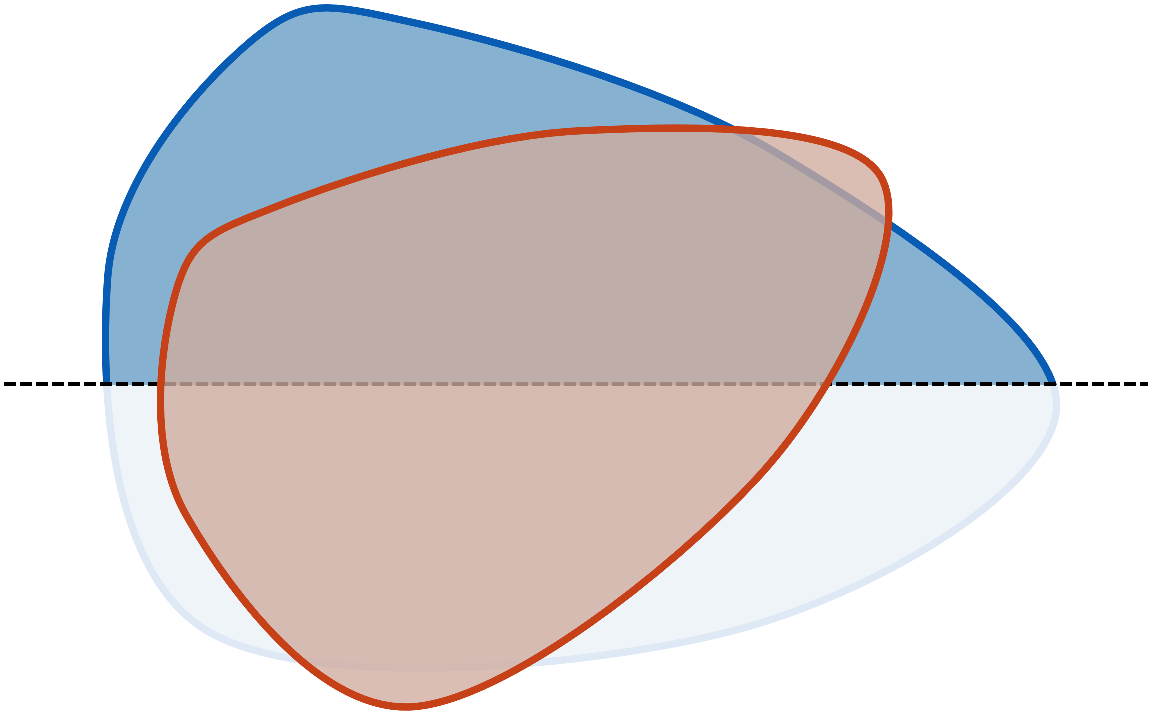
Step 2: binary search to maximize scale



Assume *momentarily* that shape is convex

Fix cut plane, center of mass, rotation

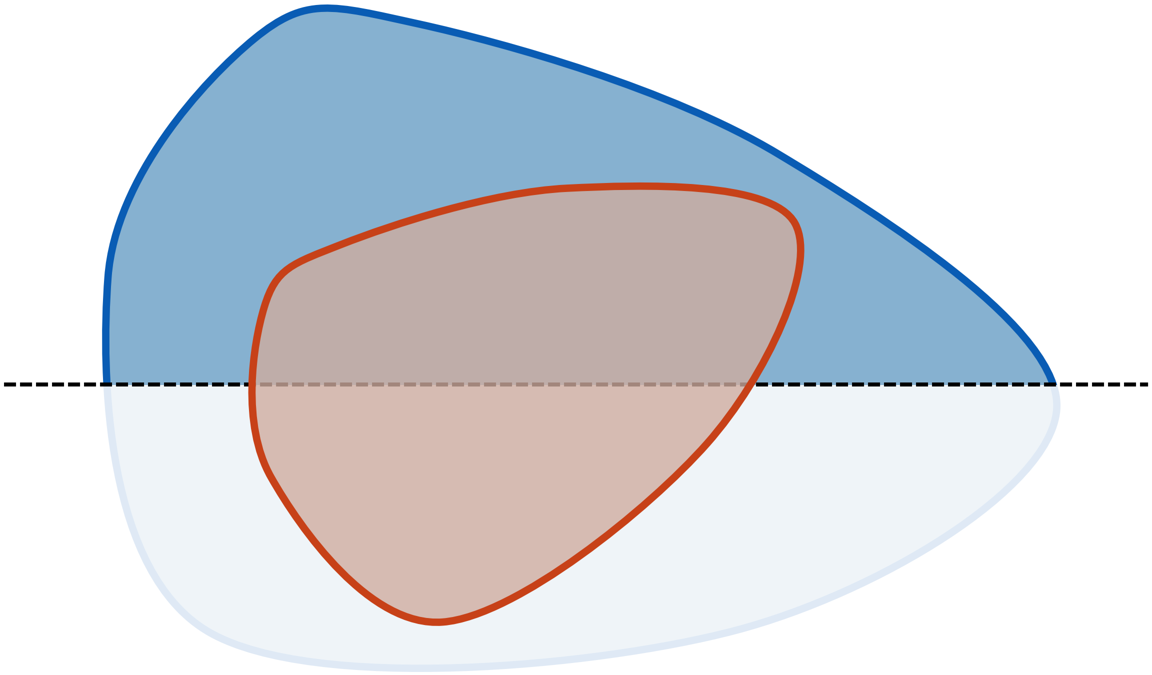
Step 2: binary search to maximize scale



Assume *momentarily* that shape is convex

Fix cut plane, center of mass, rotation

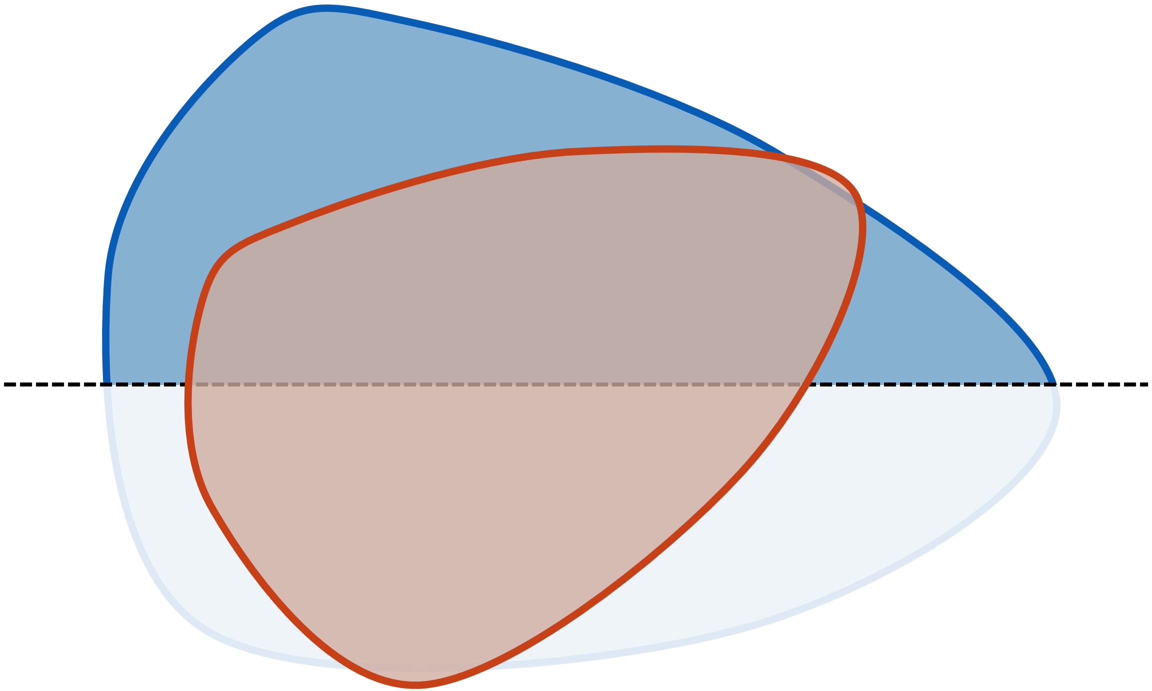
Step 2: binary search to maximize scale



Assume *momentarily* that shape is convex

Fix cut plane, center of mass, rotation

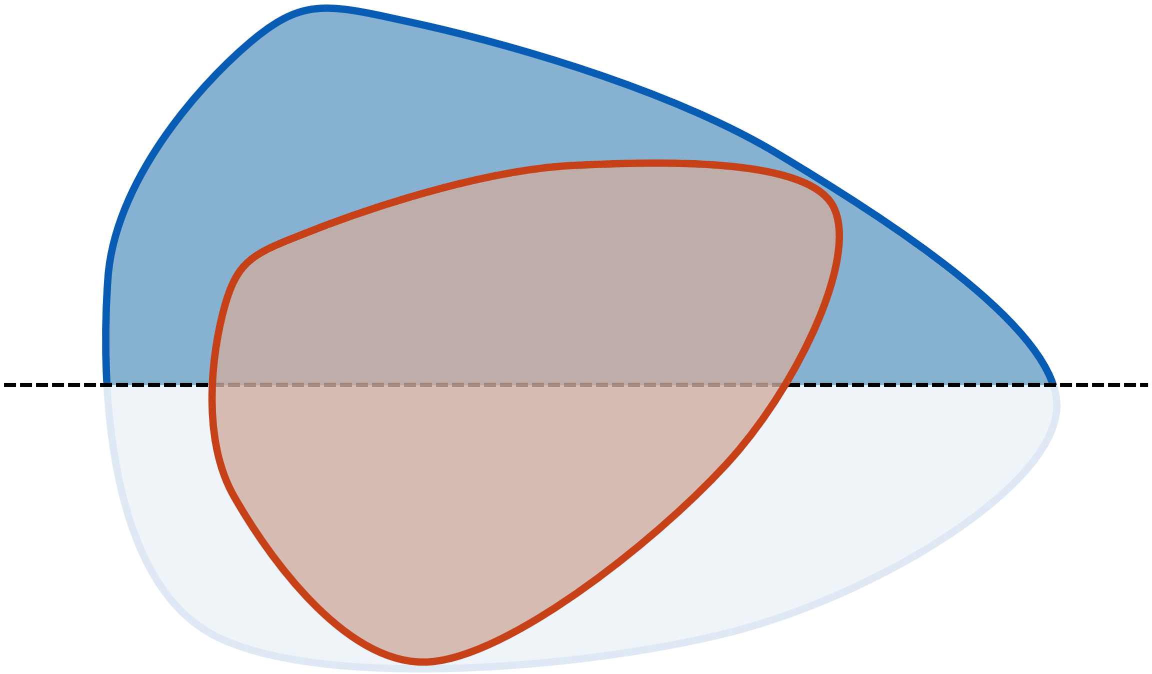
Step 2: binary search to maximize scale



Assume *momentarily* that shape is convex

Fix cut plane, center of mass, rotation

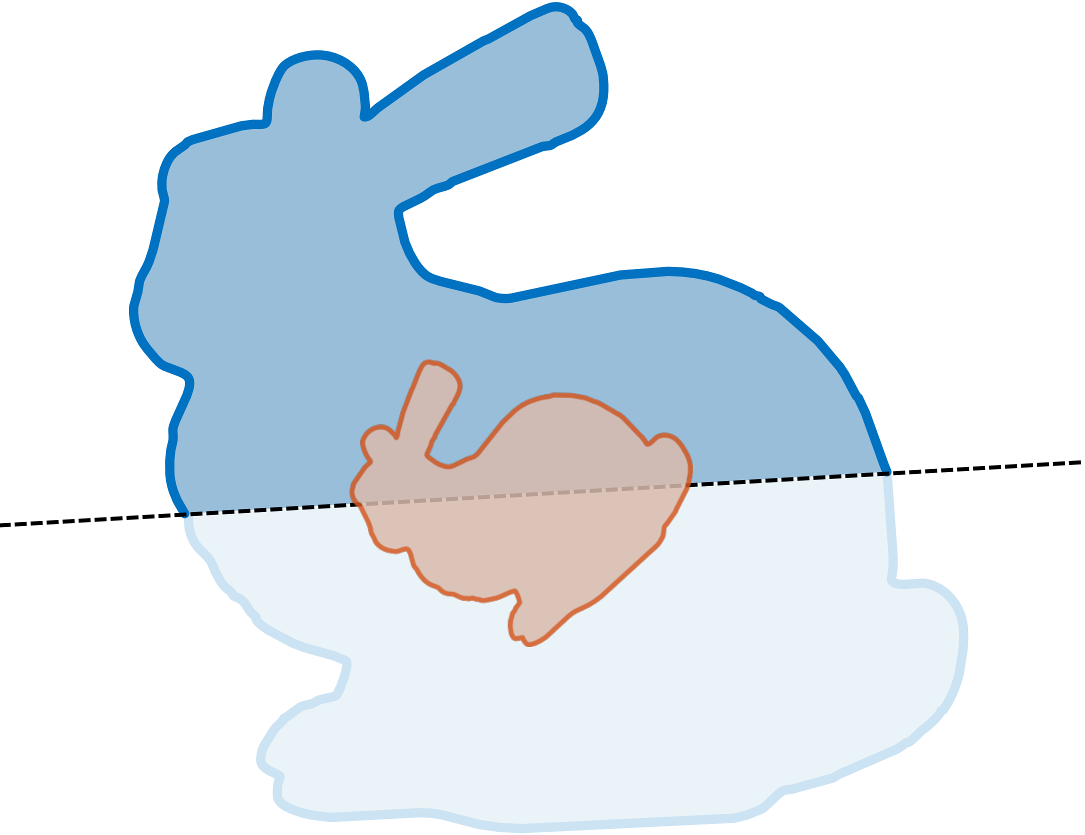
Step 2: binary search to maximize scale



Assume *momentarily* that shape is convex

Fix cut plane, center of mass, rotation

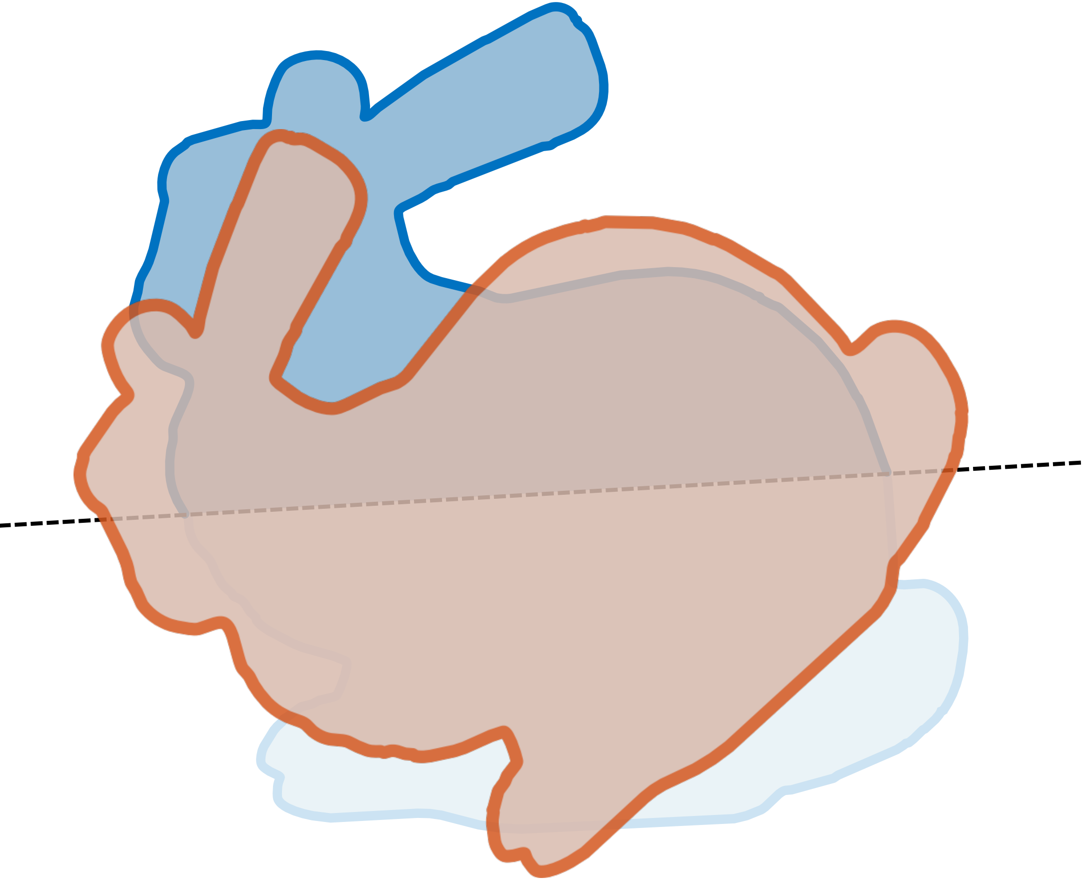
Step 2: binary search to maximize scale



For non-convex shapes
binary search is conservative,

but in practice optimal

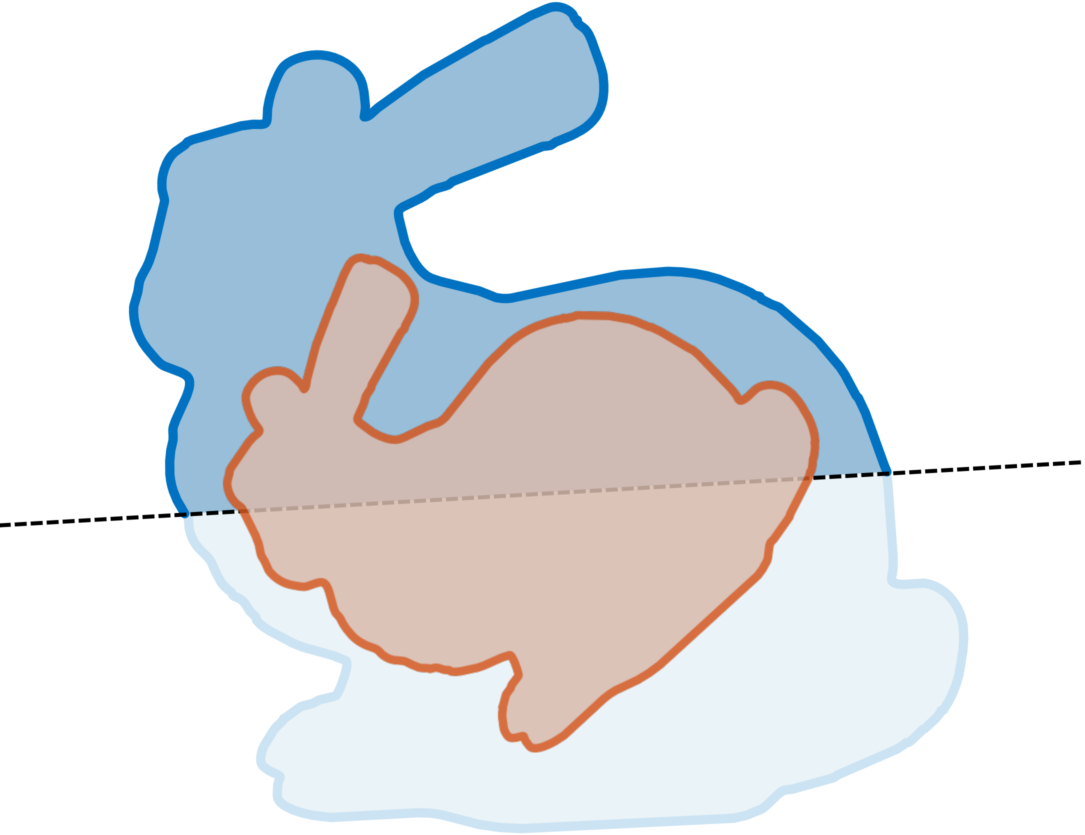
Step 2: binary search to maximize scale



For non-convex shapes
binary search is conservative,

but in practice optimal

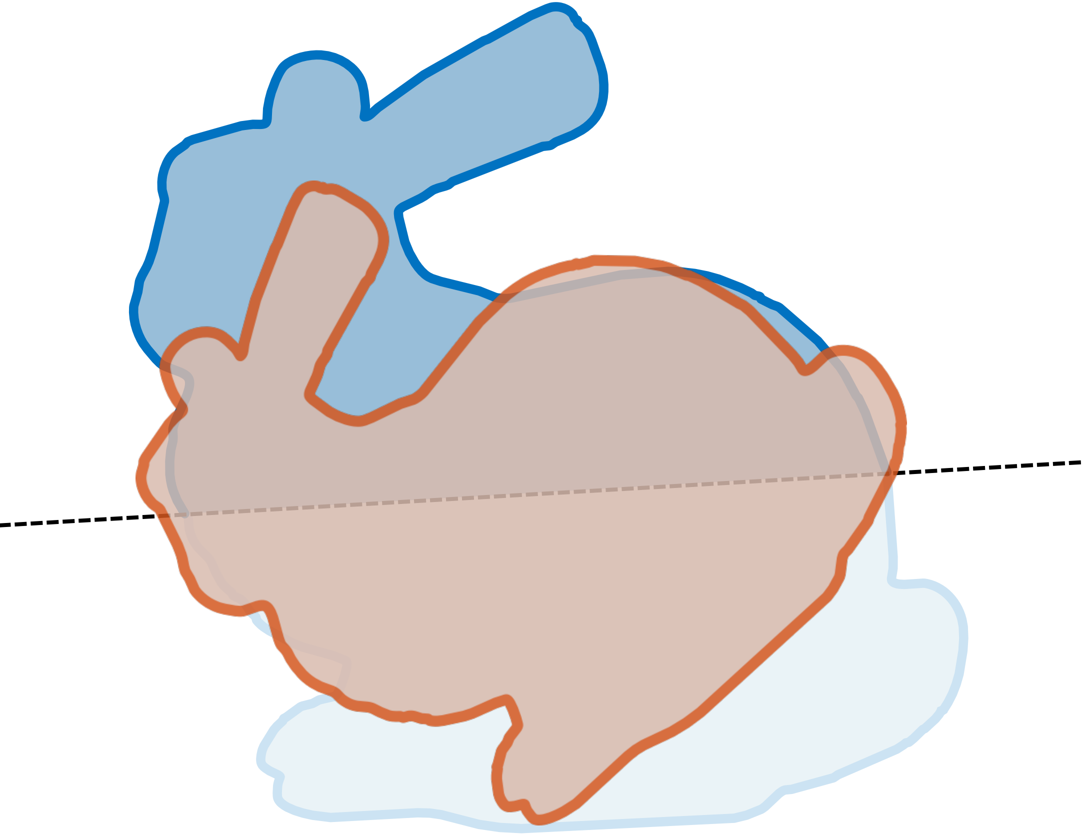
Step 2: binary search to maximize scale



For non-convex shapes
binary search is conservative,

but in practice optimal

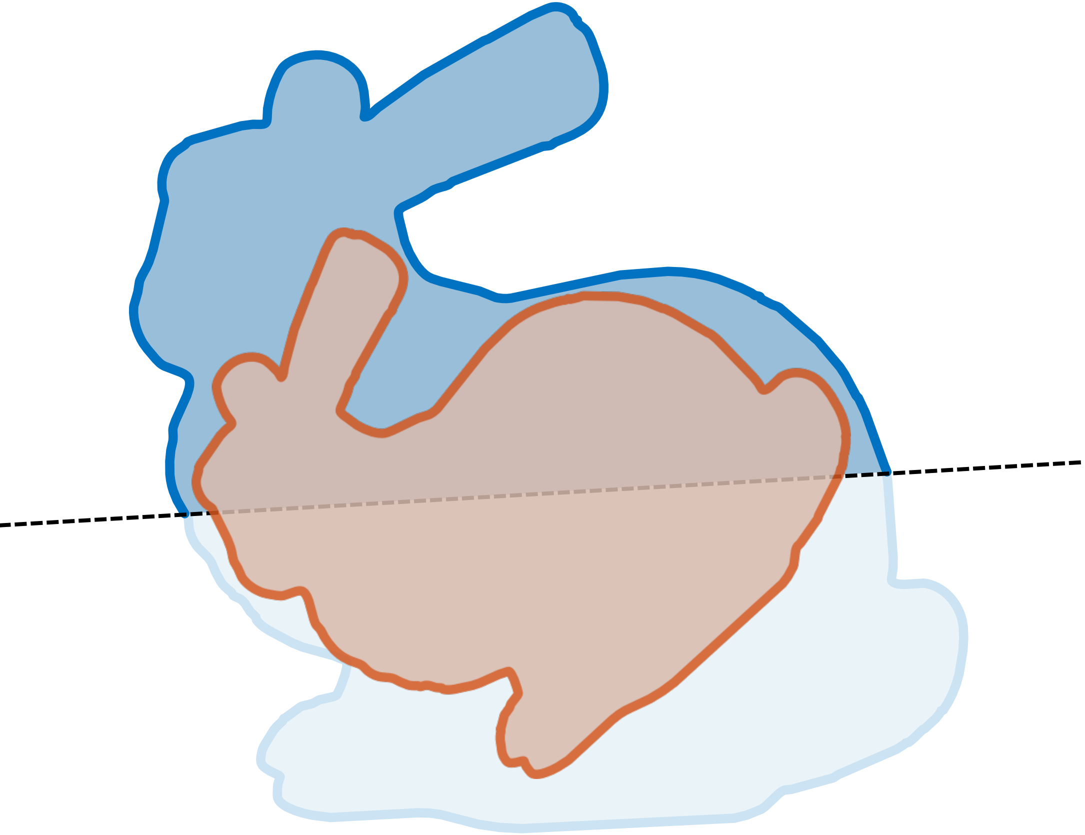
Step 2: binary search to maximize scale



For non-convex shapes
binary search is conservative,

but in practice optimal

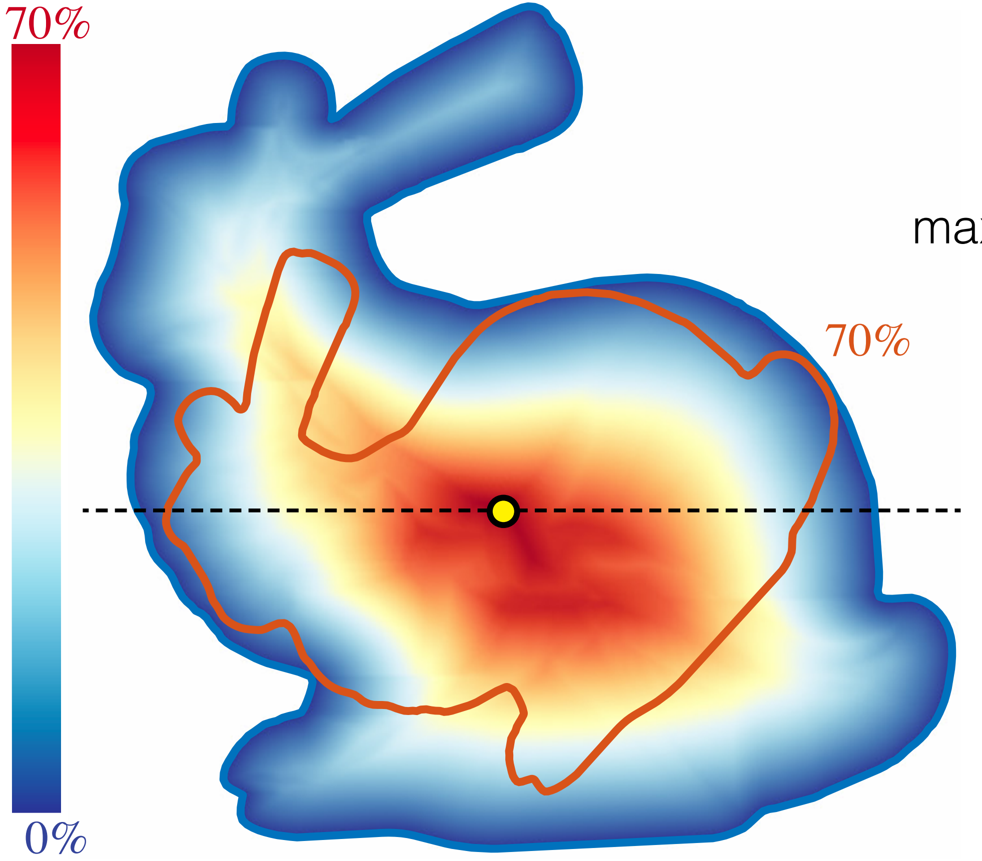
Step 2: binary search to maximize scale



For non-convex shapes
binary search is conservative,

but in practice optimal

Step 3: optimize over all parameters



maximize *scale* subject to *nesting constraint*

non-convex energy landscape

Step 3: optimize over all parameters *via particle swarm optimization*

k parameter vector as point in *n*D $\mathbf{x}_i \in \mathbb{R}^n$

Step 3: optimize over all parameters *via particle swarm optimization*

k parameter vector as point in *n*D $\mathbf{x}_i \in \mathbb{R}^n$

update each iteration according to "velocity"

$$\mathbf{x}_i \leftarrow \mathbf{x}_i + \mathbf{v}_i,$$

Step 3: optimize over all parameters *via particle swarm optimization*

k parameter vector as point in *n*D $\mathbf{x}_i \in \mathbb{R}^n$

pull velocity toward ***personal best*** *and* ***global best*** *of swarm*

$$\mathbf{v}_i \leftarrow \omega \mathbf{v}_i + \phi_p r_p \left(\boxed{\mathbf{x}_i^p} - \mathbf{x}_i \right) + \phi_g r_g \left(\boxed{\mathbf{x}^g} - \mathbf{x}_i \right),$$

$$\mathbf{x}_i \leftarrow \mathbf{x}_i + \mathbf{v}_i,$$

Step 3: optimize over all parameters *via particle swarm optimization*

k parameter vector as point in *n*D $\mathbf{x}_i \in \mathbb{R}^n$

$$\mathbf{v}_i \leftarrow \omega \mathbf{v}_i + \phi_p r_p (\mathbf{x}_i^p - \mathbf{x}_i) + \phi_g r_g (\mathbf{x}^g - \mathbf{x}_i),$$

random perturbations

$$\mathbf{x}_i \leftarrow \mathbf{x}_i + \mathbf{v}_i,$$

Naive P-Swarm would treat scale as just another parameter (coordinate)…

$$\begin{aligned} \underset{s,\mathbf{R},\mathbf{c},\mathbf{P},\mathbf{a}^+,\mathbf{a}^-}{\text{maximize}} \quad & s \\ \text{such that} \quad & \mathbf{T}(\mathcal{B}) \text{ nests in } \mathcal{A} \text{ w.r.t. } \mathbf{P}, \mathbf{a}^+, \mathbf{a}^- \end{aligned}$$

… instead optimize over all others,

$$\underset{\mathbf{R},\mathbf{c},\mathbf{P},\mathbf{a}^+,\mathbf{a}^-}{\text{maximize}} \quad f(\mathbf{R}, \mathbf{c}, \mathbf{P}, \mathbf{a}^+, \mathbf{a}^-)$$

all other parameters

… instead optimize over all others,

$$\underset{\mathbf{R},\mathbf{c},\mathbf{P},\mathbf{a}^+,\mathbf{a}^-}{\text{maximize}} \quad f(\mathbf{R},\mathbf{c},\mathbf{P},\mathbf{a}^+,\mathbf{a}^-)$$

where

$$\begin{aligned} f(\mathbf{R},\mathbf{c},\mathbf{P},\mathbf{a}^+,\mathbf{a}^-) = & \boxed{\underset{s}{\text{maximize}} \quad s} \\ \text{such that} \quad & \mathbf{T}(\mathcal{B}) \text{ nests in } \mathcal{A} \text{ w.r.t. } \mathbf{P},\mathbf{a}^+,\mathbf{a}^- \end{aligned}$$

… instead optimize over all others, and *search* for max scale

$$\underset{\mathbf{R},\mathbf{c},\mathbf{P},\mathbf{a}^+,\mathbf{a}^-}{\text{maximize}} \quad f(\mathbf{R},\mathbf{c},\mathbf{P},\mathbf{a}^+,\mathbf{a}^-)$$

where

$$f \approx \underset{s}{\text{search}}(\mathbf{R},\mathbf{c},\mathbf{P},\mathbf{a}^+,\mathbf{a}^-)$$

… instead optimize over all others, and *search* for max scale

$$\underset{\mathbf{R},\mathbf{c},\mathbf{P},\mathbf{a}^+,\mathbf{a}^-}{\text{maximize}} \quad f(\mathbf{R}, \mathbf{c}, \mathbf{P}, \mathbf{a}^+, \mathbf{a}^-)$$

where

$$f \approx \underset{s}{\text{search}}(\mathbf{R}, \mathbf{c}, \mathbf{P}, \mathbf{a}^+, \mathbf{a}^-)$$

abort search early if
upper bound < *best*

Our optimization enables fully automatic Matryoshka generation…

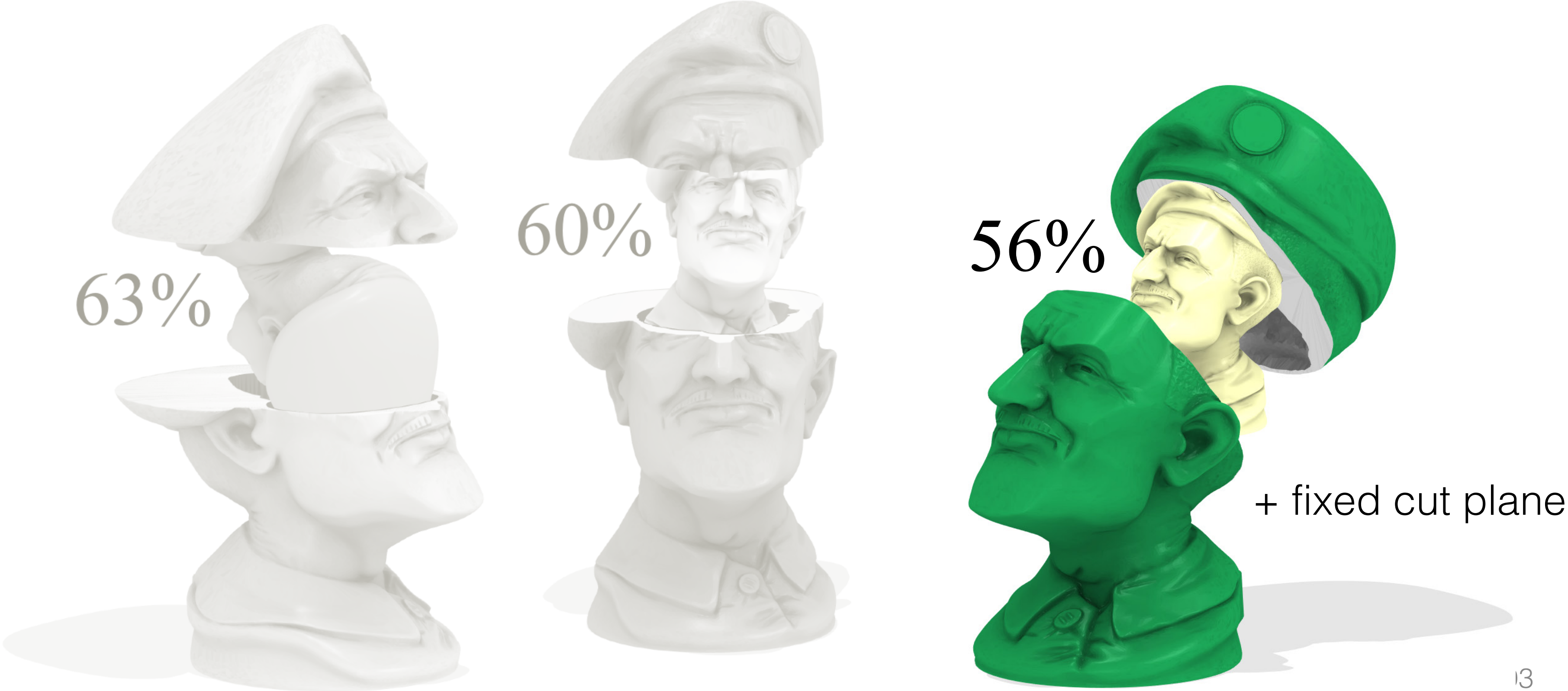
63%

fully optimized

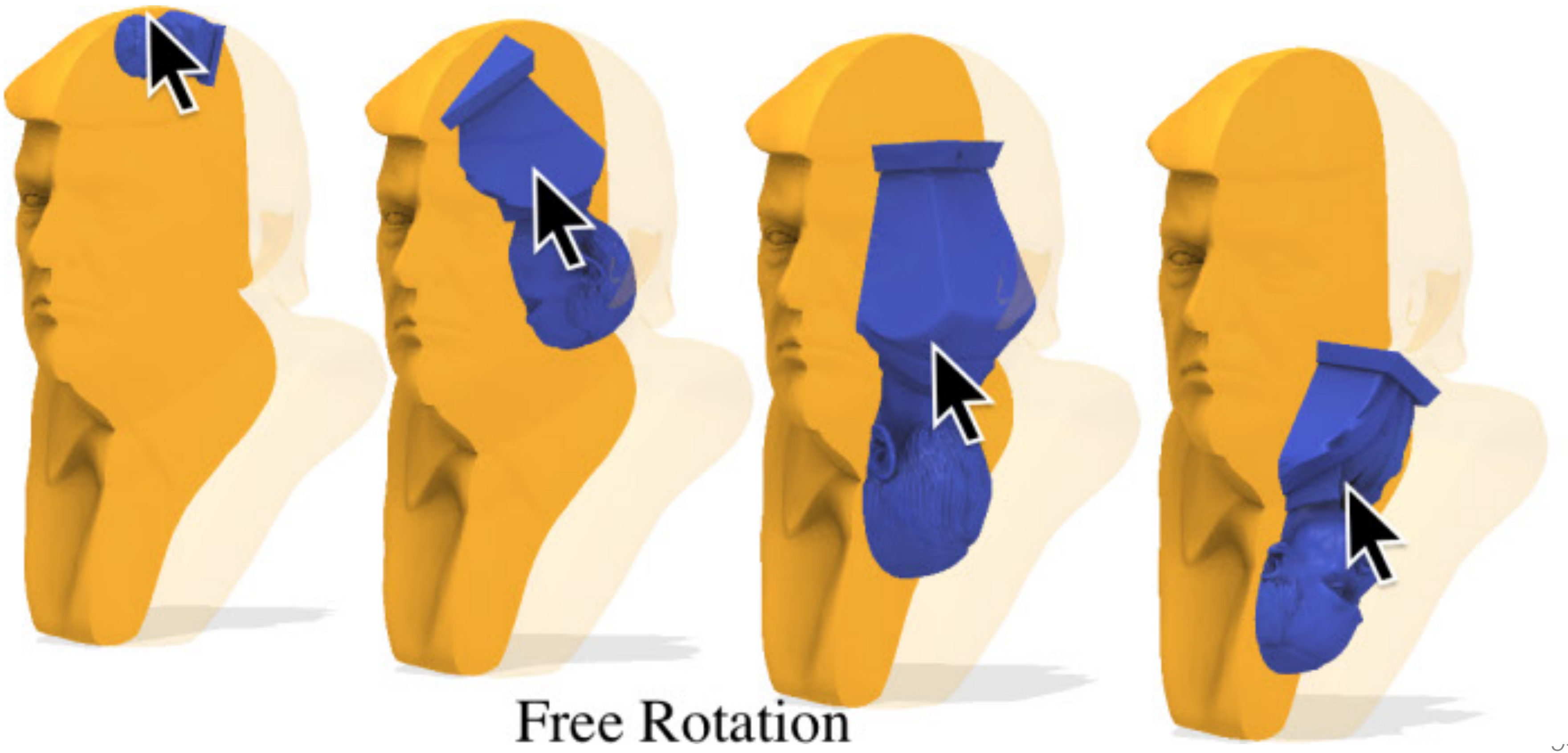
… or partially constrained interactive design

fixed upright orientation

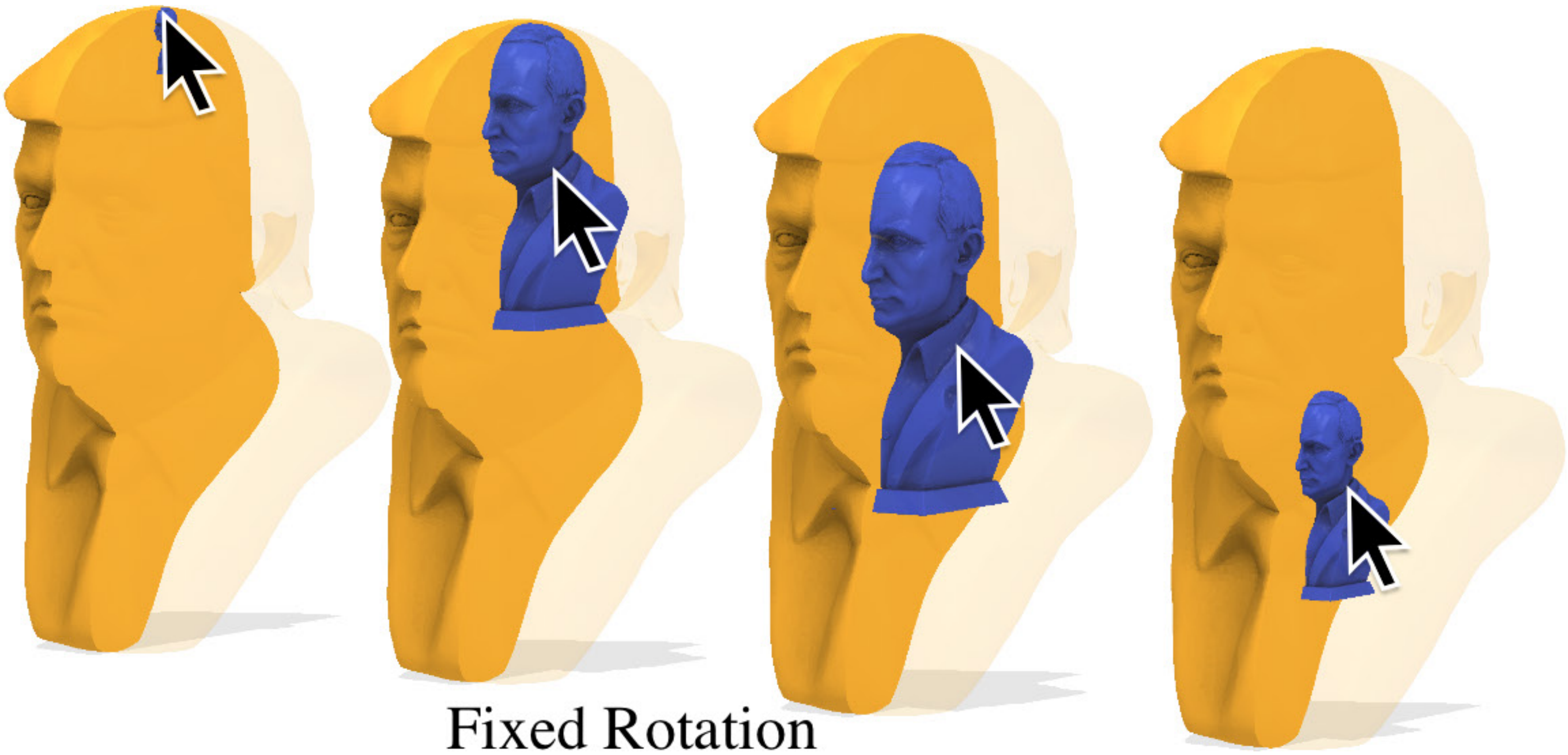
… or partially constrained interactive design



Tool performs fast enough for interaction



Tool performs fast enough for interaction



We validate our results via 3D printing

We validate our results via 3D printing

We validate our results via 3D printing

We validate our results via 3D printing

We validate our results via 3D printing

We validate our results via 3D printing

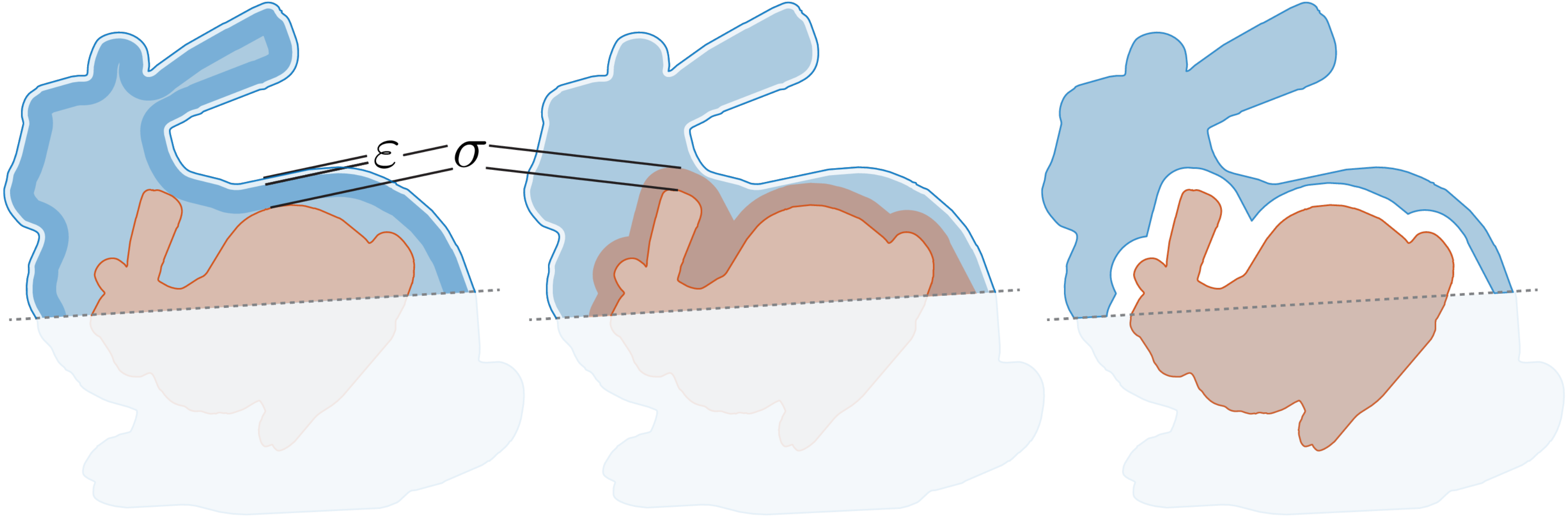
We validate our results via 3D printing

We validate our results via 3D printing

We validate our results via 3D printing

We validate our results via 3D printing

We accommodate printer tolerances by nesting *within* an offset surface



During validation During carving Result

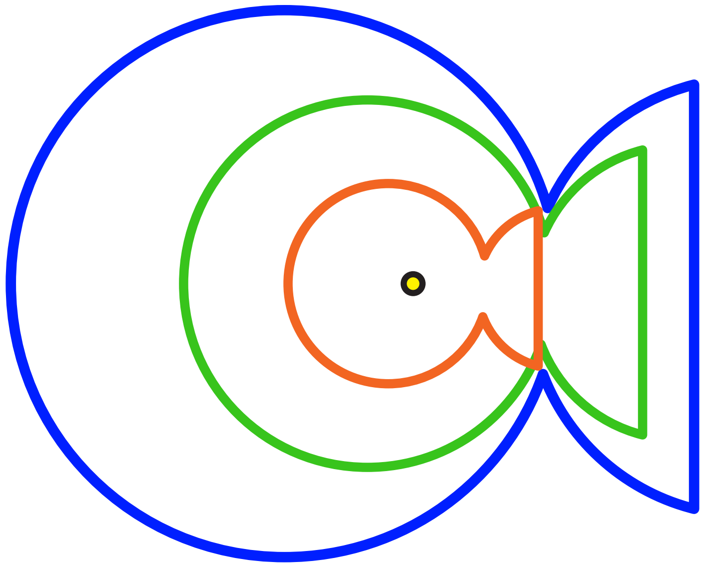
Our tools trivially generalize to nesting disparate shapes

Limitations & Future Work

- no global optimum guarantee

Limitations & Future Work

- no global optimum guarantee
- search assumption too conservative



Limitations & Future Work

- no global optimum guarantee
- search assumption too conservative
- thin shapes don't *rigidly* nest well

Limitations & Future Work

- no global optimum guarantee
- search assumption too conservative
- thin shapes don’t *rigidly* nest well
- deformable nesting?

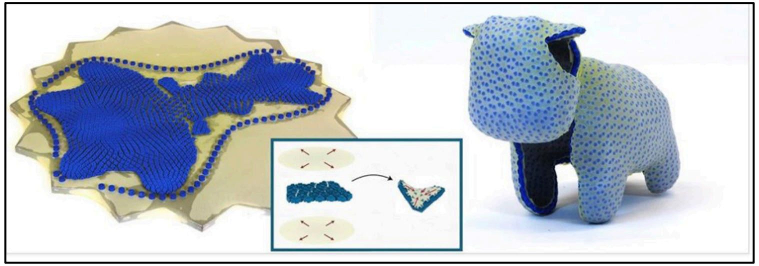
Limitations & Future Work

- no global optimum guarantee
- search assumption too conservative
- thin shapes don't *rigidly* nest well
- deformable nesting?
  1. deform during design

Prevost et al.

Limitations & Future Work

- no global optimum guarantee
- search assumption too conservative
- thin shapes don't *rigidly* nest well
- deformable nesting?
    1. deform during design
    2. nest soft physical objects



Bickel et al.

Acknowledgements…

David Levin

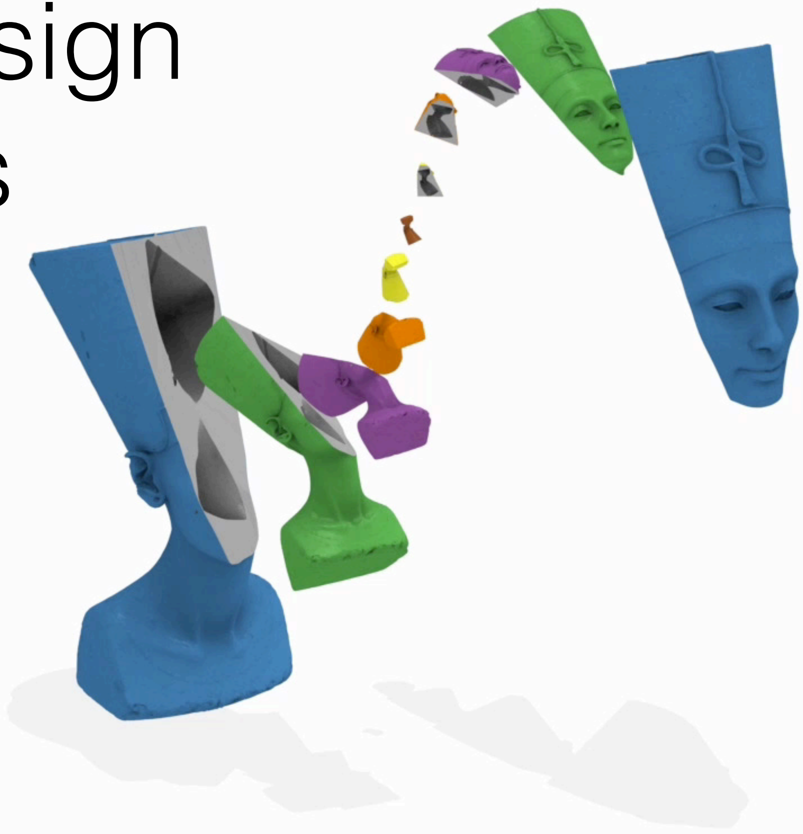
NSERC Discovery Grants (RGPIN-2017-05235 & RGPAS-2017-507938)

Connaught Fund (NR-2016-17)

Adobe Systems Inc.

Kevin Gibson, Masha Shugrina, Michael Tao, and Alex Tessier

Generalized Matryoshka
Computational Design of Nesting Objects



Alec Jacobson

jacobson@cs.toronto.edu