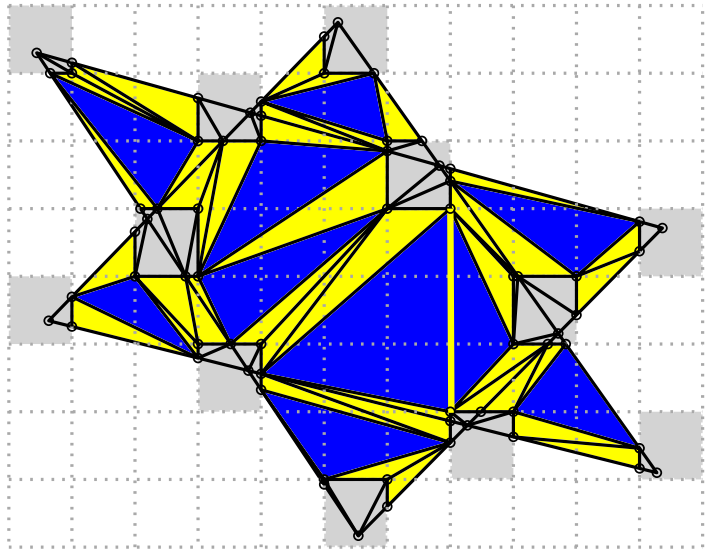
Days of department 1

Rounding polygonal meshes

Léo Valque



Float Number

$$Float = \boxed{mantis} * 2^{\boxed{exponent}}$$

Integer of d digits

Integer of k digits

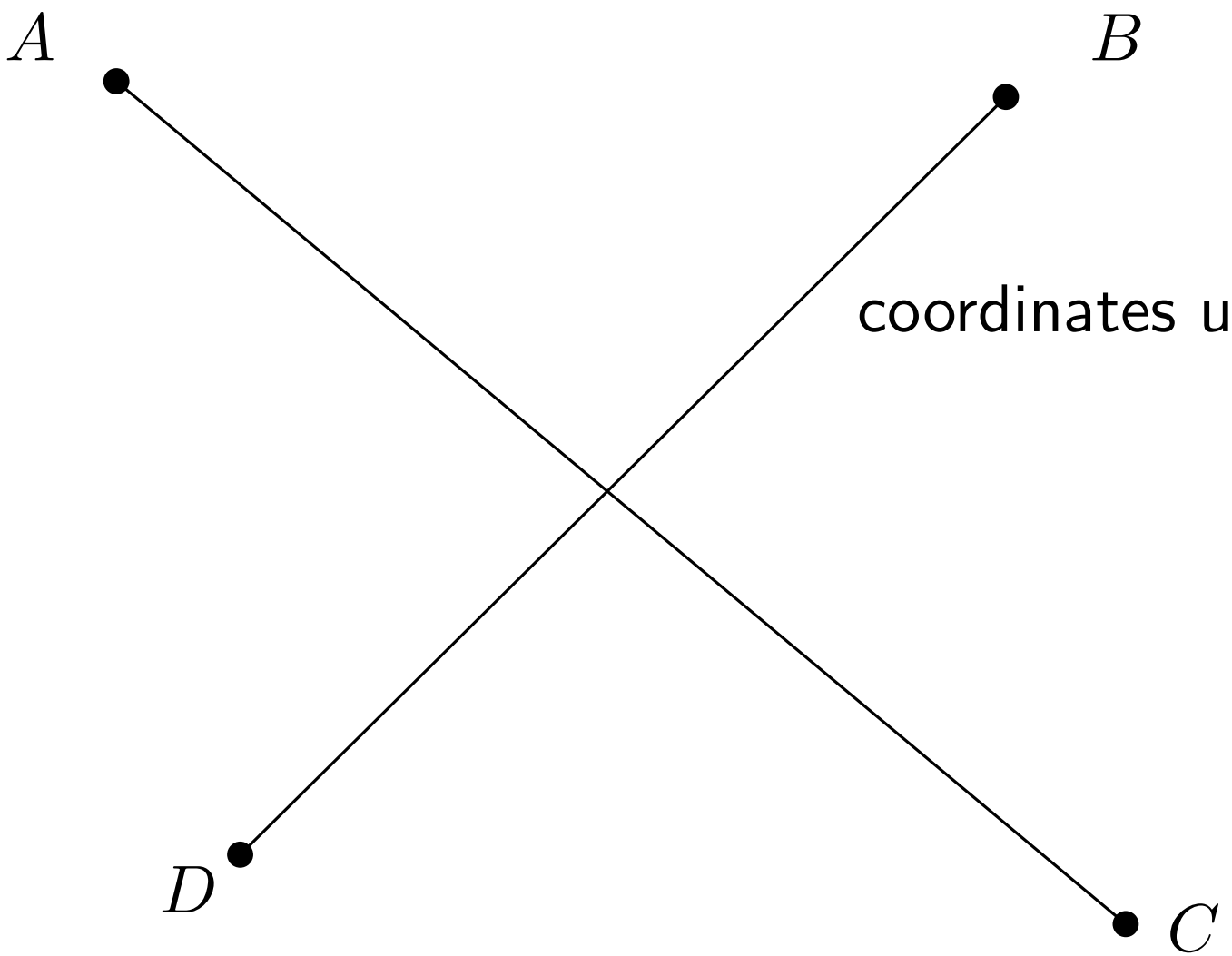
Float Number

d_1 digits d_2 digits

$$f_1 * f_2 = \boxed{\boxed{m_1} * \boxed{m_2}} * 2^{e_1+e_2}$$

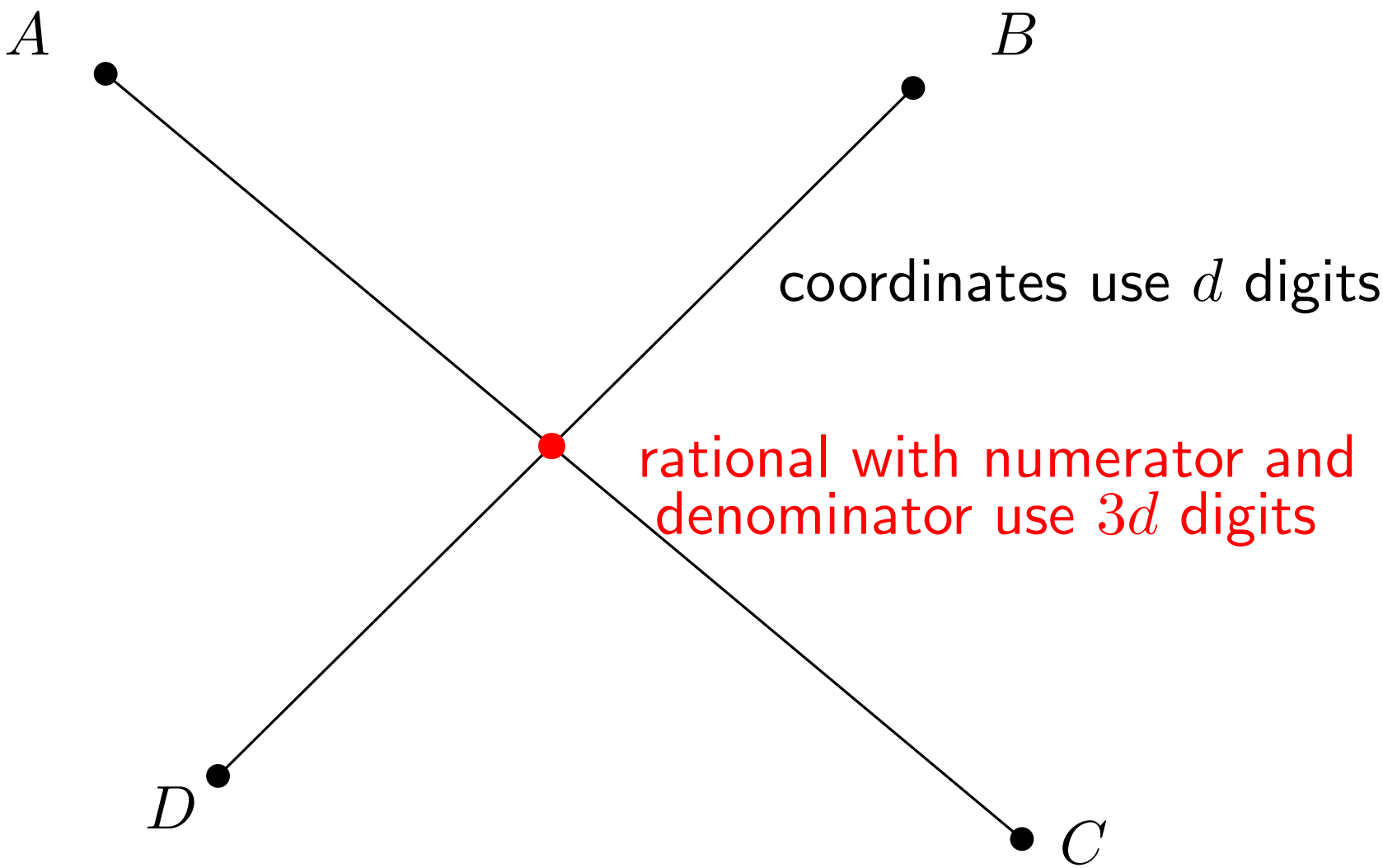
$d_1 + d_2$ digits

Intersection of Segments

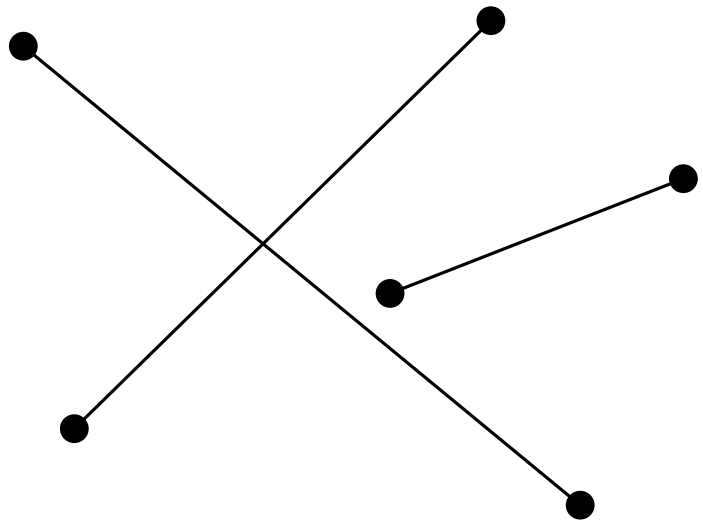


coordinates use d digits

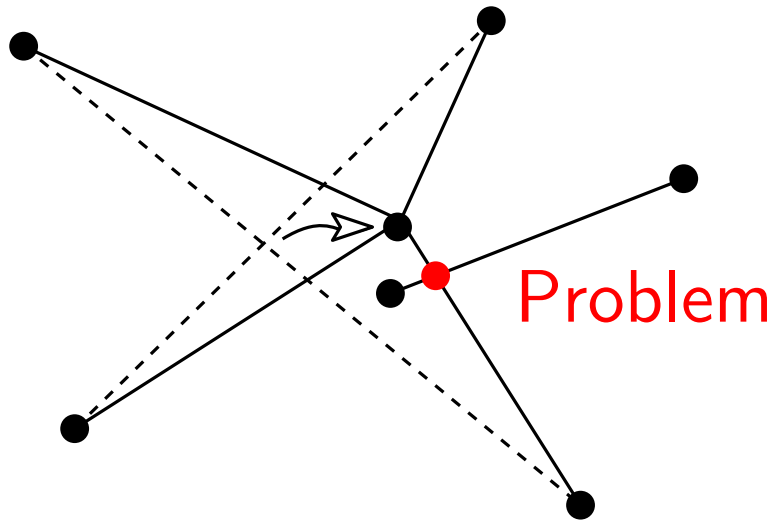
Intersection of Segments



Rounding Issues in 2D

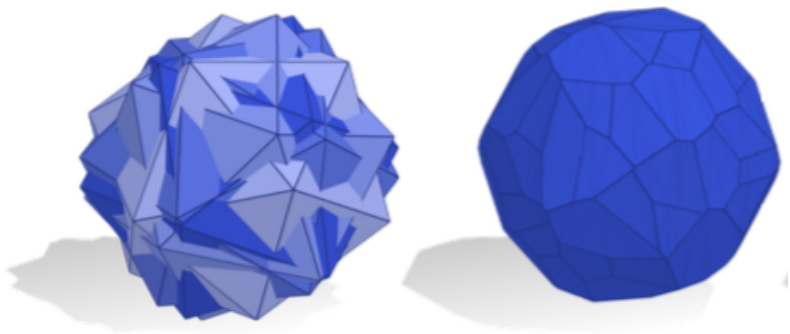


Rounding Issues in 2D



Rounding Issues in 3D

Boolean operations on solids



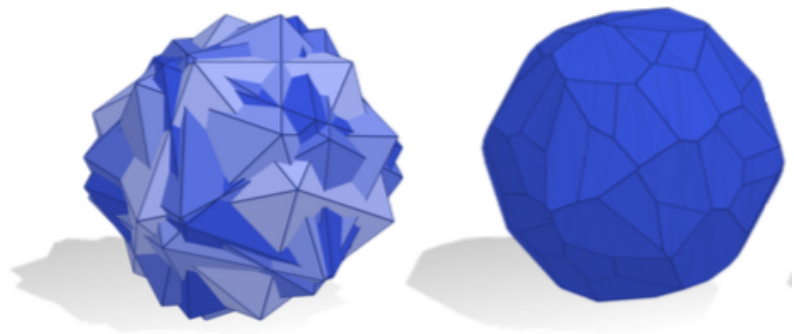
Intersection of four icosahedra

[Zhou et al. 2016]

Need to round the output coordinates

Rounding Issues in 3D

Boolean operations on solids



Intersection of four icosahedra

[Zhou et al. 2016]

Need to round the output coordinates

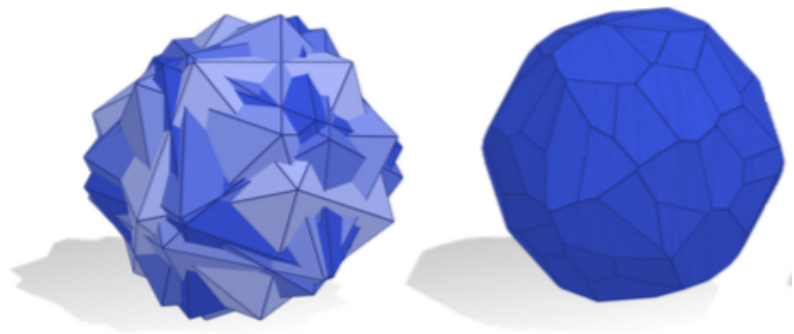
Thingiverse: 10K meshes 45% with self intersections

Mesh repair (elimination of self-intersections)

Zhou et al. [SIGGRAPH 2016] exact arithmetic

Rounding Issues in 3D

Boolean operations on solids



Intersection of four icosahedra

[Zhou et al. 2016]

Need to round the output coordinates

Thingiverse: 10K meshes 45% with self intersections

Mesh repair (elimination of self-intersections)

Zhou et al. [SIGGRAPH 2016] exact arithmetic

Naive rounding fails (self-intersections) in 2.2%

Snap Rounding Problem

Given a set of interior disjoint faces in 3D (or segments in 2D)

Round the vertices coordinates on a grid – Allow subdivisions

Preserve the geometry

Bound the distance between a face and its rounded image

Preserve the topology "up to collapses":

Snap Rounding Problem

Given a set of interior disjoint faces in 3D (or segments in 2D)

Round the vertices coordinates on a grid – Allow subdivisions

Preserve the geometry

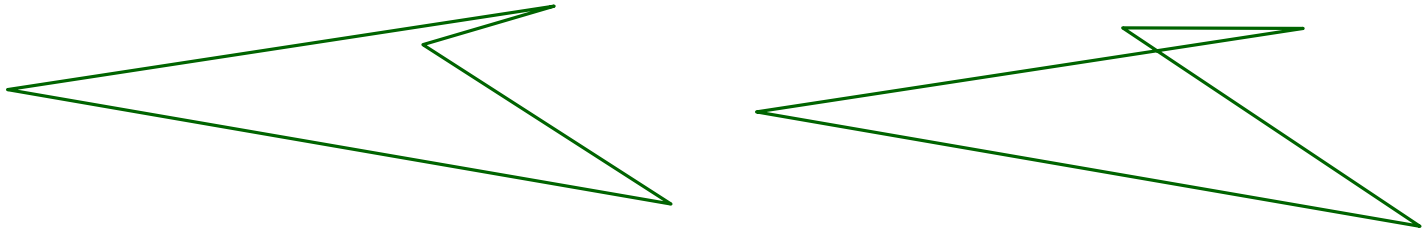
Bound the distance between a face and its rounded image

Preserve the topology "up to collapses":

Subdivisions are needed

NP-hard to round simple polygons with fixed precision, bounded Hausdorff distance, and preserving the topology

[Milenkovic, Nackman, 1990]



Snap Rounding Problem

Given a set of interior disjoint faces in 3D (or segments in 2D)

Round the vertices coordinates on a grid – Allow subdivisions

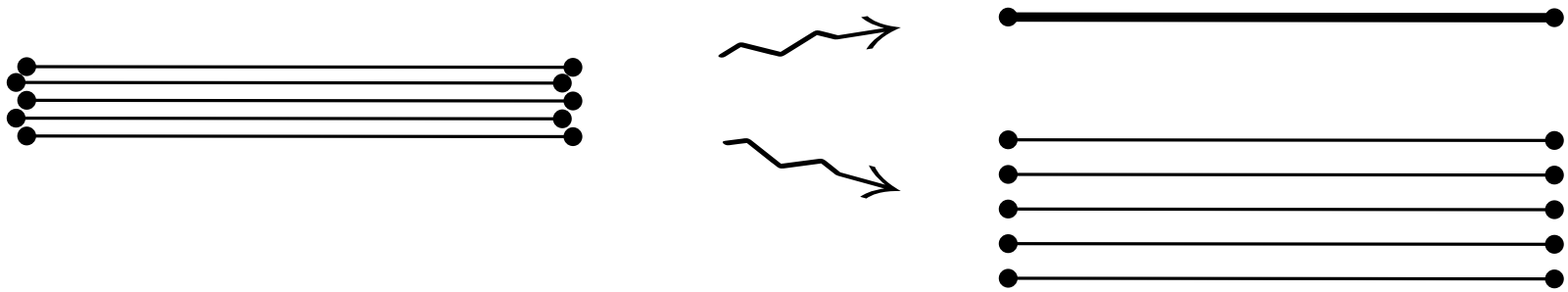
Preserve the geometry

Bound the distance between a face and its rounded image

Preserve the topology "up to collapses":

Subdivisions are needed

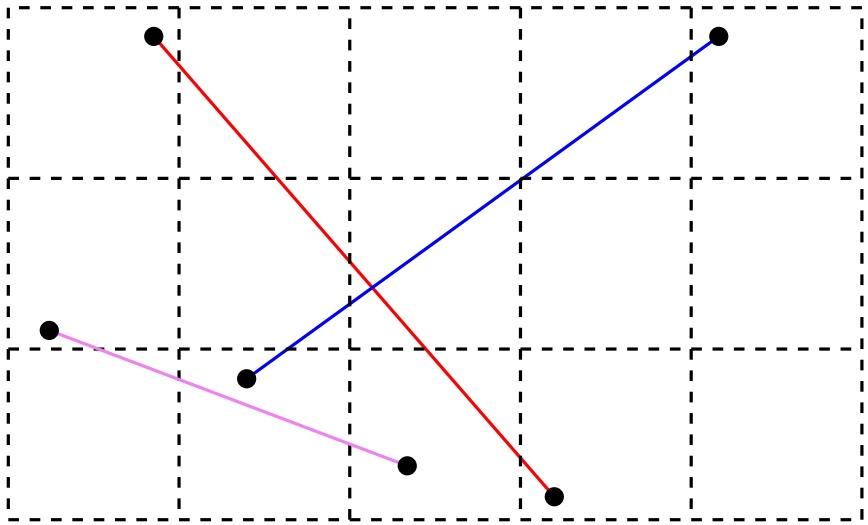
Collapses or unbounded distance may be needed



2D Snap Rounding

Well understood in 2D

1986 Greene, Yao
1997 Goodrich, Guibas, Hershberger, Tanenbaum
1998 Guibas and Marimont
1999 Hobby
2002 Halperin, Packer
2007 de Berg, Halperin, Overmars
2008 Hershberger
2013 Hershberger

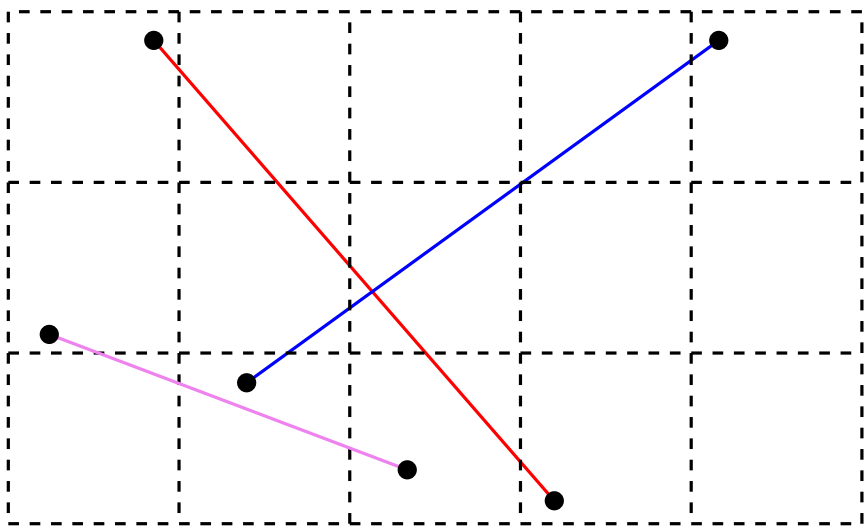


2D Snap Rounding

Well understood in 2D

We round coordinates on the integer grid [Guibas & Marimont 98]

- Pixels that contain a vertex of the arrangement are tagged *hot*
- Segments that intersect a hot pixel are subdivided in that pixel
- All vertices are rounded to their pixels centers

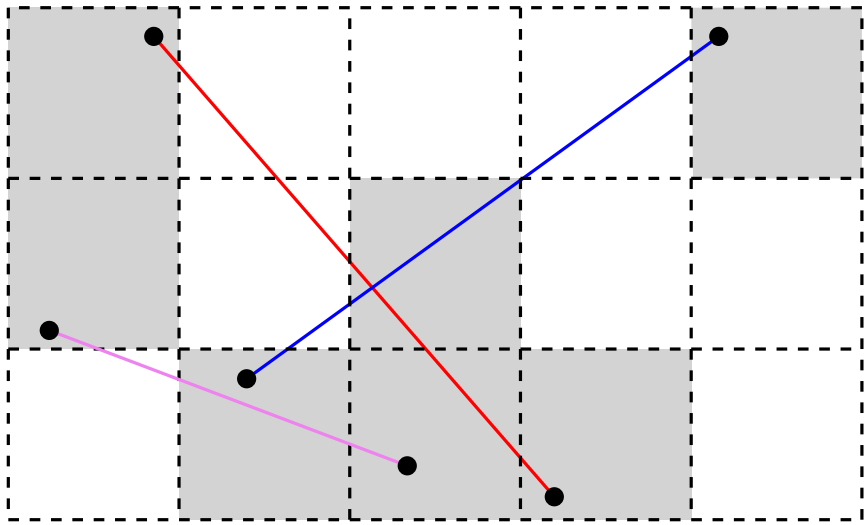


2D Snap Rounding

Well understood in 2D

We round coordinates on the integer grid [Guibas & Marimont 98]

- Pixels that contain a vertex of the arrangement are tagged *hot*
- Segments that intersect a hot pixel are subdivided in that pixel
- All vertices are rounded to to their pixels centers

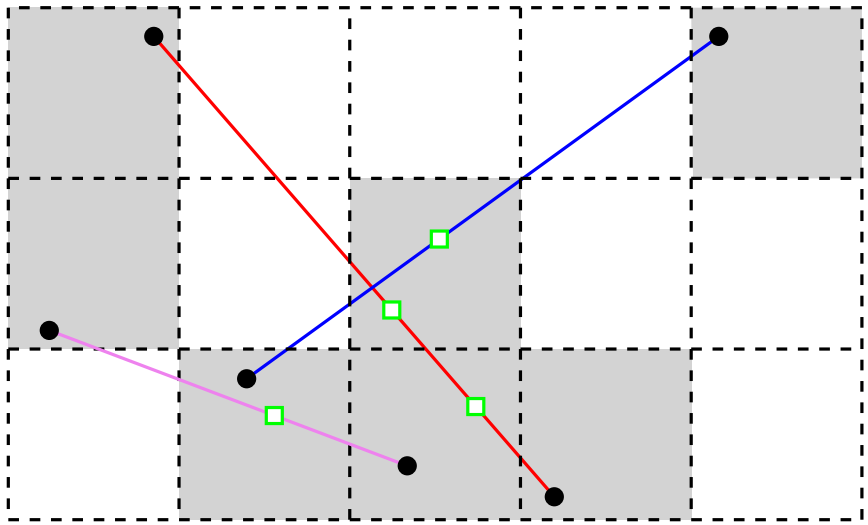


2D Snap Rounding

Well understood in 2D

We round coordinates on the integer grid [Guibas & Marimont 98]

- Pixels that contain a vertex of the arrangement are tagged *hot*
- Segments that intersect a hot pixel are subdivided in that pixel
- All vertices are rounded to to their pixels centers

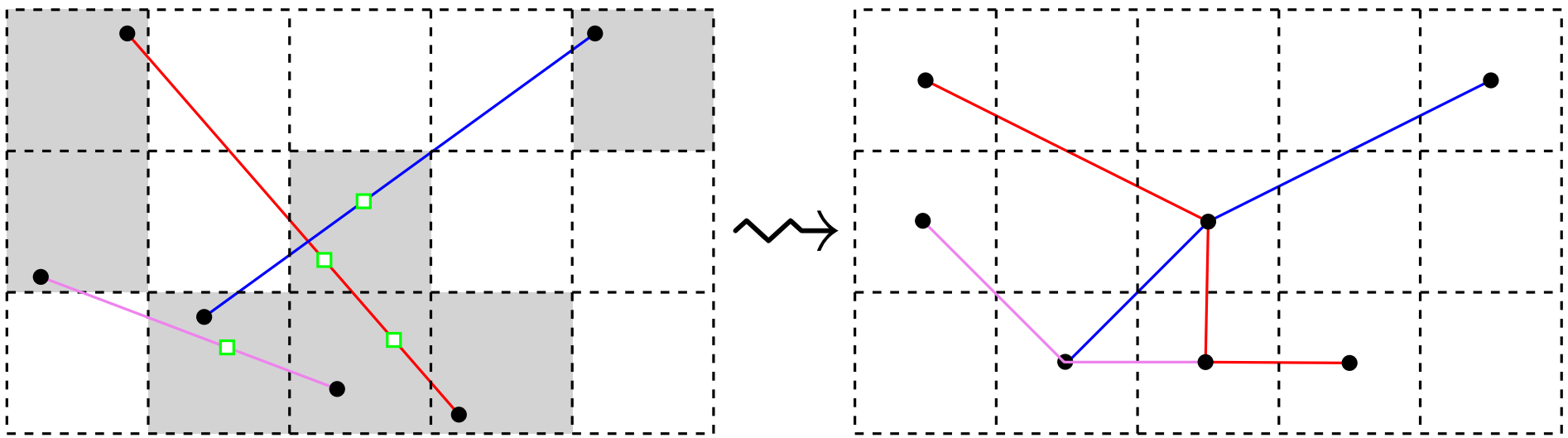


2D Snap Rounding

Well understood in 2D

We round coordinates on the integer grid [Guibas & Marimont 98]

- Pixels that contain a vertex of the arrangement are tagged *hot*
- Segments that intersect a hot pixel are subdivided in that pixel
- All vertices are rounded to their pixels centers



3D Snap Rounding

1990 CCCG, Milenkovic. Face lattices in $\mathbb{R}^d$ Incorrect in 3D

1997 CAGD, Fortune High-level algo for plane-based polyhedra
Does not generalize to vertex-based polyhedra

1999 DCG, Fortune
- The algorithm is very intricate
- Coordinates are rounded to integer multiples of about $1/n$

No satisfying solution

3D Snap Rounding

1990 CCCG, Milenkovic. Face lattices in $\mathbb{R}^d$ Incorrect in 3D

1997 CAGD, Fortune High-level algo for plane-based polyhedra
Does not generalize to vertex-based polyhedra

1999 DCG, Fortune
- The algorithm is very intricate
- Coordinates are rounded to integer multiples of about $1/n$

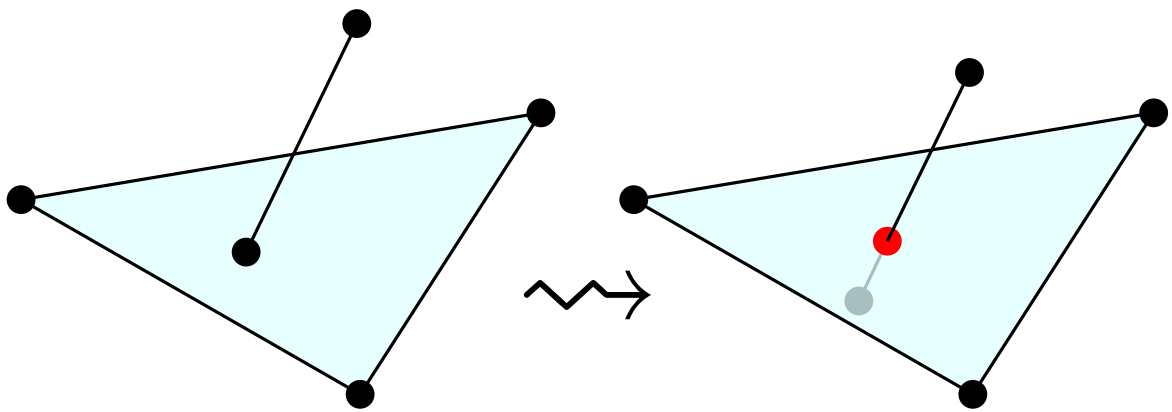
No satisfying solution

2020 DCG, S. Lazard, W. Lenhart, O. Devillers

The difficulty of 3D Snap Rounding

Assume that vertices are rounded to the center of their voxel

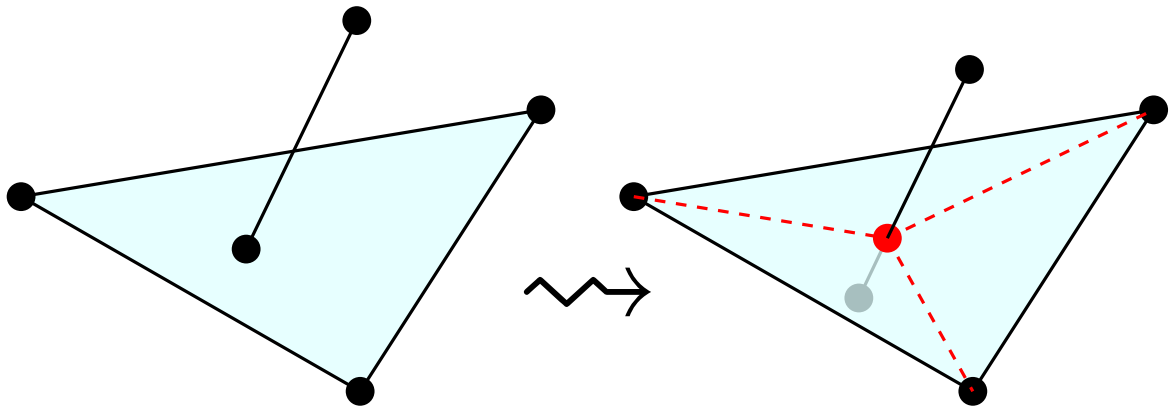
During rounding, a vertex might traverse a face



The difficulty of 3D Snap Rounding

Assume that vertices are rounded to the center of their voxel

During rounding, a vertex might traverse a face



Faces should be retriangulated

The new edges may intersect other edges when rounded

Unknown if such approaches always terminates

The difficulty of 3D Snap Rounding

Second example: Nice but flawed algorithm

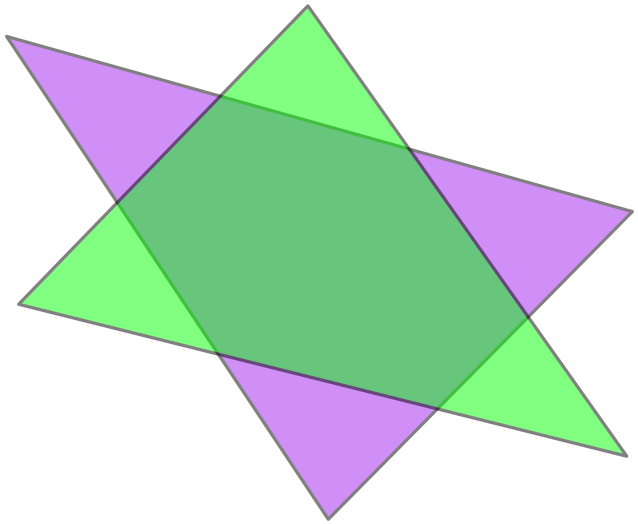
Project all edges on the xy-plane

Subdivide them as in 2D snap rounding

Add the boundaries of hot pixels (to avoid recursion) & Triangulate

Lift this triangulation vertically back on all faces

Round all vertices to the centers of their voxels



The difficulty of 3D Snap Rounding

Second example: Nice but flawed algorithm

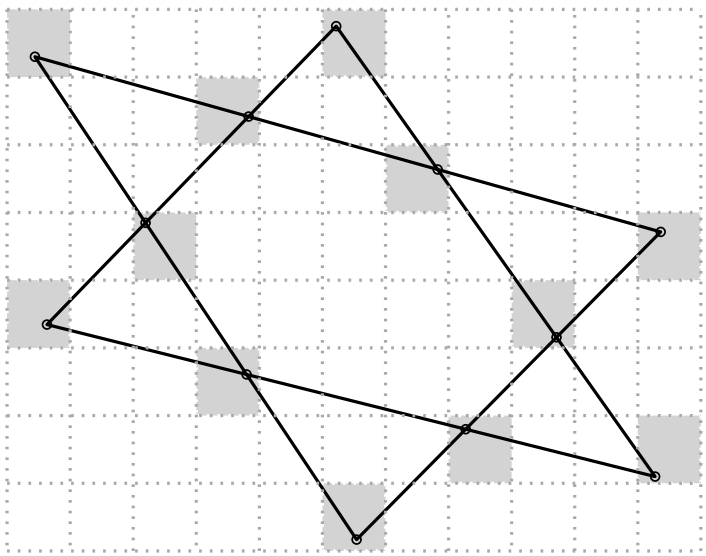
Project all edges on the xy-plane

Subdivide them as in 2D snap rounding

Add the boundaries of hot pixels (to avoid recursion) & Triangulate

Lift this triangulation vertically back on all faces

Round all vertices to the centers of their voxels



The difficulty of 3D Snap Rounding

Second example: Nice but flawed algorithm

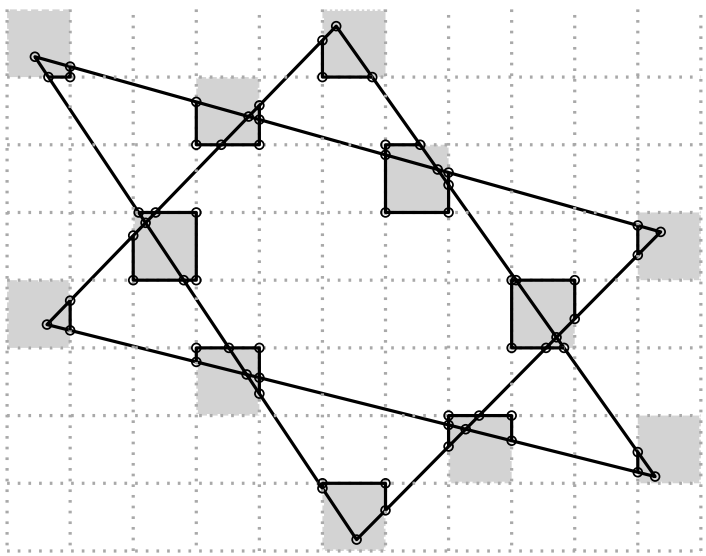
Project all edges on the xy-plane

Subdivide them as in 2D snap rounding

Add the boundaries of hot pixels (to avoid recursion) & Triangulate

Lift this triangulation vertically back on all faces

Round all vertices to the centers of their voxels



Blue: connect 3 pixels
Yellow: connect 2 pixels

Grey: in 1 pixel

The difficulty of 3D Snap Rounding

Second example: Nice but flawed algorithm

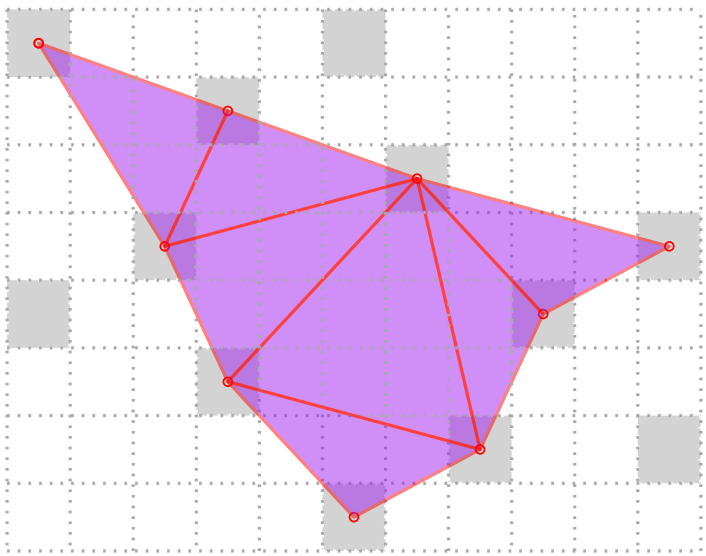
Project all edges on the xy-plane

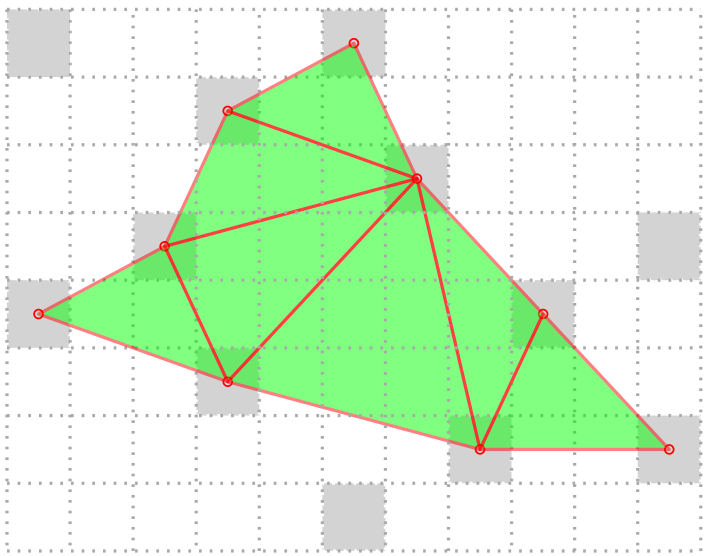
Subdivide them as in 2D snap rounding

Add the boundaries of hot pixels (to avoid recursion) & Triangulate

Lift this triangulation vertically back on all faces

Round all vertices to the centers of their voxels





The difficulty of 3D Snap Rounding

Second example: Nice but flawed algorithm

Project all edges on the xy-plane

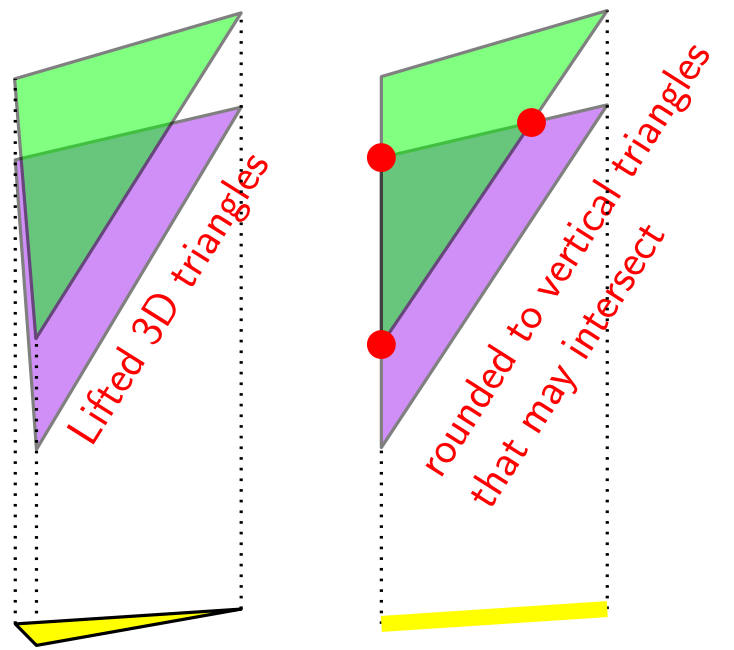
Subdivide them as in 2D snap rounding

Add the boundaries of hot pixels (to avoid recursion) & Triangulate

Lift this triangulation vertically back on all faces

Round all vertices to the centers of their voxels

Flaw:



Yellow triangle rounded to a segment

The difficulty of 3D Snap Rounding

Second example: Nice but flawed algorithm

Project all edges on the xy-plane

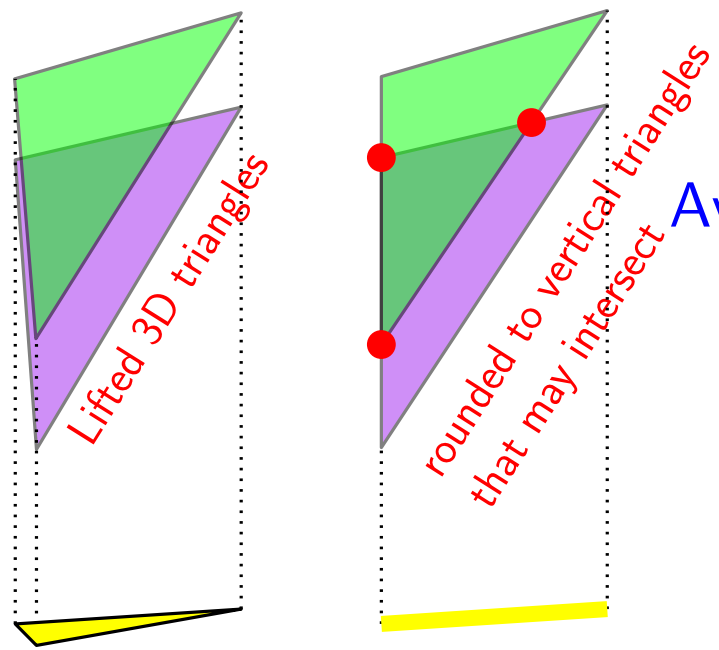
Subdivide them as in 2D snap rounding

Add the boundaries of hot pixels (to avoid recursion) & Triangulate

Lift this triangulation vertically back on all faces

Round all vertices to the centers of their voxels

Flaw:



Yellow triangle rounded to a segment

Fortune's solution [1999]:
Avoid vertical rounding of the faces
by using a finer grid
(integer multiples of $\approx 1/n$)

The difficulty of 3D Snap Rounding

Second example: Nice but flawed algorithm

Project all edges on the xy-plane

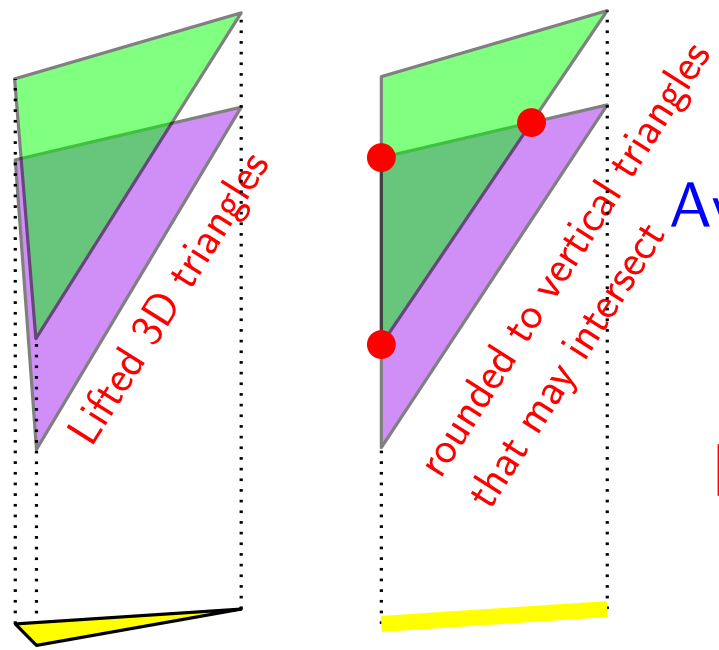
Subdivide them as in 2D snap rounding

Add the boundaries of hot pixels (to avoid recursion) & Triangulate

Lift this triangulation vertically back on all faces

Round all vertices to the centers of their voxels

Flaw:



Yellow triangle rounded to a segment

Fortune's solution [1999]:
Avoid vertical rounding of the faces
by using a finer grid
(integer multiples of $\approx 1/n$)

Nice but flawed algorithm

Our Algorithm

WHILE Some proper intersections occur:

0. add involved faces in F_C.

IF F_C has changed:

1. Collapse faces close to one another.

2. Project the faces on the floor, subdivide them and lift.

ELSE

3. Partition intersecting features into xy-parallel slabs.

4. Triangulate the faces and simulate again the rounding.

Our Algorithm

WHILE Some proper intersections occur:

0. add involved faces in F_C.

IF F_C has changed:

1. Collapse faces close to one another.

2. Project the faces on the floor, subdivide them and lift.

ELSE

3. Partition intersecting features into xy-parallel slabs.

4. Triangulate the faces and simulate again the rounding.

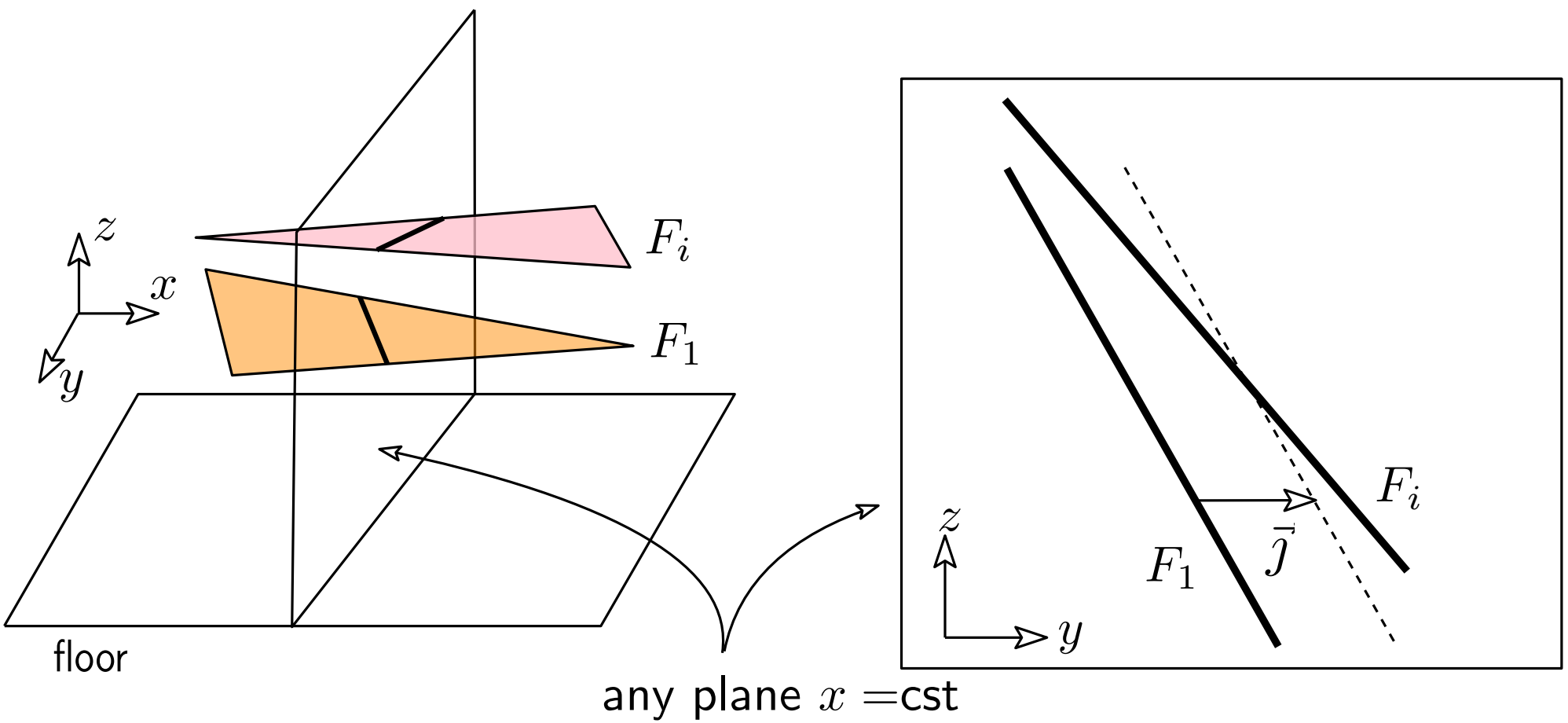
1. Collapse the faces that are close to one another

We deform iteratively the input faces ordered arbitrarily from F_1 to F_n

For i from 2 to n

Project the points of F_i along y onto $F_1, \ldots, F_{i-1}$, in order

iff the points project at distance at most 1



1. Collapse the faces that are close to one another

We deform iteratively the input faces ordered arbitrarily from F_1 to F_n

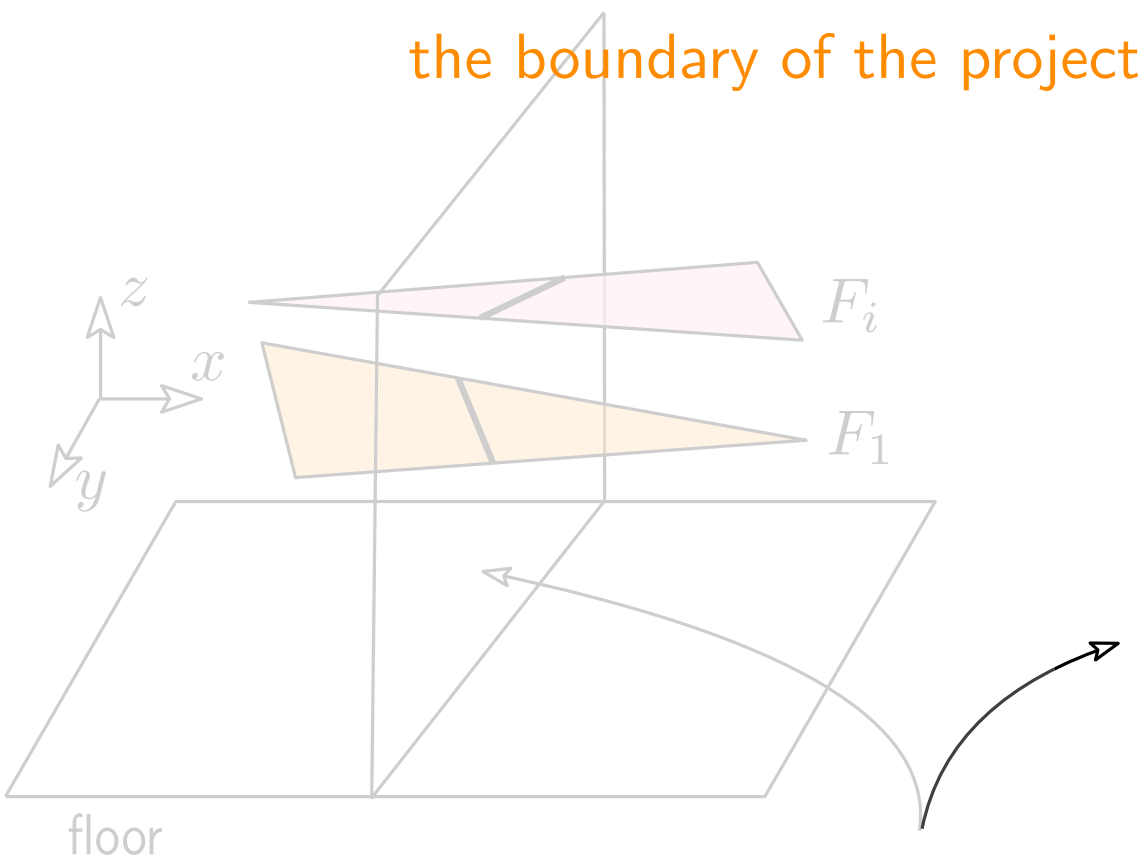
For i from 2 to n

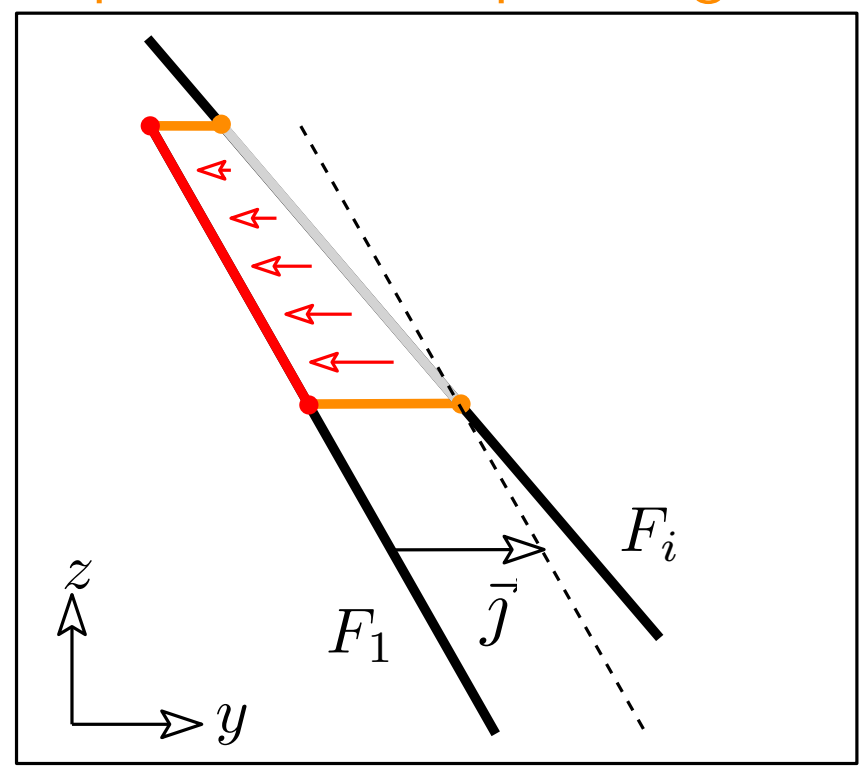
Project the points of F_i along y onto $F_1, \ldots, F_{i-1}$, in order

iff the points project at distance at most 1

Create, if needed, new faces that connect

the boundary of the projected points to their pre-image





any plane $x =$cst

1. Collapse the faces that are close to one another

We deform iteratively the input faces ordered arbitrarily from F_1 to F_n

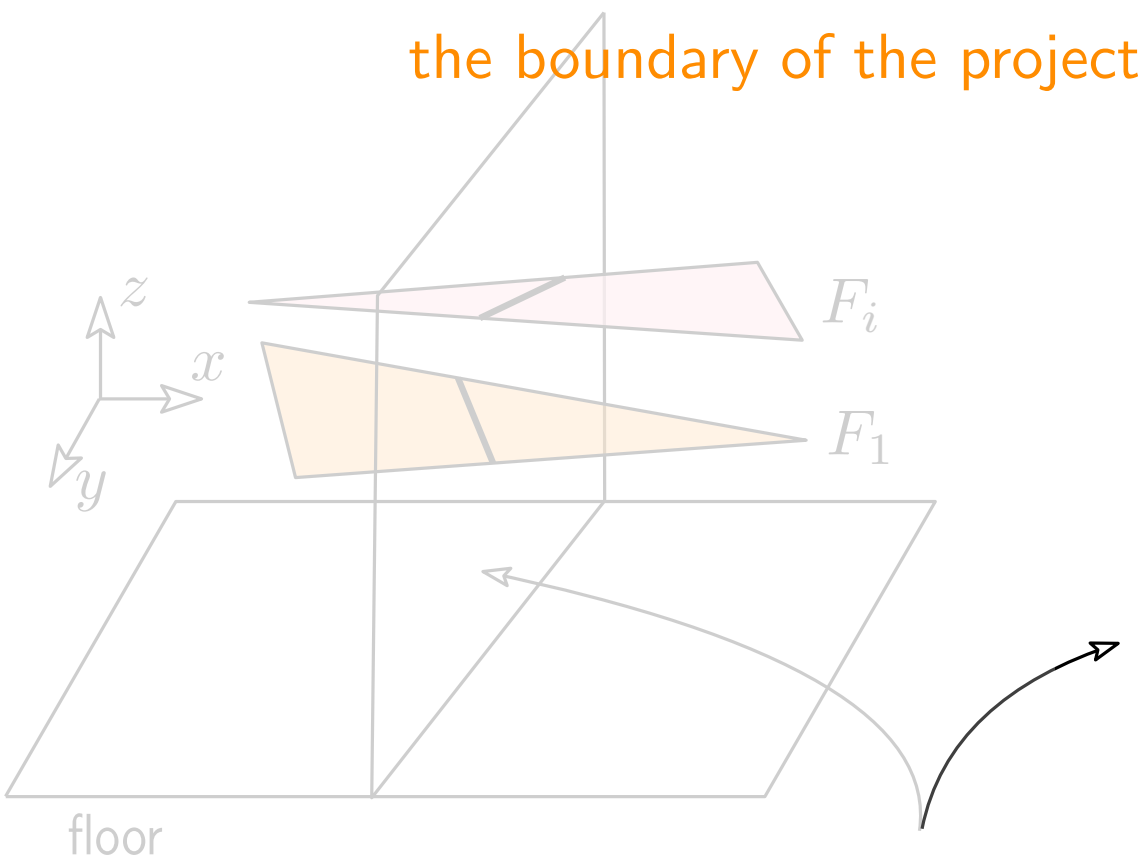
For i from 2 to n

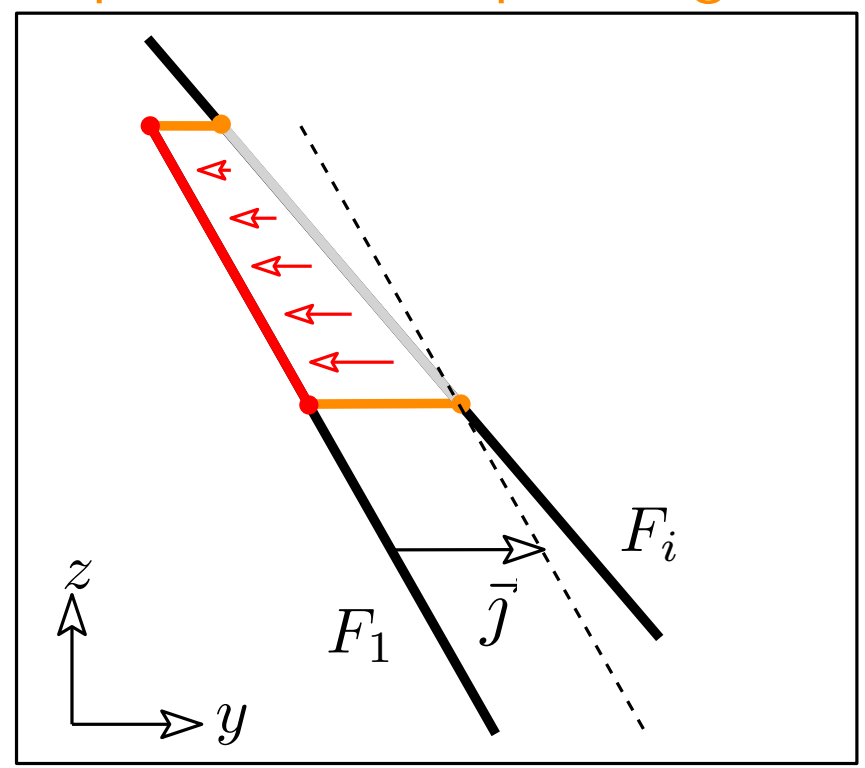
Project the points of F_i along y onto $F_1, \ldots, F_{i-1}$, in order

iff the points project at distance at most 1

Create, if needed, new faces that connect

the boundary of the projected points to their pre-image





any plane $x =$cst

Our Algorithm

WHILE Some proper intersections occur:

0. add involved faces in F_C.

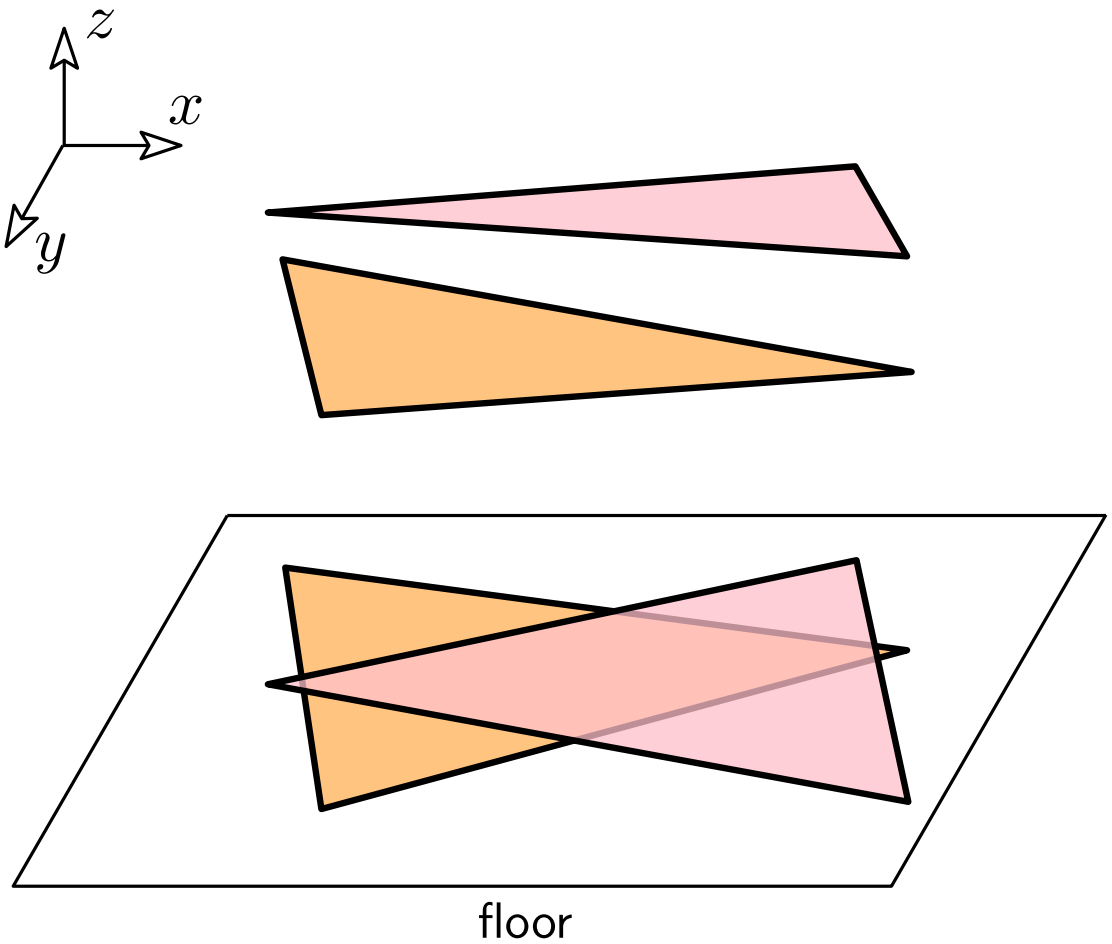
IF F_C has changed:

1. Collapse faces close to one another.
2. Project the faces on the floor, subdivide them and lift.

ELSE

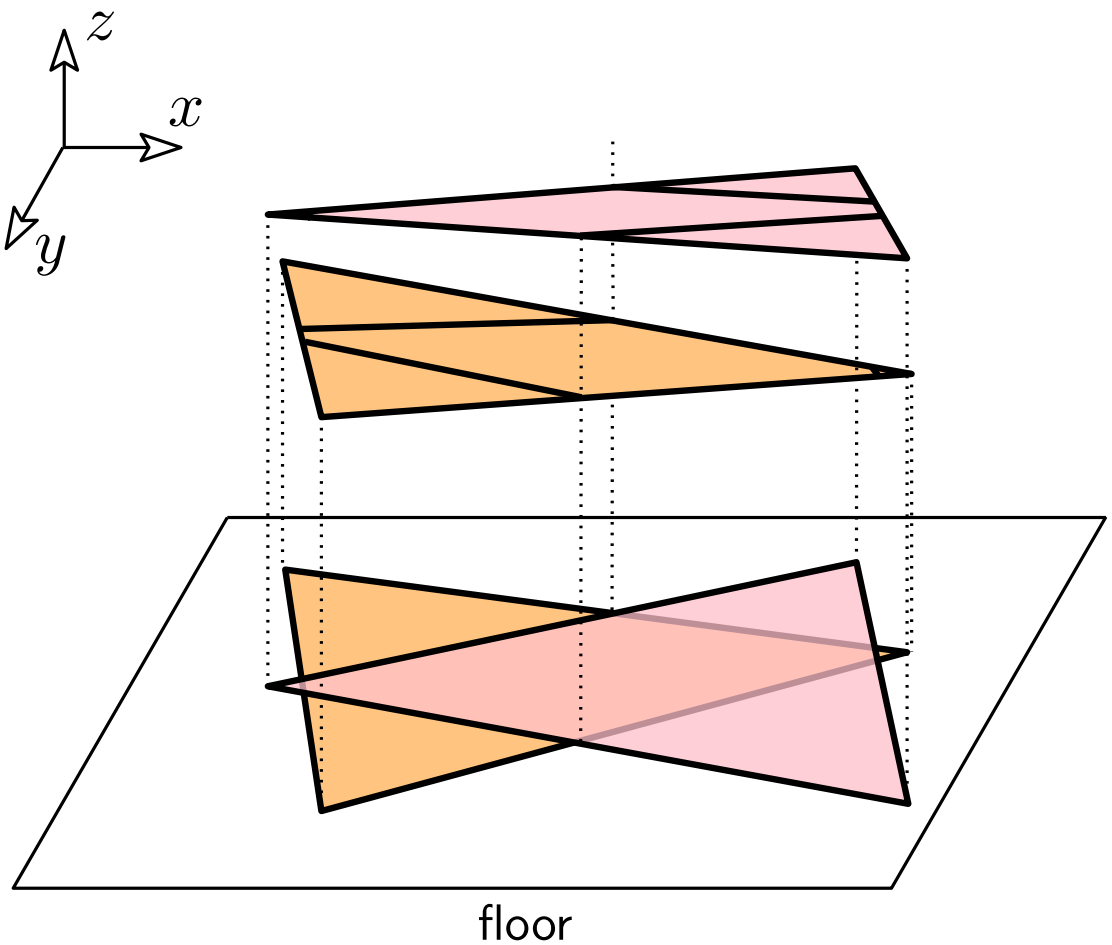
3. Partition intersecting features into xy-parallel slabs.
4. Triangulate the faces and simulate again the rounding.

2. Project faces on the floor, subdivide them and lift.



• For all pair of faces at distance at most 1. Project them on the floor and subdivide them.

2. Project faces on the floor, subdivide them and lift.



- For all pair of faces at distance at most 1. Project them on the floor and subdivide them.
- Lift it back on 3D faces and subdivide them

Our Algorithm

WHILE Some proper intersections occur:

0. add involved faces in F_C.

IF F_C has changed:

1. Collapse faces close to one another.

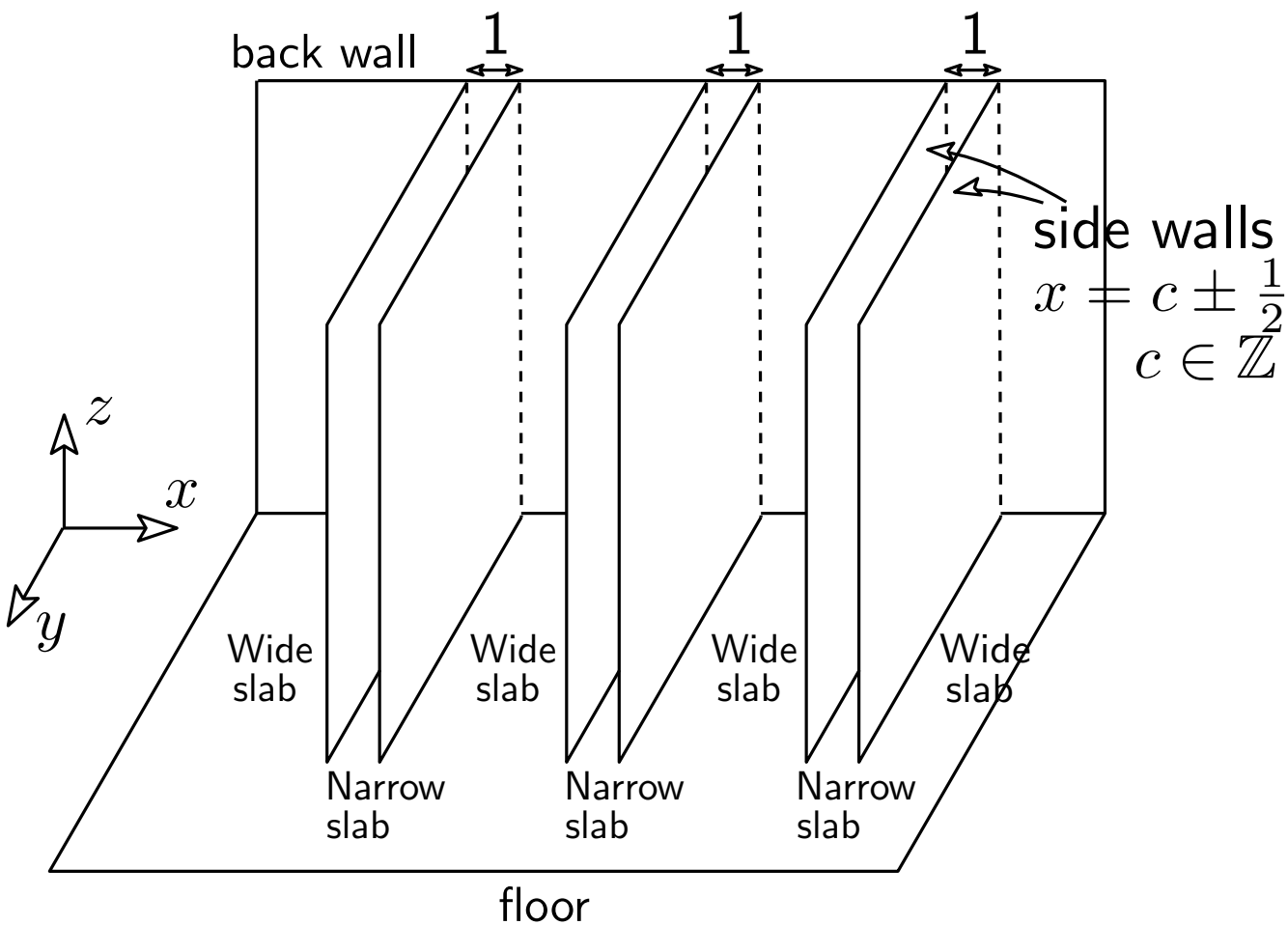
2. Project the faces on the floor, subdivide them and lift.

ELSE

3. Partition intersecting features into xy-parallel slabs.

4. Triangulate the faces and simulate again the rounding.

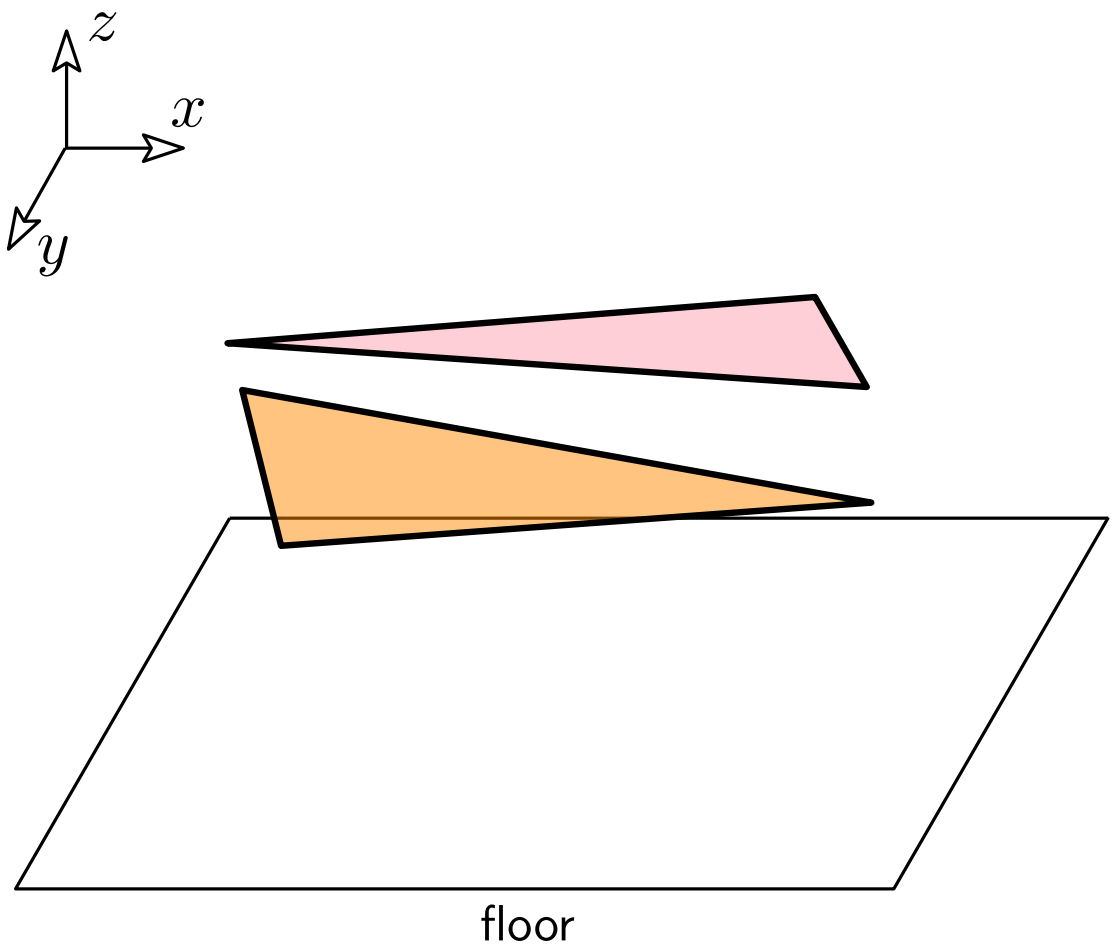
3.Partition intersecting features into xy-parallel slabs



Narrow slab: Region in between two side walls $x = c \pm \frac{1}{2},\ c \in \mathbb{Z}$

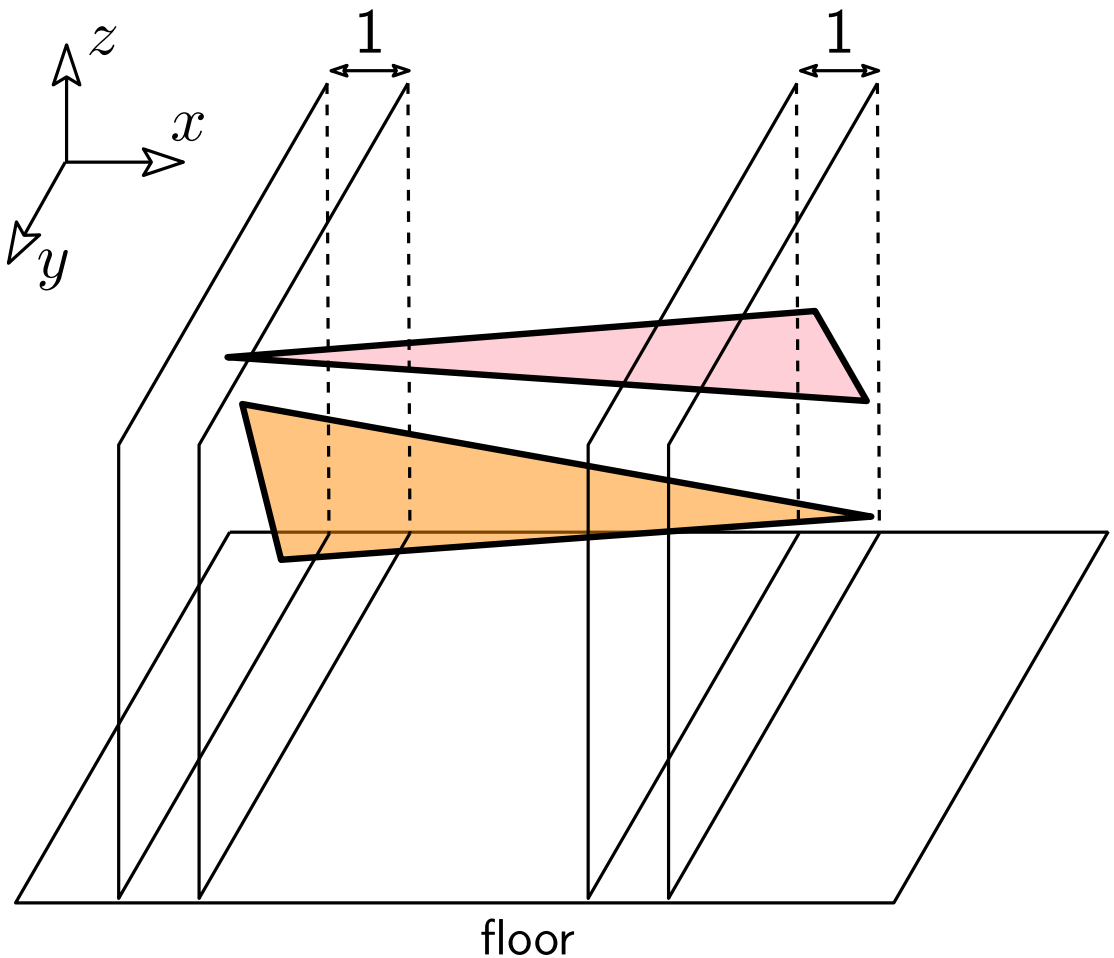
Wide slabs: Region bounded by two consecutive thin slabs

3.Partition intersecting features into xy-parallel slabs



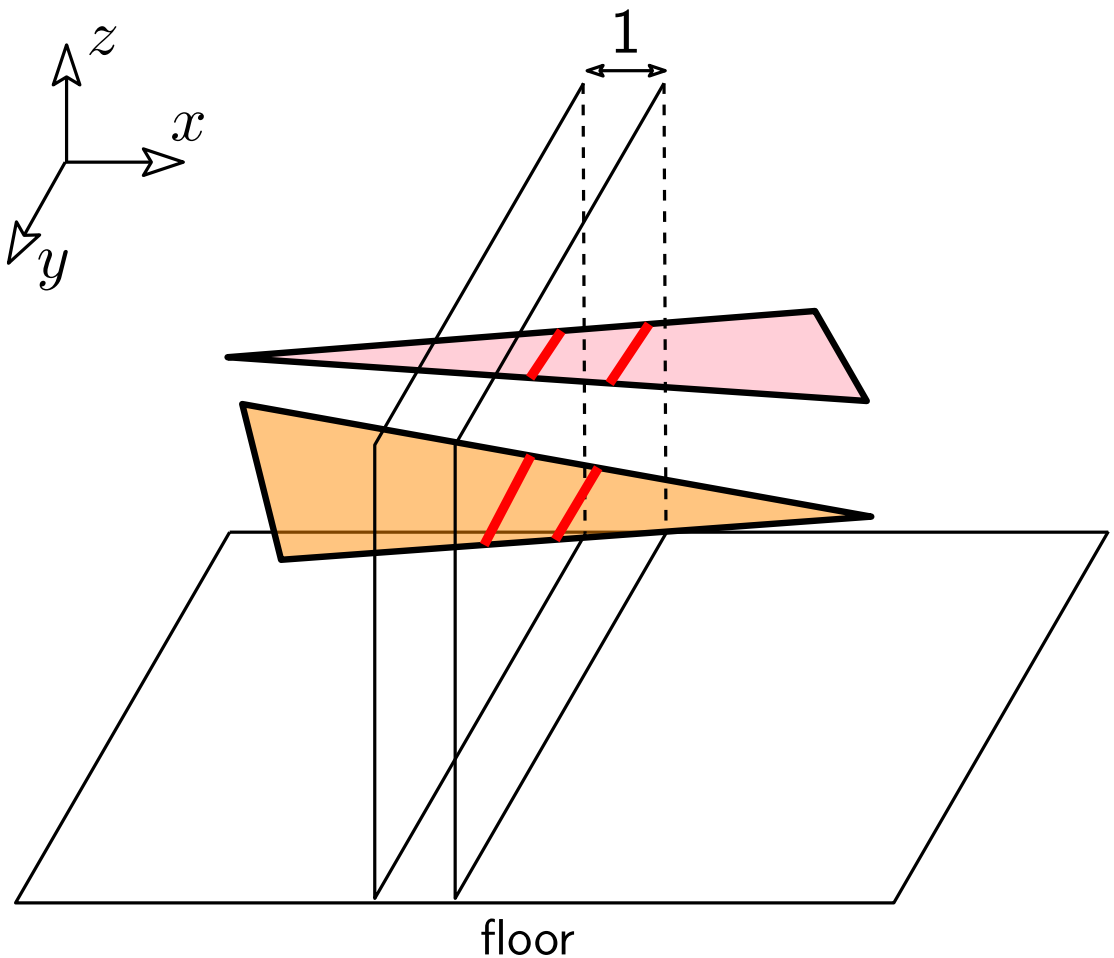
- For all pair of faces at distance that intersect during the virtual rounding.

3.Partition intersecting features into xy-parallel slabs



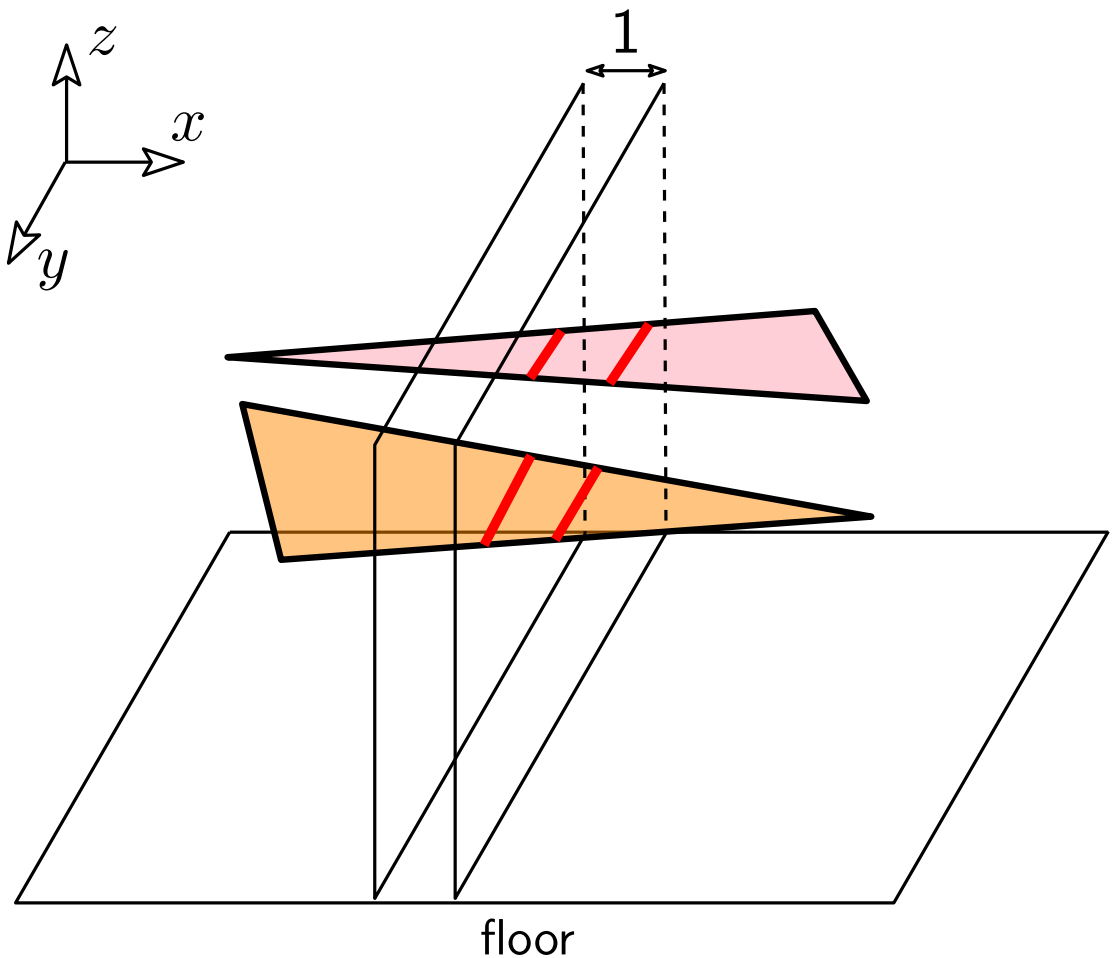
- For all pair of faces at distance that intersect during the virtual rounding.
- define a narrow slab for each of they're vertices.

3.Partition intersecting features into xy-parallel slabs



- For all pair of faces at distance that intersect during the virtual rounding.
- If they're cut, define a narrow slab for each of they're vertices.
- Subdivide the 2 faces by thoses slabs

3.Partition intersecting features into xy-parallel slabs



- For all pair of faces at distance that intersect during the virtual rounding.
- If they're cut, define a narrow slab for each of they're vertices.
- Subdivide the 2 faces by thoses slabs
- Subdivide by thoses slabs the faces at distance 1 from these 2 faces to keep faces consistent on the floor.

Our Algorithm

WHILE Some proper intersections occur:

0. add involved faces in F_C.

IF F_C has changed:

1. Collapse faces close to one another.

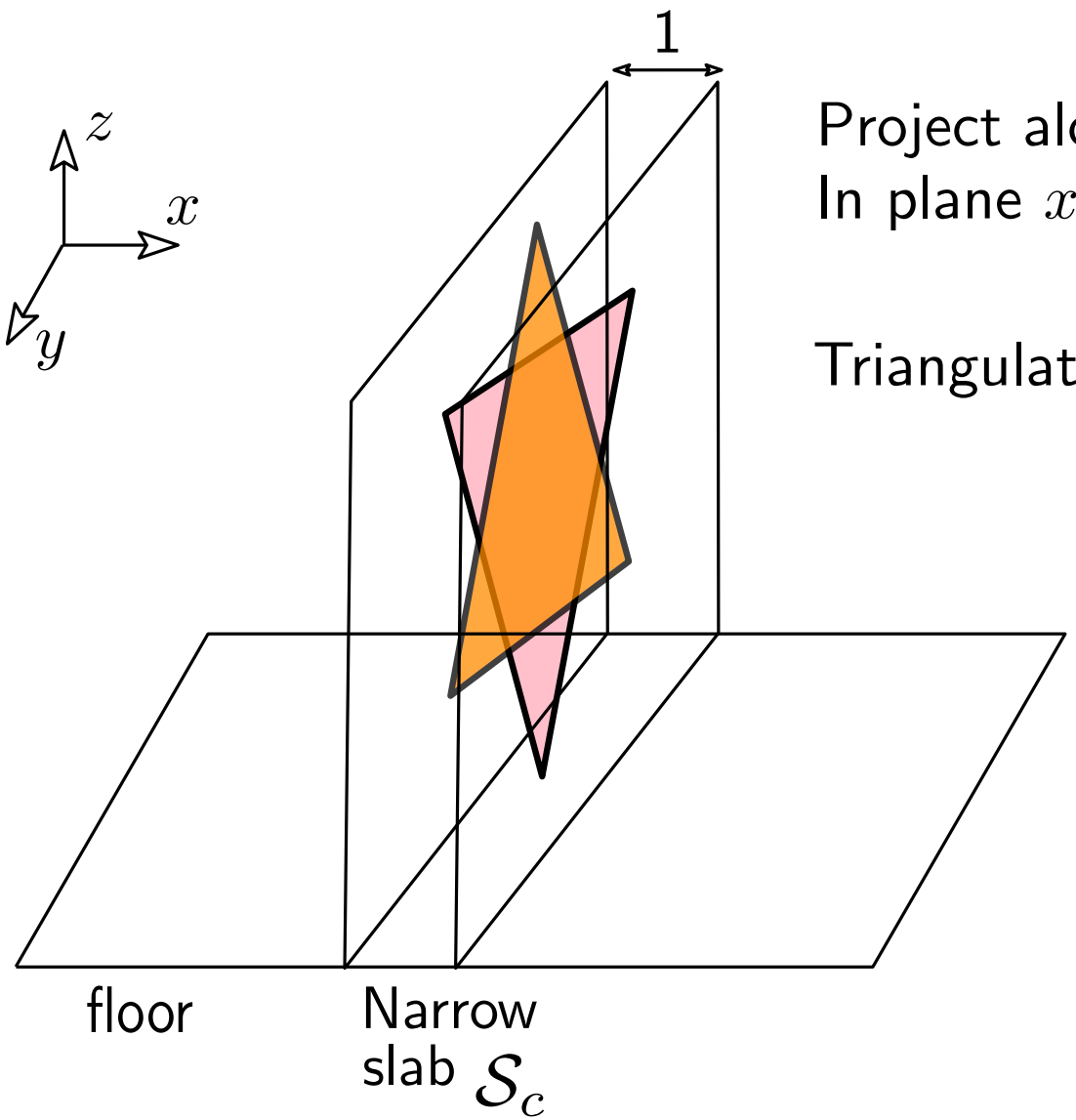
2. Project the faces on the floor, subdivide them and lift.

ELSE

3. Partition intersecting features into xy-parallel slabs.

4. Triangulate the faces and simulate again the rounding.

4. Triangulate the faces in narrow slabs



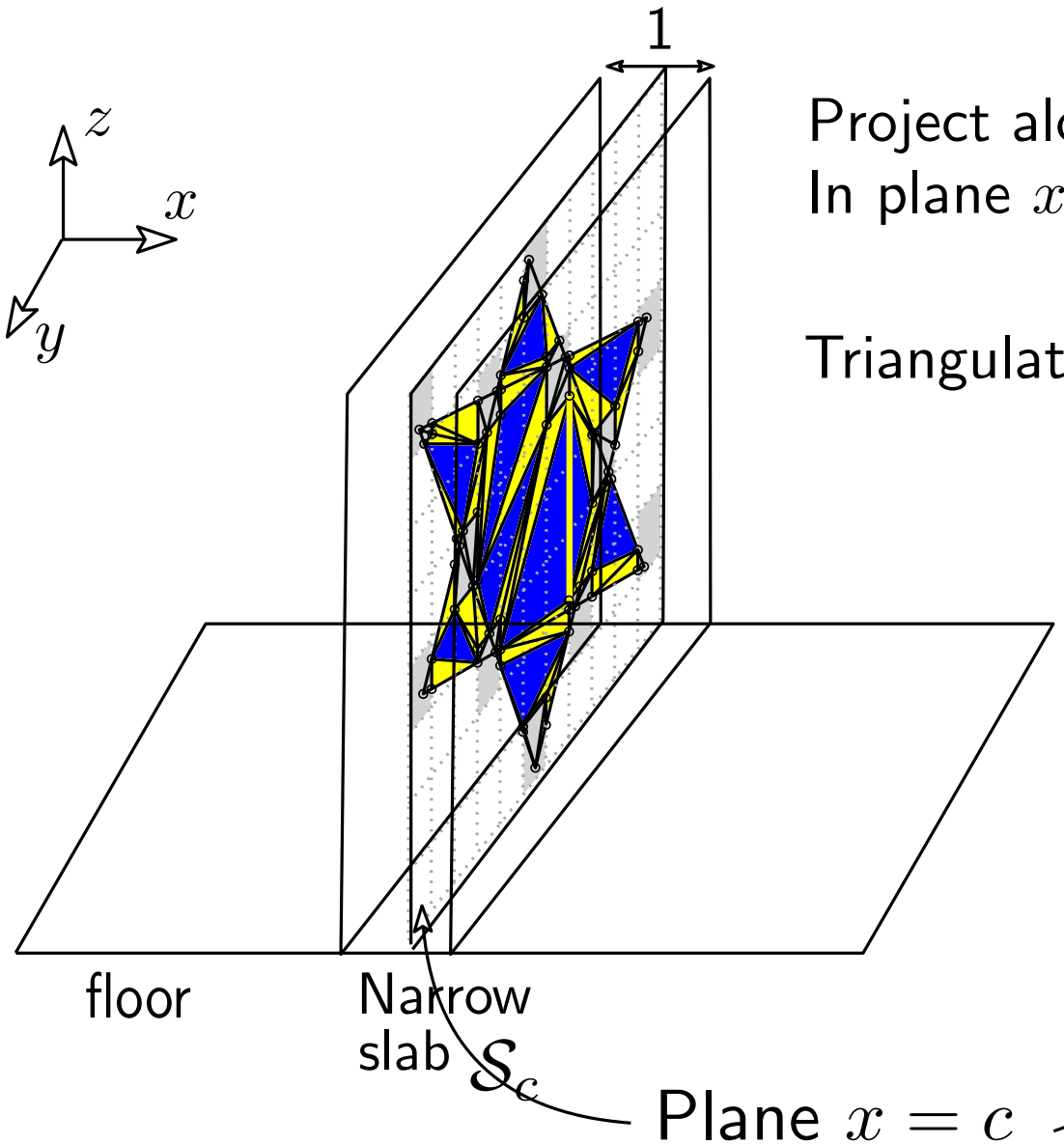
Project along x all faces on $x = c$

In plane $x = c$, subdivide all edges

as in 2D snap rounding

Triangulate and lift back on the faces in F_C

4. Triangulate the faces in narrow slabs

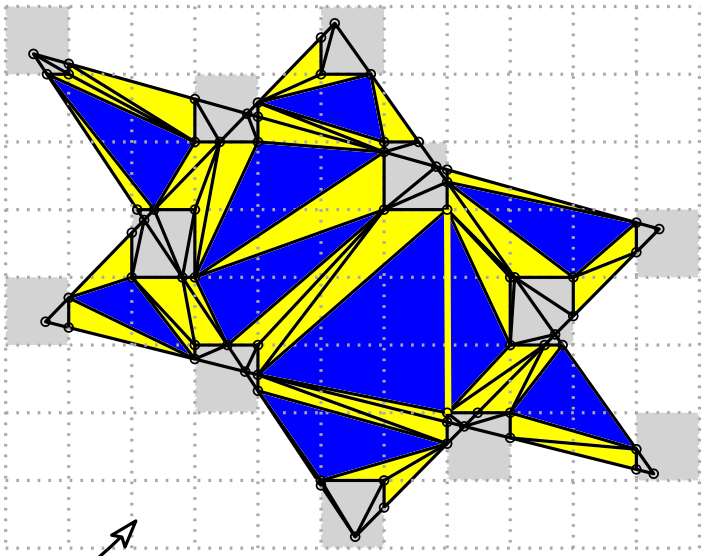


Project along x all faces on $x = c$

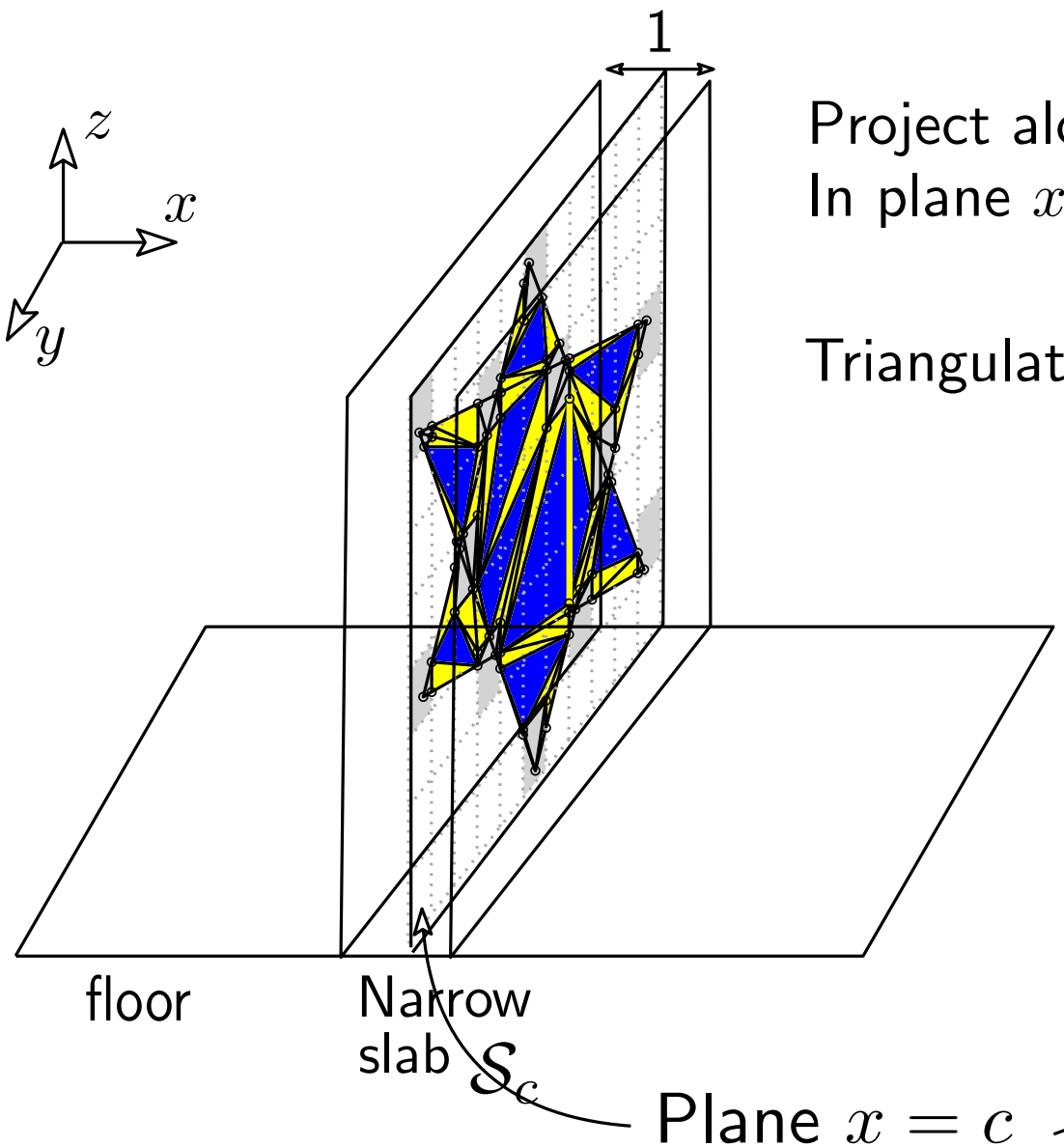
In plane $x = c$, subdivide all edges

as in 2D snap rounding

Triangulate and lift back on the faces in F_C

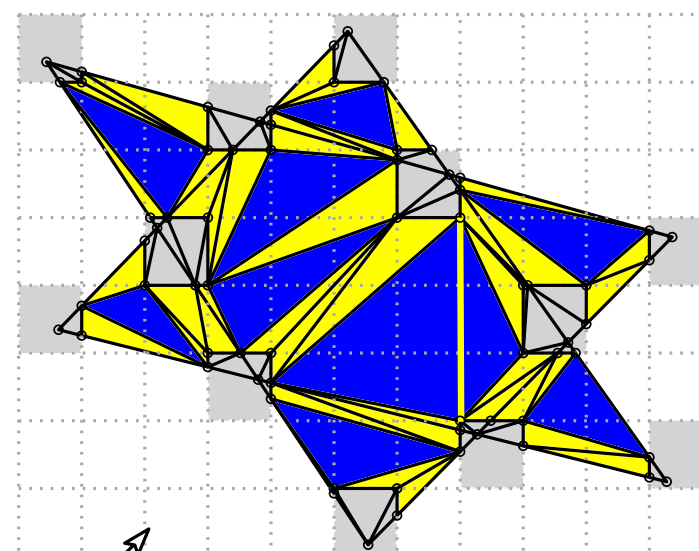


4. Triangulate the faces in narrow slabs



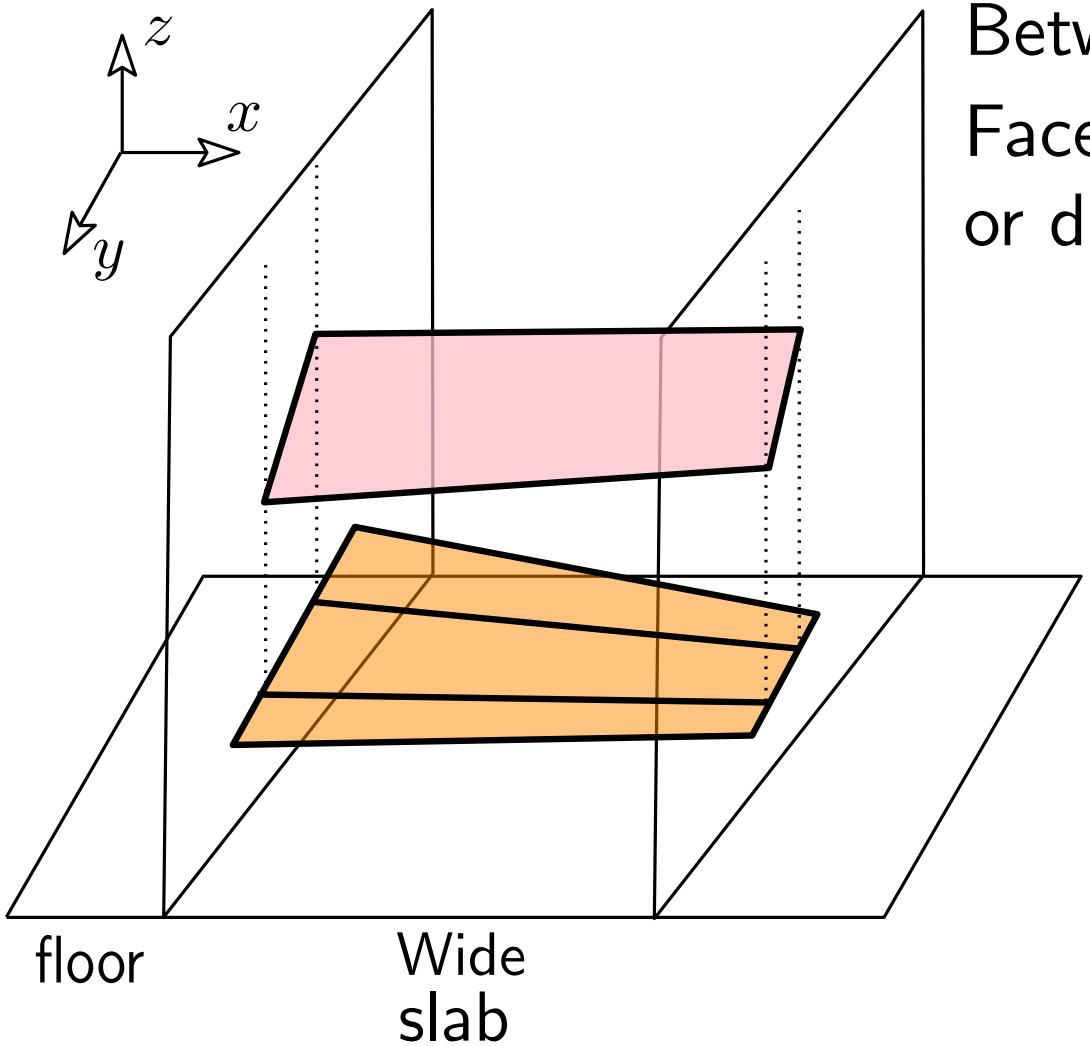
Project along x all faces on $x = c$
In plane $x = c$, subdivide all edges
as in 2D snap rounding

Triangulate and lift back on the faces in F_C



This may create vertices in the side-wall boundaries $x = c \pm \frac{1}{2}$
shared with the adjacent thick slabs

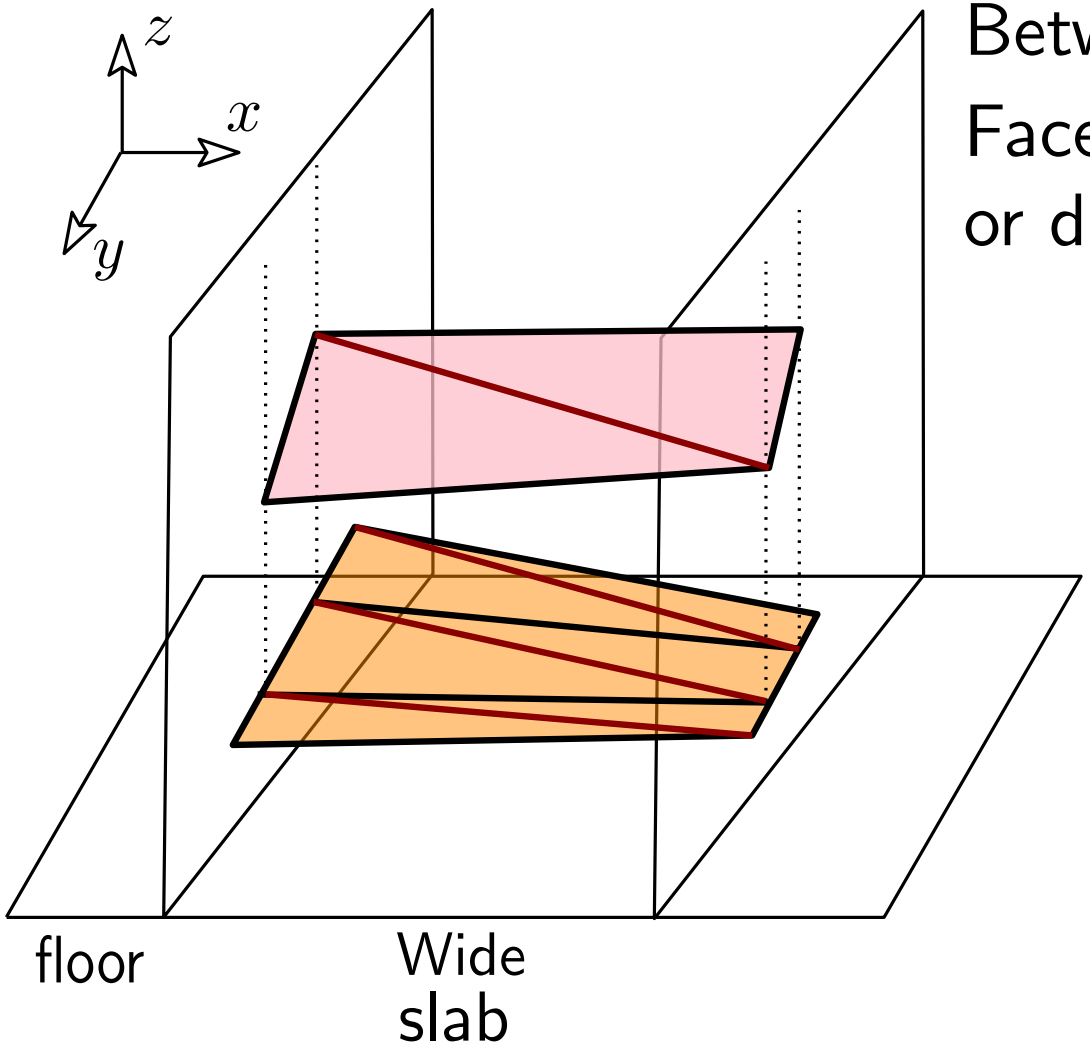
4. Triangulate the faces between narrow slabs



Between narrow slabs

Faces are trapezoids with identical or disjoint floor projections

4. Triangulate the faces between narrow slabs

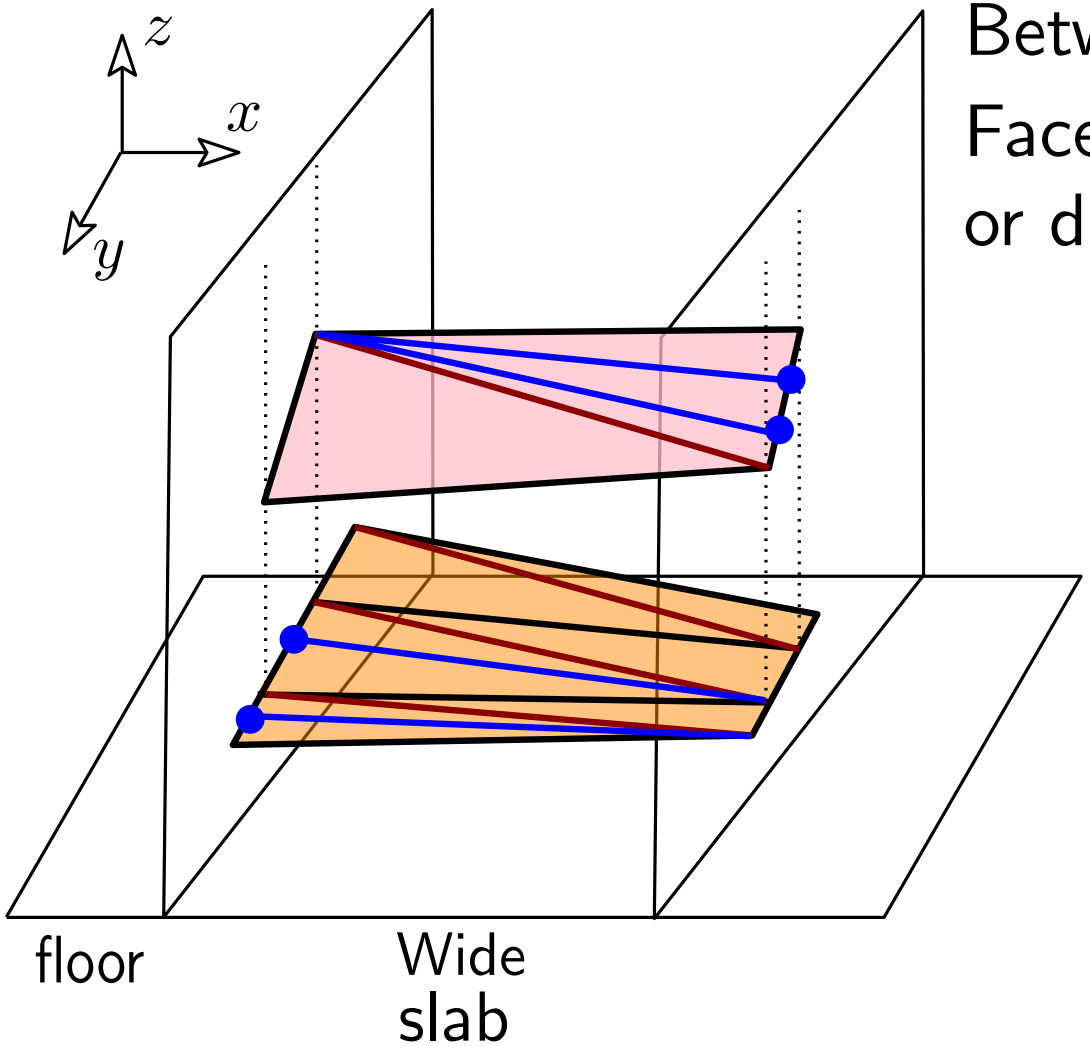


Between narrow slabs

Faces are trapezoids with identical or disjoint floor projections

- Triangulate with diagonals whose floor projections are consistent

4. Triangulate the faces between narrow slabs



Between narrow slabs

Faces are trapezoids with identical or disjoint floor projections

- Triangulate with diagonals whose floor projections are consistent
- Further triangulate with the vertices just created in the adjacent narrow slabs

Our Algorithm

WHILE Some proper intersections occur:

0. add involved faces in F_C.

IF F_C has changed:

1. Collapse faces close to one another.

2. Project the faces on the floor, subdivide them and lift.

ELSE

3. Partition intersecting features into xy-parallel slabs.

4. Triangulate the faces and simulate again the rounding.

Recap

Three Critical Operations:

1. Collapse the faces that are close to one another
2. Project faces on the floor, subdivide them and lift.
3. Partition the space into xy-parallel slabs

Rather simple algorithm Delicate proof of correctness

Preserve the geometry L_∞ Hausdorff distance $\leqslant 3/2$

Preserve the topology "up to collapses"

Bad worst-case complexity: $O(n^{17})$ space and $O(n^{44})$ time

$O(n^5)$ space and $O(n^{6.5})$ time under some hypotheses

We conjecture $O(n)$ and $O(n \log n)$ time
under stronger hypotheses.

Conclusion and current works

First algorithm for snap rounding sets of polygons in 3D on a fixed-size grid

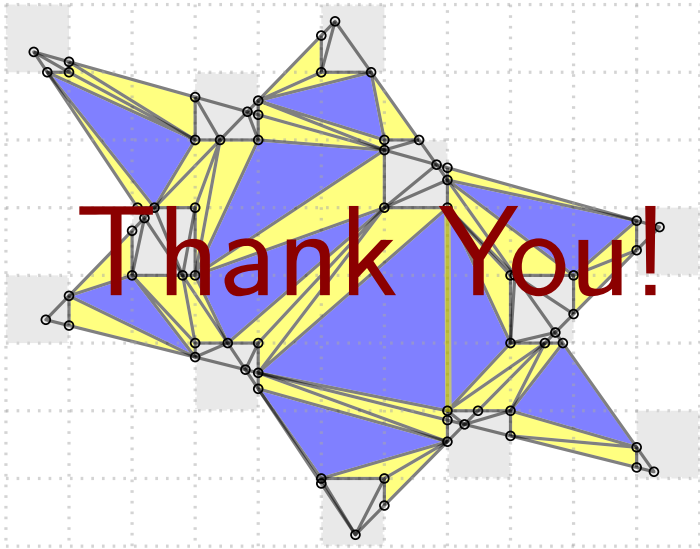
We have a first implementation of this algorithm.

.We quickly test it on 5.390 of meshes by rounding coordinates from double to simple precision.

.641 of them present a proper intersection after naive rounding.

.188 of them a round in less than 10min by our algorithm.

Debug and Optimisation are in progress to improve this result.


Thank You!