CSCI 4560/6560 Computational Geometry

https://www.cs.rpi.edu/~cutler/classes/computationalgeometry/S22/

Lecture 6:
Half-Space Intersections

Outline for Today

- Homework 2 Questions?
- Last Time: Monotone Polygons & Improved Triangulation Algorithm
- Motivation: Manufacturing by Mold Casting
- Dual Representation: Planar Constraints
- Half-Plane / Half-Space Intersection
- Incremental Linear Programming
- Related Application: Japanese Wood Joints
- Related Application: Automatic Robotic Part Sorting
- Next Time: Point Location

Homework 2

- Each Halfedge stores:
    - **vertex** at end of directed edge
    - **symmetric** halfedge
    - **face** to left of edge
    - **next** points to the Halfedge counter-clockwise around face on left

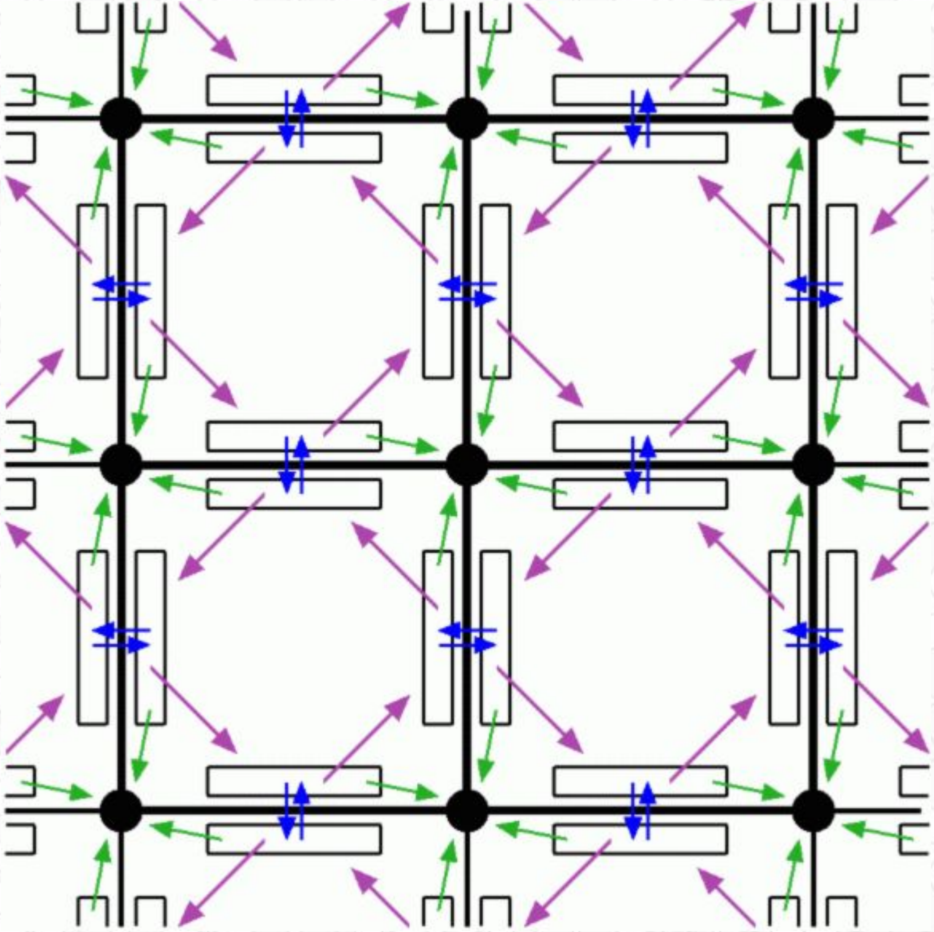


Image from Justin Legakis

Homework 2

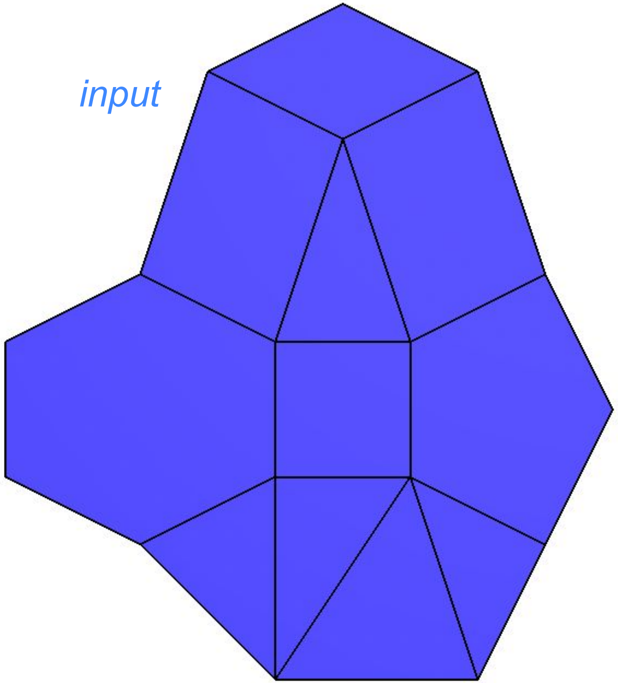
- Use CGAL's Surface Mesh (Halfedge) data structure

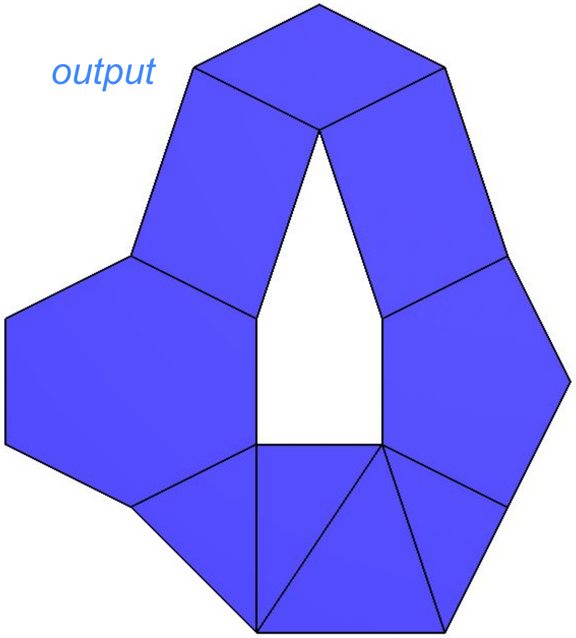
use case 1

- Input: all edges
- Output: all faces on any boundary

use case 2

- Input: 1 edge on a boundary
- Output: all faces on that boundary





How to Debug Installation Problems?

- Google the error message
  - likely someone else has had the problem and there may be a solution out there for you to try
- Read the error message carefully
  - Google to understand the terms and information in the error
- Read the installation documentation carefully
- Start over – and make sure you have
  - good internet
  - good power
  - enough hard drive
  - enough time to finish install

How to Read Software Documentation?

- Read carefully, start at the introduction, understand the organization of the documentation
- Understand the expectations of the functions (requirements on function arguments, etc)
- CGAL classes have
  - An overview section, which breaks implementation into categories,
  - hyperlinks to related pages (good, but sometimes navigation may be confusing)

What is “Bad” about (some) Software Documentation?

How do we write Good Software Documentation?
What can we do to avoid creating more “Bad” Software Documentation?

- Hyperlinks & navigation can be confusing
- Avoid duplicate/redundant information
- Search bar - would be nice to be able to filter by type, etc.
- Functions (overridden) with same name - unclear which one I want
- Documentation assumptions may be unclear to newbies
- Include usage examples for every function – e.g., cppreference.com
- Include time complexity of the function
- Enumerate all of the exceptions (errors) that can happen
- What do you need to #include to use this function
- Description of all input parameters & output & types

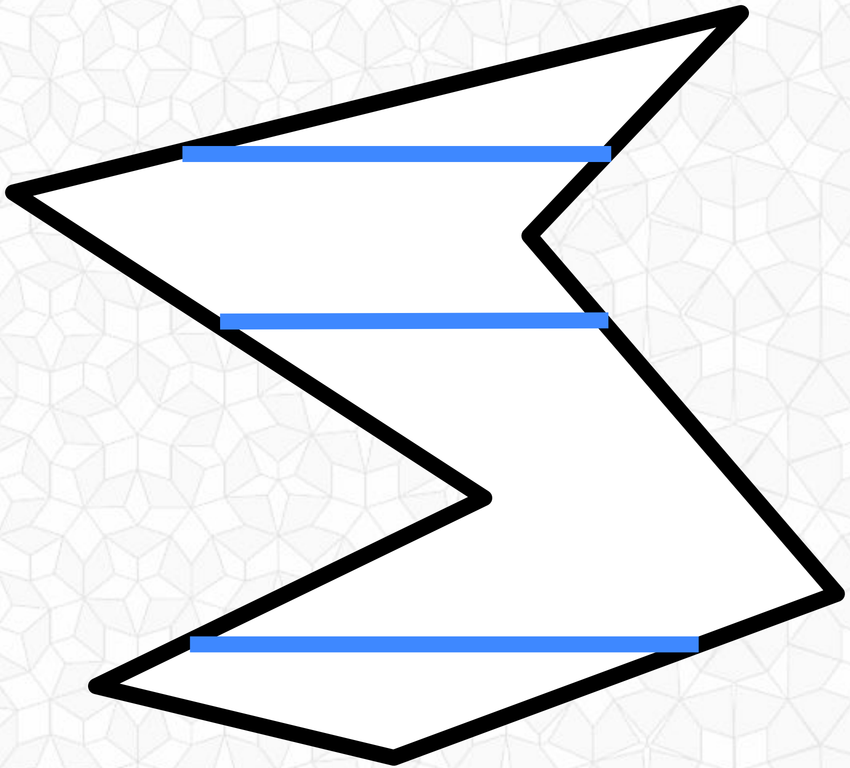
Outline for Today

- Homework 2 Questions?
- Last Time: Monotone Polygons & Improved Triangulation Algorithm
- Motivation: Manufacturing by Mold Casting
- Dual Representation: Planar Constraints
- Half-Plane / Half-Space Intersection
- Incremental Linear Programming
- Related Application: Japanese Wood Joints
- Related Application: Automatic Robotic Part Sorting
- Next Time: Point Location

Definition: Monotone with Respect to Y-Axis

- The intersection of the polygon with any line perpendicular to the y-axis is connected.

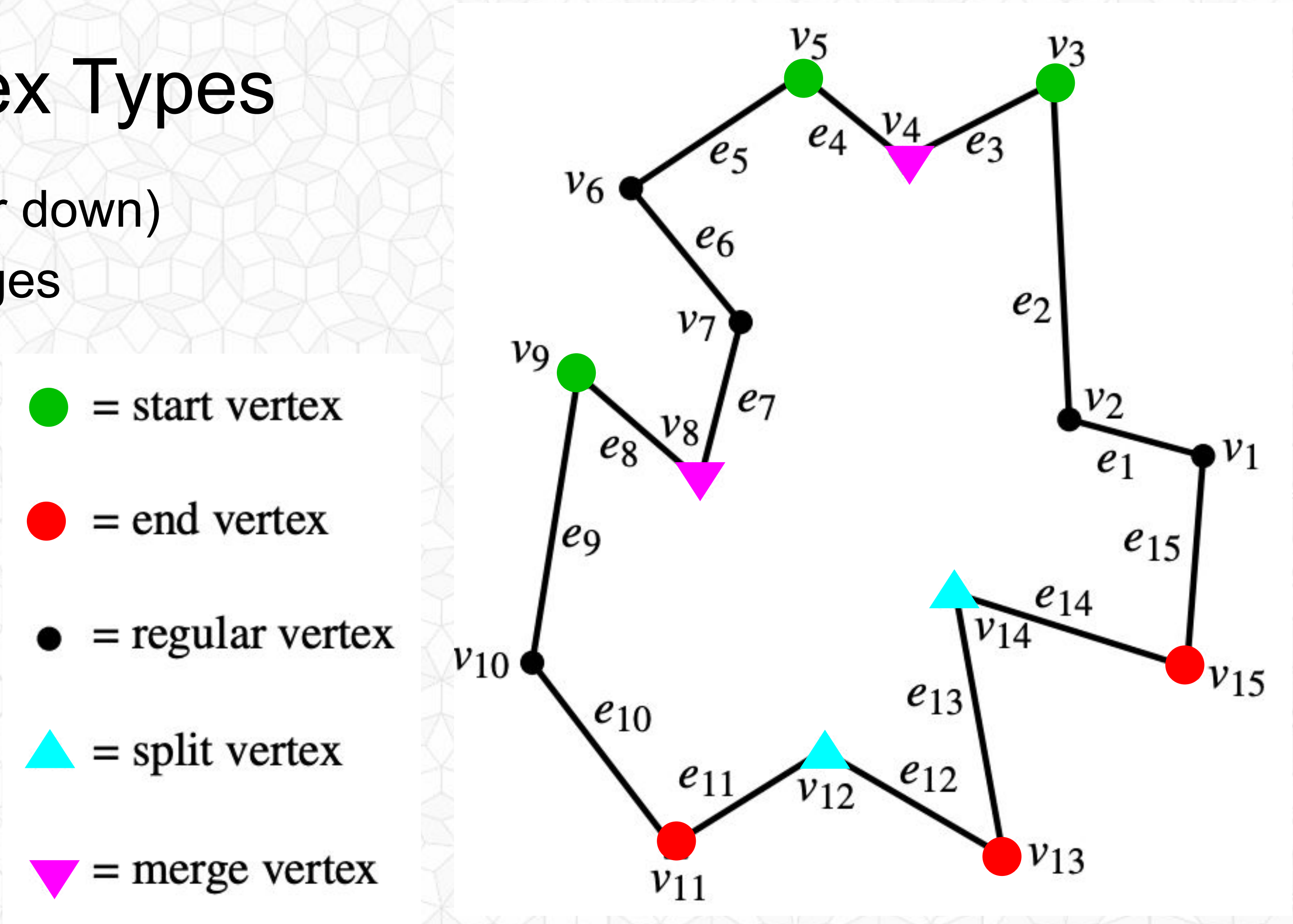
- The intersection is either
  - empty (above or below the polygon),
  - one point (top or bottom vertex), or
  - *a line segment.*



Identify Vertex Types

- Direction (up or down) of adjacent edges

- Interior angle at vertex (> 180° or < 180°)

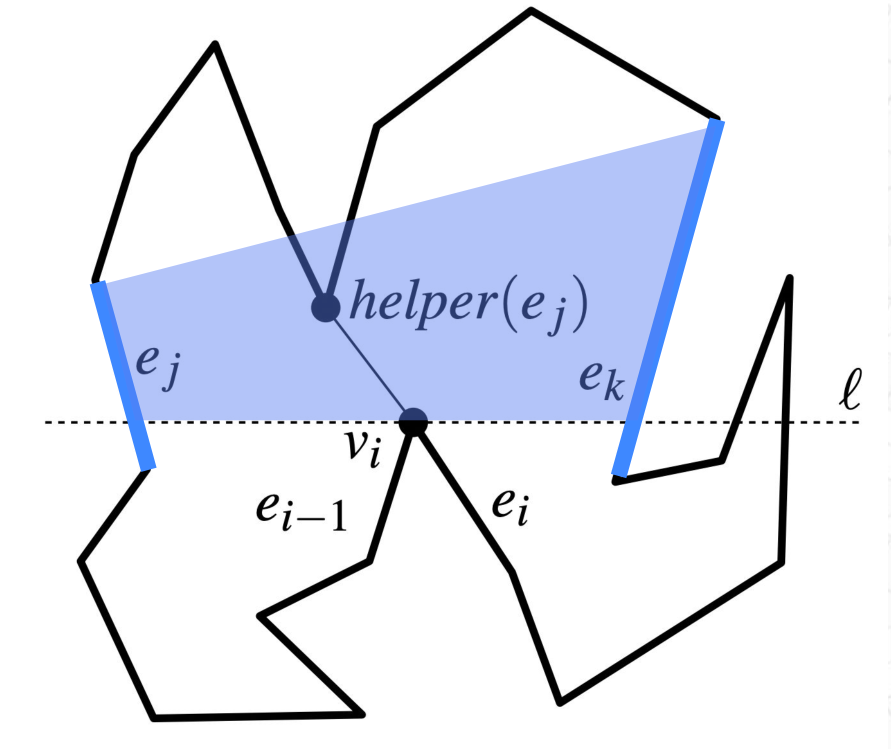


Computational Geometry Algorithms and Applications, de Berg, Cheong, van Kreveld and Overmars, Chapter 3

How do we decide what to connect them to?

- Perform line sweep from top to bottom
- When we find split vertex v_i, connect it to a vertex above us…
- Which vertex?

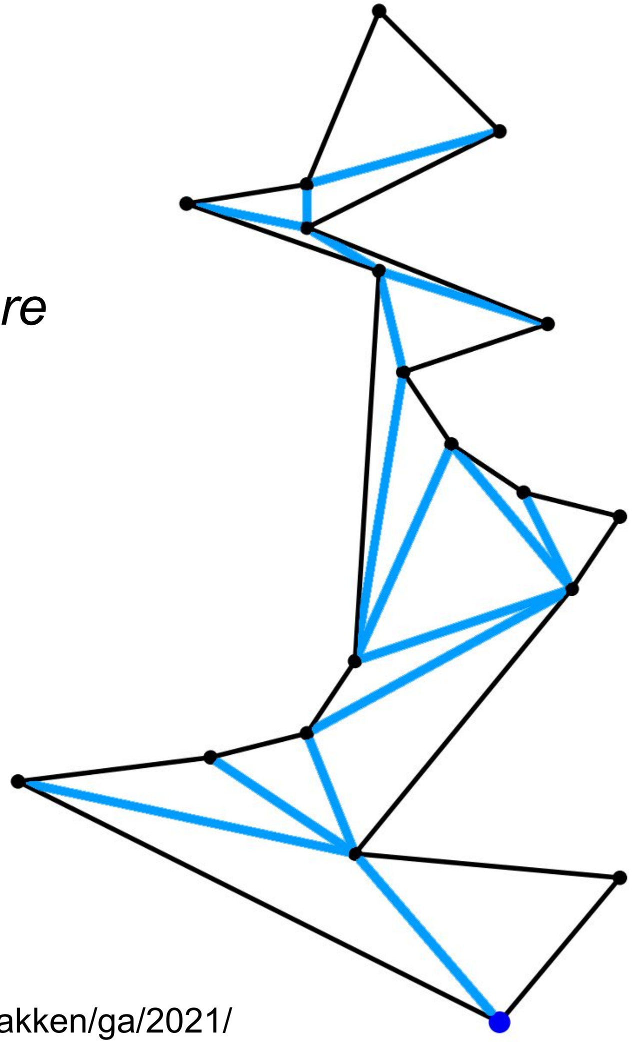
- Find **line to left, e_j, and to right, e_k,** of v_i on the current sweep line.
- Locate the lowest point between these two lines (a merge vertex)
- If none, take the upper end point of edge e_j or edge e_k



Computational Geometry Algorithms and Applications, de Berg, Cheong, van Kreveld and Overmars, Chapter 3

Triangulate a Monotone Polygon

- Sort all of the points vertically
- Push top two points onto a *stack data structure*
- Process the remaining points,
  one at a time, from top to bottom
- If you can…
  - make a triangle with the new point
    and the last two points on the stack
  - & remove 1 point
  - & repeat
- If not, push the new point on the stack

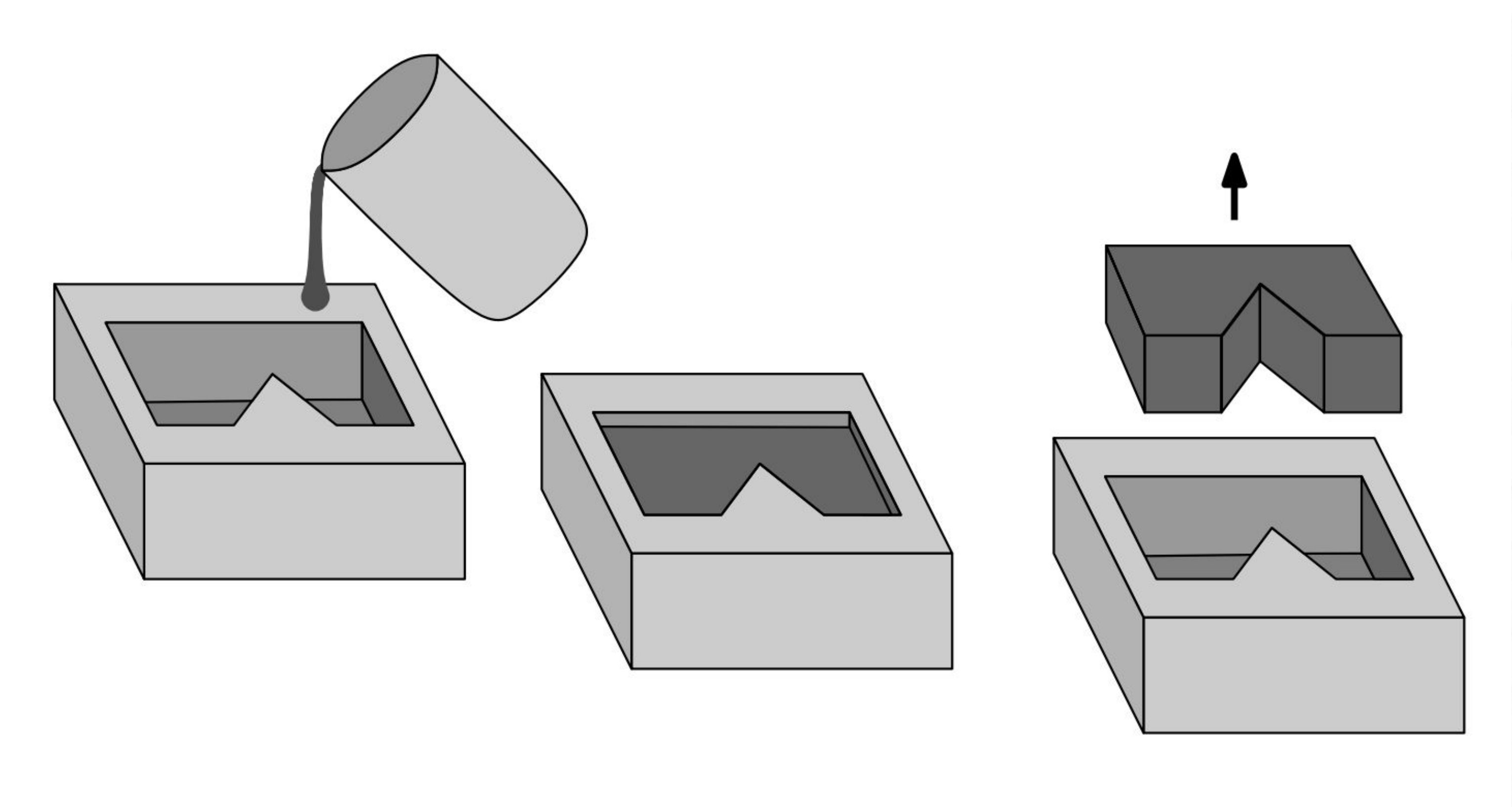


Frank Staals, http://www.cs.uu.nl/docs/vakken/ga/2021/

Outline for Today

- Homework 2 Questions?
- Last Time: Monotone Polygons & Improved Triangulation Algorithm
- Motivation: Manufacturing by Mold Casting
- Dual Representation: Planar Constraints
- Half-Plane / Half-Space Intersection
- Incremental Linear Programming
- Related Application: Japanese Wood Joints
- Related Application: Automatic Robotic Part Sorting
- Next Time: Point Location

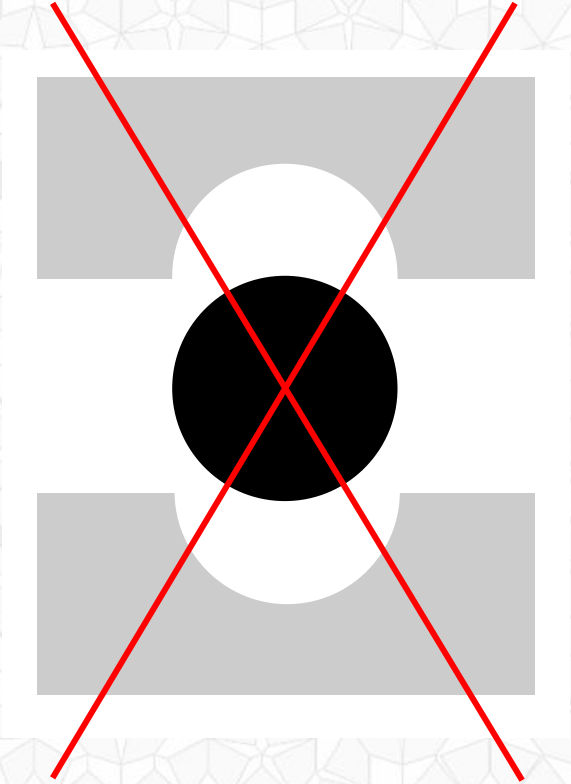
Motivation: Manufacturing by Mold Casting



Computational Geometry Algorithms and Applications, de Berg, Cheong, van Kreveld and Overmars, Chapter 4

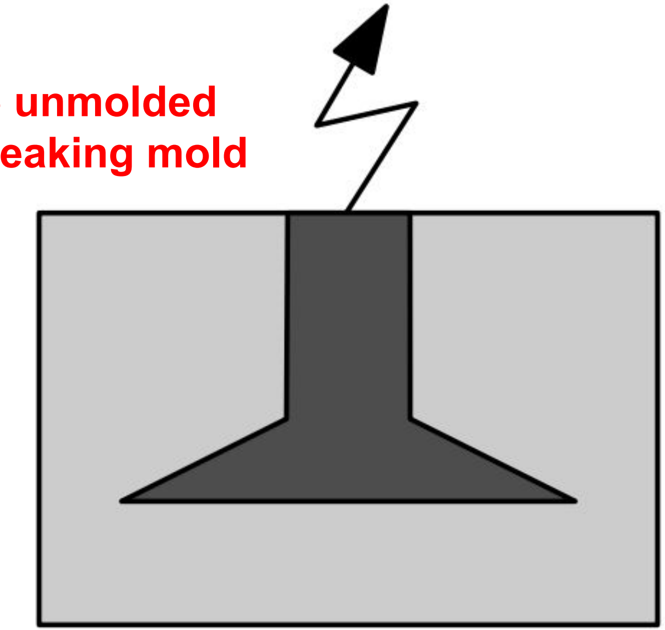
“Rules” for the Mold Casting Problem

- Single piece mold
- Cannot break mold
- Rigid mold
    - *not flexible, e.g., silicone*
- Polyhedral objects
    - *no curved surfaces*
- Must remove object using translation only, no rotation
    - *cannot mold a screw*

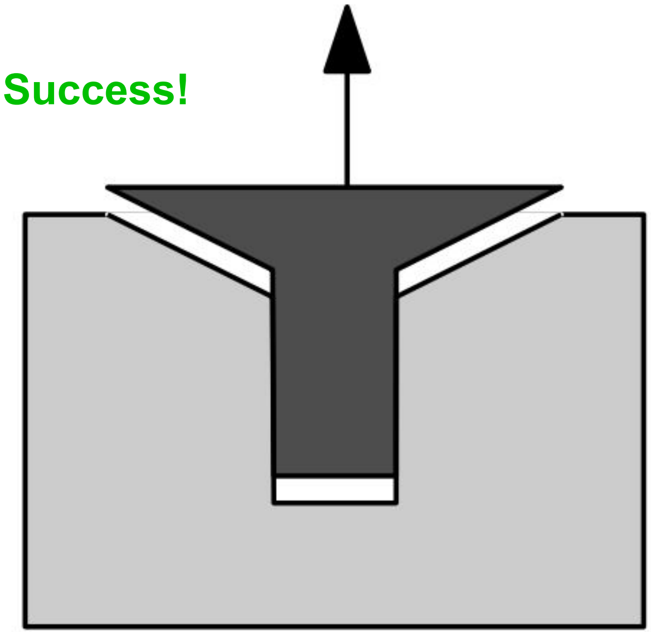


Motivation: Manufacturing by Mold Casting

Failure!
Cannot be unmolded
without breaking mold



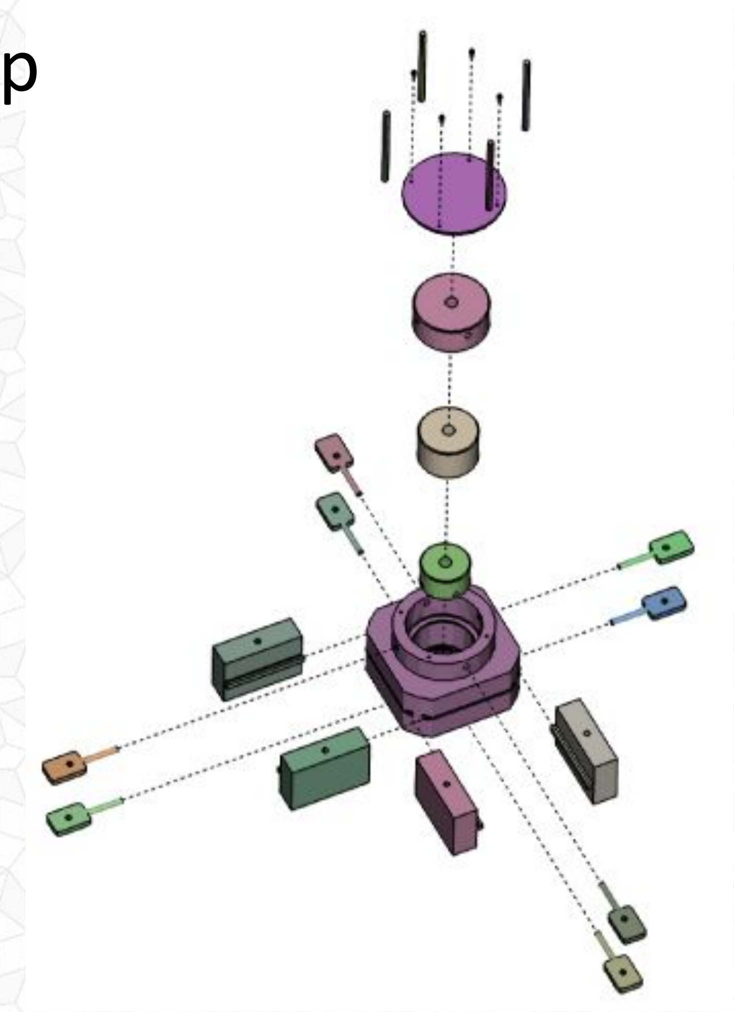
Success!



Computational Geometry Algorithms and Applications,
de Berg, Cheong, van Kreveld and Overmars, Chapter 4

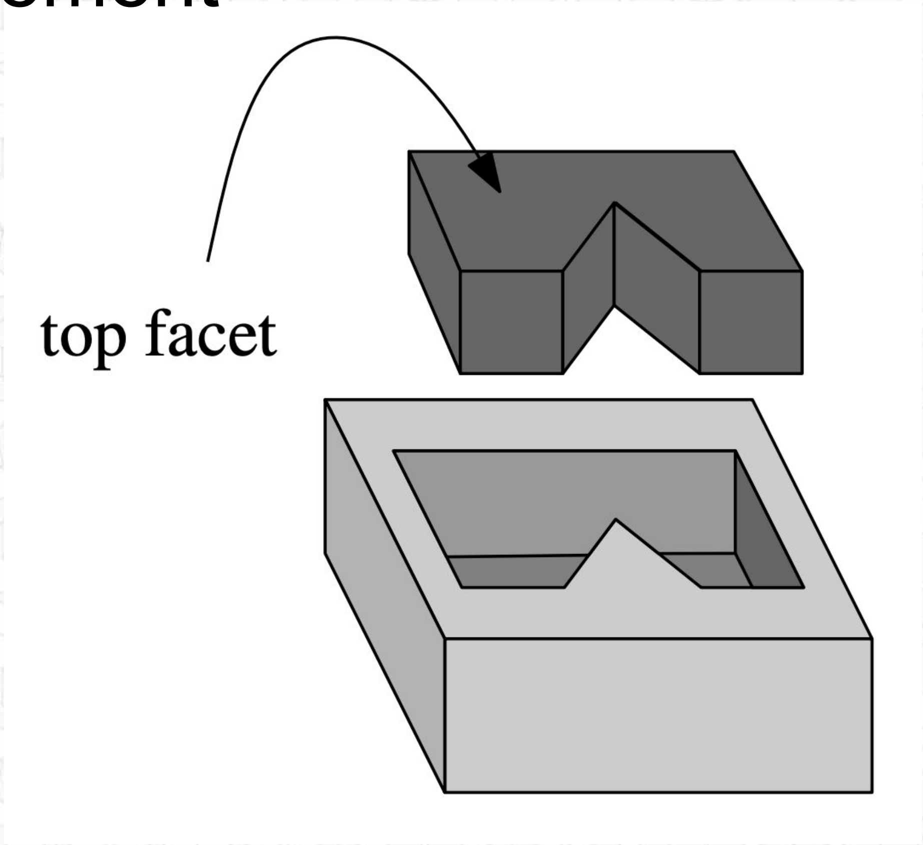
“Designing Effective Step-by-step Assembly Instructions” Agrawala et al., SIGGRAPH 2003

- Inspired by robotics planning research
- Need to solve planning & presentation simultaneously for best result



“Castable” Problem Statement

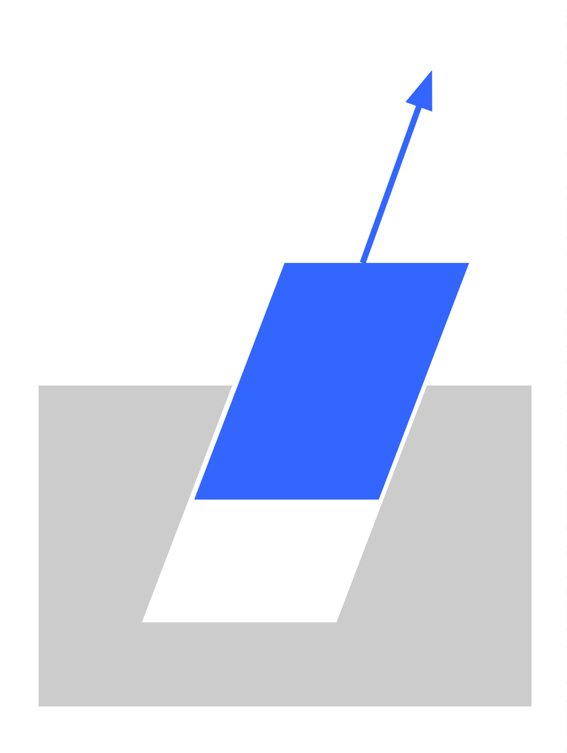
- Given a polyhedron with polygonal facets, can it be cast from a single mold?
- What is the shape of the mold?
  - How is the part oriented in the mold?
  - Which is the top facet?
- What direction is the object translated to remove it from the mold?



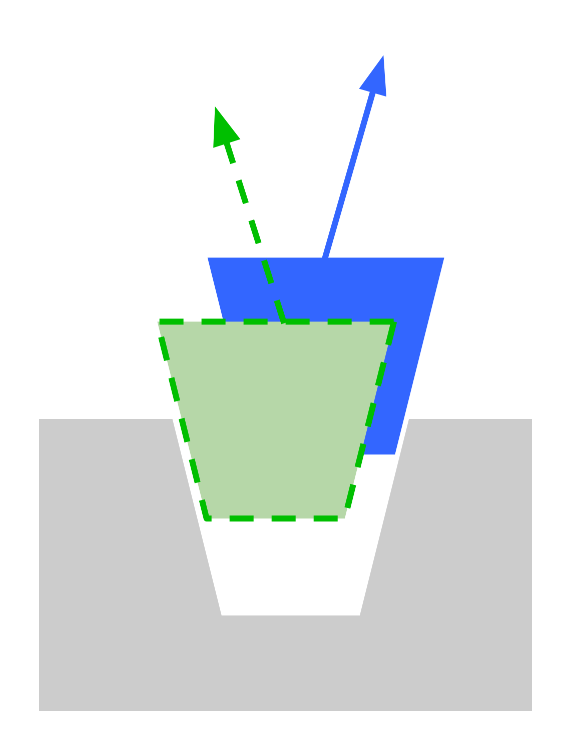
Computational Geometry Algorithms and Applications,
de Berg, Cheong, van Kreveld and Overmars, Chapter 4

Problem Statement

- The translation direction is not necessarily perpendicular to the top facet of the mold!



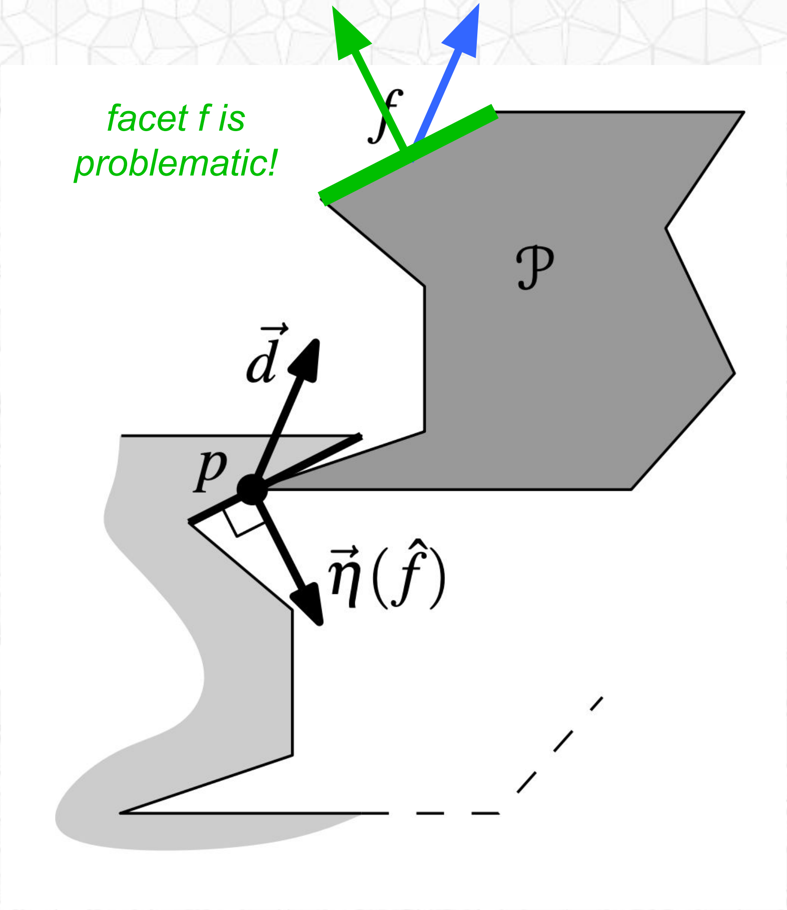
- The translation direction may not be unique *– there may be multiple answers!*



Outline for Today

- Homework 2 Questions?
- Last Time: Monotone Polygons & Improved Triangulation Algorithm
- Motivation: Manufacturing by Mold Casting
- Dual Representation: Planar Constraints
- Half-Plane / Half-Space Intersection
- Incremental Linear Programming
- Related Application: Japanese Wood Joints
- Related Application: Automatic Robotic Part Sorting
- Next Time: Point Location

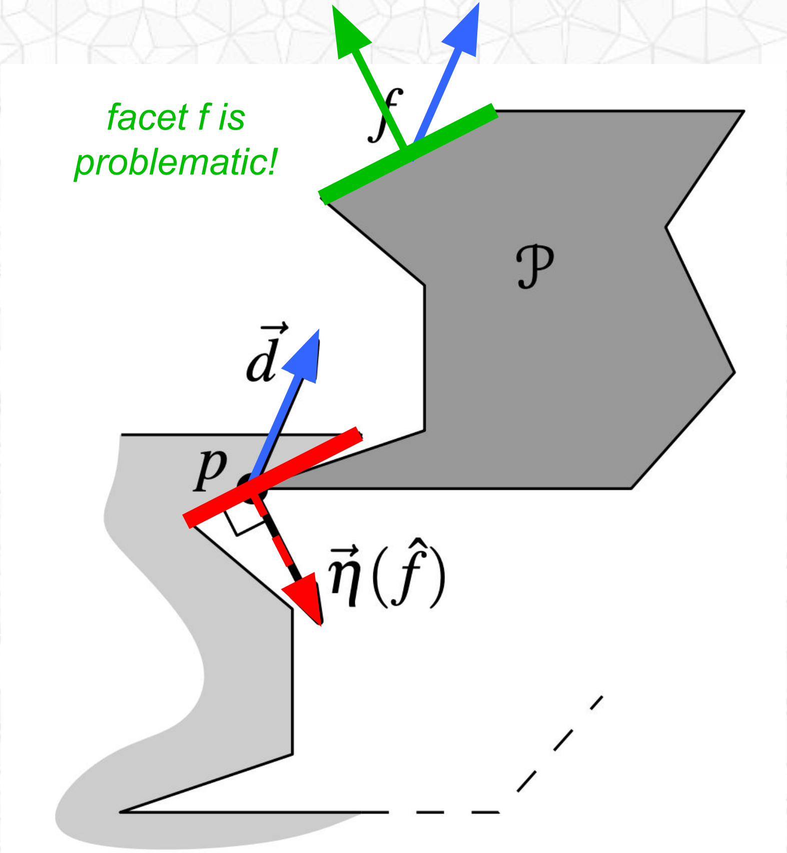
Lemma 4.1 The polyhedron *P* can be removed from its mold by a translation in direction *d* if and only if *d* makes an angle of at least 90° with the outward normal of all ordinary facets of *P*.



Computational Geometry Algorithms and Applications, de Berg, Cheong, van Kreveld and Overmars, Chapter 4

Lemma 4.1 The polyhedron *P* can be removed from its mold by a translation in direction *d* if and only if *d* makes an angle of at least 90° with the outward normal of all ordinary facets of *P*.

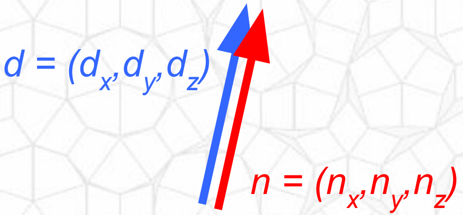
If the piece collides with mold facet $\hat{f}$ it must have angle > 90°, which would imply an angle < 90° with the corresponding piece facet *f*



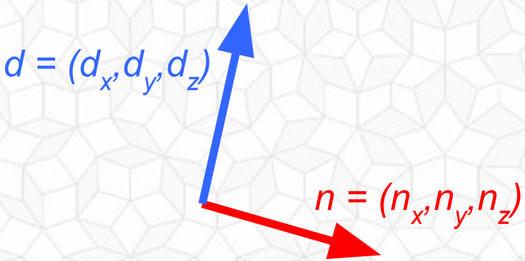
Computational Geometry Algorithms and Applications,
de Berg, Cheong, van Kreveld and Overmars, Chapter 4

Definition: Dot Product

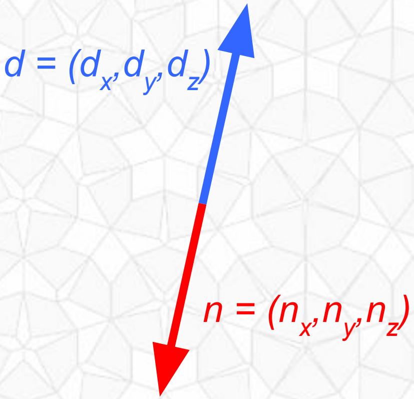
- A unit vector has length = 1: $sqrt(n_x^2 + n_y^2 + n_z^2) = 1$
- The dot product of two unit vectors is: $d_x*n_x + d_y*n_y + d_z*n_z$



Dot product = 1
When d and n are parallel
in the same direction



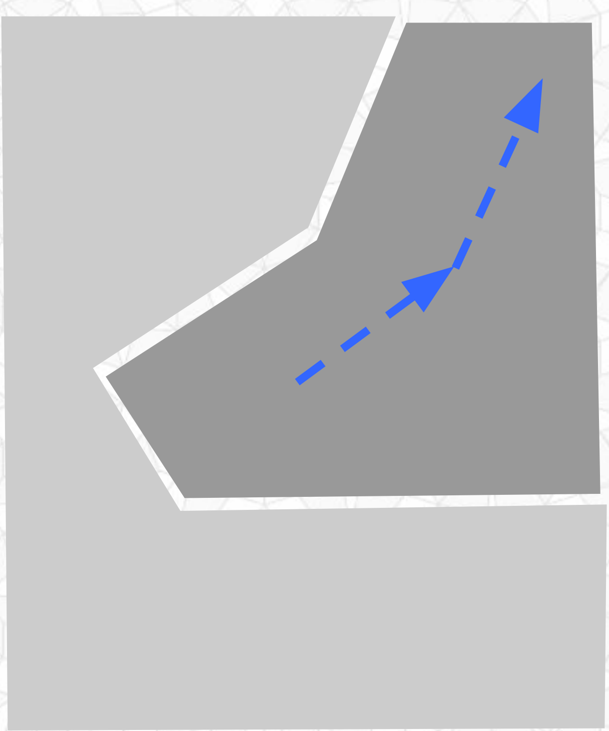
Dot product = 0
When d and n are
perpendicular (90°)



Dot product = -1
When d and n are parallel in
the opposite directions

Lemma 4.1 The polyhedron *P* can be removed from its mold by a translation in direction *d*
if and only if *d* makes an angle of at least 90° with the outward normal of all ordinary facets of *P*.

Note: It will NOT be necessarily to *change direction* during unmolding. If the object can be removed from the mold, a single direction is sufficient.



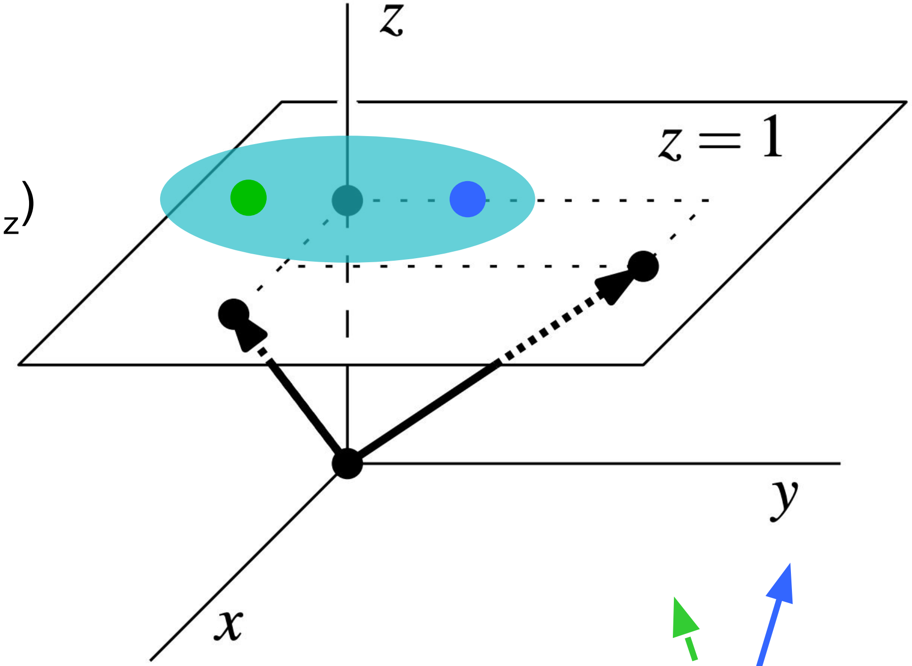
Computational Geometry Algorithms and Applications, de Berg, Cheong, van Kreveld and Overmars, Chapter 4

“Dual” Representation

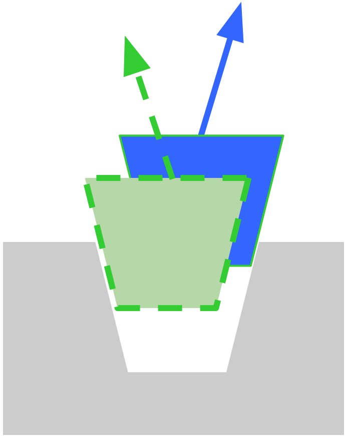
- Every upwards direction d = (d_x,d_y,d_z) can be represented as a point on the z=1 plane: d = $(d_x,d_y,1)$

- Not a unit vector, that’s ok
- *Convert our 3D problem to 2D*

- All valid solutions to the unmolding problem form **a region on the plane.**



Computational Geometry Algorithms and Applications, de Berg, Cheong, van Kreveld and Overmars, Chapter 4



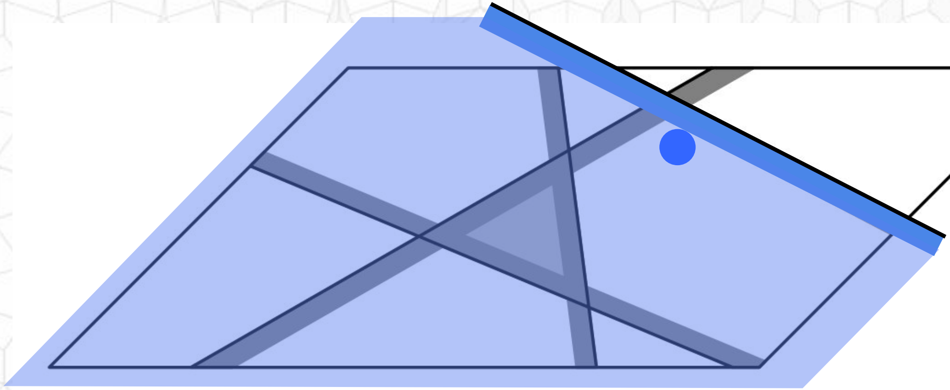
Outline for Today

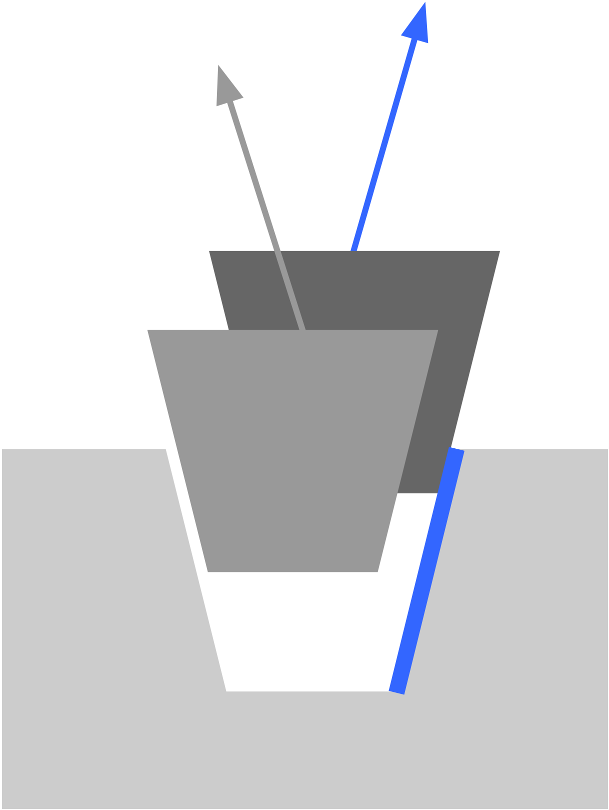
- Homework 2 Questions?
- Last Time: Monotone Polygons & Improved Triangulation Algorithm
- Motivation: Manufacturing by Mold Casting
- Dual Representation: Planar Constraints
- Half-Plane / Half-Space Intersection
- Incremental Linear Programming
- Related Application: Japanese Wood Joints
- Related Application: Automatic Robotic Part Sorting
- Next Time: Point Location

- Each facet places a *linear constraint* on the valid unmolding directions

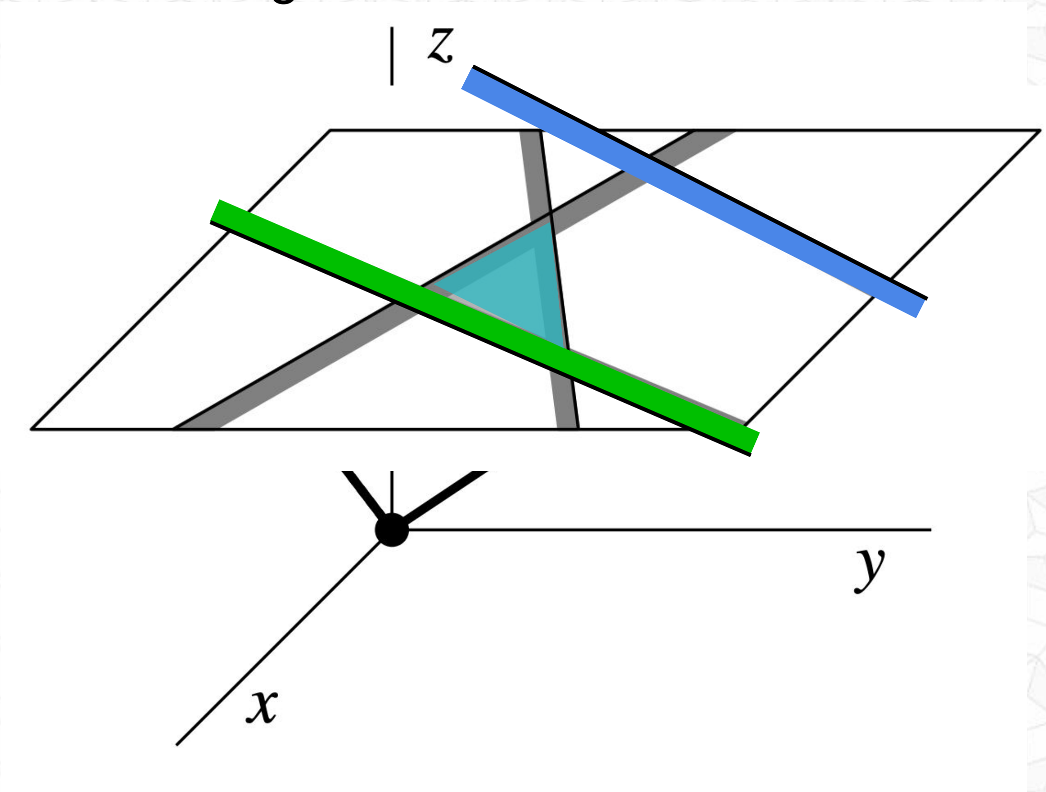
$$n_x d_x + n_y d_y + n_z \leq 0$$

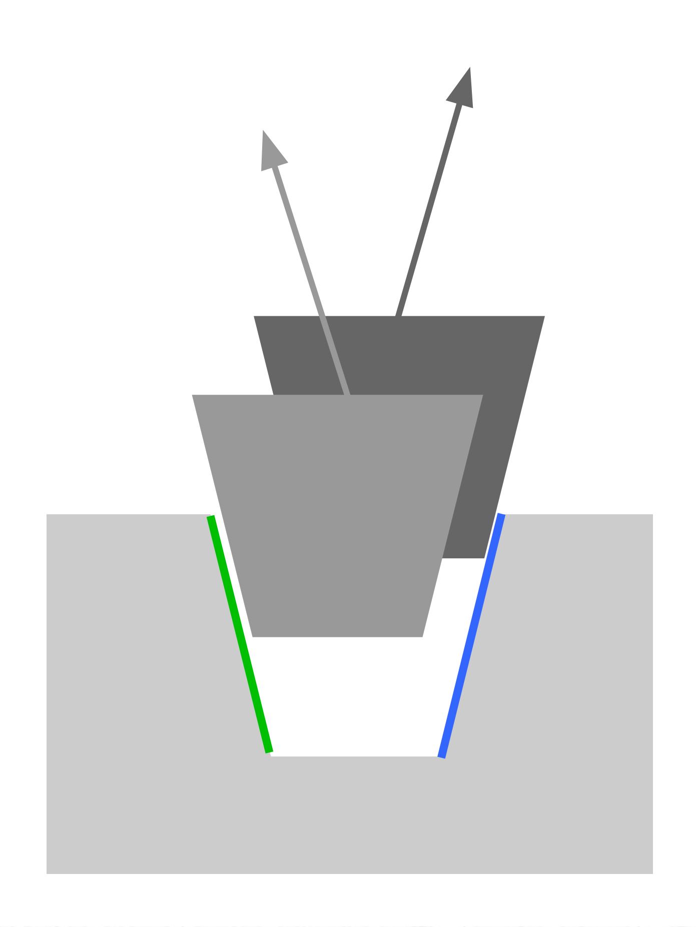
- This half-plane / half-space space can be visualized on our dual representation z=1



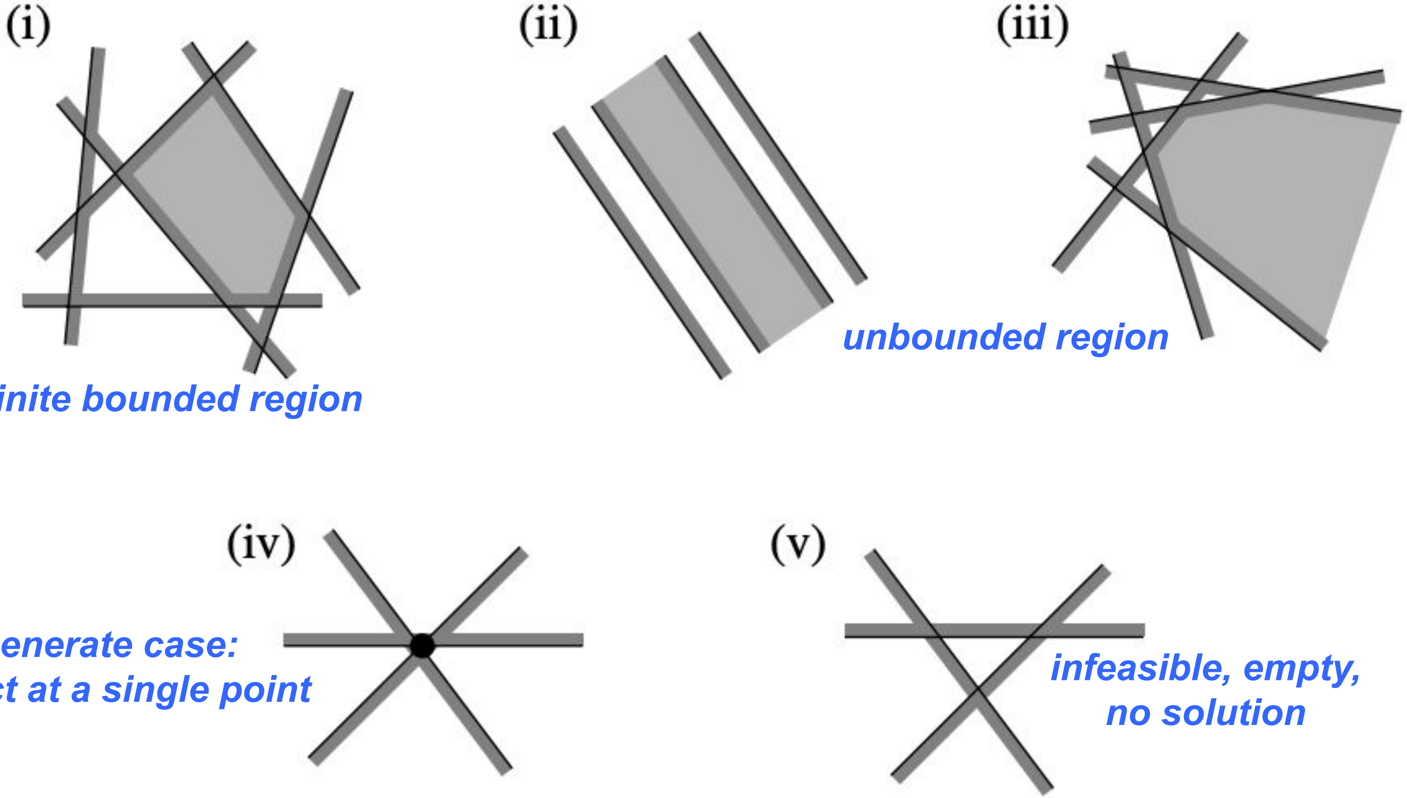


- That region is the intersection of the linear constraints from each face of the piece.
- That region is convex!





Half Space Intersection



finite bounded region

unbounded region

degenerate case: intersect at a single point

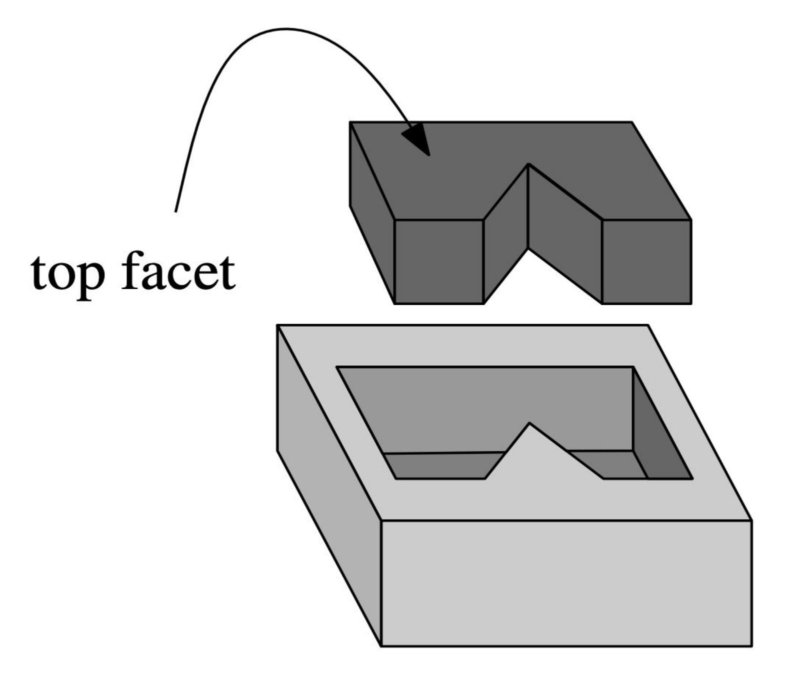
infeasible, empty, no solution

Computational Geometry Algorithms and Applications, de Berg, Cheong, van Kreveld and Overmars, Chapter 4

Is it Castable? Algorithm

- Given an input polyhedron with *n* facets
- Try each facet as the "top" facet

- Intersect the half-spaces of all other facets
- If it is non-empty, we have a solution!

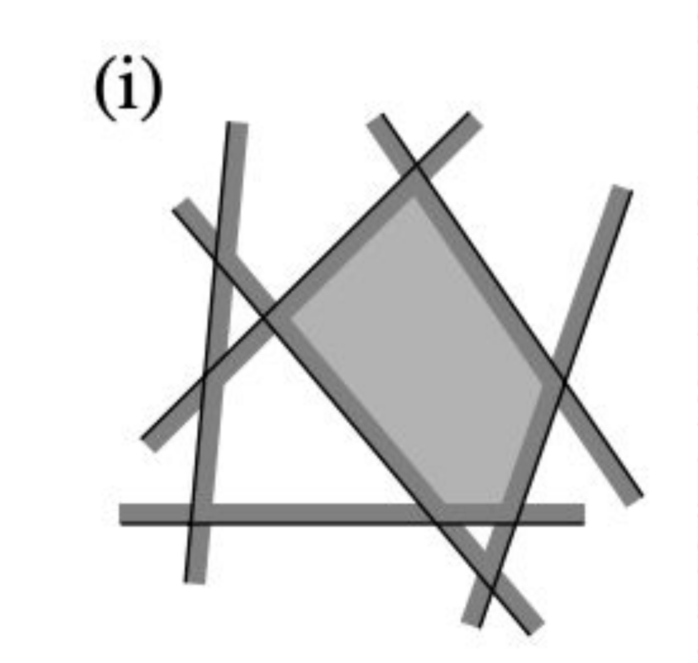


Computational Geometry Algorithms and Applications,
de Berg, Cheong, van Kreveld and Overmars, Chapter 4

Compute Halfspace Intersection

- Given *n* linear constraints (*n* halfspaces)
- Intersection will be a convex region in the z=1 plane with at most n edges

- Let's compute intersection via Divide & Conquer:
    - Split half spaces into two groups
    - Compute intersection
    - Merge intersections

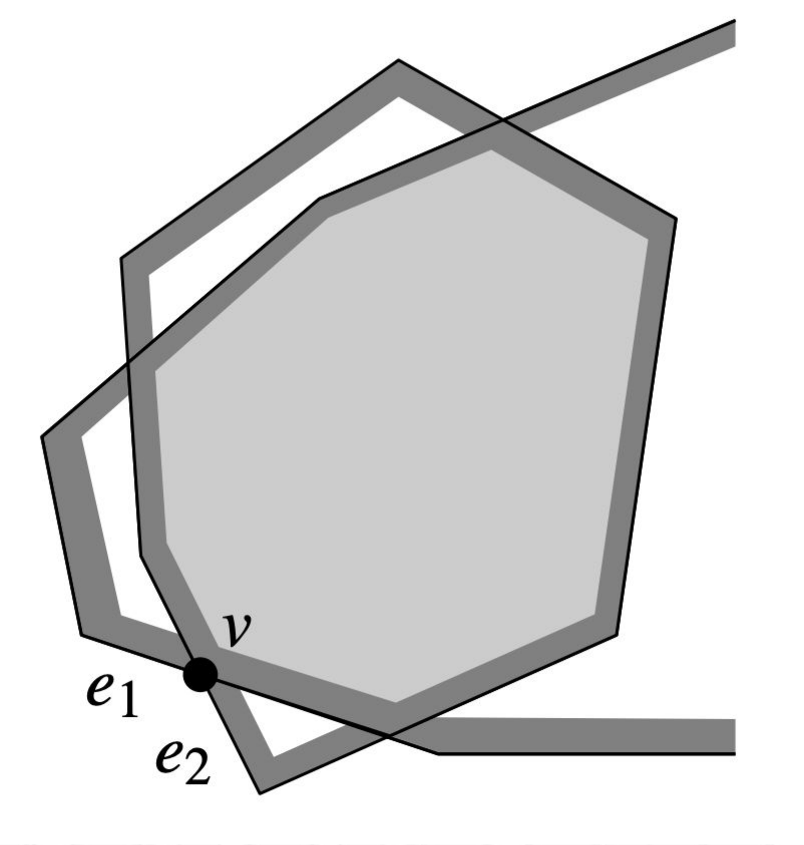


Computational Geometry Algorithms and Applications,
de Berg, Cheong, van Kreveld and Overmars, Chapter 4

Merge Two Convex Regions

- From previous lecture, we can compute the intersection/overlay general (non-convex) polygonal shapes in *O(n log n + k log n)*
  - *k* is the complexity, # of faces on output polygon
  - In this case *k* ≤ *n*

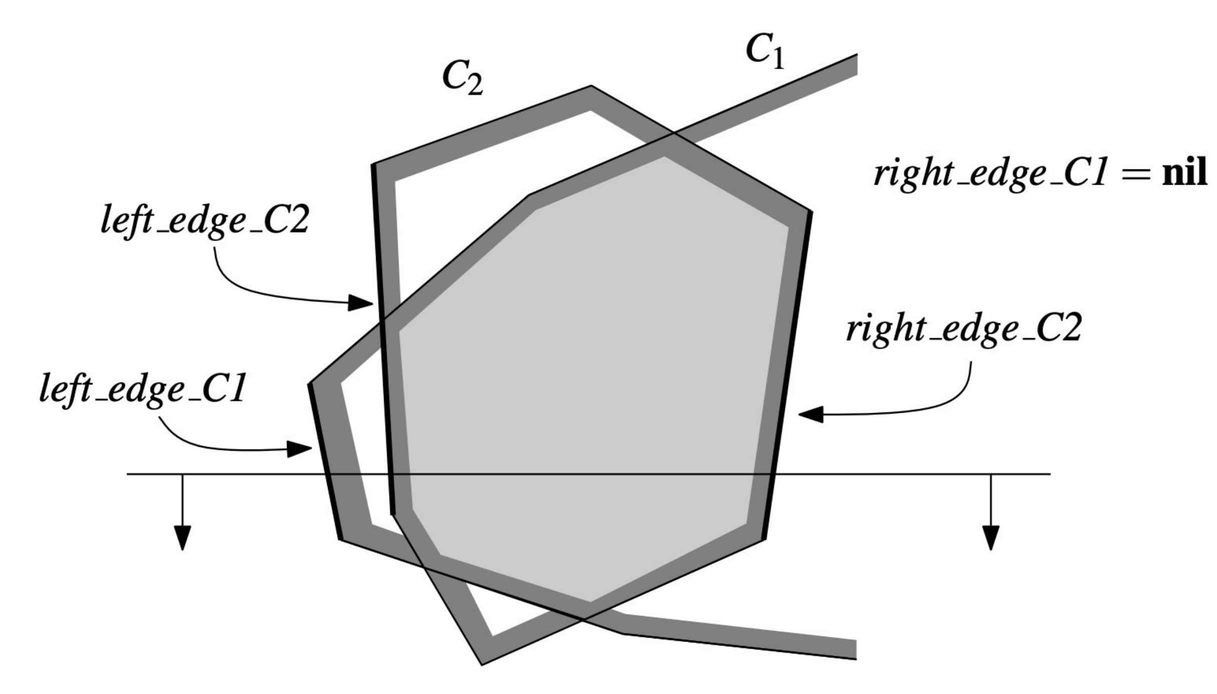
- Potential Complication? The shapes may be *unbounded*



Computational Geometry Algorithms and Applications, de Berg, Cheong, van Kreveld and Overmars, Chapter 4

Plane Sweep to Compute Overlay

- Worst case sweep line horizontal complexity is constant, not *n*
- Track left & right faces of each shape C_1 & C_2
- We can handle unbounded by setting one or more of these edges to *NULL*



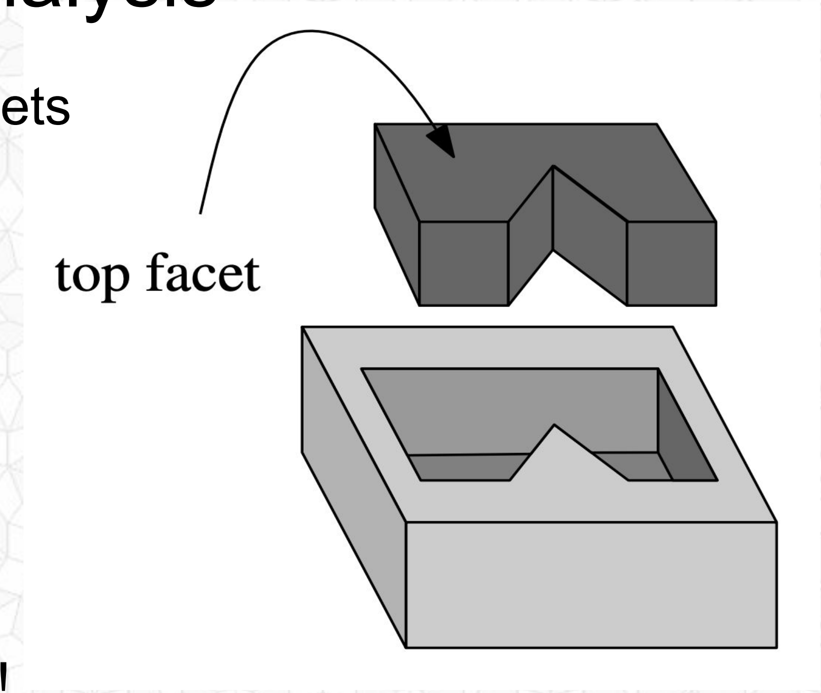
Computational Geometry Algorithms and Applications,
de Berg, Cheong, van Kreveld and Overmars, Chapter 4

Is it Castable? Algorithm Analysis

- Given an input polyhedron with *n* facets
- Try each facet as the "top" facet

- Intersect the half-spaces of all other facets
  - Merge 2 convex regions
  - Divide & Conquer Recursion

- If it is non-empty, we have a solution!
- Overall:



Computational Geometry Algorithms and Applications, de Berg, Cheong, van Kreveld and Overmars, Chapter 4

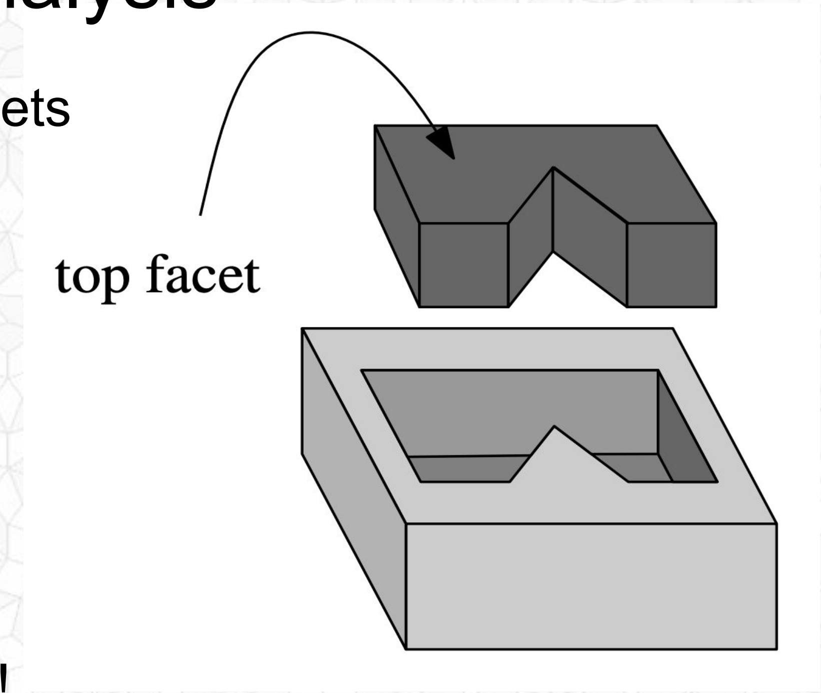
Is it Castable? Algorithm Analysis

- Given an input polyhedron with *n* facets
- Try each facet as the “top” facet

  → O(n)
- Intersect the half-spaces of all other facets
  - Merge 2 convex regions

    → O(n)
  - Divide & Conquer Recursion

    → O(n log n)
- If it is non-empty, we have a solution!
- Overall:



Computational Geometry Algorithms and Applications, de Berg, Cheong, van Kreveld and Overmars, Chapter 4

Is it Castable? Algorithm Analysis

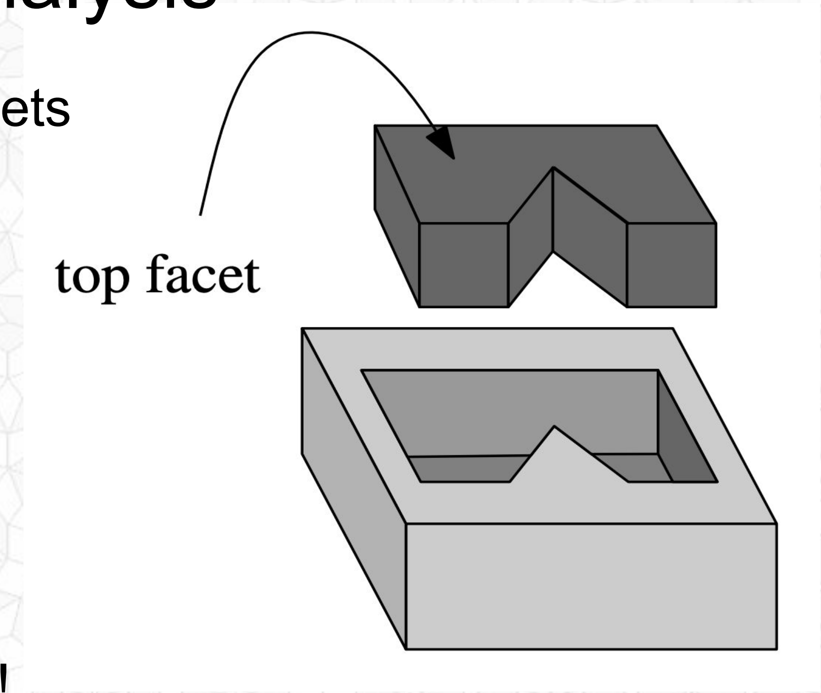
- Given an input polyhedron with *n* facets
- Try each facet as the “top” facet

  → O(n)
- Intersect the half-spaces of all other facets
  - Merge 2 convex regions

    → O(n)
  - Divide & Conquer Recursion

    → O(n log n)
- If it is non-empty, we have a solution!
- Overall: → $O(n^2 \log n)$

  Can we do better?



Computational Geometry Algorithms and Applications, de Berg, Cheong, van Kreveld and Overmars, Chapter 4

Is it Castable? Algorithm Analysis

- Given an input polyhedron with *n* facets
- Try each facet as the “top” facet

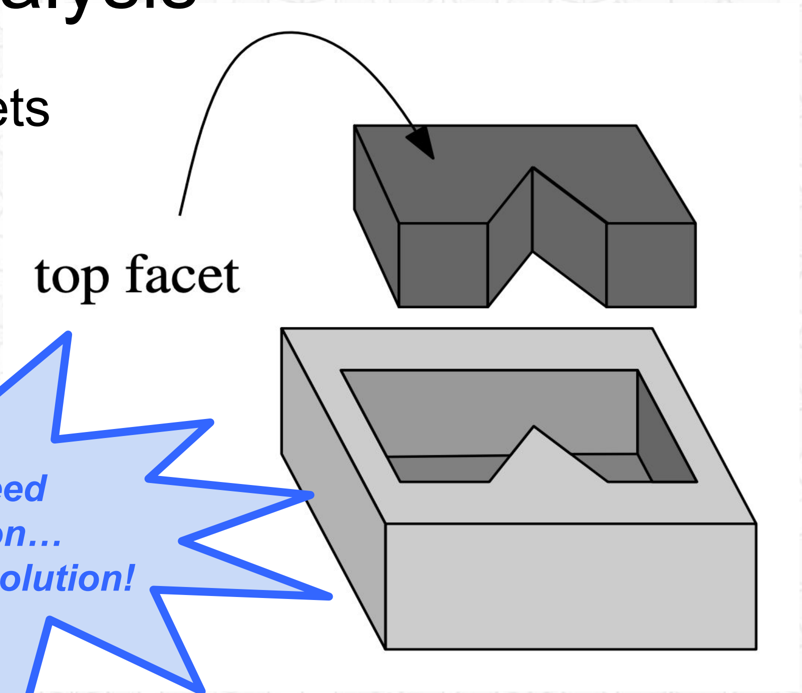
  → O(n)
- Intersect the half-spaces of all other facets
  - Merge 2 convex regions

    → O(n)
  - Divide & Conquer

    → O(n log n)
- If it is non-empty, we have a solution.
- Overall: → $O(n^2 \log n)$

Can we do better?

We don’t need every solution… we only need 1 solution!

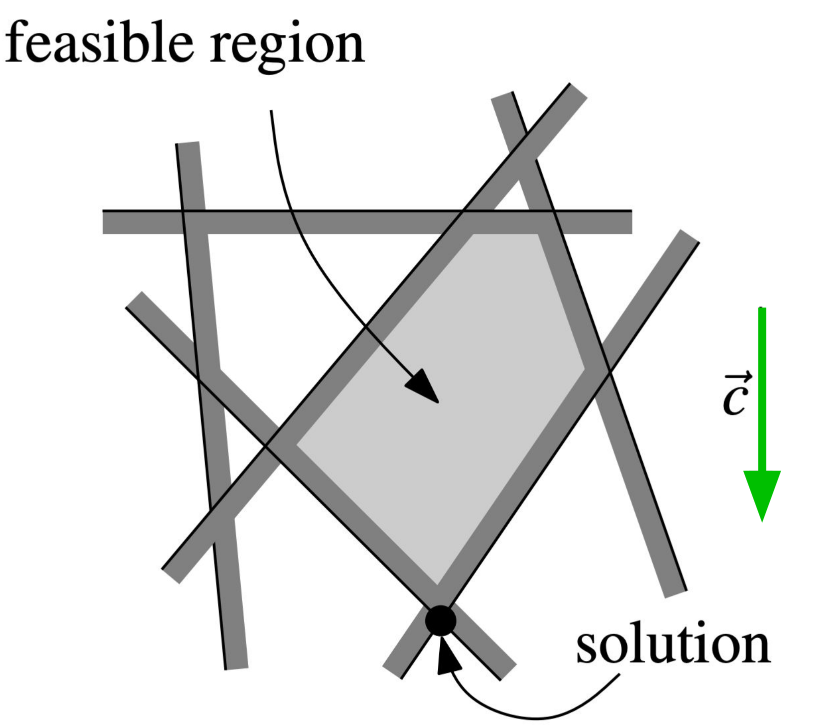


Computational Geometry Algorithms and Applications, de Berg, Cheong, van Kreveld and Overmars, Chapter 4

Outline for Today

- Homework 2 Questions?
- Last Time: Monotone Polygons & Improved Triangulation Algorithm
- Motivation: Manufacturing by Mold Casting
- Dual Representation: Planar Constraints
- Half-Plane / Half-Space Intersection
- Incremental Linear Programming
- Related Application: Japanese Wood Joints
- Related Application: Automatic Robotic Part Sorting
- Next Time: Point Location

Linear Optimization, a.k.a. Linear Programming



objective function

$$\text{Maximize} \quad c_1x_1 + c_2x_2 + \cdots + c_dx_d$$

$$\text{Subject to} \quad \left.\begin{array}{rcl} a_{1,1}x_1 + \cdots + a_{1,d}x_d & \leqslant & b_1 \\ a_{2,1}x_1 + \cdots + a_{2,d}x_d & \leqslant & b_2 \\ & \vdots & \\ a_{n,1}x_1 + \cdots + a_{n,d}x_d & \leqslant & b_n \end{array}\right\}$$

constraints

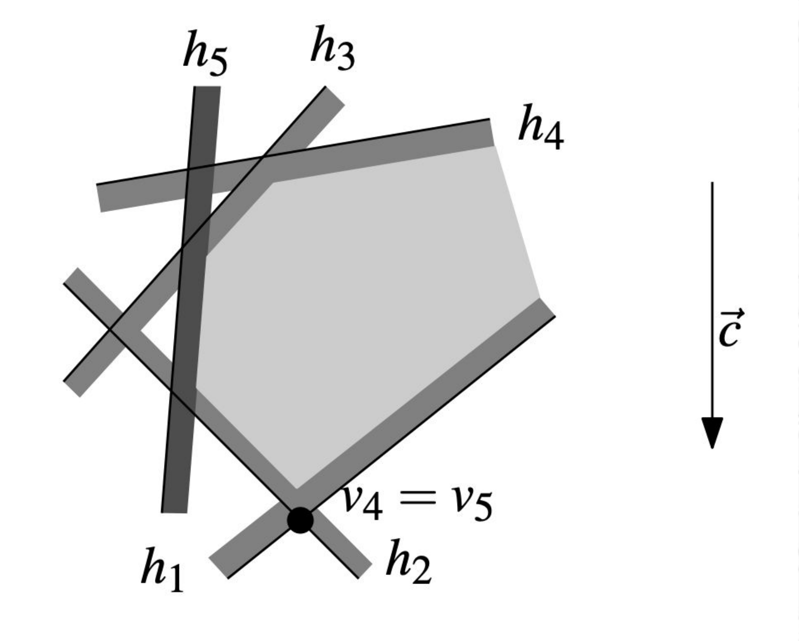
Computational Geometry Algorithms and Applications,
de Berg, Cheong, van Kreveld and Overmars, Chapter 4

Linear Programming - Incremental Solution

- Order the half-space constraints in some order: $h_1, h_2, h_3, \ldots h_n$
- We will solve incremental versions of the problem: $C_1, C_2, C_3, \ldots C_n$
- Which have optimal solutions:

  $v_1, v_2, v_3, \ldots v_n$

- C_i has with half-space constraints $\{ h_1, h_2, h_3, \ldots h_i \}$ with solution v_i

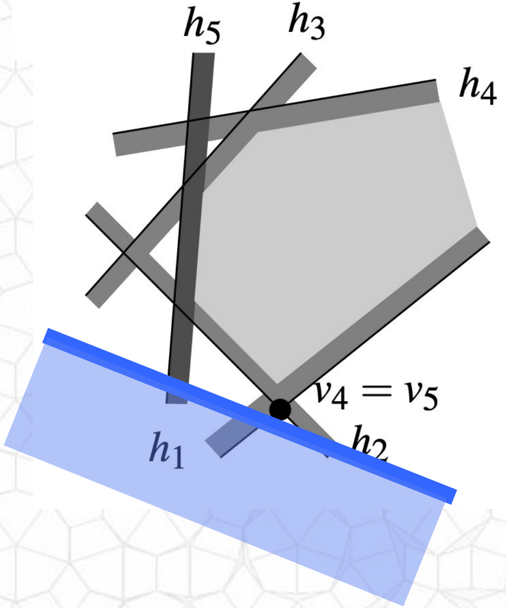


Computational Geometry Algorithms and Applications,
de Berg, Cheong, van Kreveld and Overmars, Chapter 4

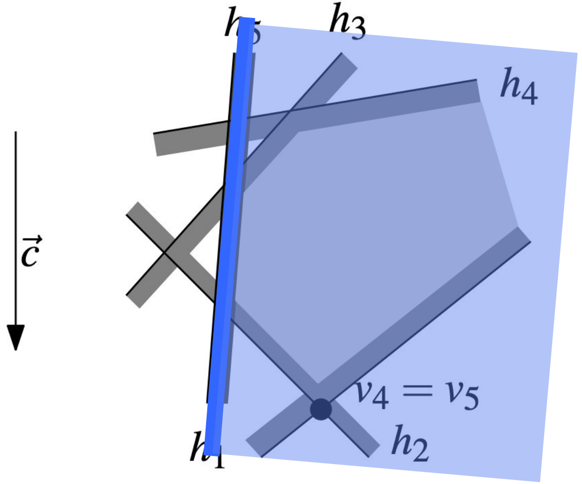
Linear Programming - Incremental Solution

- At each step, we will add in the next halfspace constraint h_{i+1}

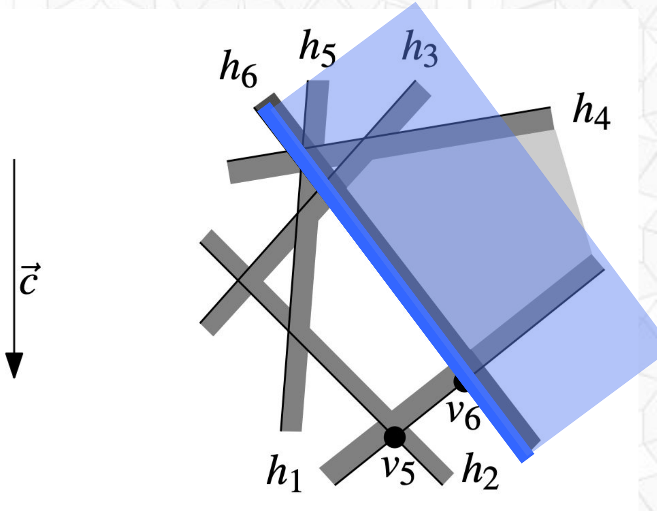
Infeasible - no solution



Satisfied: $v_1 = v_{i+1}$



Satisfied: compute new v_{i+1}



Computational Geometry Algorithms and Applications,
de Berg, Cheong, van Kreveld and Overmars, Chapter 4

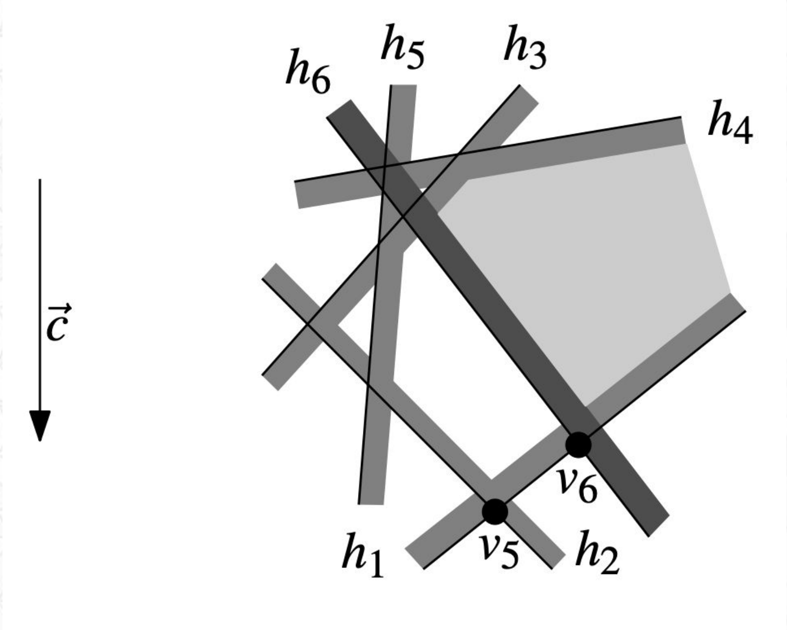
Computing New Solution v_{i+1}

- It must lie on the constraint h_{i+1}
- Must intersect with all previous halfspaces

- *Note: We are not computing or storing the feasible region, only the solution point v_i*

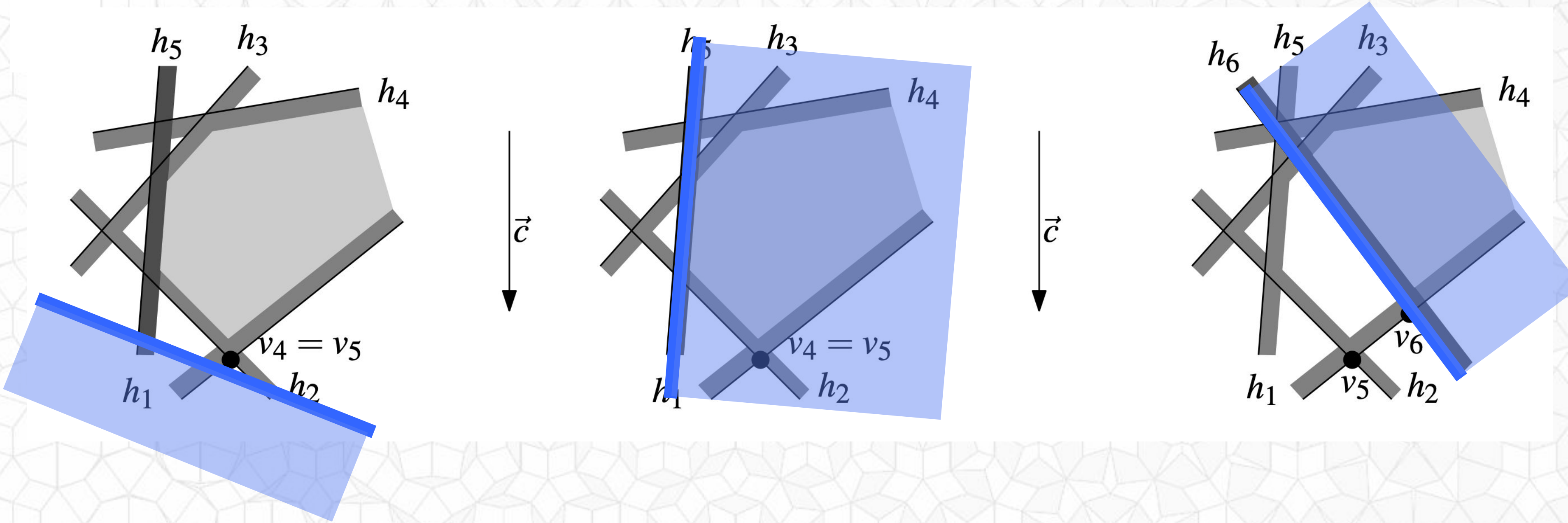
- *What is the running time?*

Satisfied: compute new v_{i+1}



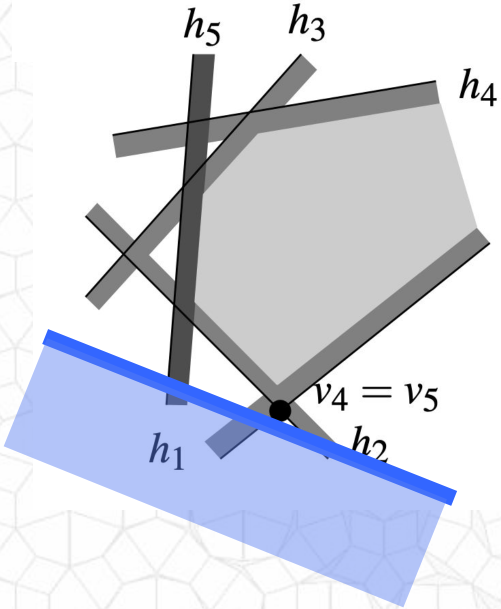
Incremental Solution - Analysis

Infeasible - no solution **Satisfied: $v_1 = v_{i+1}$** **Satisfied: compute new v_{i+1}**



Incremental Solution - Analysis

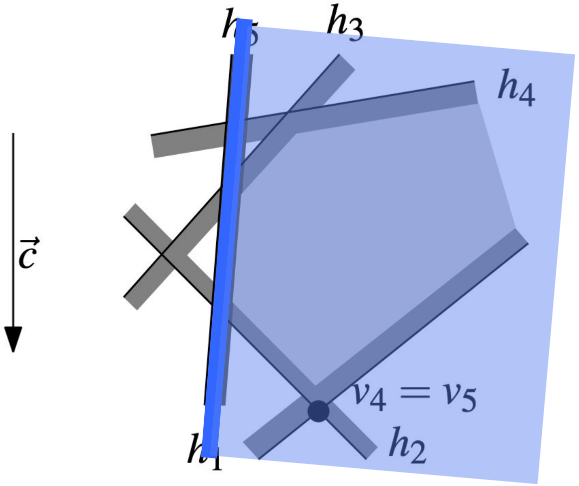
Infeasible - no solution



→ ***O(1)***

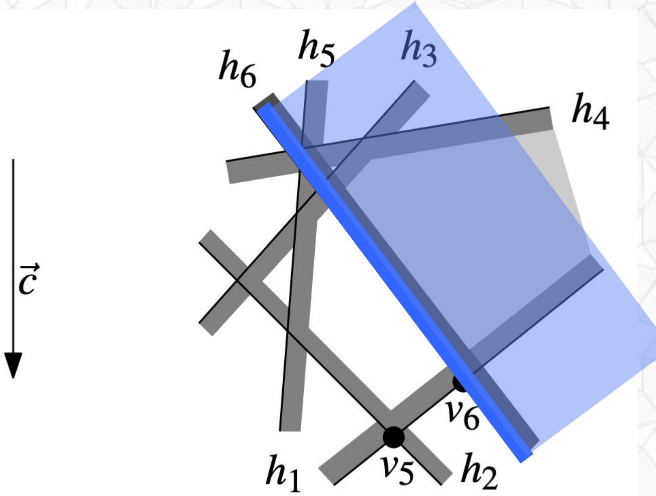
short circuit exit!

Satisfied: $v_1 = v_{i+1}$



→ ***O(1)***

Satisfied: compute new v_{i+1}



→ ***O(n)***

Incremental Solution - Analysis

- Order the half-space constraints in some order: $h_1, h_2, h_3, \ldots h_n$
- We will solve incremental versions of the problem: $C_1, C_2, C_3, \ldots C_n$

→

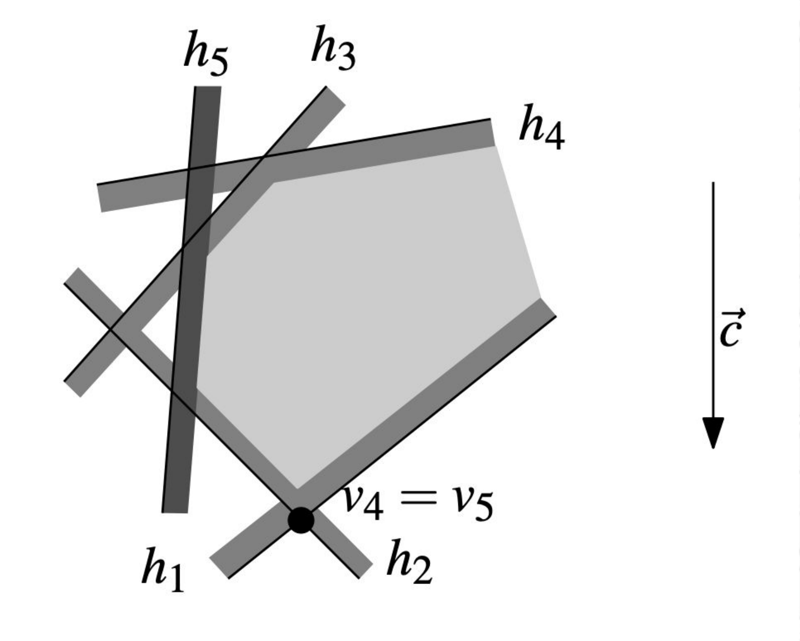
- Which have optimal solutions:

  $v_1, v_2, v_3, \ldots v_n$

- C_i has with half-space constraints $\{ h_1, h_2, h_3, \ldots h_i \}$ with solution v_i

Overall:

→



Incremental Solution - Analysis

- Order the half-space constraints in some order: $h_1, h_2, h_3, \ldots h_n$
- We will solve incremental versions of the problem: $C_1, C_2, C_3, \ldots C_n$

→ ***O(n)***

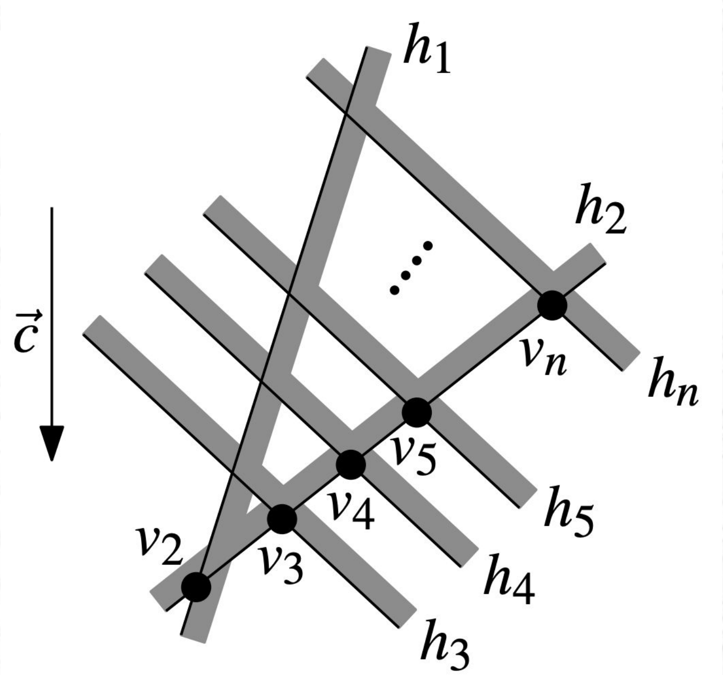
- Which have optimal solutions:

$v_1, v_2, v_3, \ldots v_n$

- C_i has with half-space constraints $\{ h_1, h_2, h_3, \ldots h_i \}$ with solution v_i

Overall:

→ O(n^2) worst case



Computational Geometry Algorithms and Applications, de Berg, Cheong, van Kreveld and Overmars, Chapter 4

Incremental Solution - Analysis

- Order the half-space constraints in some order: $h_1, h_2, h_3, \ldots h_n$
- We will solve incremental versions of the problem: $C_1, C_2, C_3, \ldots C_n$

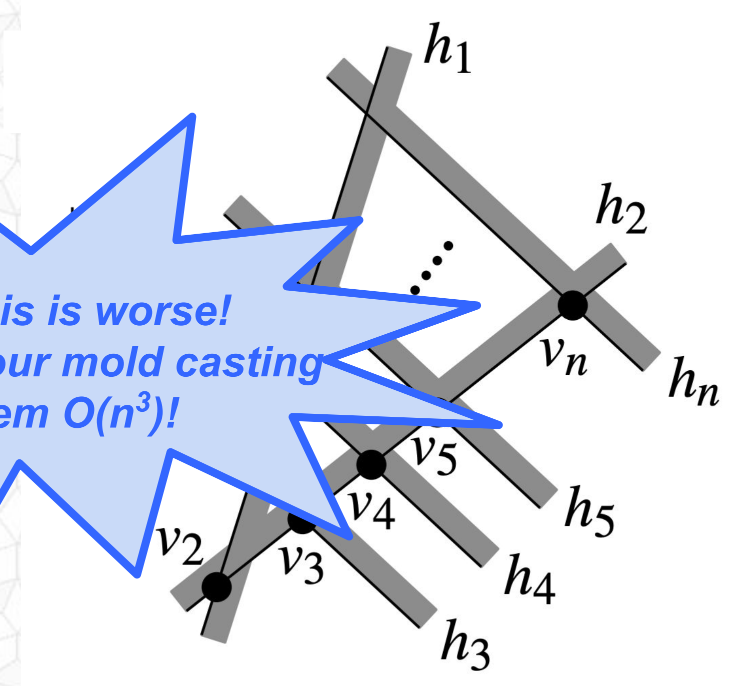
→ ***O(n)***

- Which have optimal solutions:

$v_1, v_2, v_3, \ldots v_n$

- C_i has with half-space c
{ $h_1, h_2, h_3, \ldots h_i$ } with so

Overall:
→ ***O(n^2) worst case***



Ach! This is worse!
This makes our mold casting problem O(n^3)!

Randomized Linear Programming

randomize the order of the halfspaces

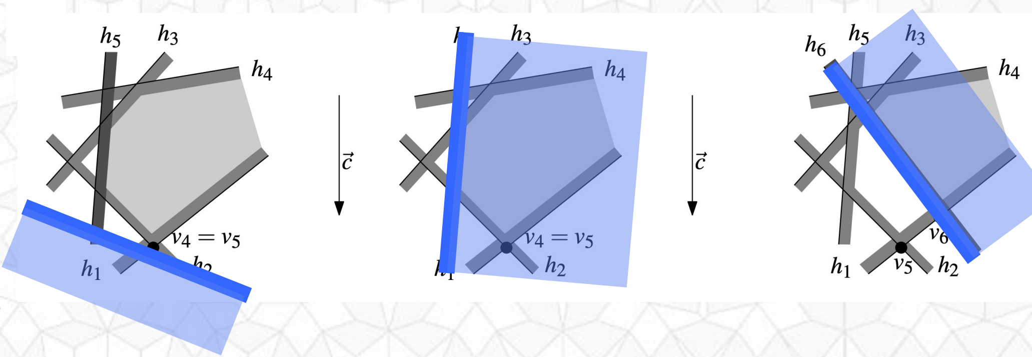
- Order the half-space constraints in some order: $h_1, h_2, h_3, \ldots h_n$
- We will solve incremental versions of the problem: $C_1, C_2, C_3, \ldots C_n$

→ ***O(n)***

- Which have optimal solutions:

  $v_1, v_2, v_3, \ldots v_n$

- C_i has with half-space constraints $\{ h_1, h_2, h_3, \ldots h_i \}$ with solution v_i



→ ***O(1)***

short circuit exit!

→ ***O(1)***

→ ***O(n)***

Can be shown that the case to recompute the solution is rare…

Overall:

→ O(n) expected case

Randomized Linear Programming

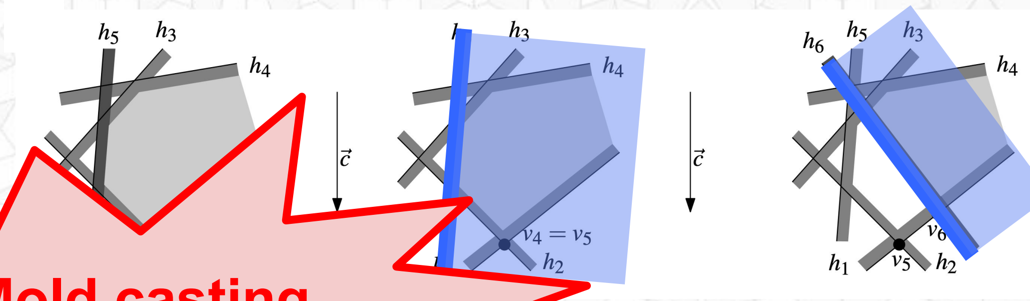
randomize the order of the halfspaces

- Order the half-space constraints in some order: $h_1, h_2, h_3, \ldots h_n$
- We will solve incremental versions of the problem: $C_1, C_2, C_3, \ldots C_n$

→ ***O(n)***

- Which have optimal solutions:

$v_1, v_2, v_3, \ldots v_n$



- C_i has with half-spa

$\{ h_1, h_2, h_3, \ldots h_i \}$ with so

Mold casting

→ $O(n^2)$!

→ ***O(1)***

→ ***O(n)***

exit.

Overall:

→ O(n) expected case

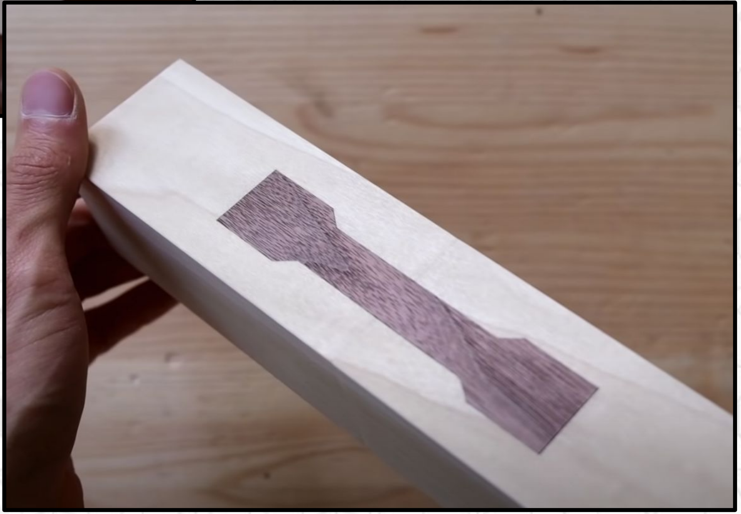
Can be shown that the case to recompute the solution is rare…

Outline for Today

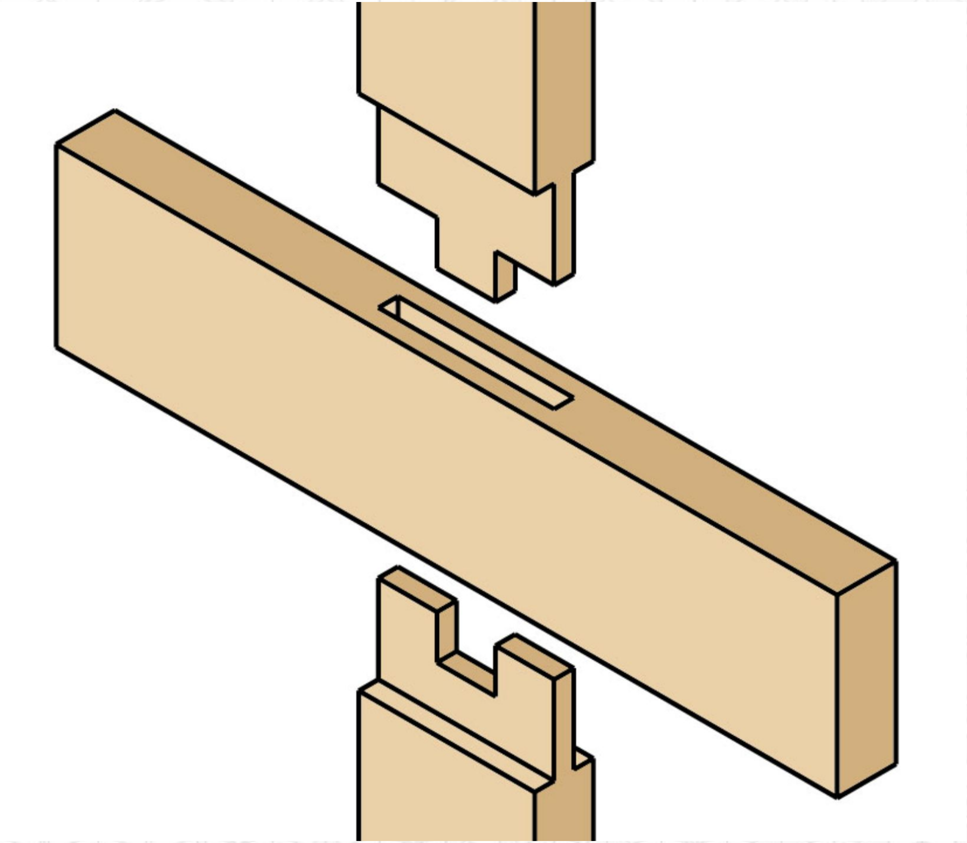
- Homework 2 Questions?
- Last Time: Monotone Polygons & Improved Triangulation Algorithm
- Motivation: Manufacturing by Mold Casting
- Dual Representation: Planar Constraints
- Half-Plane / Half-Space Intersection
- Incremental Linear Programming
- Related Application: Japanese Wood Joints
- Related Application: Automatic Robotic Part Sorting
- Next Time: Point Location

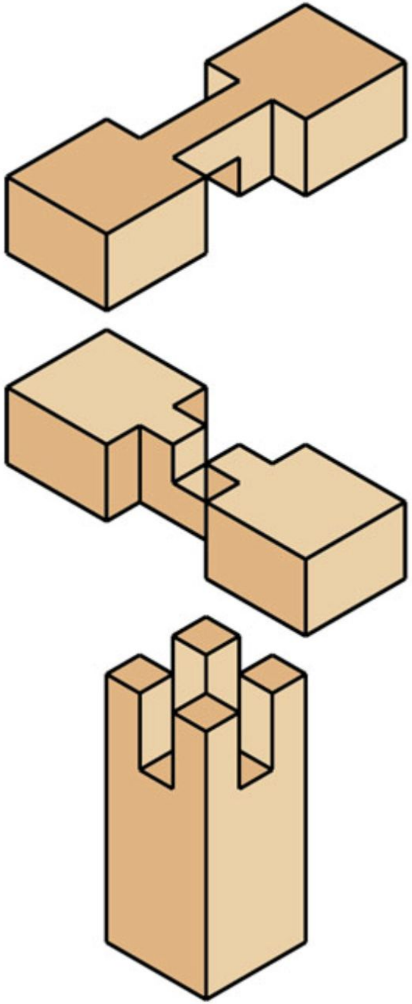
The Art of Traditional Japanese Wood Joinery

Dylan Iwakuni
https://www.youtube.com/watch?v=3KqIIOyuo1Q&t=17s



MECH DRAFTING Vasileios I. Koutsovoulos

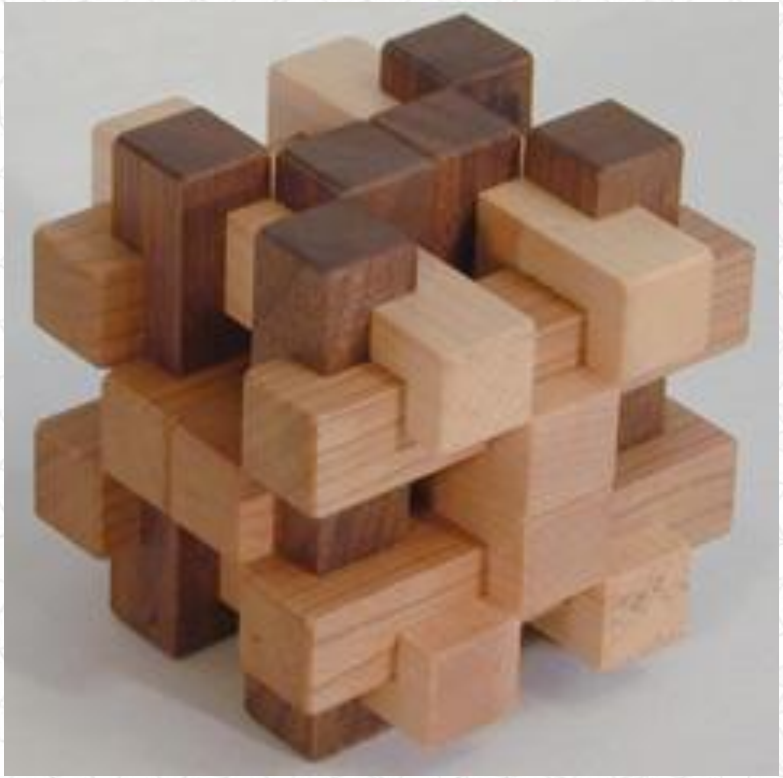




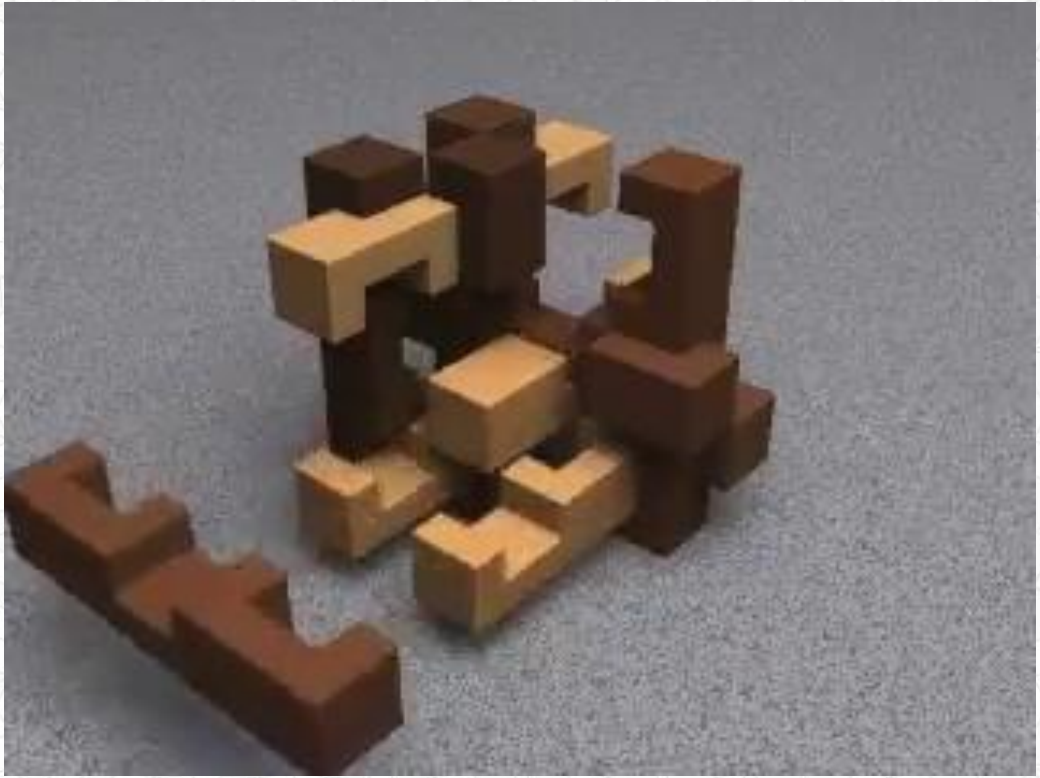
https://mechdrafting.net/en/portfolio-item/japanese-joinery

Justin Legakis ~1999

18 Piece Burr
Bill Cutler Puzzles



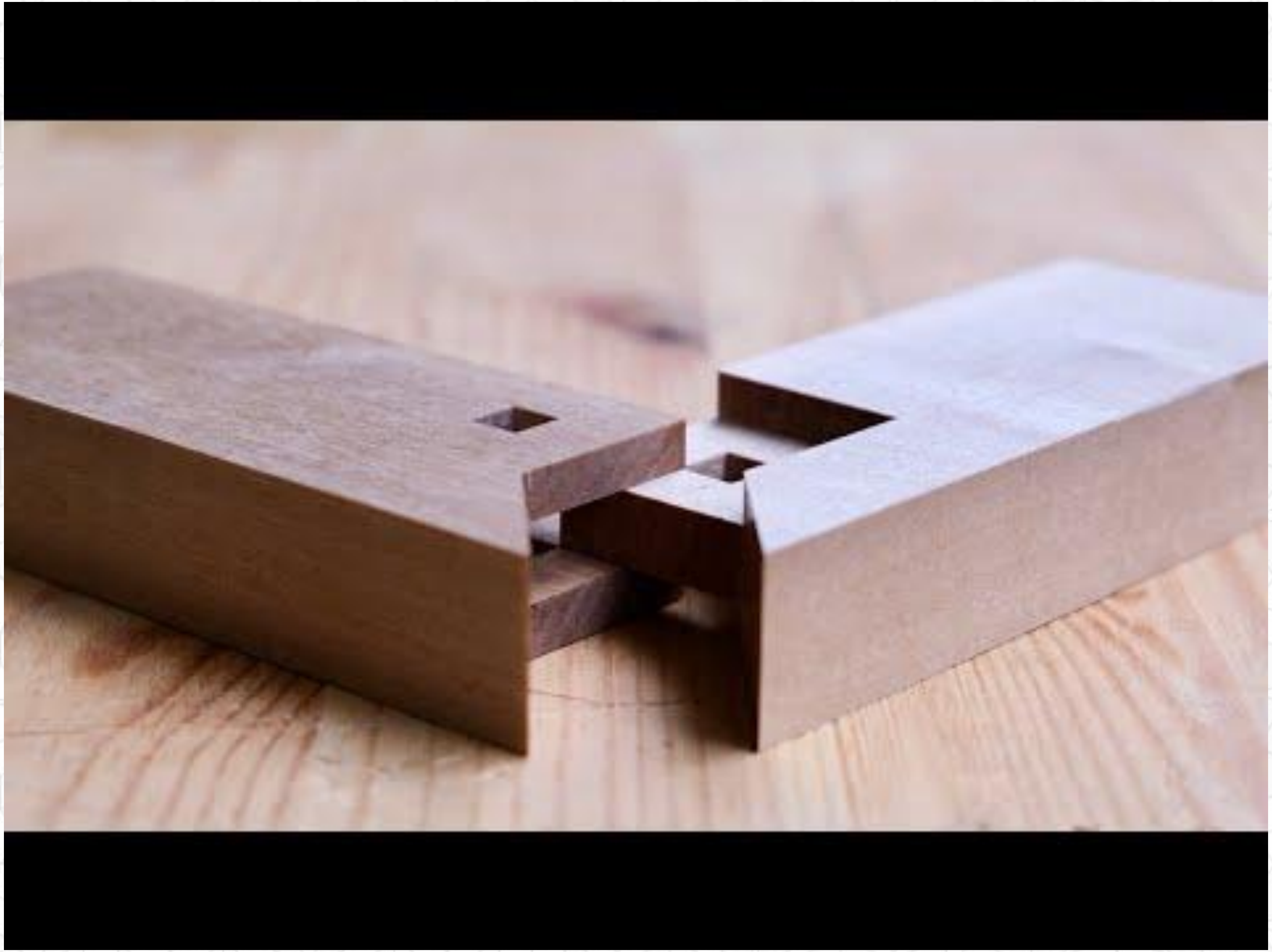
http://billcutlerpuzzles.com/stock/18piece.html



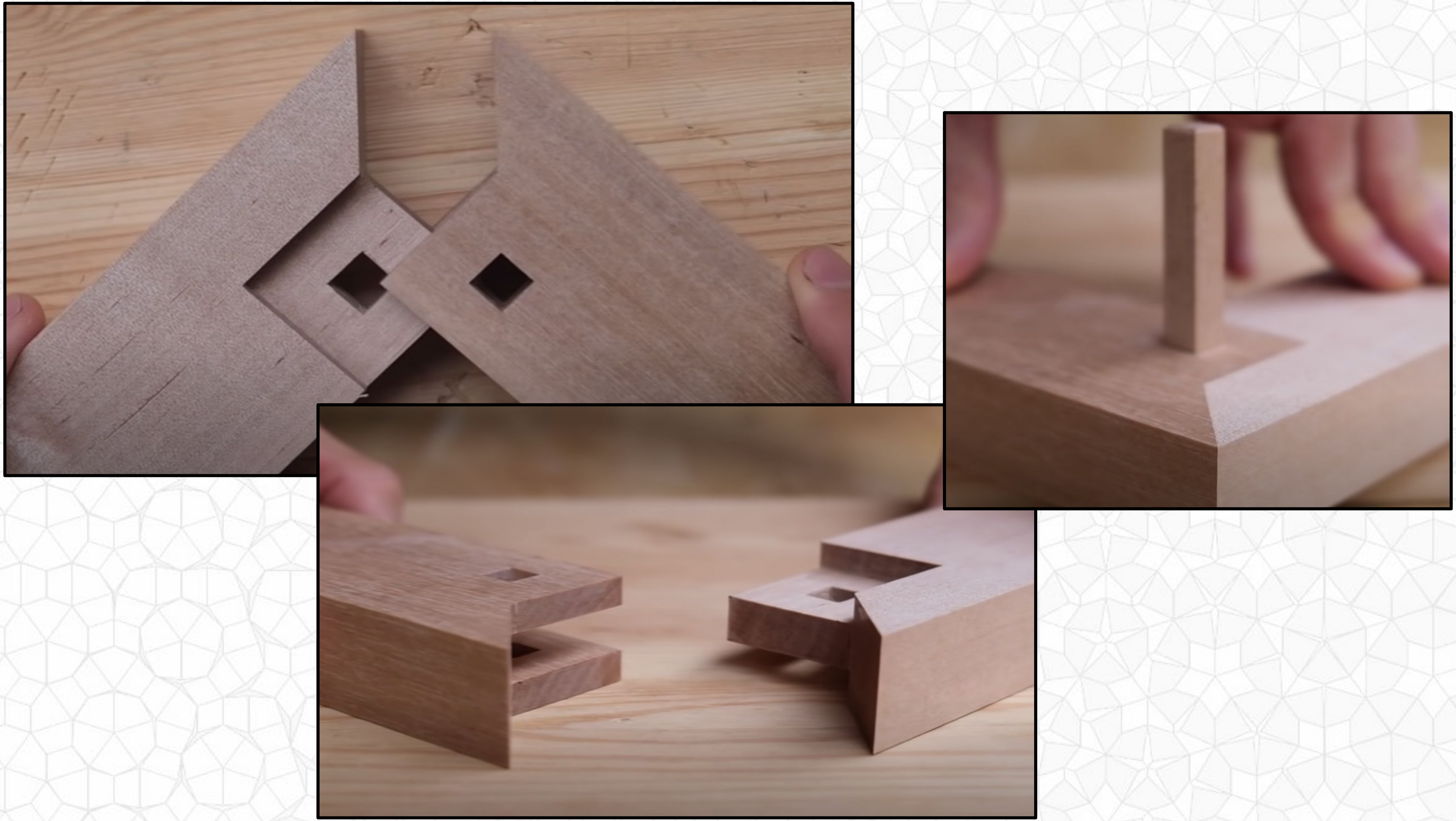
http://legakis.net/justin/gallery_burr.html

Japanese Joinery - Kane Tsugi

Dylan
Iwakuni



https://www.youtube.com/watch?v=P-ODWGUfBEM



Mysterious Japanese Joinery

Dylan Iwakuni

https://www.youtube.com/watch?v=GtdQoT7saz0

Outline for Today

- Homework 2 Questions?
- Last Time: Monotone Polygons & Improved Triangulation Algorithm
- Motivation: Manufacturing by Mold Casting
- Dual Representation: Planar Constraints
- Half-Plane / Half-Space Intersection
- Incremental Linear Programming
- Related Application: Japanese Wood Joints
- Related Application: Automatic Robotic Part Sorting
- Next Time: Point Location

Robotics: Automatic Part Sorting & Orienting

"Design of Part Feeding and Assembly Processes with Dynamics", Song,Trinkle, Kumar, & Pang, MEAM 2004.

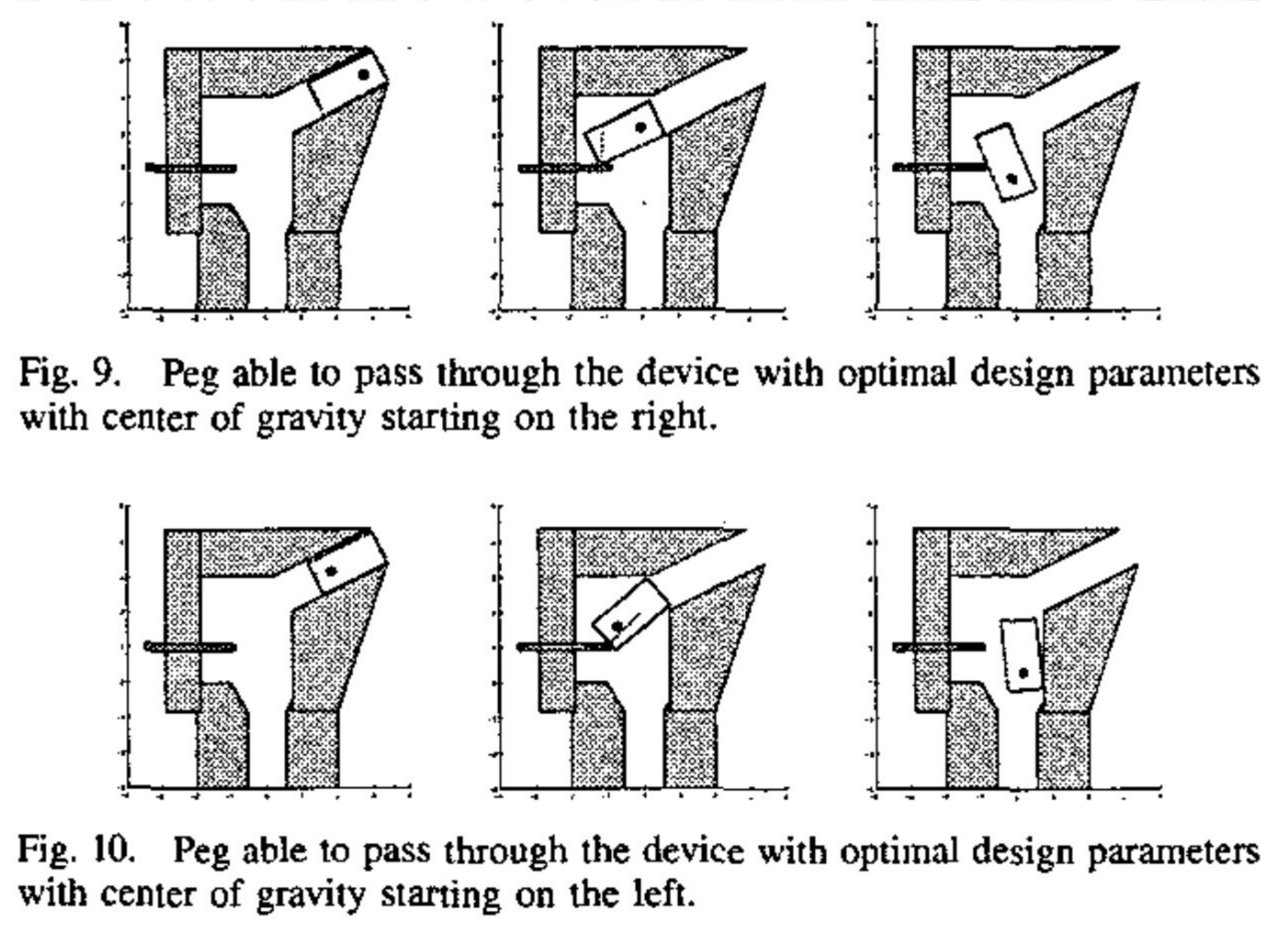


Fig. 9. Peg able to pass through the device with optimal design parameters with center of gravity starting on the right.

Fig. 10. Peg able to pass through the device with optimal design parameters with center of gravity starting on the left.

Robotics: Automatic Part Sorting & Orienting

“Using Simulation for Planning and Design of Robotic Systems with Intermittent Contact”,
Stephen Berard,
RPI PhD 2009.

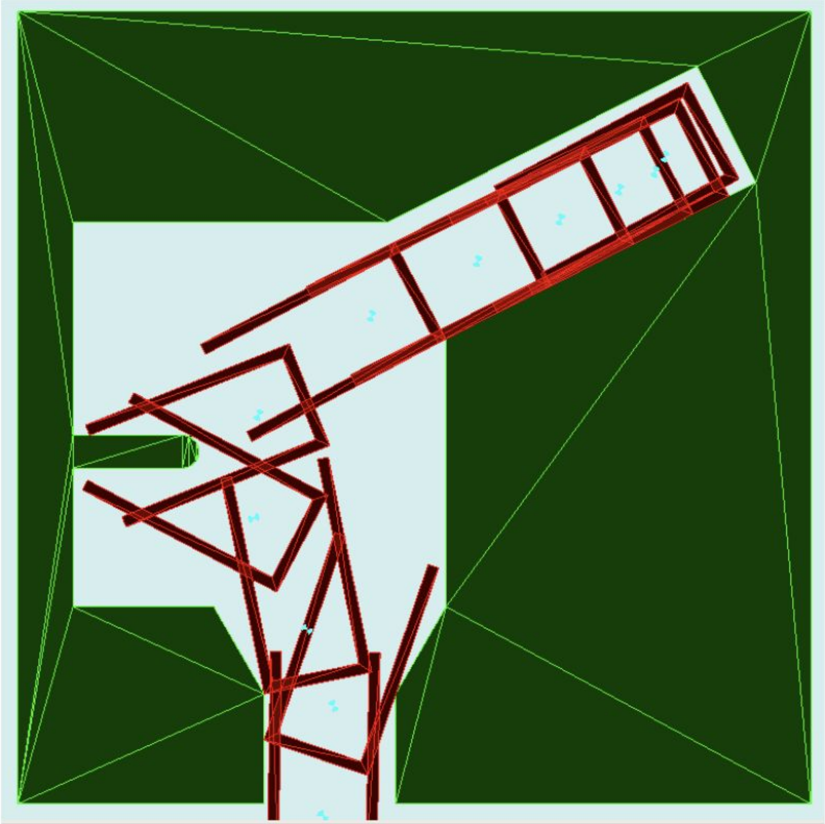


Figure 4.2: Snapshots of the gravity-fed part in the feeder.

Outline for Today

- Homework 2 Questions?
- Last Time: Monotone Polygons & Improved Triangulation Algorithm
- Motivation: Manufacturing by Mold Casting
- Dual Representation: Planar Constraints
- Half-Plane / Half-Space Intersection
- Incremental Linear Programming
- Related Application: Japanese Wood Joints
- Related Application: Automatic Robotic Part Sorting
- Next Time: Point Location