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 counterclockwise around face on left

Image from Justin Legakis
Homework 2

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

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

- 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°)

- = start vertex
- = end vertex
- = regular vertex
\[\text{△}\] = split vertex
\[\text{◆}\] = merge vertex
How do we decide what to connect them to?

- Perform line sweep from top to bottom
- When we find split vertex $v_j$, 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 $e_k$
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

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
“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!
“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?
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 $90^\circ$ with the outward normal of all ordinary facets of $P$. 
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 $90^\circ$ with the outward normal of all ordinary facets of $P$.

If the piece collides with mold facet $\hat{f}$ it must have angle $> 90^\circ$, which would imply an angle $< 90^\circ$ 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: \( d_x * n_x + d_y * n_y + d_z * n_z \)

\[ d = (d_x, d_y, d_z) \]
\[ n = (n_x, n_y, n_z) \]

- 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 $90^\circ$ 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.
“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

(i) finite bounded region

(ii) unbounded region

(iii) degenere case: intersect at a single point

(iv) 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!
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
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 $k \leq n$

- Potential Complication?
  The shapes may be unbounded
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 $NULL$
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
  $\rightarrow O(n)$
- Intersect the half-spaces of all other facets
  - Merge 2 convex regions
    $\rightarrow O(n)$
  - Divide & Conquer Recursion
    $\rightarrow O(n \log n)$
- If it is non-empty, we have a solution!
- Overall:
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?
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: → $O(n^2 \log n)$

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

Can we do better?
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

Maximize \[ c_1 x_1 + c_2 x_2 + \cdots + c_d x_d \]

Subject to \[
\begin{align*}
    a_{1,1} x_1 + \cdots + a_{1,d} x_d &\leq b_1 \\
    a_{2,1} x_1 + \cdots + a_{2,d} x_d &\leq b_2 \\
    \vdots \\
    a_{n,1} x_1 + \cdots + a_{n,d} x_d &\leq b_n
\end{align*}
\]

feasible region

solution

objective function

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 \( v_i \)
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?
Incremental Solution - Analysis

Infeasible - no solution

Satisfied: \( v_1 = v_{i+1} \)

Satisfied: compute new \( v_{i+1} \)
Incremental Solution - Analysis

Infeasible - no solution

Satisfied: \( v_1 = v_{i+1} \)

Satisfied: compute new \( v_{i+1} \)

\( \rightarrow O(1) \)

\( \rightarrow O(1) \)

\( \rightarrow O(n) \)

\( \rightarrow O(1) \)

short circuit exit!
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: $C_1, C_2, C_3, \ldots, C_n$

  $\rightarrow$

- 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:**

$\rightarrow$
Incremental Solution - Analysis

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

$\rightarrow O(n)$

- Which have optimal solutions:
  $v_1$, $v_2$, $v_3$, … $v_n$

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

Overall:
$\rightarrow O(n^2)$ worst case
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: $C_1, C_2, C_3, \ldots C_n$

  $\rightarrow O(n)$

- 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:

$\rightarrow O(n^2)$ worst case
Randomized Linear Programming

- 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: \( C_1, C_2, C_3, \ldots C_n \)

\[ \rightarrow O(n) \]

- 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 \( v_i \)

\[ \rightarrow O(1) \quad \text{short circuit exit!} \]

Overall:

\[ \rightarrow O(n) \text{ expected case} \]

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

- 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: \( C_1, C_2, C_3, \ldots C_n \)

\[ \rightarrow O(n) \]

- Which have optimal solutions: \( v_1, v_2, v_3, \ldots v_n \)
- \( C_i \) has with half-spaces \( \{ h_1, h_2, h_3, \ldots h_i \} \) with solution \( v_i \)

Overall: \[ \rightarrow O(n) \] expected case

Mold casting \[ \rightarrow O(n^2)! \]

\[ \rightarrow O(1) \]

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
Japanese Joinery - Kane Tsugi

Dylan Iwakuni

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

Hand Cutting the "Mysterious Joinery"

Hand工具で刻む 謎の継手

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.
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