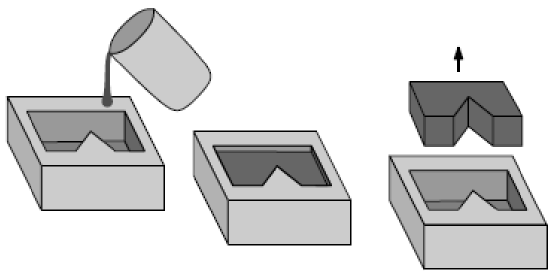
Linear Programming
Manufacturing with Molds

Min-Te Sun, Ph.D.

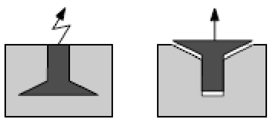
Casting

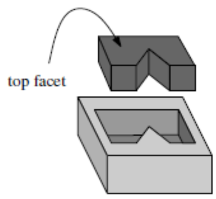
- Can a mold be removed by a single translation?



The Geometry of Casting

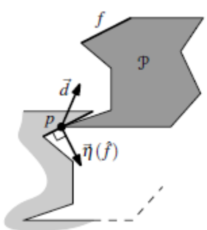
- We model a manufactured object as a 3-dimensional polyhedron P bounded by planar facets
  - An ordinary facet f is any facet other than top facet, which has a corresponding facet in the mold denoted as $\widehat{f}$





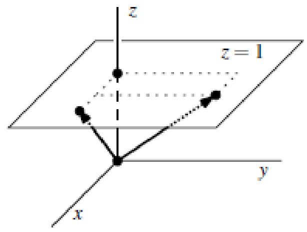
Check if a Casting be Removed

- The polyhedron P can be removed from its mold by a translation in direction $\vec{d}$ if and only if $\vec{d}$ makes an angle of at least 90° with the outward normal of all ordinary facets of P



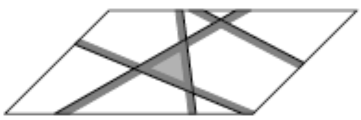
Learned from the Proof

- If a casting can be removed at all by a series of translations, there exists a direction such that it can be removed by a single translation
- If a outward normal of a ordinary facet is (η_x, η_y, η_z) and the direction is (d_x, d_y, 1), then two makes an angle greater than 90° if and only if $\eta_x d_x + \eta_y d_y + \eta_z <= 0$



Finding If a Casting Can Be Removed

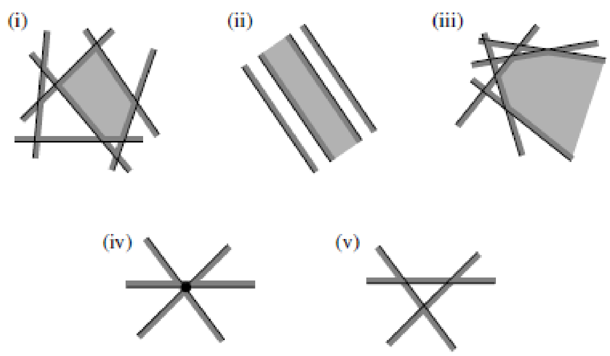
- Let P be a polyhedron with n facets. In $O(n^2)$ expected time and using O(n) storage it can be decided whether P is castable. Moreover, if P is castable, a mold and a valid direction for removing P from it can be computed in the same amount of time.



Half-Plane Intersection

- Let $H = \{h_1,h_2,\ldots,h_n\}$ be a set of linear constraints in two variables, that is, constraints of the form $a_ix+b_iy <= c_i$, where at least one of a_i and b_i is non-zero, we would like to find the set of all points $(x,y) \in R^2$ that satisfy all n constraints at the same time
  - Each constraint is a half-plane in R^2
  - The result is a convex set

Five Different Cases



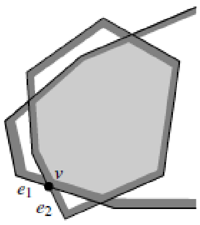
A Divide-and-Conquer Algorithm

Algorithm IntersectHalfplanes(H)
Input. A set H of n half-planes in the plane.
Output. The convex polygonal region $C := \bigcap_{h \in H} h$.

```
if card(H) = 1
  then C ← the unique half-plane h ∈ H
  else Split H into sets H1 and H2 of size ⌈n/2⌉ and ⌊n/2⌋.
       C1 ← IntersectHalfplanes(H1)
       C2 ← IntersectHalfplanes(H2)
       C ← IntersectConvexRegions(C1, C2)
```

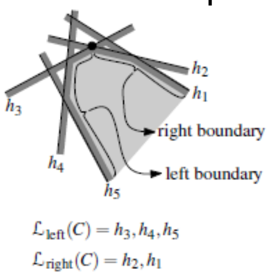
How to Do IntersectConvexRegions?

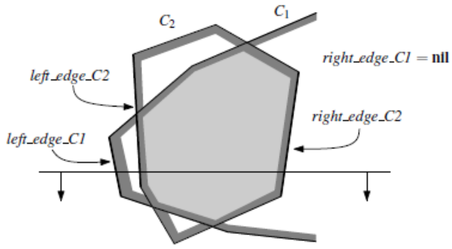
- Covered in Chap 2 already!
- The overlay of two polygons can be computed in O((n+k)logn)
  - k is the # of intersection points => k <= n
  - IntersectConvexRegions can be done in O(nlogn)
    - The divide-n-conquer algorithm can be done in O(nlog^{2}n)
- Can we do better?



More Efficient IntersectConvexRegions

- Break boundary of the convex region into left and right chains and apply plane sweep algorithm
  - y_{start} = infinite if the top is unbounded
  - The event points are the top of segments

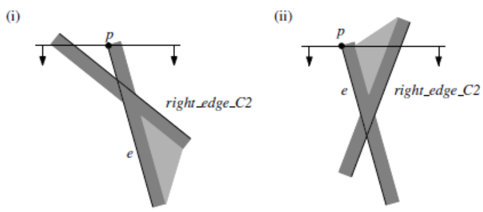




Three Cases

- Let *p* be the upper endpoint of a left boundary *e,* then

1. *p* lies in between *left_edge_C2* and *right_edge_C2*
2. *e* intersects *right_edge_C2*
   - both edges contribute an edge to *C* starting at the intersection point; or
   - both edges contribute an edge ending there
3. *e* intersects *left_edge_C2*



Time Complexity Analysis

- Since each case only needs O(1) to handle, IntersectConvexRegions can be complete in O(n)
- The common intersection of a set of n half-planes in the plane can be computed in O(nlogn) time and linear storage

Incremental Linear Programming

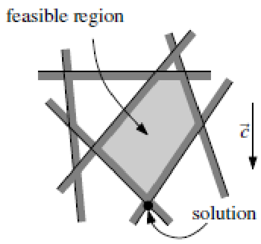
- A linear optimization problem is described as follows:

$$\begin{array}{ll} \text{Maximize} & c_1x_1 + c_2x_2 + \cdots + c_dx_d \\ \\ \text{Subject to} & a_{1,1}x_1 + \cdots + a_{1,d}x_d \leqslant b_1 \\ & a_{2,1}x_1 + \cdots + a_{2,d}x_d \leqslant b_2 \\ & \vdots \\ & a_{n,1}x_1 + \cdots + a_{n,d}x_d \leqslant b_n \end{array}$$

- $c_1x_1+c_2x_2+\ldots+c_dx_d$ is called the objective function
- The intersection of these half-spaces is called the *feasible region* of the linear program

Low-Dimensional Linear Programming

- The objective function can be viewed as a direction in R^d
  - maximizing $c_1x_1+c_2x_2+\ldots+c_dx_d$ means finding a point ($x1, \ldots, xd$) that is extreme in the direction $c = (c_1, \ldots, c_d)$
- This can be solved by Simplex algorithm
  - designed mainly for high dimension
  - does not perform efficiently in low-dimensional (2-D in our case) linear programming

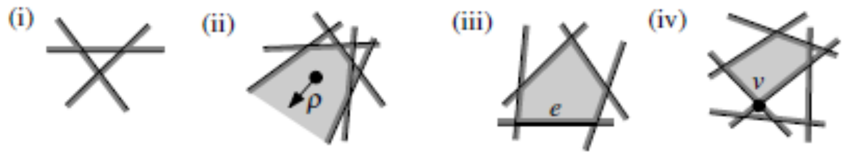


2-D Linear Programming

- The set of *n* linear constraints in our 2-dimensional linear programming problem is denoted by *H*
- The vector defining the objective function is $c=(c_x, c_y)$
  - The objective function is $f_c(p) = c_x p_x + c_y p_y$
- Our goal is to find a point $p \in R^2$ such that $p \in \cap H$ and $f_c(p)$ is maximized
  - LP is denoted as (H, c)

Four Cases for the Solution of LP

- (iii) can be considered as a case of (iv) if proper priority is given to the points on the line



Definition of C_i

$$m_1 := \begin{cases} p_x \leqslant M & \text{if } c_x > 0 \\ -p_x \leqslant M & \text{otherwise} \end{cases}$$

and

$$m_2 := \begin{cases} p_y \leqslant M & \text{if } c_y > 0 \\ -p_y \leqslant M & \text{otherwise} \end{cases}$$

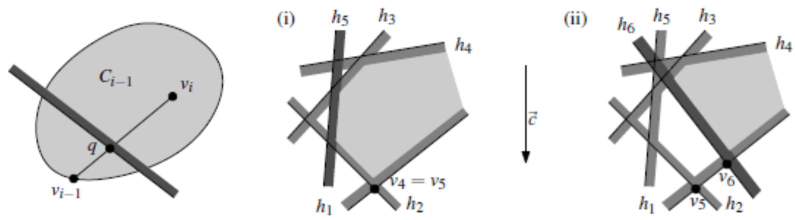
- $C_i = m_1 \cap m_2 \cap h_1 \cap h_2 \cap \ldots \cap h_i$
- $C0 \supseteq C1 \supseteq C2 \ldots \supseteq Cn = C$
- We denote the optimal vertex of C_i by v_i

Adding One More Half-Plane

- *Let* 1 <= *i* <= *n, and let C_i and v_i be defined as above. Then we have*

1. If $v_{i-1} \in h_i$, then $v_i = v_{i-1}$

2. If $v_{i-1} \in h_i$, then either $C_i = \varnothing$ or $v_i \in l_i$, where l_i is the line bounding h_i



Problem Transformation

- Assume that the current optimal vertex *vi*−1 is not contained in the next half-plane *hi*. The problem we have to solve can be stated as follows:

  Find the point *p* on *i* that maximizes $f_c(p)$, subject to the constraints $p \in h$, for $h \in H_{i-1}$

- *A* 1-*dimensional linear program* can be solved in linear time
  - If case 2 of previous result arises, then we can compute the new optimal vertex vi, or decide that the linear program is infeasible, in O(i) time

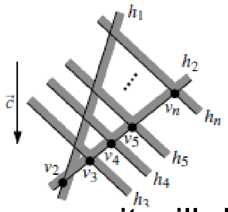
Incremental LP Algorithm

Algorithm 2DBOUNDEDLP($H, \vec{c}, m_1, m_2$)
Input. A linear program $(H \cup \{m_1, m_2\}, \vec{c})$, where H is a set of n half-planes, $\vec{c} \in \mathbb{R}^2$, and m_1, m_2 bound the solution.
Output. If $(H \cup \{m_1, m_2\}, \vec{c})$ is infeasible, then this fact is reported. Otherwise, the lexicographically smallest point p that maximizes $f_{\vec{c}}(p)$ is reported.

```
Let v_0 be the corner of C_0.
Let h_1, ..., h_n be the half-planes of H.
for i ← 1 to n
    do if v_{i-1} ∈ h_i
         then v_i ← v_{i-1}
         else v_i ← the point p on ℓ_i that maximizes f_c(p), subject to the
              constraints in H_{i-1}.
              if p does not exist
                then Report that the linear program is infeasible and quit.
return v_n
```

Time Complexity

- The time complexity is $\sum_{i=1}^{n} O(i) = O(n^2)$.
- Is it that bad?
  - When case 1 or the other half of case 2 occur, the computation is much faster
  - Worst case =>



  - How do we make sure it will do better?

Randomized LP Algorithm

Algorithm 2DRANDOMIZEDBOUNDEDLP($H, \vec{c}, m_1, m_2$)
Input. A linear program $(H \cup \{m_1, m_2\}, \vec{c})$, where H is a set of n half-planes, $\vec{c} \in \mathbb{R}^2$, and m_1, m_2 bound the solution.
Output. If $(H \cup \{m_1, m_2\}, \vec{c})$ is infeasible, then this fact is reported. Otherwise, the lexicographically smallest point p that maximizes $f_{\vec{c}}(p)$ is reported.

```
1. Let v_0 be the corner of C_0.
2. Compute a random permutation h_1, ..., h_n of the half-planes by calling
   RANDOMPERMUTATION(H[1···n]).
3. for i ← 1 to n
4.     do if v_{i−1} ∈ h_i
5.          then v_i ← v_{i−1}
6.          else v_i ← the point p on ℓ_i that maximizes f_c(p), subject to the
                 constraints in H_{i−1}.
7.               if p does not exist
8.                 then Report that the linear program is infeasible and quit.
9. return v_n
```

Time Complexity Analysis

- The 2-dimensional linear programming problem with n constraints can be solved in O(n) randomized expected time using worst-case linear storage
- Let X_i be a random variable, which is 1 if $v_{i-1} \in h_i$, and 0 otherwise
  - The total time in line 6 is $\sum_{i=1}^{n} O(i) \cdot X_i$.
  - The expected value is $\mathrm{E}[\sum_{i=1}^{n} O(i) \cdot X_i] = \sum_{i=1}^{n} O(i) \cdot \mathrm{E}[X_i]$.
  - $\mathrm{E}[X_i]$ is exactly the probability that *$v_i-1 \in h_i$, which is 2/i in worst case* => $\sum_{i=1}^{n} O(i) \cdot \frac{2}{i} = O(n)$.

How to Determine if H is Unbounded in Direction of c?

- A linear program (H,c) is unbounded if and only if there is a vector d with d・c > 0 such that d・η(h) >= 0 for all h ∈ H and the linear program (H,c) is feasible, where H = {h ∈ H :η(h)・d = 0}
- Let H be a set of half-planes and η(h) be the outward normal of h, H_{min} is the subset of H such that the inner product of η(h) and c is minimum
  - H_{par} be the set of planes parallel to plane in H_{min}
  - H_{par} could be empty

Description of Algorithm for Unbounded Determination

- Let H, H_{min}, and H_{par} be defined as previously and l_{i^*} denotes the bound of h_{i^*} whose $\eta(h_{i^*})$ and c has min inner product, then
  - If $l_{i^*} \cap h_j$ is unbounded in the direction c for every half-plane h_j in H \ (H_{min} U H_{par}), then (H, c) is unbounded along a ray contained in l_{i^*}
  - If $l_{i^*} \cap h_j$ is bounded in the direction c for some h_j in H \ (H_{min} U H_{par}), then the linear program ({h_{i^*}, h_j}, c) is bounded
- Whether H is unbounded in c, and the ray of the unbounded direction d if it is or the 2 planes that bound c can be computed in O(n)

UnboundedLP Algorithm

Algorithm UnboundedLP(H, c)

Input. A linear program (H, c) where H is a set of n half-planes and c is the vector defining the objective function.

Output. If (H, c) is unbounded, the output is a ray that is contained in the feasible region. If (H, c) is bounded, the output either consists of two half-planes h_{i^*} and h_{j^*} from H such that ($\{h_{i^*}, h_{j^*}\}$, c) is bounded, or it is reported that the linear program is infeasible.

1. For each half-plane h_i in H, compute the angle $ø_i$
2. Let hi be a half-plane with $ø_i = \min_{1 \leq j \leq n} ø_j$.
3. $H_{min} \leftarrow \{h_j \text{ in } H \mid \eta_j = \eta_i\}$
4. $H_{par} \leftarrow \{h_j \text{ in } H \mid \eta_j = -\eta_i\}$
5. $H' \leftarrow H \setminus (H_{min} \cup H_{par})$
6. Compute the intersection of the half-planes in $H_{min} \cup H_{par}$.
7. If the intersection is empty
8. then Report that (H, c) is infeasible.
9. else Let h_{i^*} in Hmin be the half-plane whose bounding line bounds the intersection.
10. if there is a half-plane h_{j^*} in H' such that $l_{i^*} \cap h_{j^*}$ is bounded in direction c
11. then Report that ($\{h_{i^*}, h_{j^*}\}$, c) is bounded.
12. else Report that (H, c) is unbounded along the ray $l_{i^*} \cap (\cap H')$.

Homework Assignment 2

Page 60 ~ 61

- 3.2
- 3.7

Page 91

- 4.2
- 4.8
- 4.16