Computational Geometry

Lecture 5: Casting a polyhedron
CAD/CAM systems allow you to design objects and test how they can be constructed.

Many objects are constructed using a mold.
A general question: Given an object, can it be made with a particular design process?

For casting, can the object be removed from its mold without breaking the cast?
Casting

Objects to be made are 3D polyhedra

Its boundary is like a planar graph, but the coordinates of vertices are 3D
First the 2D version: can we remove a 2D polygon from a mold?
Certain removal directions may be good while others are not

Computational Geometry
Lecture 5: Casting a polyhedron
What top facet should we use?

When can we even begin to move the object out?

What kind of movements do we allow?

An edge of the polygon should not directly run into the touching mold edge.
Assume the top facet fixed; we can try all

An edge of the polygon should not directly run into the touching mold edge

Let us consider *translations* only
**Observe:** For a given top facet, if the object can be translated over some (small) distance, then it can be translated all the way out.

Consider a point $p$ that at first translates away from its mold side, but later runs into the mold ...
A polygon can be removed from its cast *by a single translation* if and only if there is a direction so that every polygon edge does not cross the adjacent mold edge.

Sequences of translations do not help to remove more shapes than a single translation.
Circle of directions

We need a representation of directions in 2D

Every polygon edge requires the removal direction to be in a semi-circle

⇒ compute the common intersection of a set of circular intervals (semi-circles)
Line of directions

We only need to represent upward directions: we can use points on the line $y = 1$

Every polygon edge requires the removal direction to be in a half-line

$\Rightarrow$ compute the common intersection of a set of half-lines in 1D
The common intersection of a set of half-lines in 1D

- Determine the endpoint $p_l$ of the rightmost left-bounded half-line
- Determine the endpoint $p_r$ of the leftmost right-bounded half-line
- The common intersection is $[p_l, p_r]$ (can be empty)
The algorithm takes only $O(n)$ time for $n$ half-lines.

Note: we need not sort the endpoints.
Casting in 3D

Can we do something similar in 3D?

Again each facet must not move into the corresponding mold facet
Representing directions in 3D

The circle of directions for 2D becomes a sphere of directions for 3D; the line of directions for 2D becomes a plane of directions for 3D.

Which directions represented in the plane does a facet rule out as removal directions?
Consider the outward normal vectors of all facets.

An allowed removal direction must make an angle of at least $\pi/2$ with every facet (except the topmost one).

$\Rightarrow$ every facet makes a half-plane invalid.
We get: common intersection of half-planes in the plane

The problem of deciding castability of a polyhedron with $n$ facets, with a given top facet, where the polyhedron must be removed from the cast by a single translation can be solved by common intersection of $n-1$ half-planes
Common intersection of half-planes

Half-planes in the plane:

- \( y \geq m \cdot x + c \)
- \( y \leq m \cdot x + c \)
- \( x \geq c \)
- \( x \leq c \)
An approach

Take the first set:

\[ y \geq m \cdot x + c \]

Sort by angle, and add incrementally
Introduction
Common intersection computation
Linear programming in 2D
Casting in 3D
Common intersection of half-planes
Incremental common intersection

Computational Geometry
Lecture 5: Casting a polyhedron
Common intersection computation
Linear programming in 2D
Casting in 3D
Common intersection of half-planes
Incremental common intersection
Computational Geometry Lecture 5: Casting a polyhedron
Introduction
Common intersection computation
Linear programming in 2D
Casting in 3D
Common intersection of half-planes
Incremental common intersection

Computational Geometry Lecture 5: Casting a polyhedron
Introduction
Common intersection computation
Linear programming in 2D
Casting in 3D
Common intersection of half-planes
Incremental common intersection

Computational Geometry
Lecture 5: Casting a polyhedron
The boundary of the valid region is a polygonal convex chain that is unbounded at both sides.

The next half-plane has a steeper bounding line and will always contribute to the next valid region.
Maintain the contributing bounding lines in decreasing angular order

For the new half-plane, remove any no longer contributing bounding lines from the end

Then add the line bounding the new half-plane
After sorting on angle, this takes only $O(n)$ time.

The half-planes bounded from above give a similar chain.

Intersecting the two chains is simple with a left-to-right scan.
Incremental common intersection

Half-planes with vertical bounding lines can be added by restricting the region even more.

This can also be done in linear time.
**Theorem:** The common intersection of $n$ half-planes in the plane can be computed in $O(n \log n)$ time

The common intersection may be empty, or a convex polygon that can be bounded or unbounded.
The common intersection of half-planes cannot be computed faster (we are sorting the lines along the boundary)

The region we compute represents *all mold removal directions*

…

… but to determine castability, we only need one!
Linear programming

We will find the *lowest* point in the common intersection

Notice that half-planes are linear constraints

**Minimize** $y$

**Subject to**

\[
\begin{align*}
y &\geq m_1 \cdot x + c_1 \\
y &\geq m_2 \cdot x + c_2 \\
& \vdots \\
y &\geq m_i \cdot x + c_i \\
y &\leq m_{i+1} \cdot x + c_{i+1} \\
& \vdots \\
y &\leq m_n \cdot x + c_n
\end{align*}
\]
Linear programming in usual setting:

**Minimize** \( c_1 \cdot x_1 + \cdots + c_k \cdot x_k \)

**Subject to**

\[
\begin{align*}
\sum_{i=1}^{n} a_{1,i} \cdot x_1 + \cdots + a_{k,i} \cdot x_k & \leq b_1 \\
\sum_{i=1}^{n} a_{1,i} \cdot x_1 + \cdots + a_{k,i} \cdot x_k & \leq b_2 \\
\vdots & \\
\sum_{i=1}^{n} a_{1,n} \cdot x_1 + \cdots + a_{k,n} \cdot x_k & \leq b_n
\end{align*}
\]

where \( a_{1,1}, \ldots, a_{k,n}, b_1, \ldots, b_n, c_1, \ldots, c_k \) are given coefficients.

LP with \( k \) unknowns (dimensions) and \( n \) inequalities.
LP with $k$ unknowns (dimensions) and $n$ inequalities: $k$-dimensional linear programming

The subspace that is the common intersection is the feasible region. If it is empty, the LP is infeasible

The vector $(c_1, \ldots, c_k)^T$ is the objective vector or cost vector

If the LP has solutions with arbitrarily low cost, then the LP is unbounded

Note: The feasible region may be bounded while the LP is bounded
LP for casting

It is 2-dimensional linear programming with \( n \) constraints.

We only want to decide feasibility, so we can choose any objective function.

We will make it ourselves easy.
Let \( h_1, \ldots, h_n \) be the constraints and \( \ell_1, \ldots, \ell_n \) their bounding lines.

Find any two constraints \( h_1 \) and \( h_2 \) with \( \ell_1 \) and \( \ell_2 \) non-parallel.

Rotate \( h_1 \) and \( h_2 \) over an angle \( \alpha \) around the origin to make \( \ell_1 \cap \ell_2 \) the optimal solution for the objective function minimize \( y \). Rotate all other constraints over \( \alpha \) too.
Incremental LP

Solve the LP with the rotated constraints

If the rotated LP is infeasible, then so is the unrotated version

If the rotated LP gives an optimal solution \((p_x, p_y)\), then rotate if over an angle \(-\alpha\) around the origin to get the removal direction for the original position of the polyhedron
The algorithm adds the constraints $h_3, \ldots, h_n$ incrementally and maintains the optimum so far

Let $H_i = \{h_1, \ldots, h_i\}$

Let $v_i$ be the optimum for $H_i$ (unless we already have infeasibility)
LP for casting

The incremental step: suppose we know $v_{i-1}$ and want to add $h_i$

There are two possibilities:

- If $v_{i-1} \in h_i$, then $v_i = v_{i-1}$
- If $v_{i-1} \notin h_i$, then either the LP is infeasible, or $v_i$ lies on $\ell_i$
Incremental LP
Algorithm \textsc{LPforCasting}(H)

1. Let \( h_1, h_2, \) and \( v_2 \) be as chosen
2. \textbf{for} \( i \leftarrow 3 \) \textbf{to} \( n \)
3. \textbf{do if} \( v_{i-1} \in h_i \)
   4. \textbf{then} \( v_i \leftarrow v_{i-1} \)
5. \textbf{else} \( v_i \leftarrow \) the point \( p \) on \( \ell_i \) that minimizes \( y \), subject to the constraints in \( H_{i-1} \).
6. \textbf{if} \( p \) does not exist
7. \textbf{then} Report that the LP is infeasible, and quit.
8. \textbf{return} \( v_n \)
If \( v_{i-1} \not\in h_i \), how do we find the point \( p \) on \( \ell_i \)?
Efficiency

If \( v_{i-1} \in h_i \), then the incremental step takes only \( O(1) \) time

If \( v_{i-1} \notin h_i \), then the incremental step takes \( O(i) \) time

The LP-for-casting algorithm takes \( O(n^2) \) time in the worst case
Randomized algorithm

Algorithm \textsc{RandomizedLPforCasting}(H)
1. Let $h_1$, $h_2$, and $v_2$ be as chosen
2. Let $h_3, h_4, \ldots, h_n$ be in a random order
3. for $i \leftarrow 3$ to $n$
4. do if $v_{i-1} \in h_i$
5. then $v_i \leftarrow v_{i-1}$
6. else $v_i \leftarrow$ the point $p$ on $\ell_i$ that minimizes $y$, subject to the constraints in $H_{i-1}$.
7. if $p$ does not exist
8. then Report that the LP is infeasible, and quit.
9. return $v_n$
Putting in random order

The constraints may be given in any order, the algorithm will just reorder them

- Let $j$ be a random integer in $[3, n]$
- Swap $h_j$ and $h_n$
- Recursively shuffle $h_3, \ldots, h_{n-1}$

Putting in random order takes $O(n)$ time
Expected running time

Every one of the \((n - 3)!\) orders is equally likely

The **expected time** taken by the algorithm is the *average* time over all orders

\[
\frac{1}{(n-3)!} \cdot \sum_{\Pi \text{ permutation}} \text{time if the random order is } \Pi
\]
If the order of the constraints $h_3, \ldots, h_n$ is random, what is the probability that $v_{i-1} \in h_i$?

We use **backwards analysis**: consider the situation *after* $h_i$ is inserted, and $v_i$ is computed (either by $v_i = v_{i-1}$, or somewhere on $\ell_i$)
Expected running time

Only if one of the dashed lines was $\ell_i$, the last step where $h_i$ was added was expensive and took $O(i)$ time
If $h_i$ does not bound the feasible region, or not at $v_i$, then the addition step was cheap and took $O(1)$ time.
Expected running time

There are $i - 2$ half-planes that could have been one of the lines defining $v_i$

Since the order was random, each of the $i - 2$ half-planes has the same probability to be the last one added, and only 2 of these caused the expensive step

- 2 out of $i - 2$ cases: expensive step; $O(i)$ time for $i$-th addition
- $i - 4$ out of $i - 2$ cases: cheap step; $O(1)$ time for $i$-th addition
Expected time for $i$-th addition:

$$\frac{i-4}{i-2} \cdot O(1) + \frac{2}{i-2} \cdot O(i) = O(1)$$

Total running time:

$$O(n) + \sum_{i=3}^{n} O(1) = O(n) \text{ expected time}$$
The optimal solution may not be unique, if the feasible region is bounded from below by a horizontal line. How to solve it?

There may be many lines from $\ell_3, \ldots, \ell_i$ passing through $v_i$; how does this affect the probability of an expensive step?
Degenerate cases
Degenerate cases

In degenerate cases, the probability that the last addition was expensive is even smaller: $1/(i - 2)$, or 0

Without any adaptations, the running time holds
**Theorem:** Castability of a simple polyhedron with $n$ facets, given a top facet, can be decided in $O(n)$ expected time

**Theorem:** 2-dimensional linear programming with $n$ constraints can be solved in $O(n)$ expected time

**Question:** What does “expected time” mean? Expectation over what?
Question: Can you imagine whether we can also solve 3-dimensional linear programming efficiently?