# Motion Planning

O'Rourke, Chapter 8

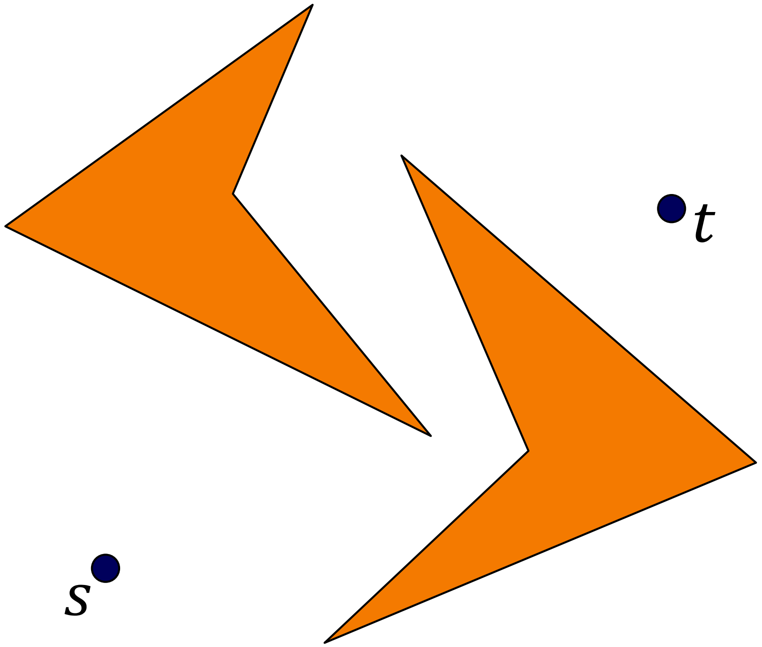
# Outline

- Translating a polygon
- Moving a ladder

# Shortest Path (Point-to-Point)

Goal:

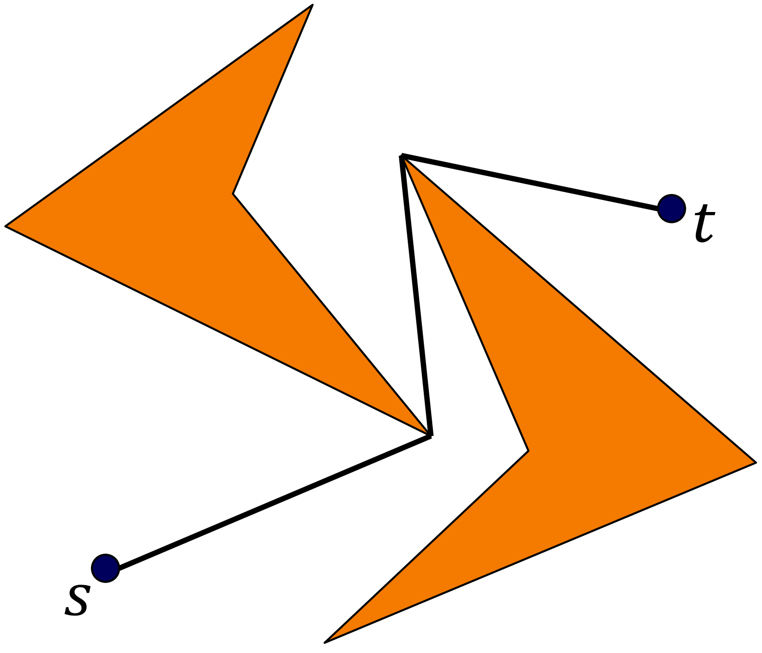
Given disjoint polygons in the plane, and given positions $s$ and $t$, find the **shortest** path from $s$ to $t$ that avoids the polygons.



# Shortest Path (Point-to-Point)

<u>Goal</u>:

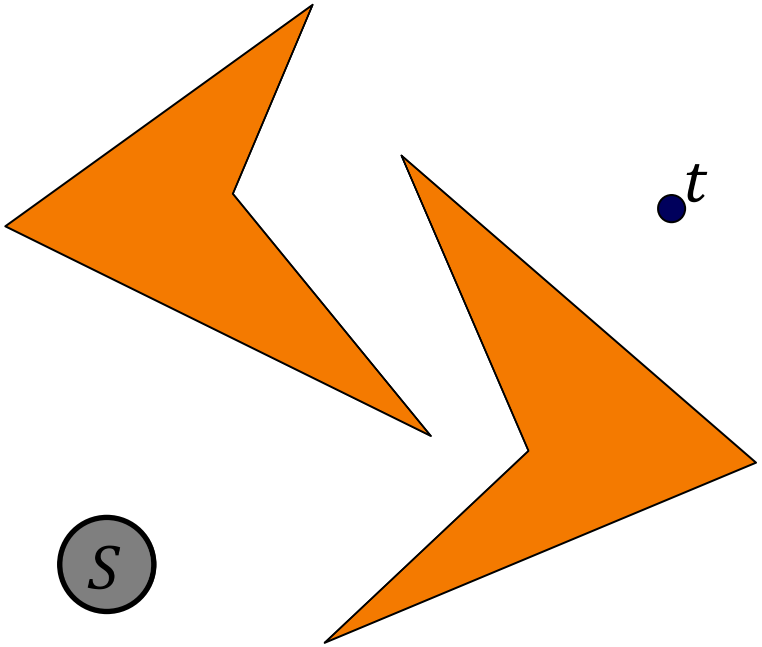
Given disjoint polygons in the plane, and given positions $s$ and $t$, find the **shortest** path from $s$ to $t$ that avoids the polygons.



# Any Path (Shape-to-Point)

Goal:

Given disjoint polygons in the plane, and given a shape $S$ and a position $t$, determine if there is **any** path taking $S$ to $t$ avoiding the polygons.



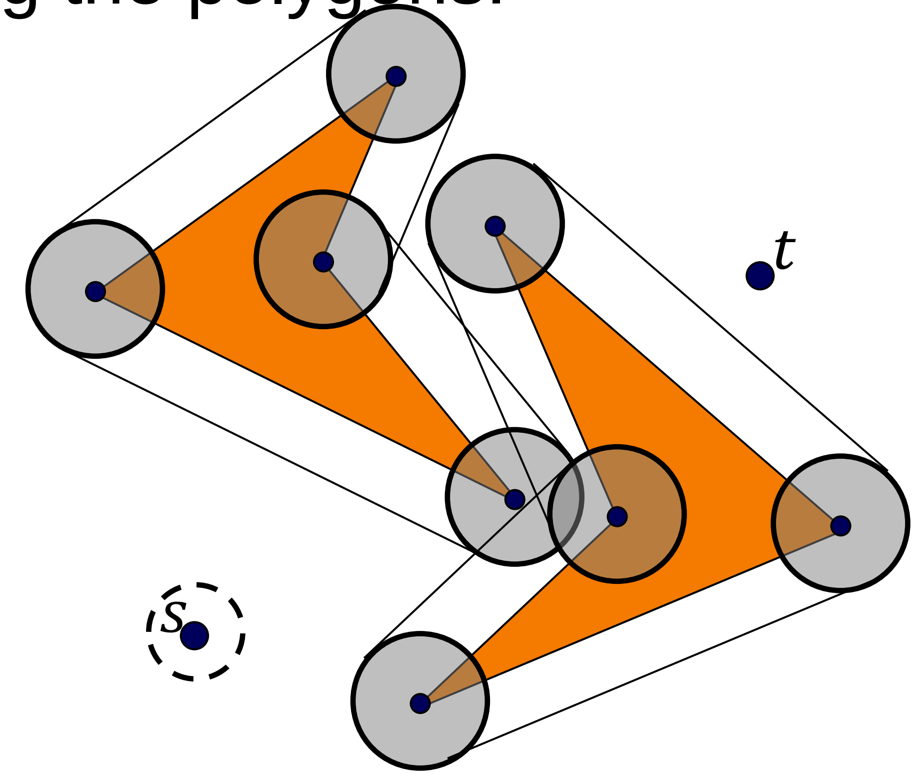
# Any Path (Shape-to-Point)

Goal:

Given disjoint polygons in the plane, and given a shape $S$ and a position $t$, determine if there is **any** path taking $S$ to $t$ avoiding the polygons.

Approach:

Dilate the polygons by $S$ to transform this into a point-to-point problem.



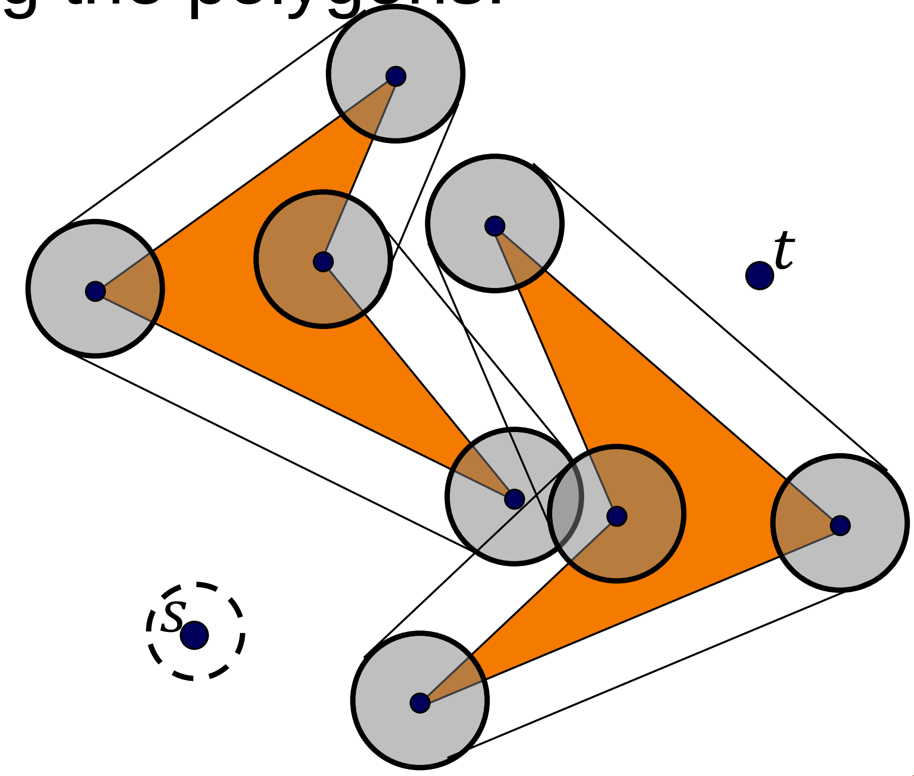
# Any Path (Shape-to-Point)

Goal:

Given disjoint polygons in the plane, and given a shape $S$ and a position $t$, determine if there is **any** path taking $S$ to $t$ avoiding the polygons.

Approach:

Dilate the polygons by $S$ to transform this into a point-to-point problem.




If $s$ can reach $t$ while avoiding the dilated polygons, $S$ can reach $t$.

# Any Path (Shape-to-Point)

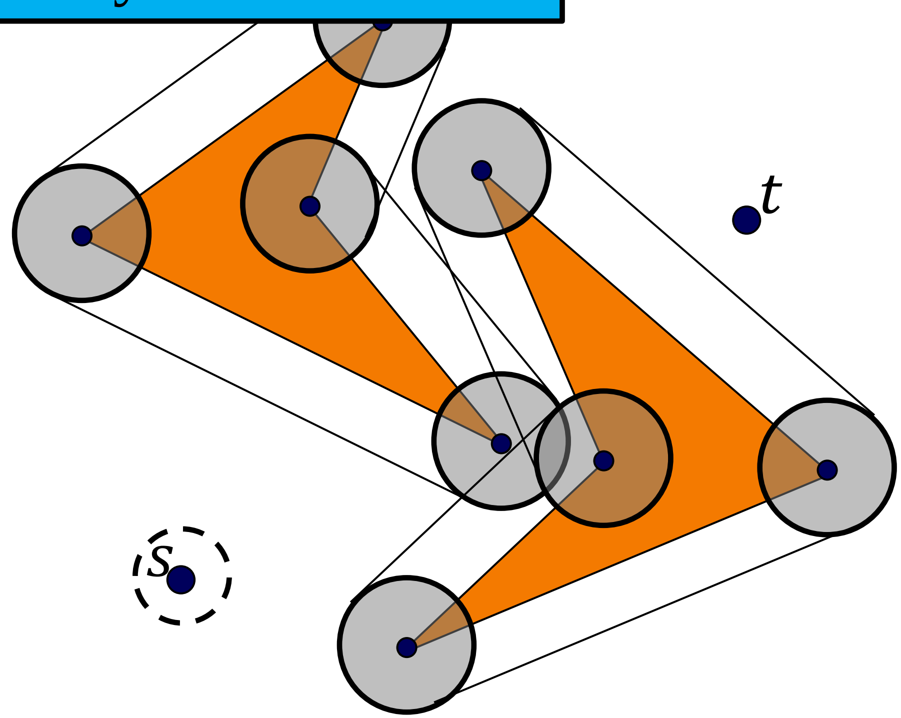
<u>Goal</u>:

Given di… …iven a shape $S$ … e is **any** path tak…

<u>Note that in this case</u>:
- $S$ is symmetric.
- $s$ is the center of $S$.
- The traced out boundary self-intersects.

<u>Approach</u>:

Dilate the polygons by $S$ to transform this into a point-to-point problem.



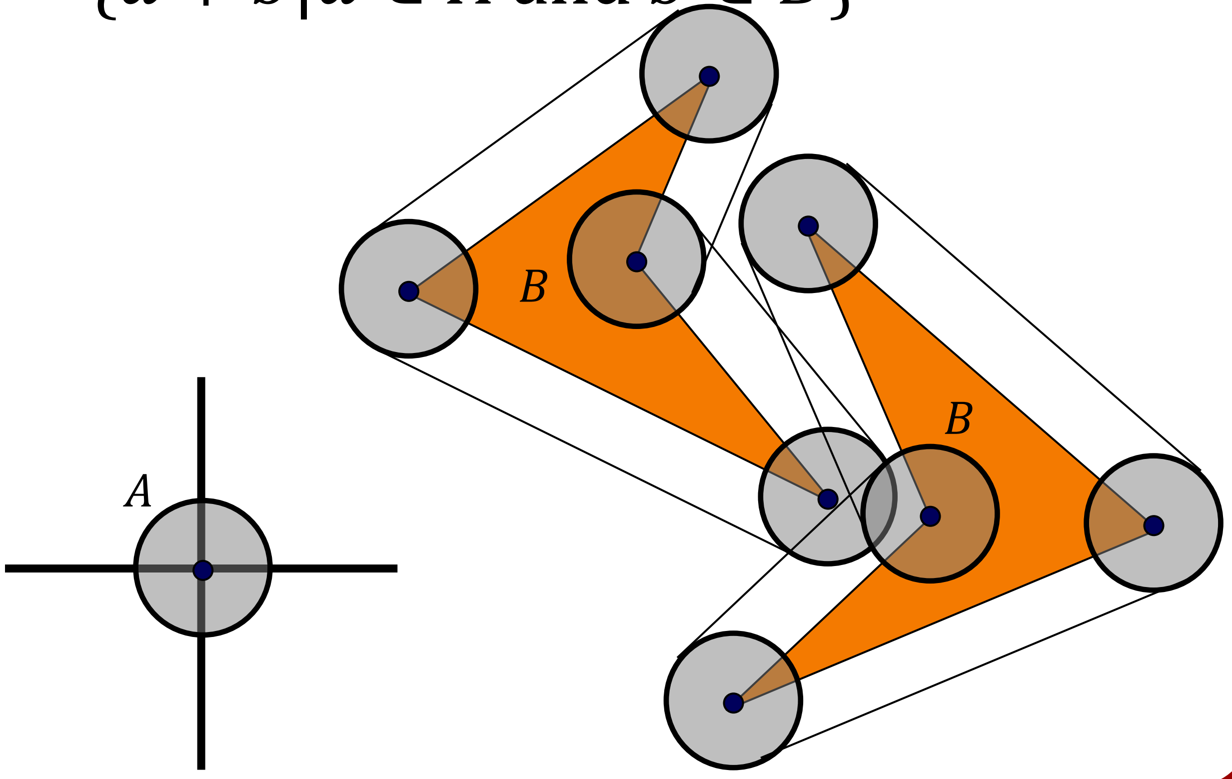

If $s$ can reach $t$ while avoiding the dilated polygons, $S$ can reach $t$.

# Minkowski Sums

<u>Definition</u>:

Given two point-sets in the plane, $A$ and $B$, the *Minkowski Sum* is the set of points:

$$A \oplus B = \{a + b | a \in A \text{ and } b \in B\}$$

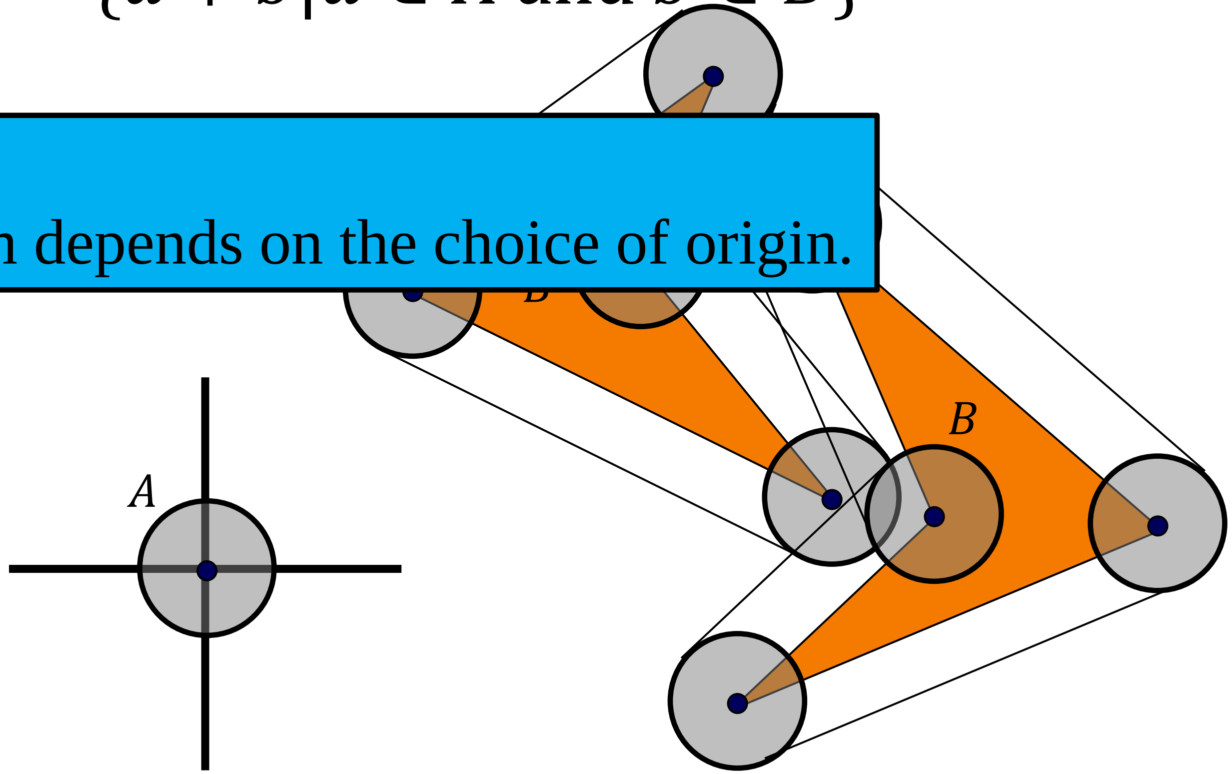


# Minkowski Sums

Definition:

Given two point-sets in the plane, $A$ and $B$, the *Minkowski Sum* is the set of points:

$$A \oplus B = \{a + b | a \in A \text{ and } b \in B\}$$



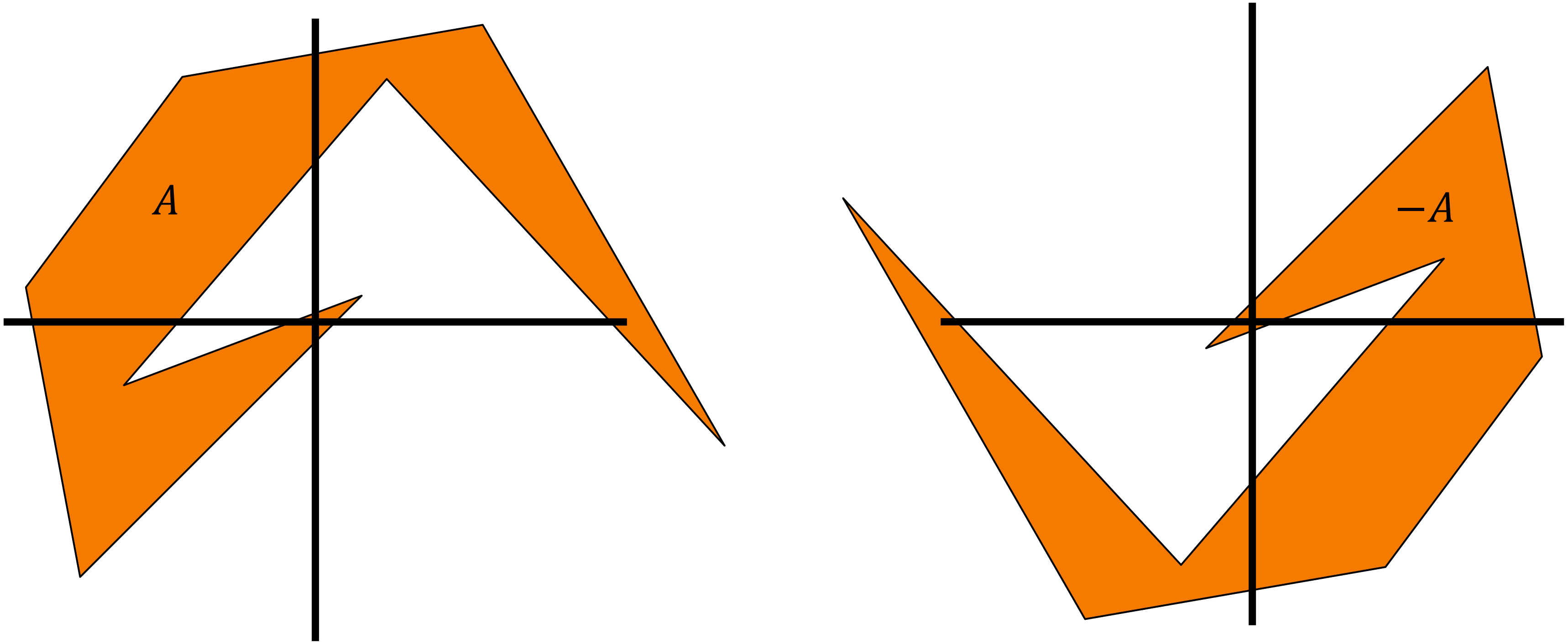

Note:
The sum depends on the choice of origin.

# Minkowski Sums

Definition:

Given a point-set $A$ in the plane, its *negation* is the set of points:

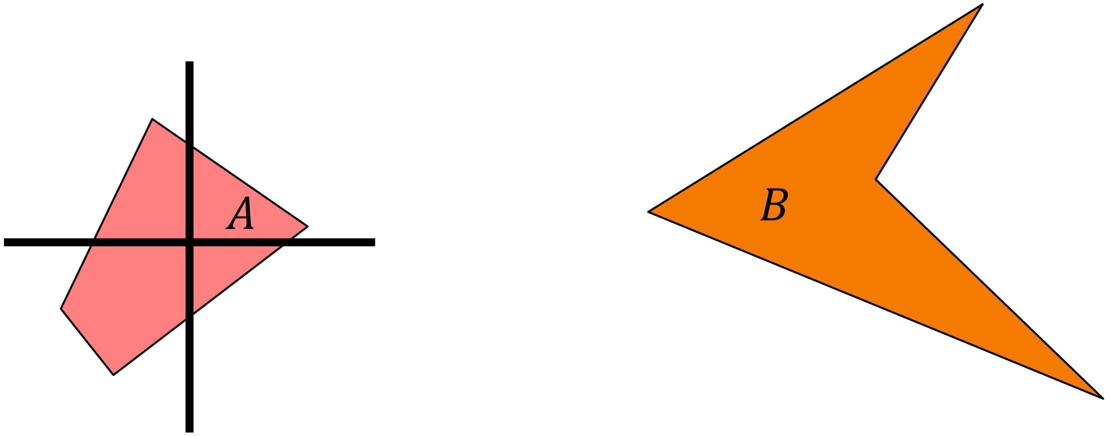
$$-A = \{-a | a \in A\}$$



# Motion Planning

<u>Claim</u>:

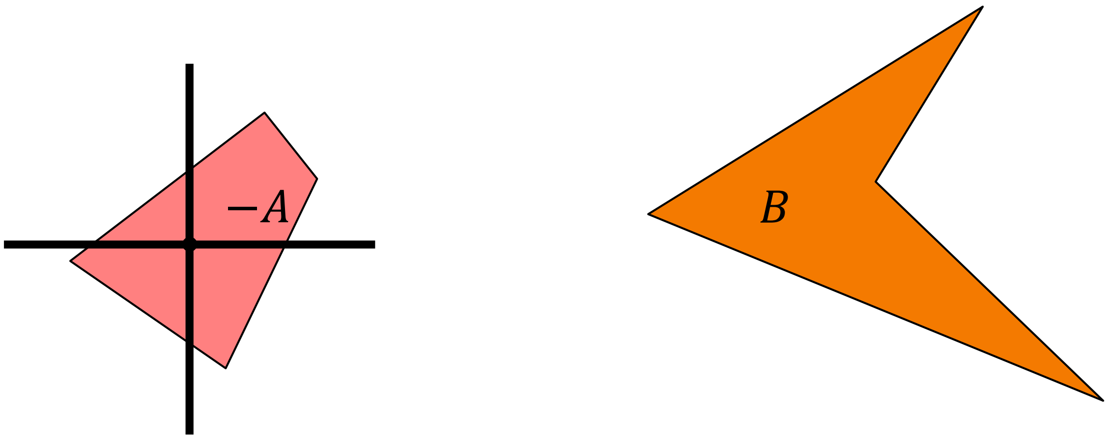
Given shapes $A$ and $B$, the translation of $A$ by $\tau$ intersects $B$ if and only if $\tau \in B \oplus (-A)$.



# Motion Planning

Claim:

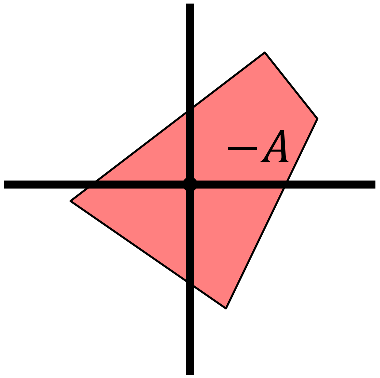
Given shapes $A$ and $B$, the translation of $A$ by $\tau$ intersects $B$ if and only if $\tau \in B \oplus (-A)$.

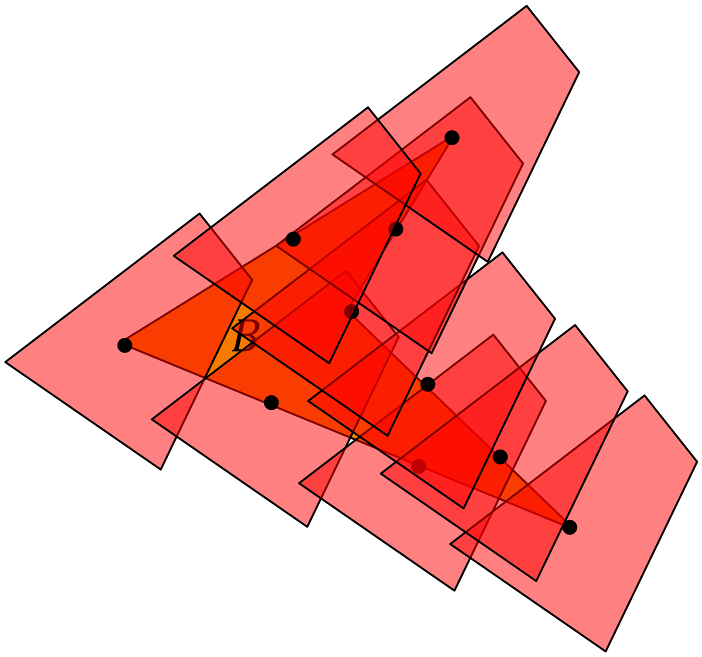


# Motion Planning

<u>Claim</u>:

Given shapes $A$ and $B$, the translation of $A$ by $\tau$ intersects $B$ if and only if $\tau \in B \oplus (-A)$.

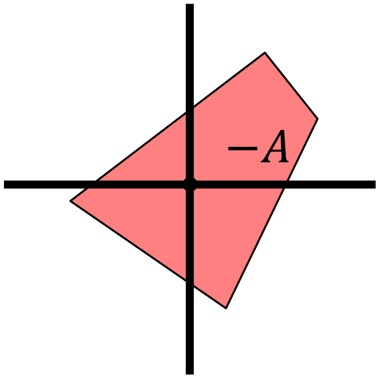


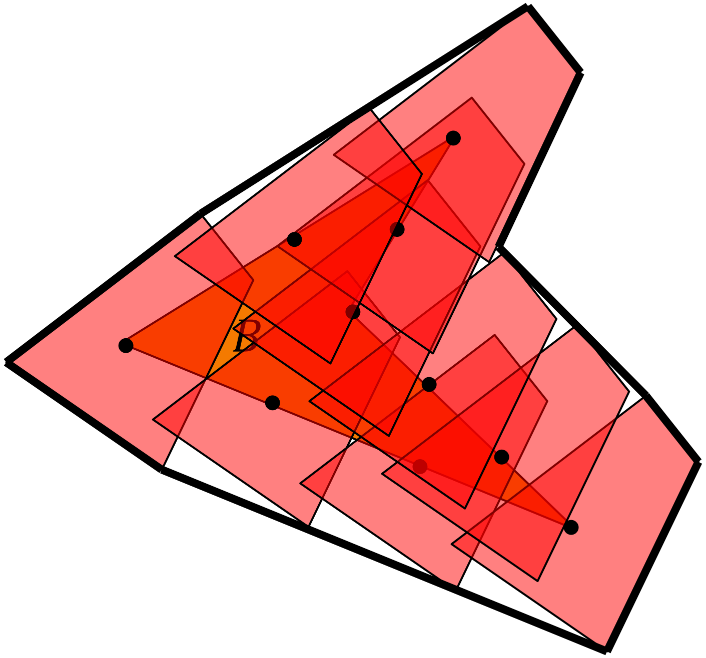



# Motion Planning

Claim:

Given shapes $A$ and $B$, the translation of $A$ by $\tau$ intersects $B$ if and only if $\tau \in B \oplus (-A)$.



# Motion Planning

Claim:

Given shapes $A$ and $B$, the translation of $A$ by $\tau$ intersects $B$ if and only if $\tau \in B \oplus (-A)$.

Proof:

The translation of $A$ by $\tau$ intersects $B$.

$\Leftrightarrow \exists a \in A, b \in B$ such that $a + \tau = b$.

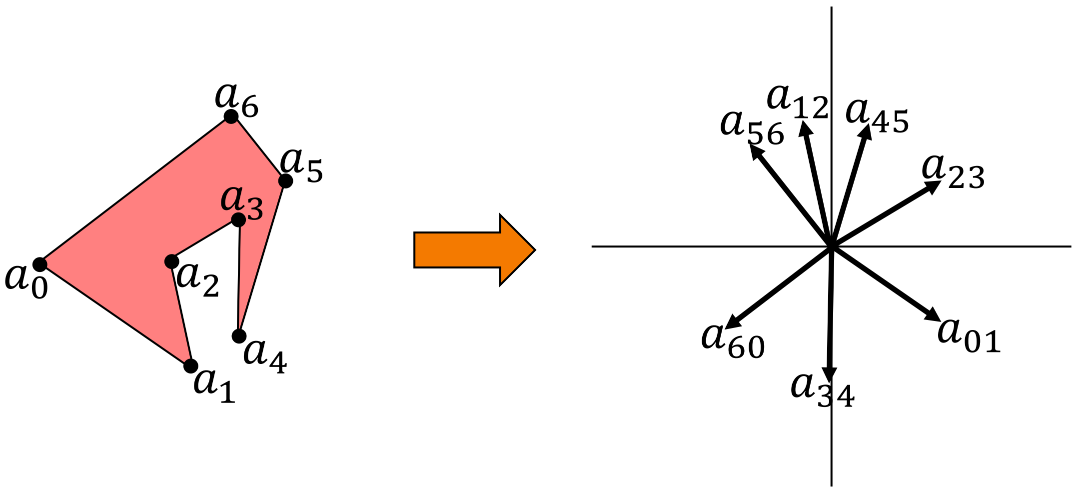
$\Leftrightarrow \exists a \in A, b \in B$ such that $\tau = b - \mathrm{a}$.

$\Leftrightarrow \tau \in B \oplus (-A)$.

# Motion Planning

Definition:

Given a polygon $A=\{a_0,\ldots,a_{n-1}\}$, the *star diagram* is the mapping from the edges of $P$ to points on the unit circle (based on the direction of the edges).

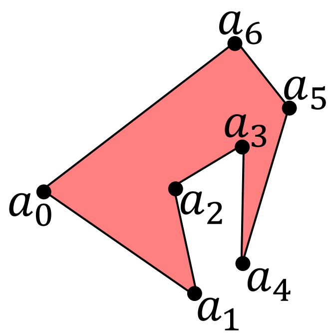


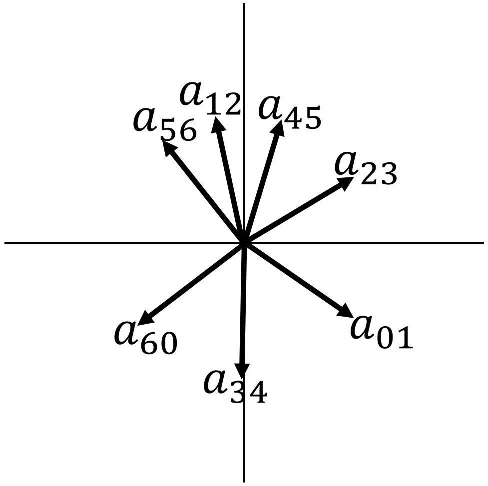
# Motion Planning

Definition:

Given a polygon $A = \{a_0, \ldots, a_{n-1}\}$, the *star diagram* is the mapping from the edges of $P$ to points on the unit circle (based on the direction of the edges).

Can think of this as a discrete Gauss map.

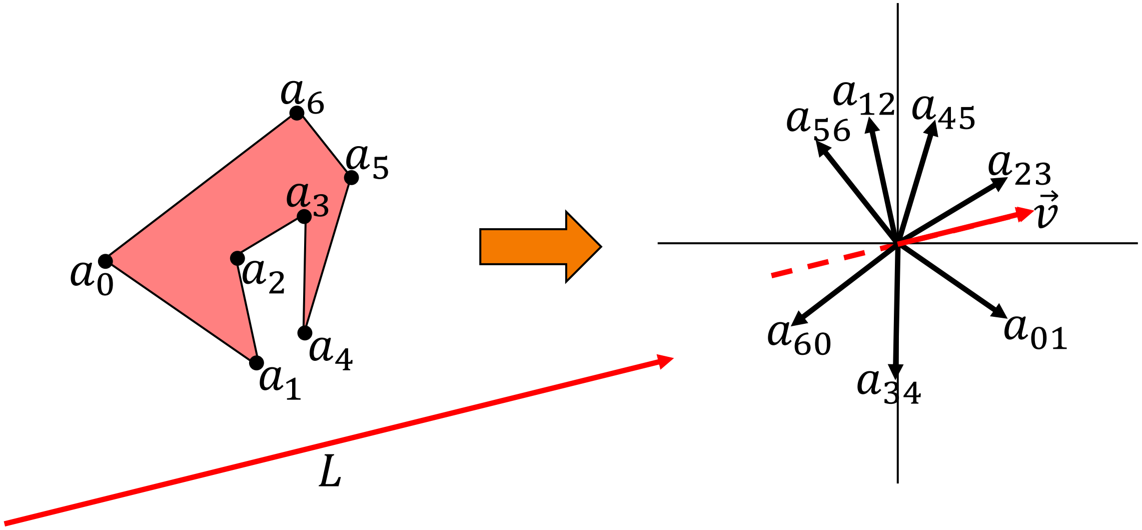


# Motion Planning

Note:

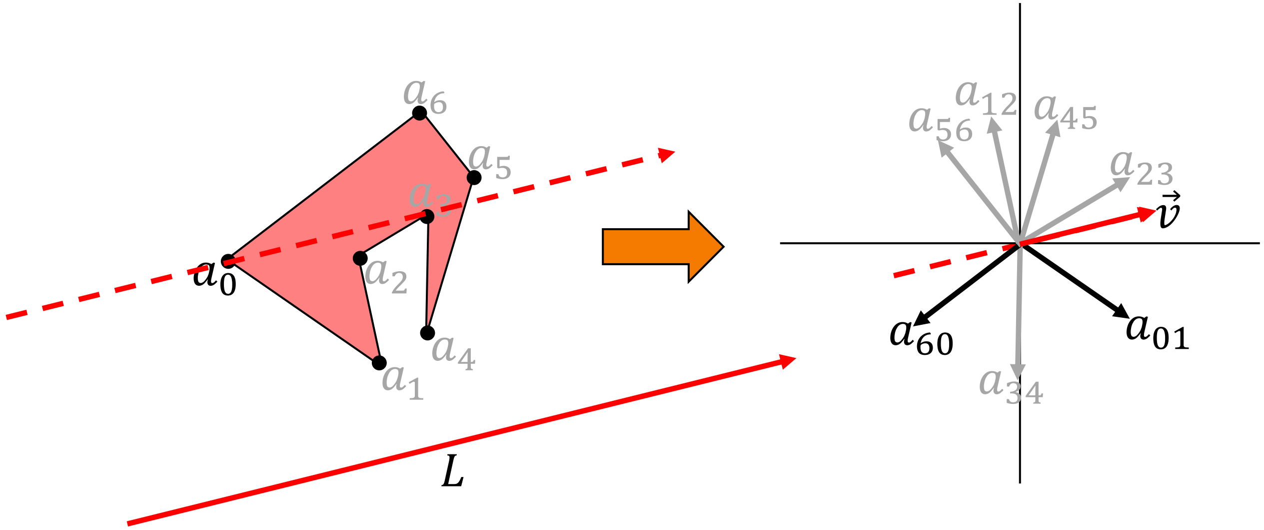
Given a (directed) line $L$ with direction $\vec{v}$, the vertex $a_i \in A$ is locally extreme w.r.t. $\vec{v}$ if the directions $a_{i-1,i}$ and $a_{i,i+1}$ are on opposite sides of $\vec{v}$.



# Motion Planning

Note:

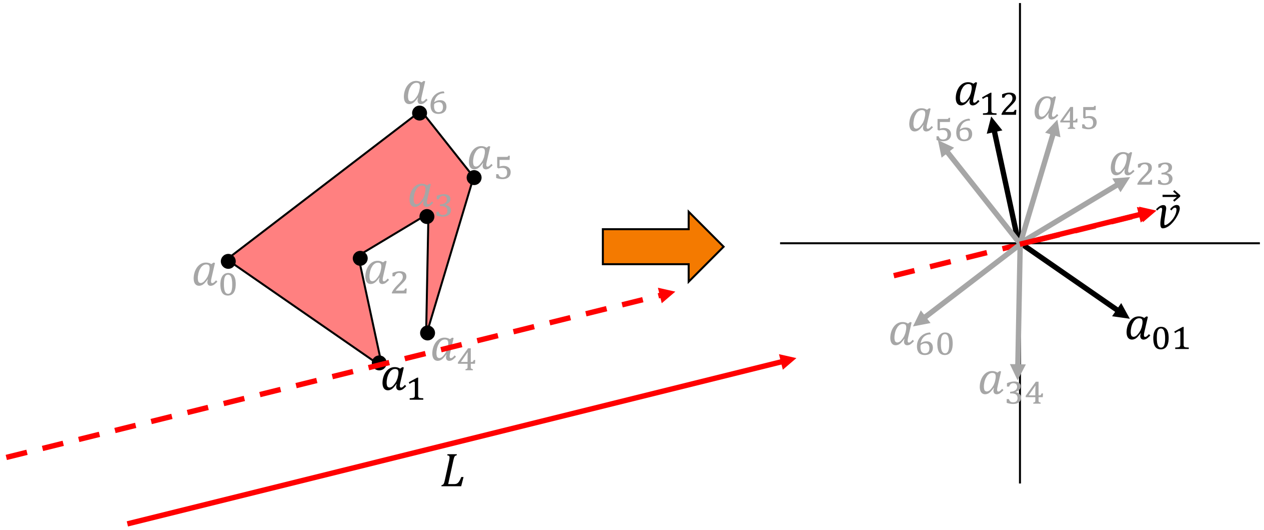
Given a (directed) line $L$ with direction $\vec{v}$, the vertex $a_i \in A$ is locally extreme w.r.t. $\vec{v}$ if the directions $a_{i-1,i}$ and $a_{i,i+1}$ are on opposite sides of $\vec{v}$.



# Motion Planning

Note:

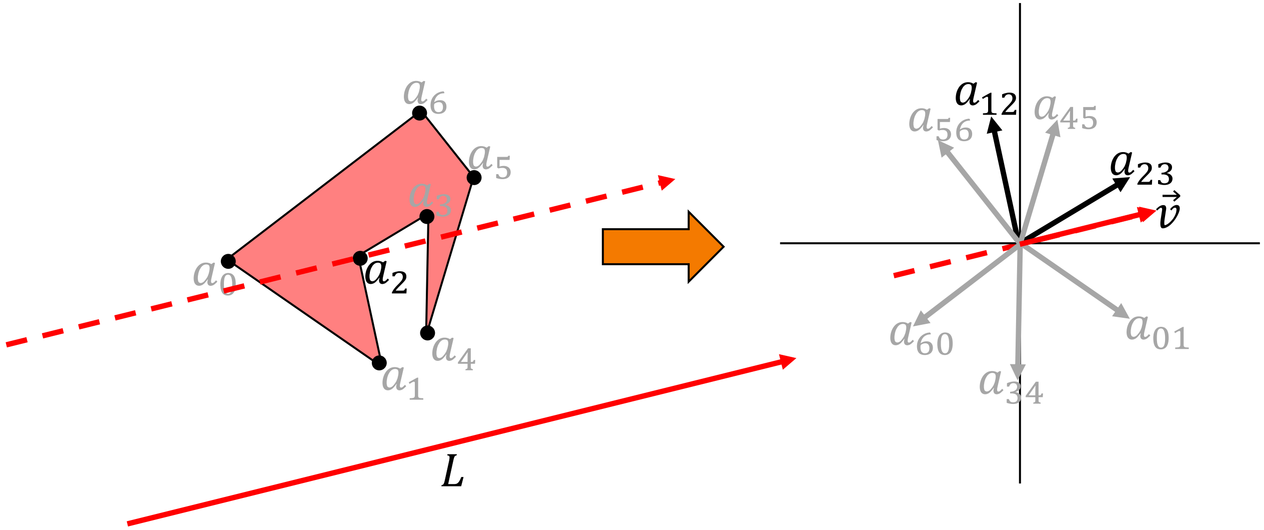
Given a (directed) line $L$ with direction $\vec{v}$, the vertex $a_i \in A$ is locally extreme w.r.t. $\vec{v}$ if the directions $a_{i-1,i}$ and $a_{i,i+1}$ are on opposite sides of $\vec{v}$.



# Motion Planning

Note:

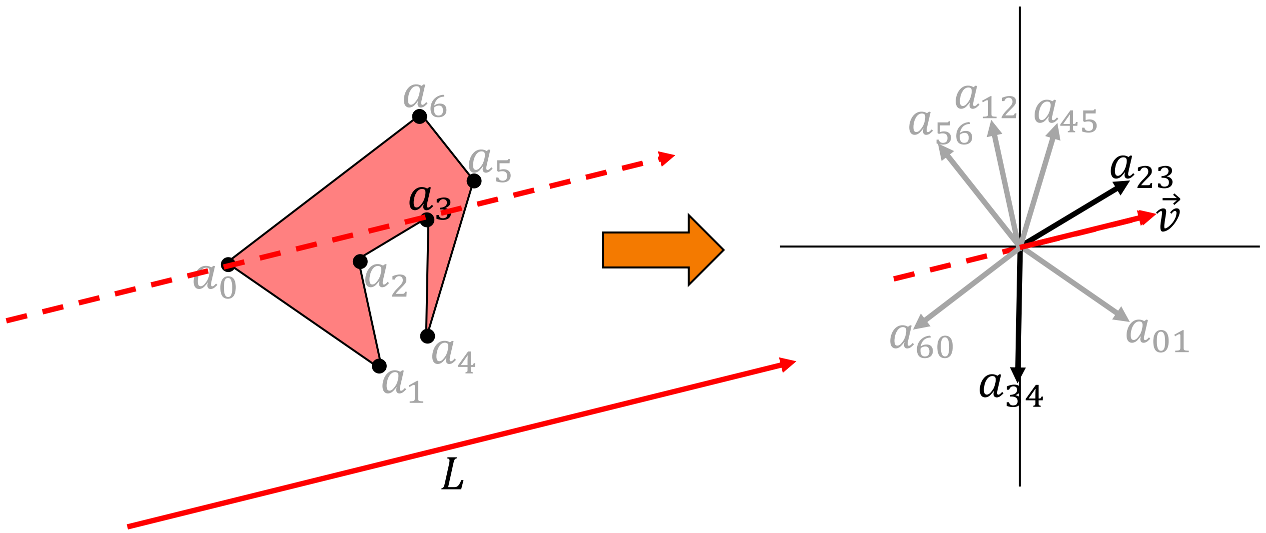
Given a (directed) line $L$ with direction $\vec{v}$, the vertex $a_i \in A$ is locally extreme w.r.t. $\vec{v}$ if the directions $a_{i-1,i}$ and $a_{i,i+1}$ are on opposite sides of $\vec{v}$.



# Motion Planning

Note:

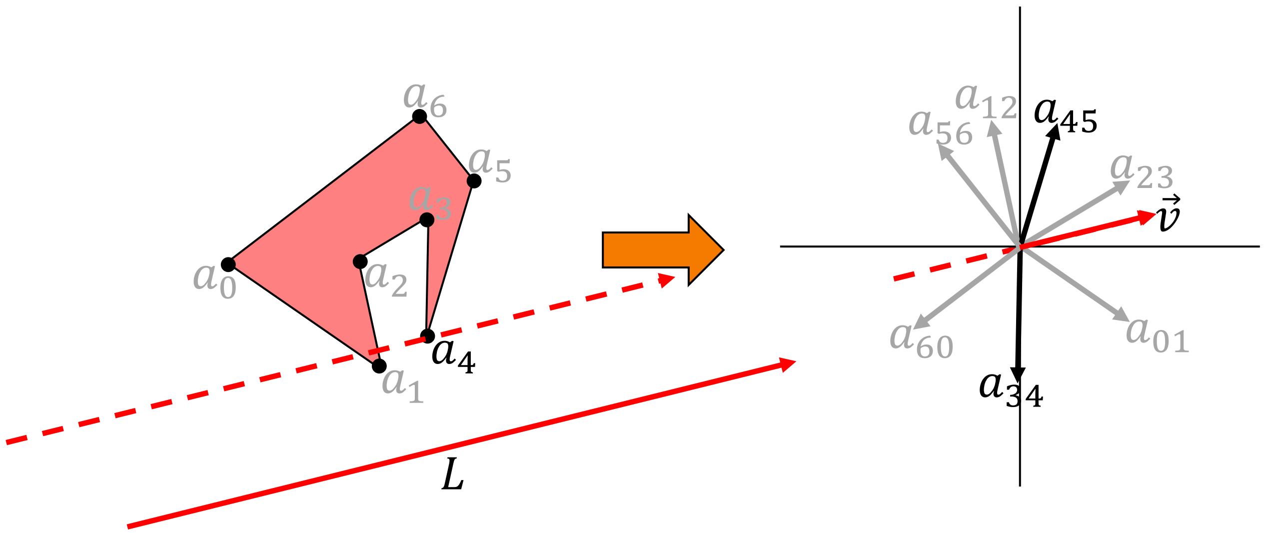
Given a (directed) line $L$ with direction $\vec{v}$, the vertex $a_i \in A$ is locally extreme w.r.t. $\vec{v}$ if the directions $a_{i-1,i}$ and $a_{i,i+1}$ are on opposite sides of $\vec{v}$.



# Motion Planning

<u>Note</u>:

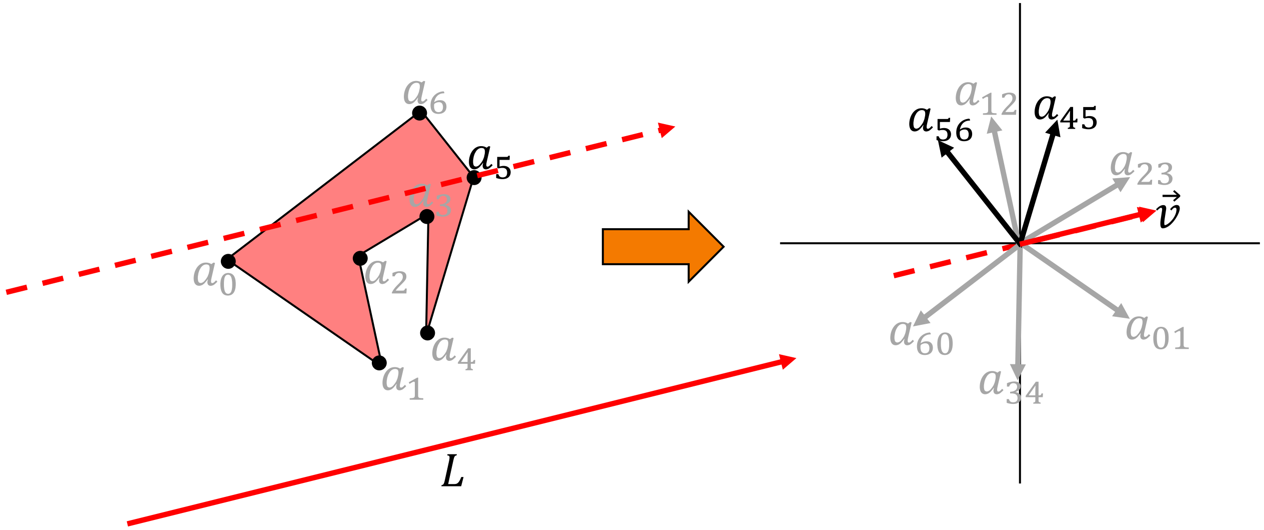
Given a (directed) line $L$ with direction $\vec{v}$, the vertex $a_i \in A$ is locally extreme w.r.t. $\vec{v}$ if the directions $a_{i-1,i}$ and $a_{i,i+1}$ are on opposite sides of $\vec{v}$.



# Motion Planning

Note:

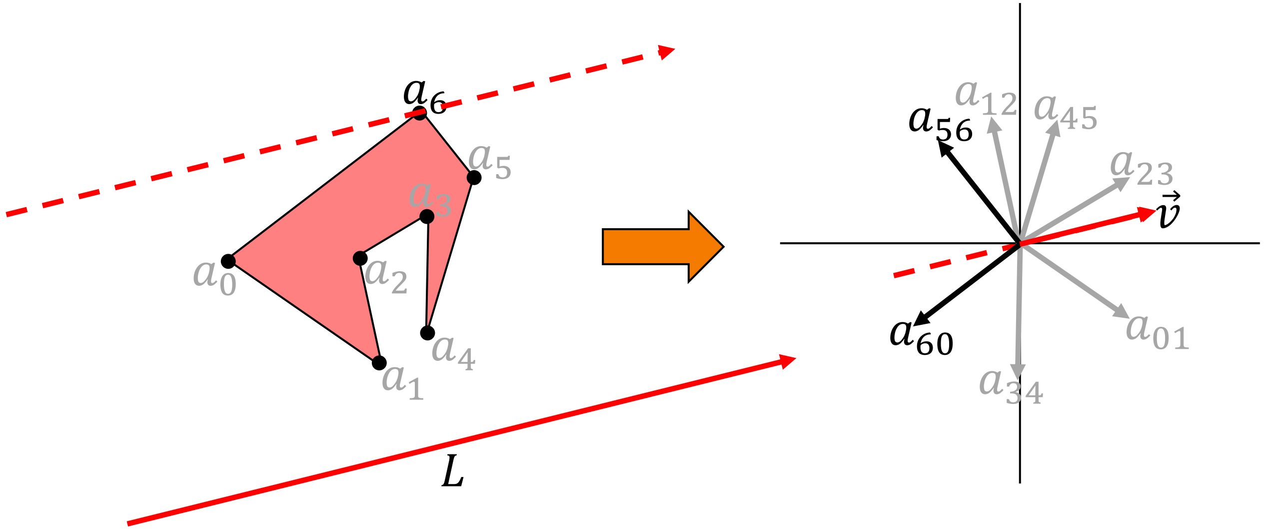
Given a (directed) line $L$ with direction $\vec{v}$, the vertex $a_i \in A$ is locally extreme w.r.t. $\vec{v}$ if the directions $a_{i-1,i}$ and $a_{i,i+1}$ are on opposite sides of $\vec{v}$.



# Motion Planning

Note:

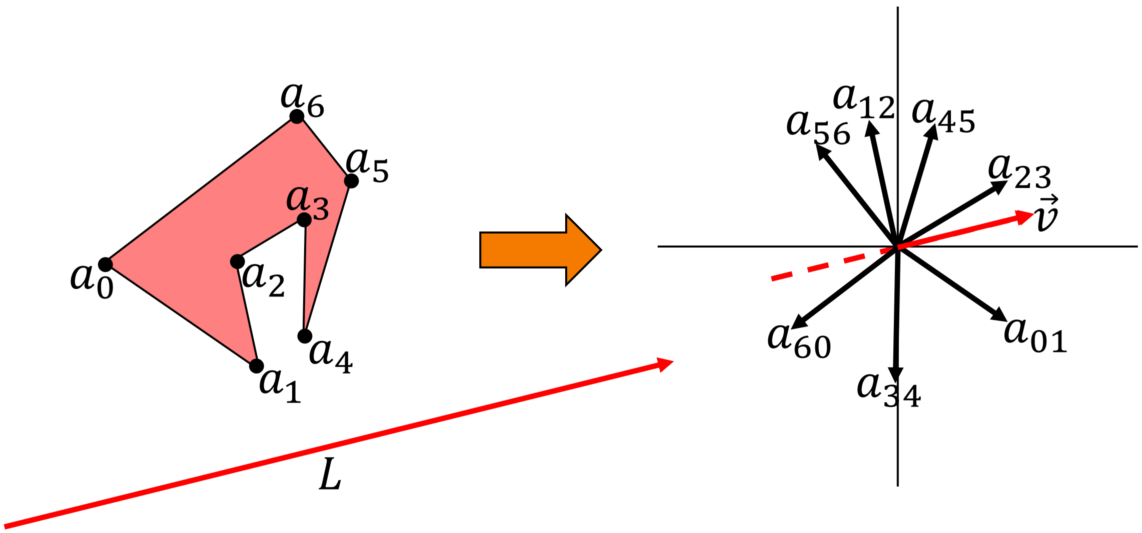
Given a (directed) line $L$ with direction $\vec{v}$, the vertex $a_i \in A$ is locally extreme w.r.t. $\vec{v}$ if the directions $a_{i-1,i}$ and $a_{i,i+1}$ are on opposite sides of $\vec{v}$.



# Motion Planning

<u>Note</u>:

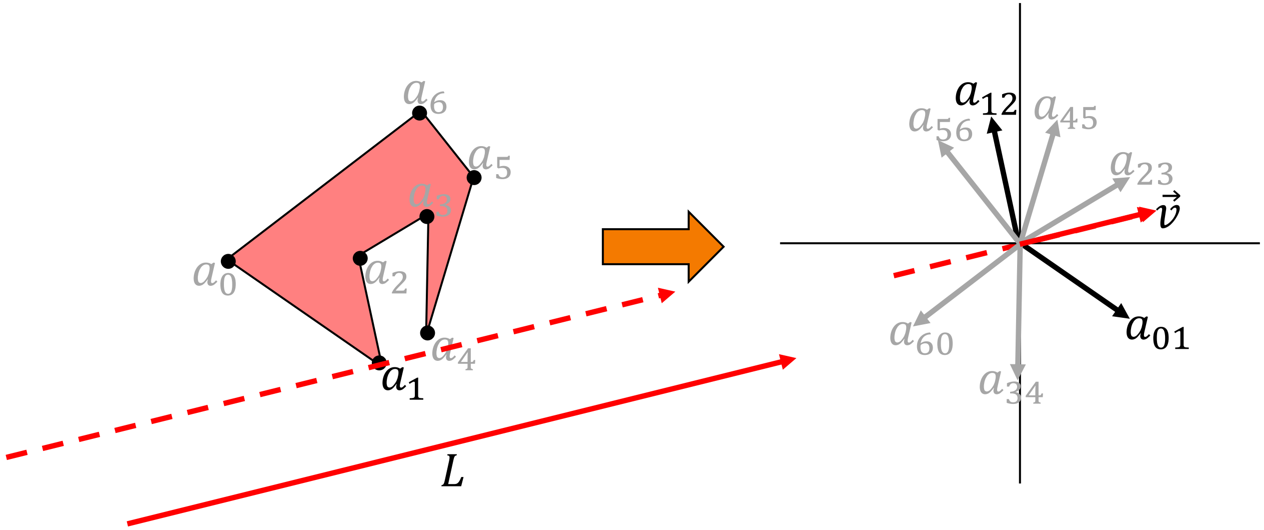
Given a (directed) line $L$ with direction $\vec{v}$, the vertex $a_i \in A$ is locally right-most w.r.t. $\vec{v}$ if it is locally extreme and the incoming edge $\overrightarrow{a_{i-1}a_i}$ points to the right of $\vec{v}$.



# Motion Planning

Note:

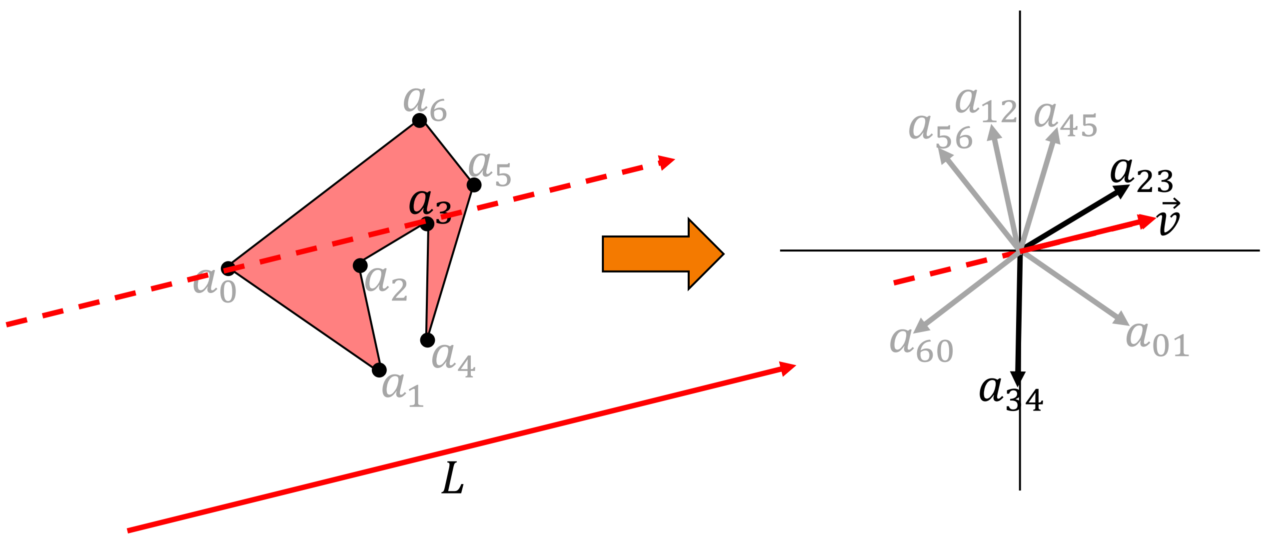
Given a (directed) line $L$ with direction $\vec{v}$, the vertex $a_i \in A$ is locally right-most w.r.t. $\vec{v}$ if it is locally extreme and the incoming edge $\overrightarrow{a_{i-1}a_i}$ points to the right of $\vec{v}$.



# Motion Planning

Note:

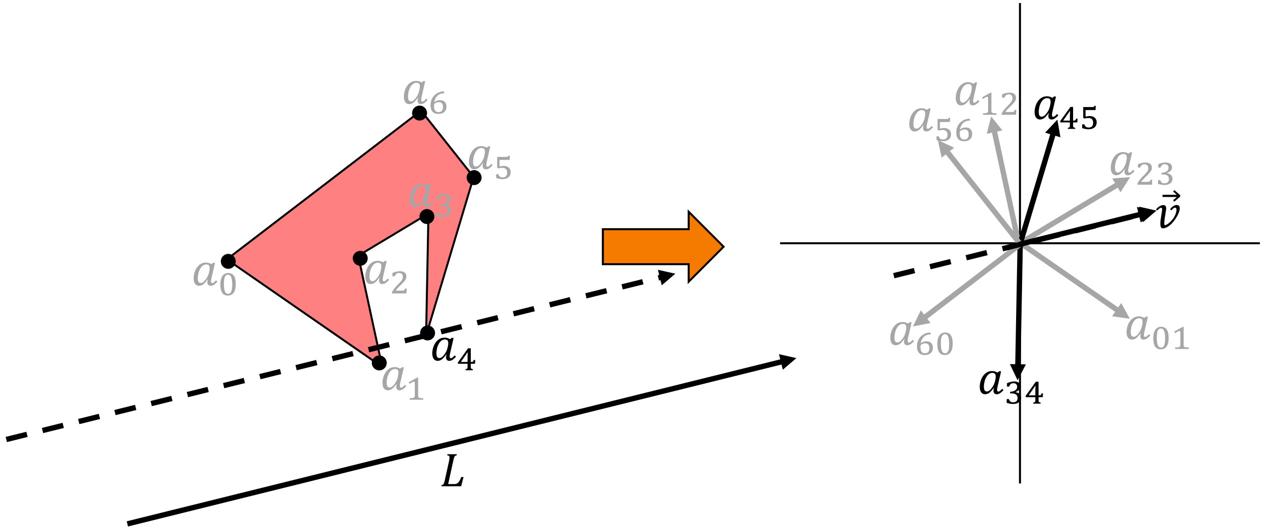
Given a (directed) line $L$ with direction $\vec{v}$, the vertex $a_i \in A$ is locally right-most w.r.t. $\vec{v}$ if it is locally extreme and the incoming edge $\overrightarrow{a_{i-1}a_i}$ points to the right of $\vec{v}$.



# Motion Planning

<u>Note</u>:

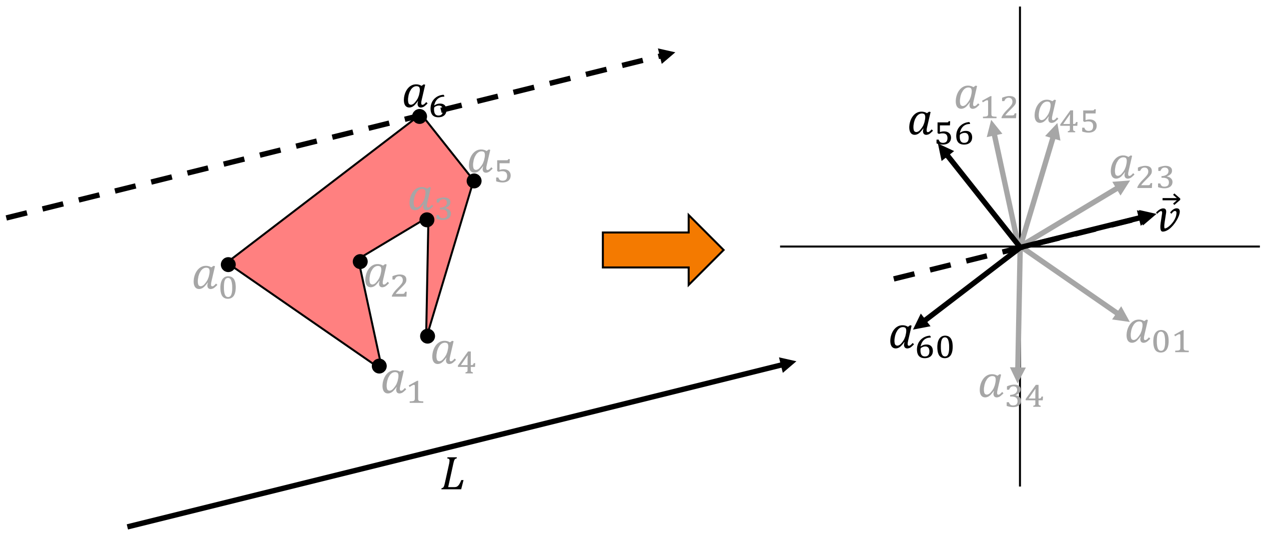
Given a (directed) line $L$ with direction $\vec{v}$, the vertex $a_i \in A$ is locally right-most w.r.t. $\vec{v}$ if it is locally extreme and the incoming edge $\overrightarrow{a_{i-1}a_i}$ points to the right of $\vec{v}$.



# Motion Planning

Note:

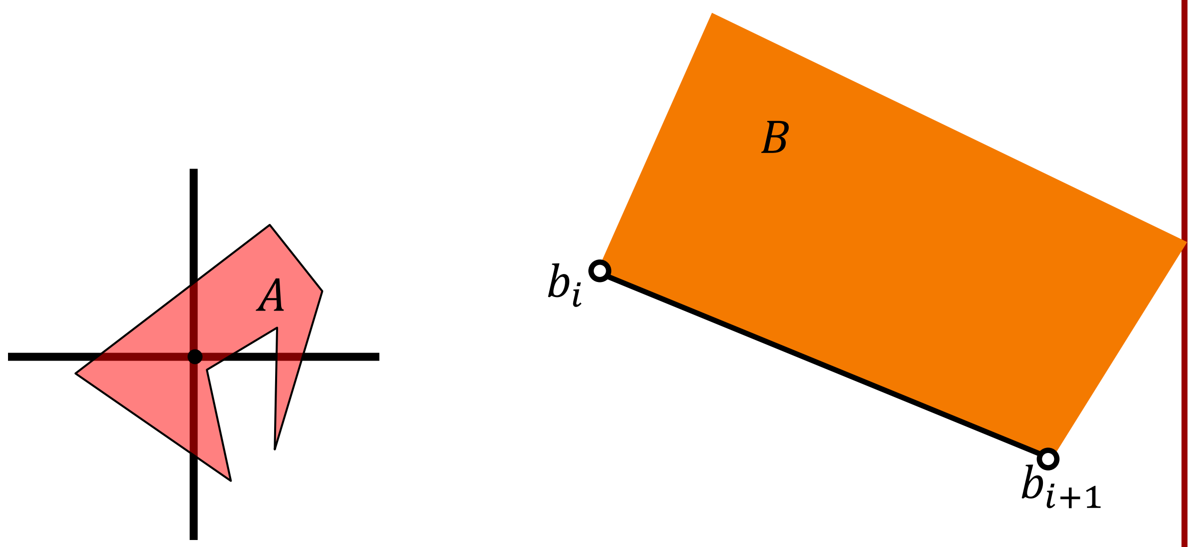
Given a (directed) line $L$ with direction $\vec{v}$, the vertex $a_i \in A$ is locally right-most w.r.t. $\vec{v}$ if it is locally extreme and the incoming edge $\overrightarrow{a_{i-1}a_i}$ points to the right of $\vec{v}$.



# Motion Planning

Key Idea 1:

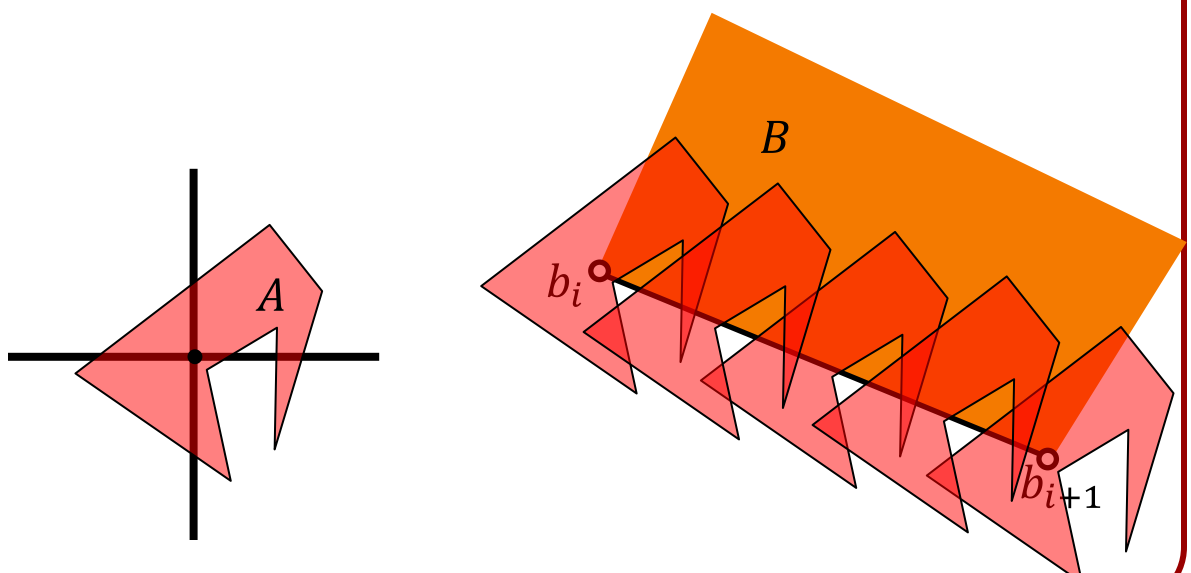
When sweeping $-A$ along an edge $\overrightarrow{b_i b_{i+1}}$, the boundary of $B \oplus (-A)$ is swept out by the vertices of $-A$ that are locally right-most with respect to the segment $\overrightarrow{b_i b_{i+1}}$.



# Motion Planning

Key Idea 1:

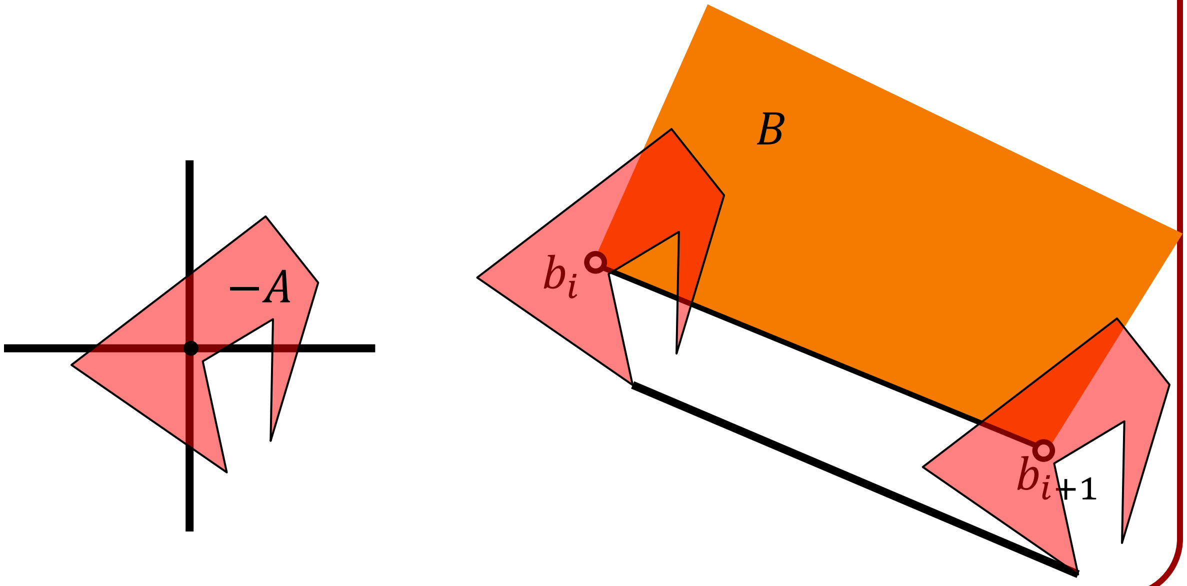
When sweeping $-A$ along an edge $\overrightarrow{b_i b_{i+1}}$, the boundary of $B \oplus (-A)$ is swept out by the vertices of $-A$ that are locally right-most with respect to the segment $\overrightarrow{b_i b_{i+1}}$.



# Motion Planning

Key Idea 1:

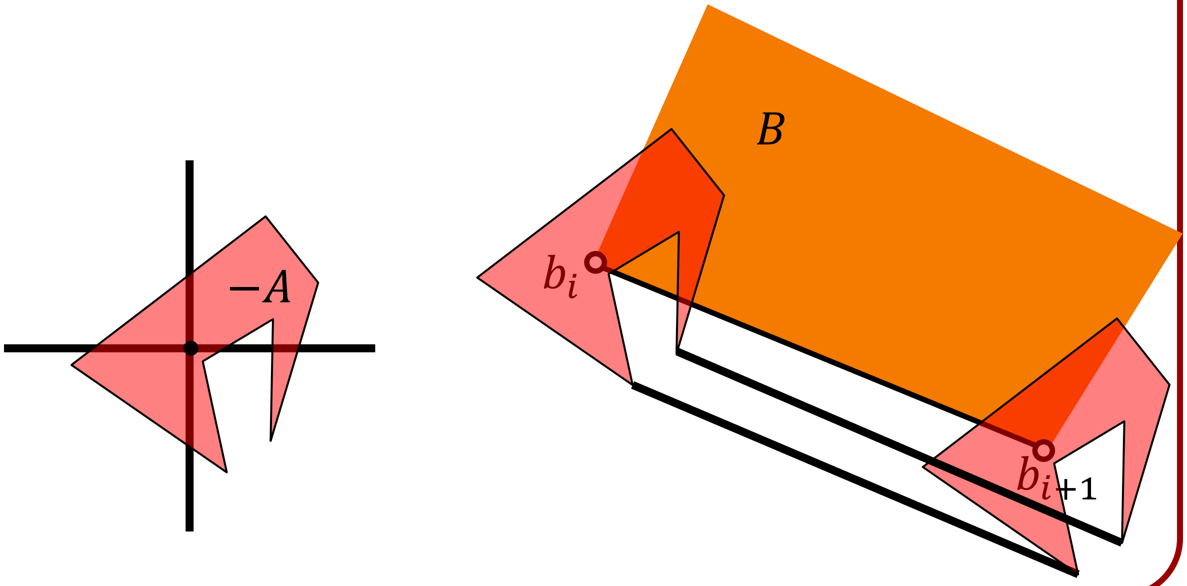
When sweeping $-A$ along an edge $\overrightarrow{b_i b_{i+1}}$, the boundary of $B \oplus (-A)$ is swept out by the vertices of $-A$ that are locally right-most with respect to the segment $\overrightarrow{b_i b_{i+1}}$.



# Motion Planning

Key Idea 1:

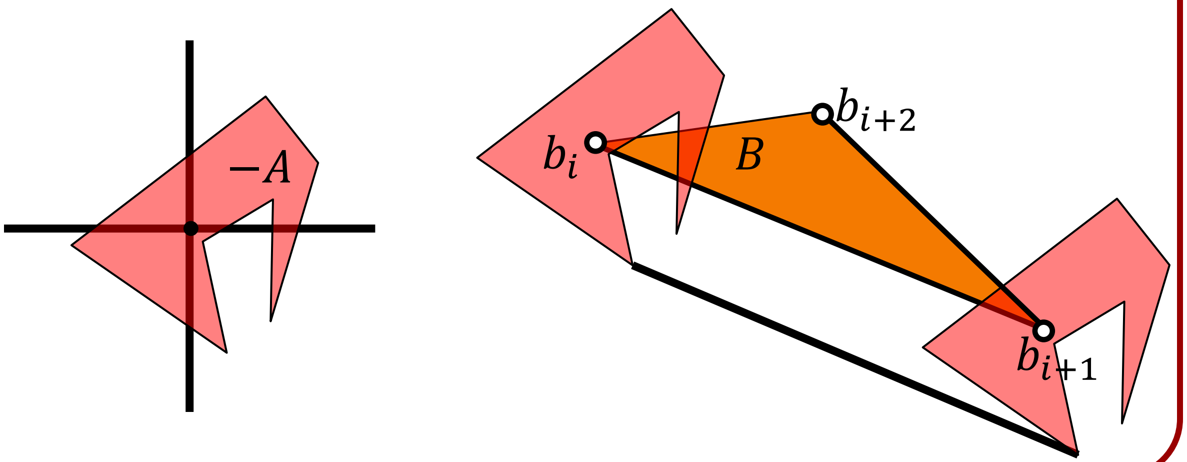
When sweeping $-A$ along an edge $\overrightarrow{b_i b_{i+1}}$, the boundary of $B \oplus (-A)$ is swept out by the vertices of $-A$ that are locally right-most with respect to the segment $\overrightarrow{b_i b_{i+1}}$.



# Motion Planning

<u>Key Idea 2</u>:

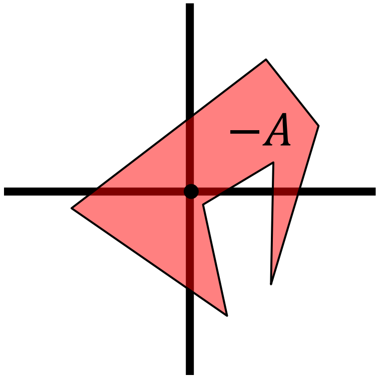
As we turn the corner from $\overrightarrow{b_i b_{i+1}}$ to $\overrightarrow{b_{i+1} b_{i+2}}$, the boundary $B \oplus (-A)$ is swept by the part of the boundary of $-A$ between the two locally right-most vertices.

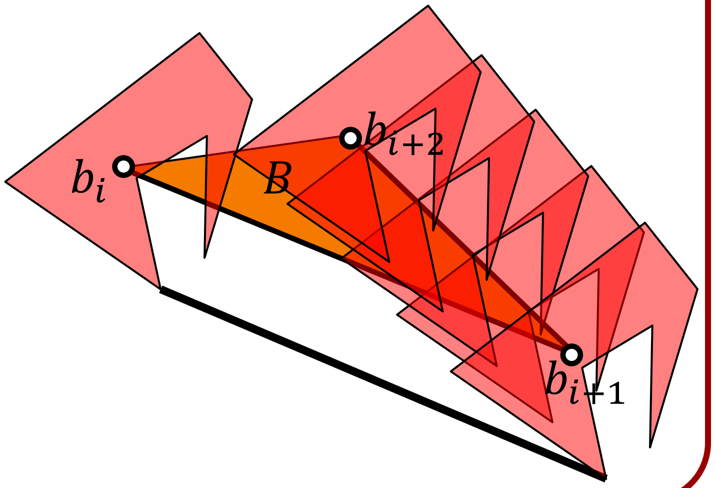


# Motion Planning

Key Idea 2:

As we turn the corner from $\overrightarrow{b_i b_{i+1}}$ to $\overrightarrow{b_{i+1} b_{i+2}}$, the boundary $B \oplus (-A)$ is swept by the part of the boundary of $-A$ between the two locally right-most vertices.

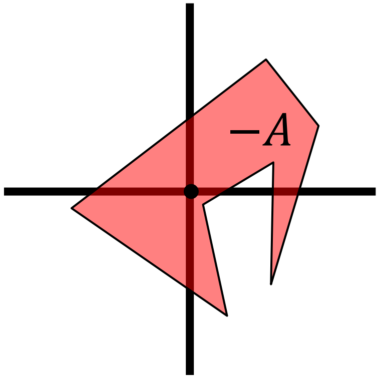


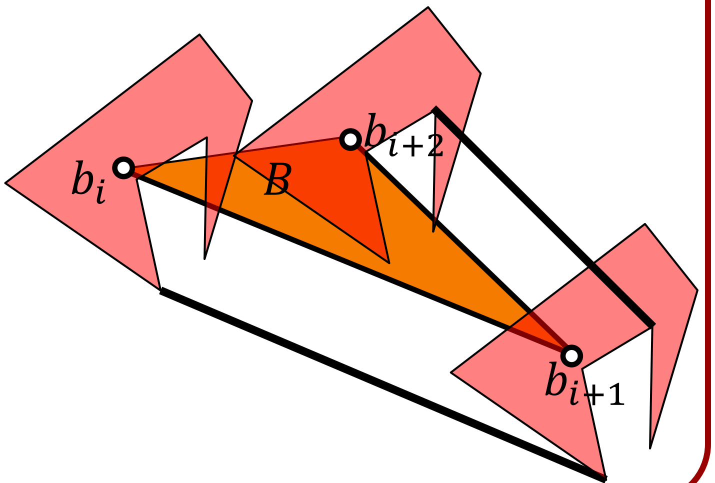



# Motion Planning

Key Idea 2:

As we turn the corner from $\overrightarrow{b_i b_{i+1}}$ to $\overrightarrow{b_{i+1} b_{i+2}}$, the boundary $B \oplus (-A)$ is swept by the part of the boundary of $-A$ between the two locally right-most vertices.

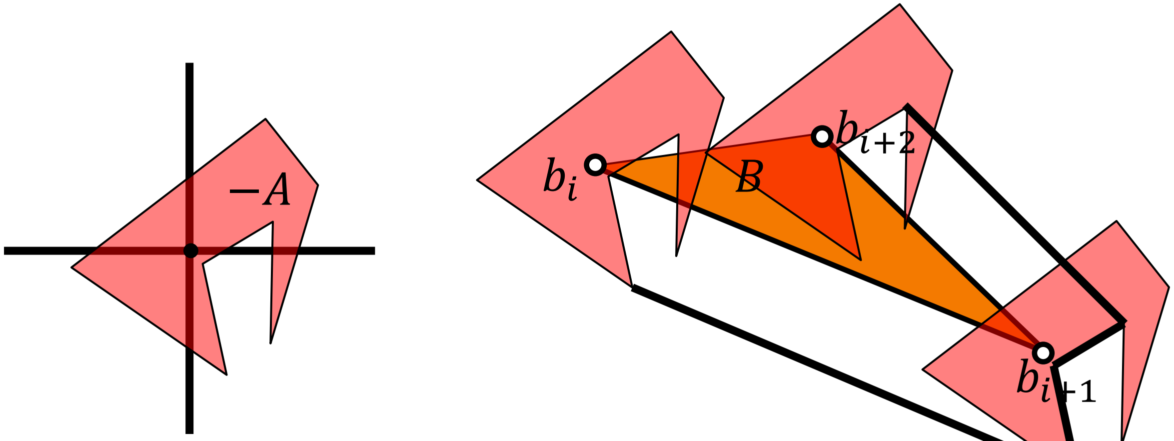


# Motion Planning

Key Idea 2:

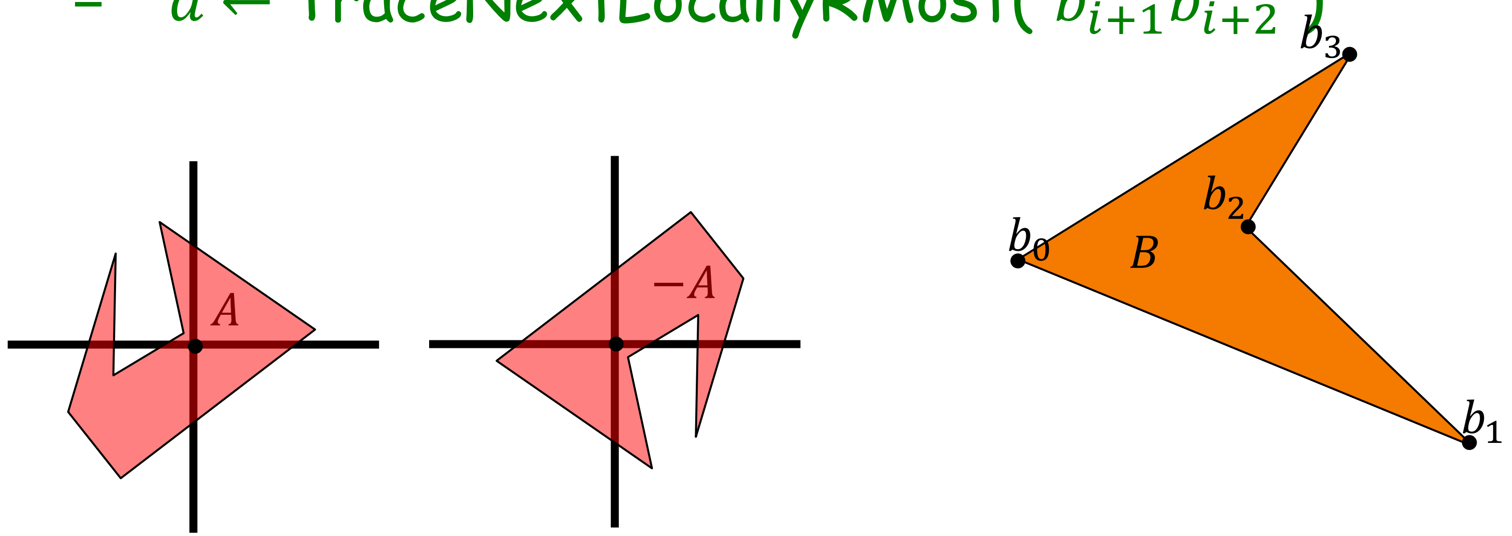
As we turn the corner from $\overrightarrow{b_i b_{i+1}}$ to $\overrightarrow{b_{i+1} b_{i+2}}$, the boundary $B \oplus (-A)$ is swept by the part of the boundary of $-A$ between the two locally right-most vertices.



# Motion Planning

MinkowskiTrace( $A$ , $B$ )

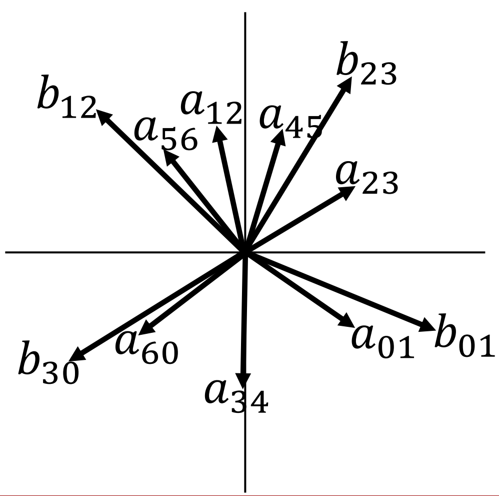
- Place $-A$ at vertex $b_0 \in B$.
- for each $a \in$ LocallyRMost( $\overrightarrow{b_0 b_1}$ )
  - for $i \in [0, |B|)$:
    - Trace $a$ along edge( $\overrightarrow{b_i b_{i+1}}$ )
    - $a \leftarrow$ TraceNextLocallyRMost( $\overrightarrow{b_{i+1} b_{i+2}}$ )

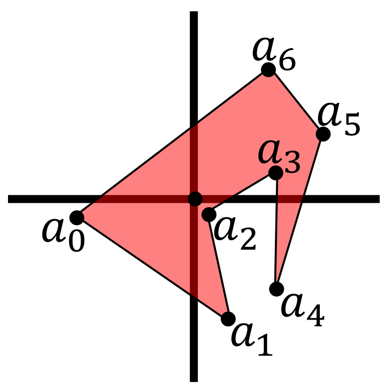


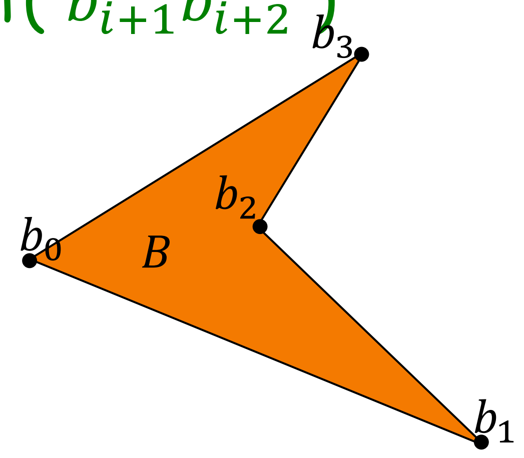
# Motion Planning

MinkowskiTrace( $A$ , $B$ )

- Place $-A$ at vertex $b_0 \in B$.
- for each $a \in$ LocallyRMost( $\overrightarrow{b_0 b_1}$ )
  - for $i \in [0, |B|)$:
    - Trace $a$ along edge( $\overrightarrow{b_i b_{i+1}}$ )
    - $a \leftarrow$ TraceNextLocallyRMost( $\overrightarrow{b_{i+1} b_{i+2}}$ )



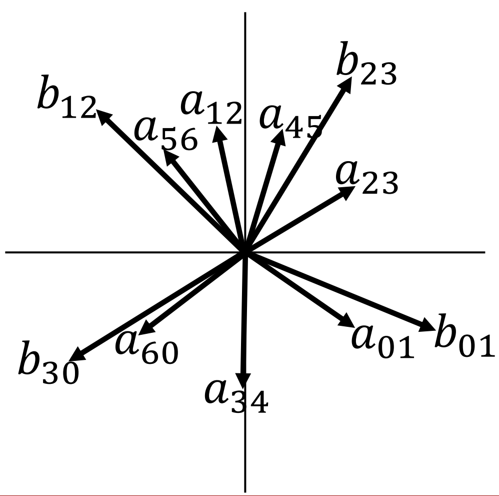



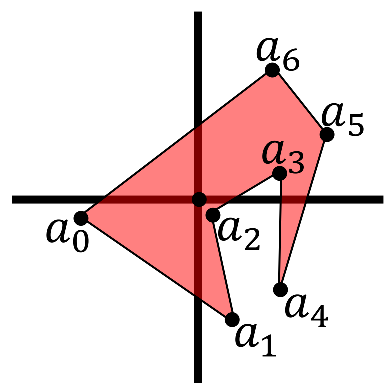



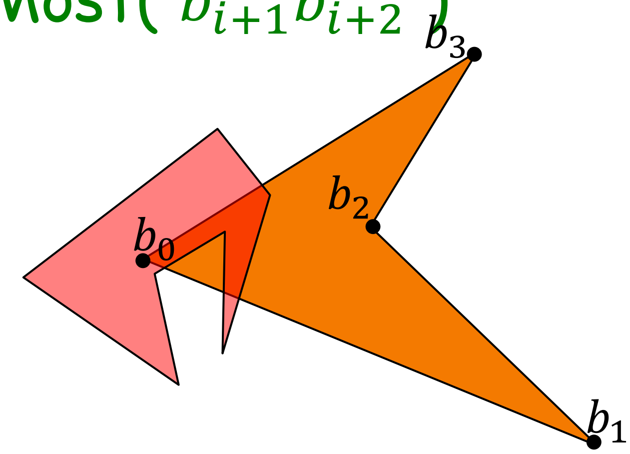
# Motion Planning

MinkowskiTrace( $A$ , $B$ )

- Place $-A$ at vertex $b_0 \in B$.
- for each $a \in$ LocallyRMost( $\overrightarrow{b_0 b_1}$ )
  - for $i \in [0, |B|)$:
    - Trace $a$ along edge( $\overrightarrow{b_i b_{i+1}}$ )
    - $a \leftarrow$ TraceNextLocallyRMost( $\overrightarrow{b_{i+1} b_{i+2}}$ )



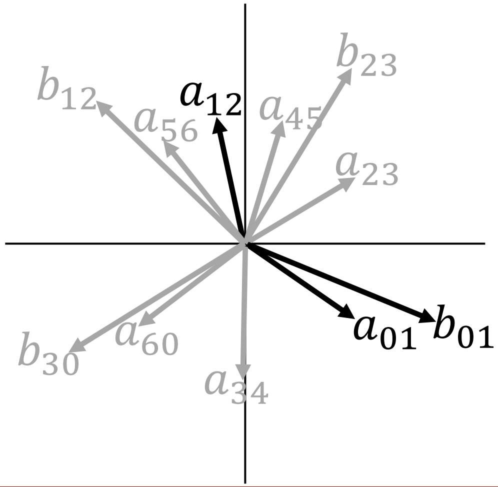



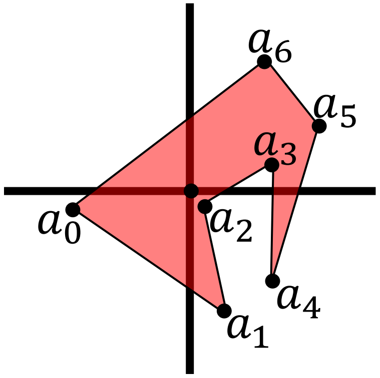



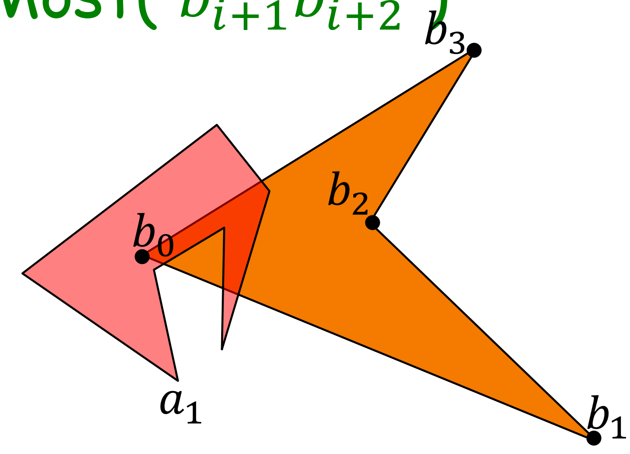
# Motion Planning

MinkowskiTrace( $A$ , $B$ )

- Place $-A$ at vertex $b_0 \in B$.
- for each $a \in$ LocallyRMost( $\overrightarrow{b_0 b_1}$ )
  - » for $i \in [0, |B|)$:
    - – Trace $a$ along edge( $\overrightarrow{b_i b_{i+1}}$ )
    - – $a \leftarrow$ TraceNextLocallyRMost( $\overrightarrow{b_{i+1} b_{i+2}}$ )



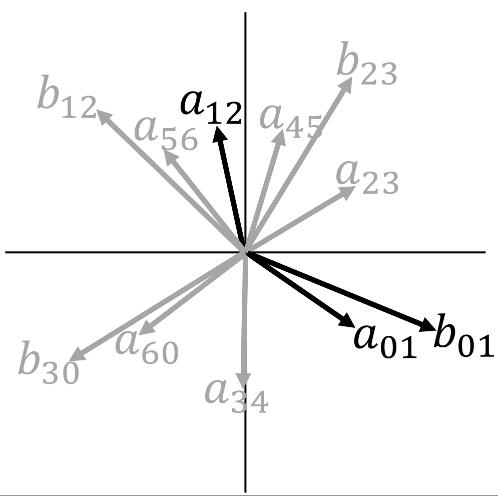



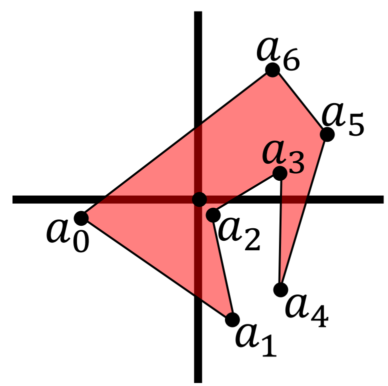



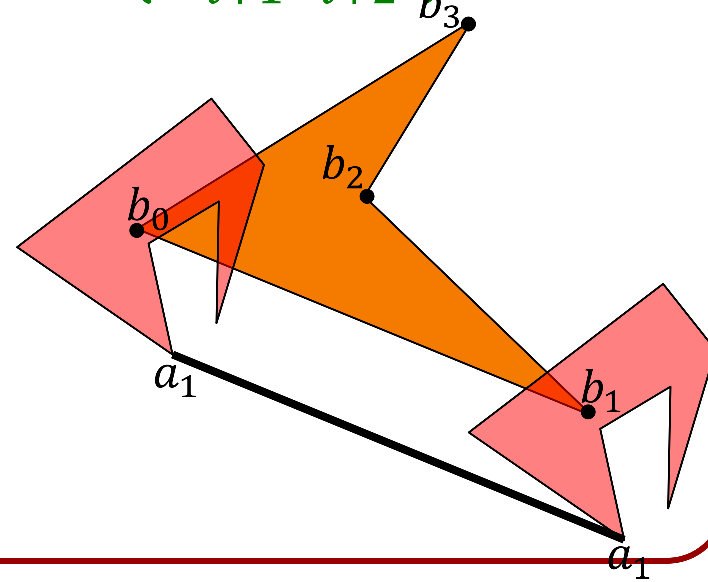
# Motion Planning

MinkowskiTrace( $A$ , $B$ )

- Place $-A$ at vertex $b_0 \in B$.
- for each $a \in$ LocallyRMost( $\overrightarrow{b_0 b_1}$ )
  - for $i \in [0, |B|)$:
    - Trace $a$ along edge( $\overrightarrow{b_i b_{i+1}}$ )
    - $a \leftarrow$ TraceNextLocallyRMost( $\overrightarrow{b_{i+1} b_{i+2}}$ )



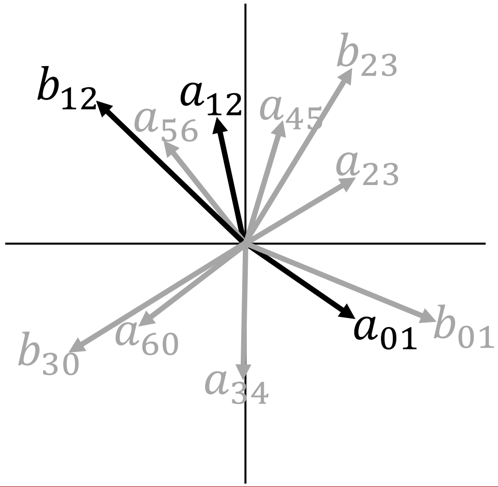



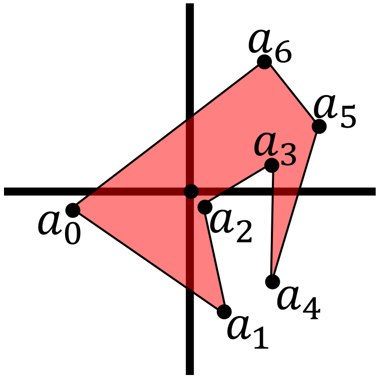



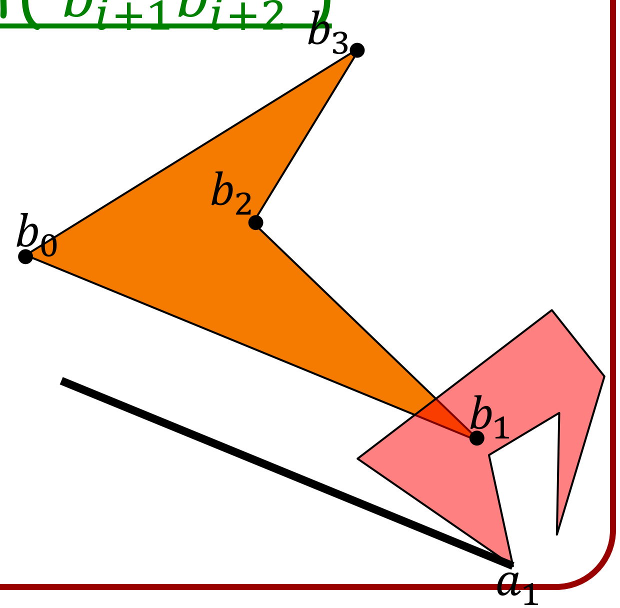
# Motion Planning

MinkowskiTrace( $A$ , $B$ )

- Place $-A$ at vertex $b_0 \in B$.
- for each $a \in$ LocallyRMost( $\overrightarrow{b_0 b_1}$ )
  - for $i \in [0, |B|)$:
    - Trace $a$ along edge( $\overrightarrow{b_i b_{i+1}}$ )
    - $a \leftarrow$ TraceNextLocallyRMost( $\overrightarrow{b_{i+1} b_{i+2}}$ )



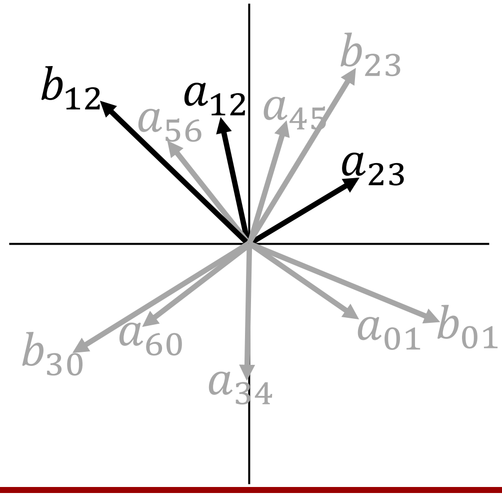



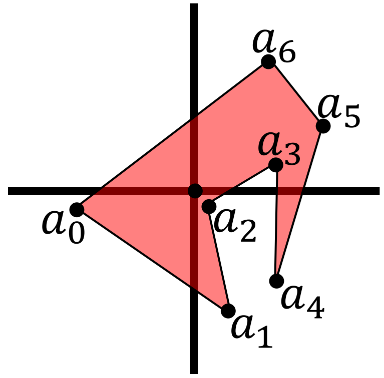



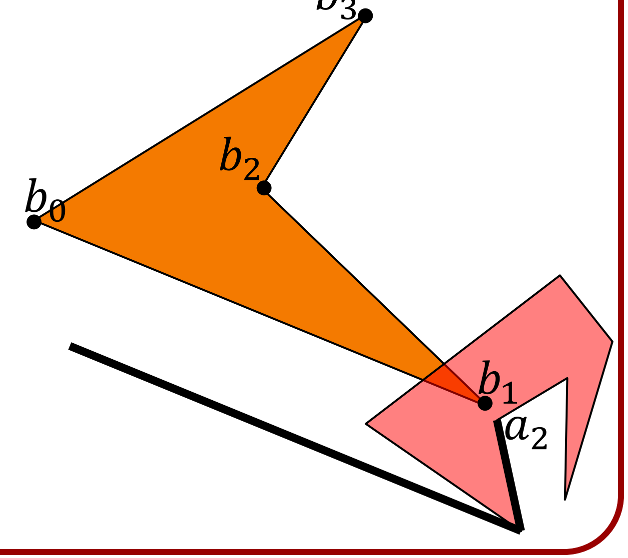
# Motion Planning

MinkowskiTrace( $A$ , $B$ )

- Place $-A$ at vertex $b_0 \in B$.
- for each $a \in$ LocallyRMost( $\overrightarrow{b_0 b_1}$ )
  - for $i \in [0, |B|)$:
    - Trace $a$ along edge( $\overrightarrow{b_i b_{i+1}}$ )
    - $a \leftarrow$ TraceNextLocallyRMost( $\overrightarrow{b_{i+1} b_{i+2}}$ )



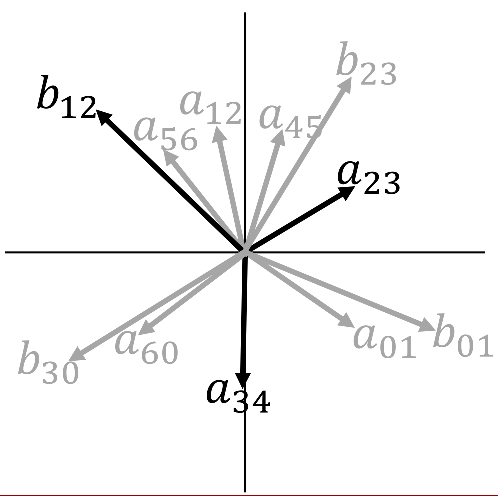



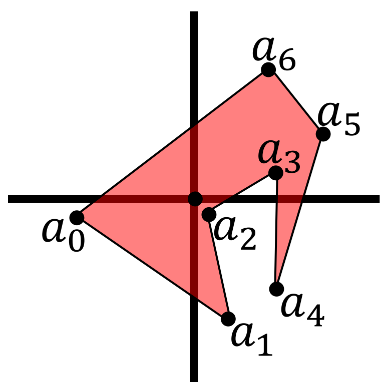



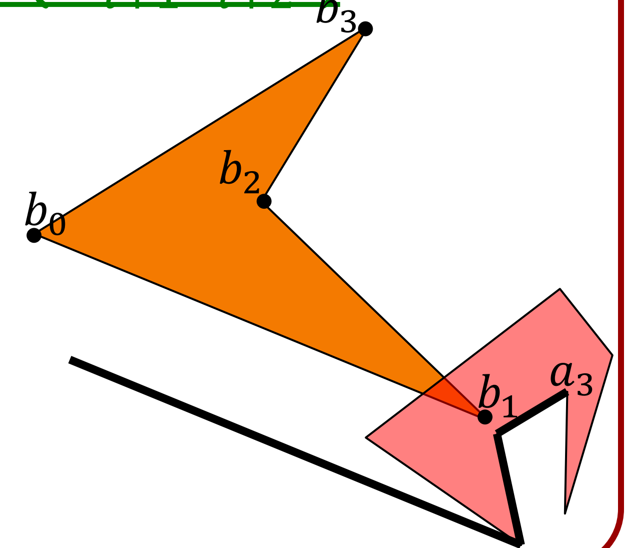
# Motion Planning

MinkowskiTrace( $A$ , $B$ )

- Place $-A$ at vertex $b_0 \in B$.
- for each $a \in$ LocallyRMost( $\overrightarrow{b_0 b_1}$ )
  - for $i \in [0, |B|)$:
    - Trace $a$ along edge( $\overrightarrow{b_i b_{i+1}}$ )
    - $a \leftarrow$ TraceNextLocallyRMost( $\overrightarrow{b_{i+1} b_{i+2}}$ )



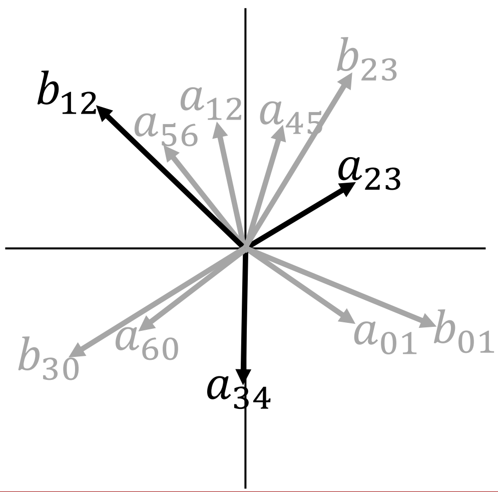



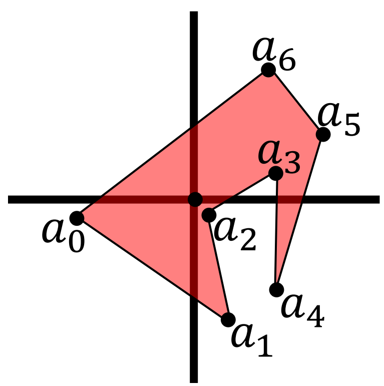



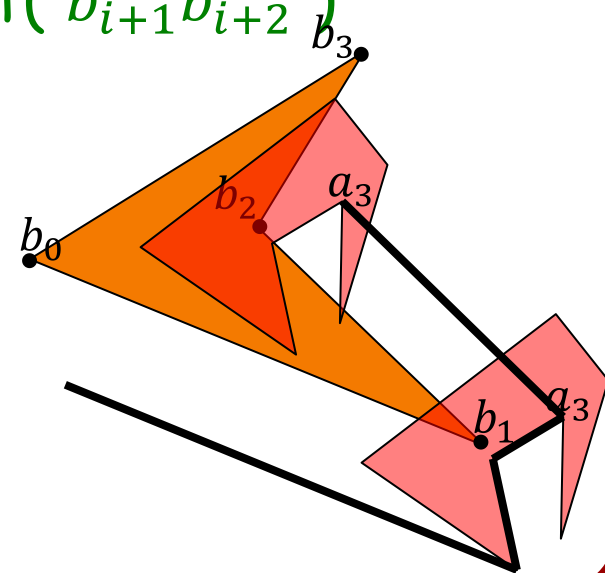
# Motion Planning

MinkowskiTrace( $A$ , $B$ )

- Place $-A$ at vertex $b_0 \in B$.
- for each $a \in$ LocallyRMost( $\overrightarrow{b_0 b_1}$ )
  - for $i \in [0, |B|)$:
    - <u>Trace $a$ along edge( $\overrightarrow{b_i b_{i+1}}$ )</u>
    - $a \leftarrow$ TraceNextLocallyRMost( $\overrightarrow{b_{i+1} b_{i+2}}$ )



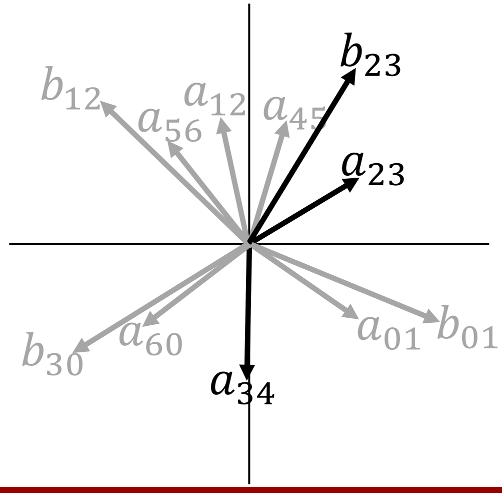



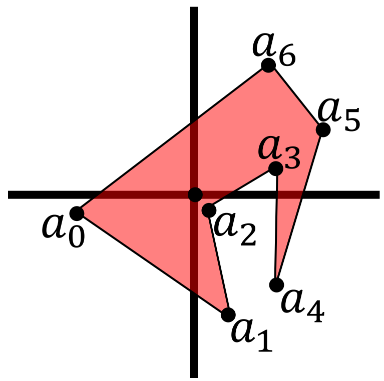



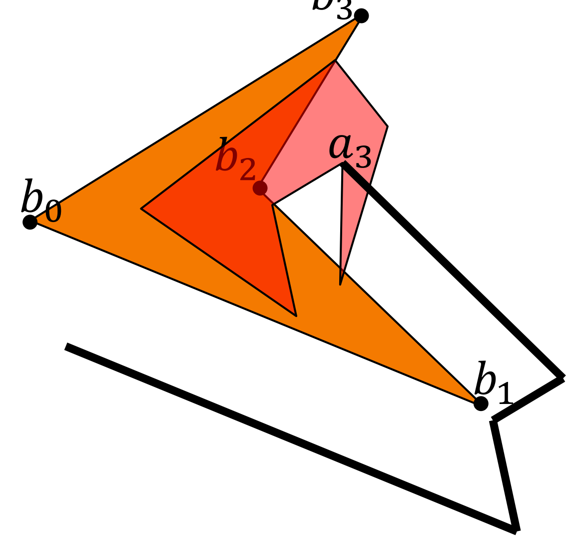
# Motion Planning

MinkowskiTrace( $A$ , $B$ )

- Place $-A$ at vertex $b_0 \in B$.
- for each $a \in$ LocallyRMost( $\overrightarrow{b_0 b_1}$ )
  - for $i \in [0, |B|)$:
    - Trace $a$ along edge( $\overrightarrow{b_i b_{i+1}}$ )
    - <u>$a \leftarrow$ TraceNextLocallyRMost( $\overrightarrow{b_{i+1} b_{i+2}}$ )</u>



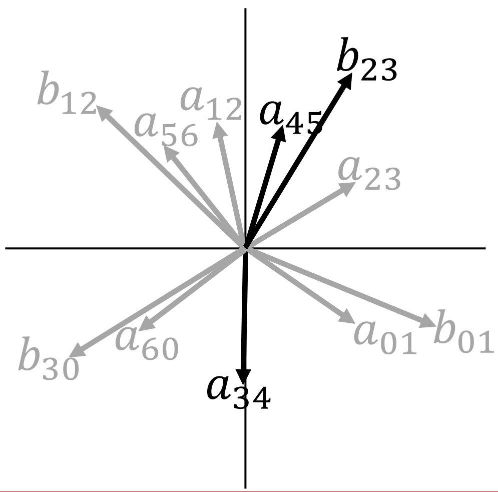



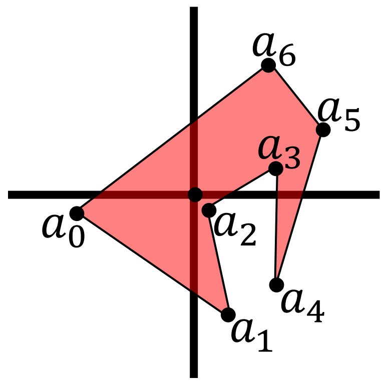



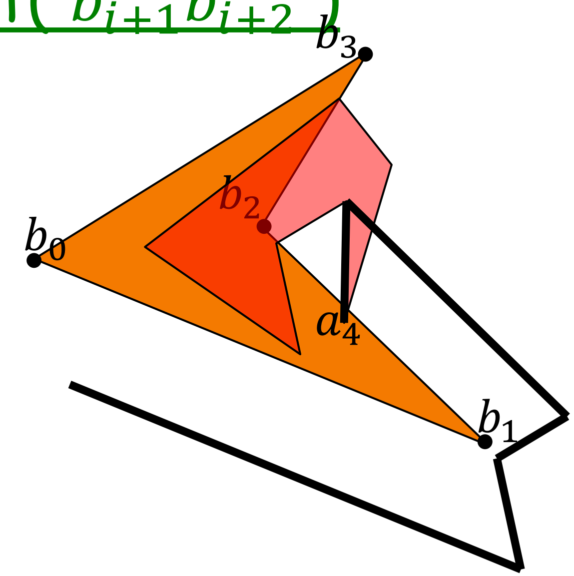
# Motion Planning

MinkowskiTrace( $A$ , $B$ )

- Place $-A$ at vertex $b_0 \in B$.
- for each $a \in$ LocallyRMost( $\overrightarrow{b_0 b_1}$ )
  - for $i \in [0, |B|)$:
    - Trace $a$ along edge( $\overrightarrow{b_i b_{i+1}}$ )
    - <u>$a \leftarrow$ TraceNextLocallyRMost( $\overrightarrow{b_{i+1} b_{i+2}}$ )</u>



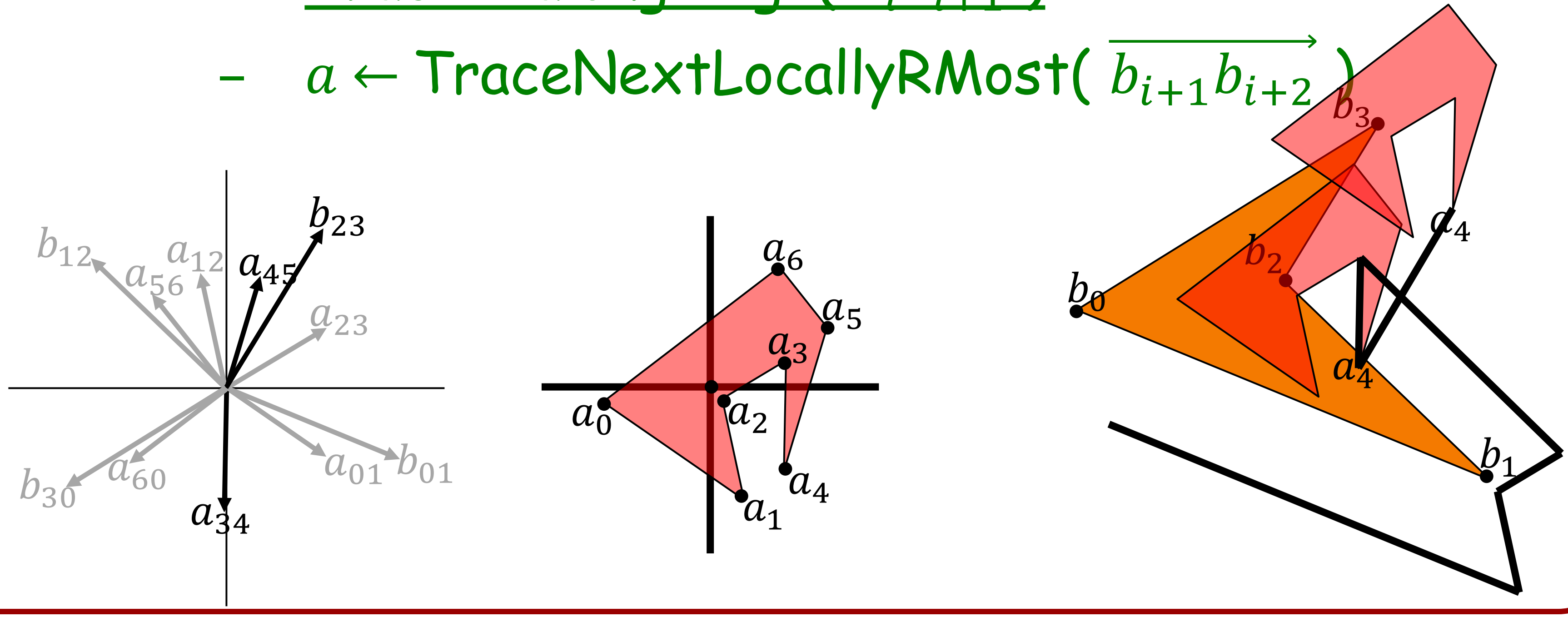



# Motion Planning

MinkowskiTrace( $A$ , $B$ )

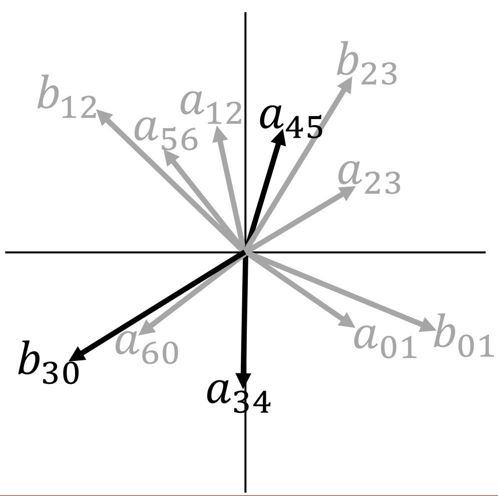
- Place $-A$ at vertex $b_0 \in B$.
- for each $a \in$ LocallyRMost( $\overrightarrow{b_0 b_1}$ )
  - for $i \in [0, |B|)$:
    - Trace $a$ along edge( $\overrightarrow{b_i b_{i+1}}$ )
    - $a \leftarrow$ TraceNextLocallyRMost( $\overrightarrow{b_{i+1} b_{i+2}}$ )

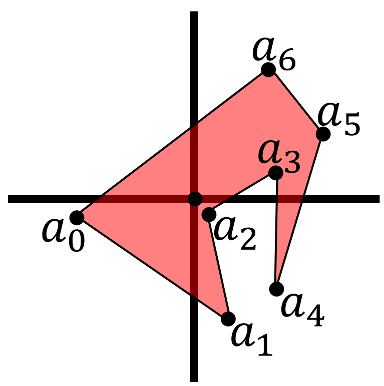


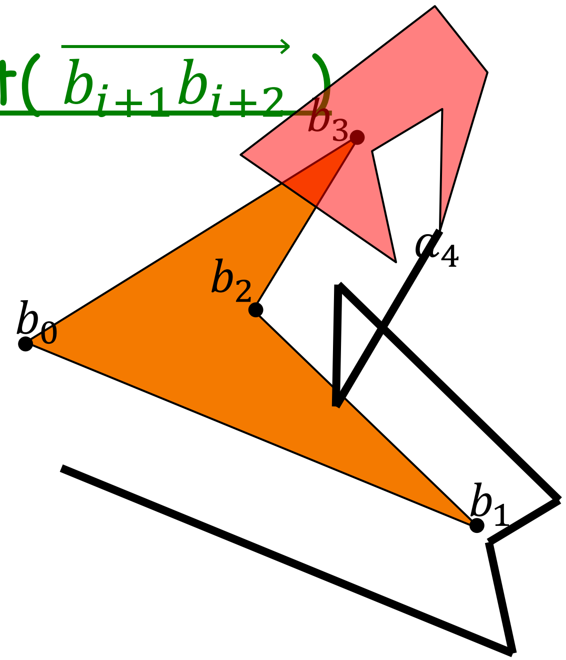
# Motion Planning

MinkowskiTrace( $A$ , $B$ )

- Place $-A$ at vertex $b_0 \in B$.
- for each $a \in$ LocallyRMost( $\overrightarrow{b_0 b_1}$ )
  - for $i \in [0, |B|)$:
    - Trace $a$ along edge( $\overrightarrow{b_i b_{i+1}}$ )
    - <u>$a \leftarrow$ TraceNextLocallyRMost( $\overrightarrow{b_{i+1} b_{i+2}}$ )</u>



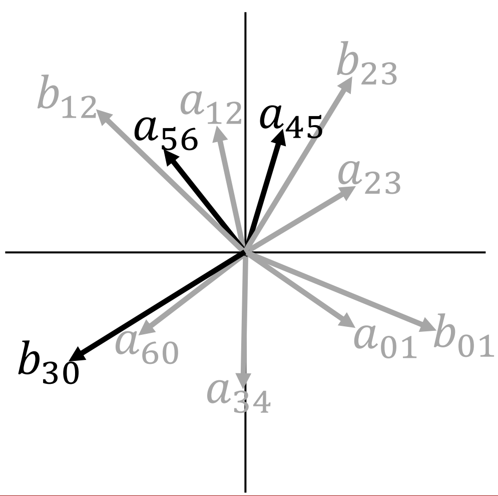



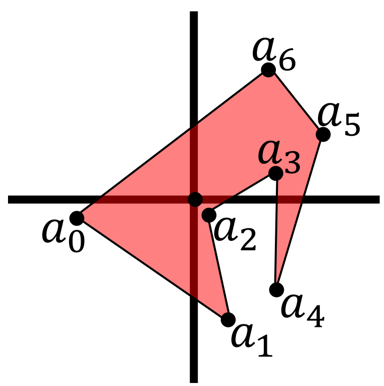



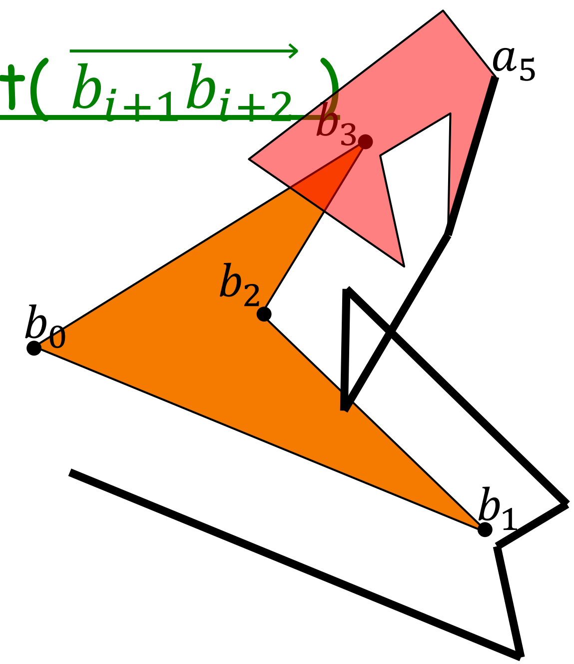
# Motion Planning

MinkowskiTrace( $A$ , $B$ )

- Place $-A$ at vertex $b_0 \in B$.
- for each $a \in$ LocallyRMost( $\overrightarrow{b_0 b_1}$ )
  - for $i \in [0, |B|)$:
    - Trace $a$ along edge( $\overrightarrow{b_i b_{i+1}}$ )
    - <u>$a \leftarrow$ TraceNextLocallyRMost( $\overrightarrow{b_{i+1} b_{i+2}}$ )</u>



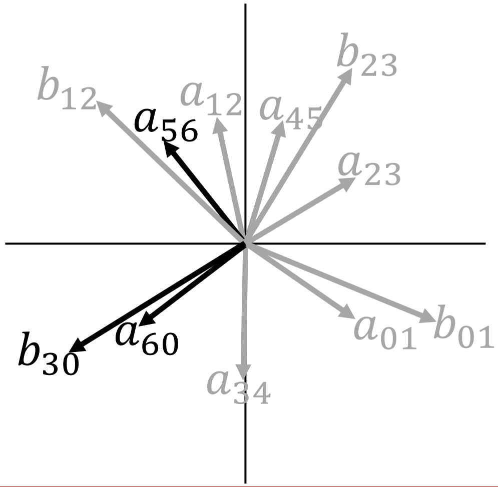



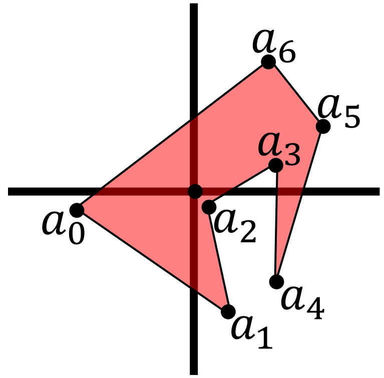



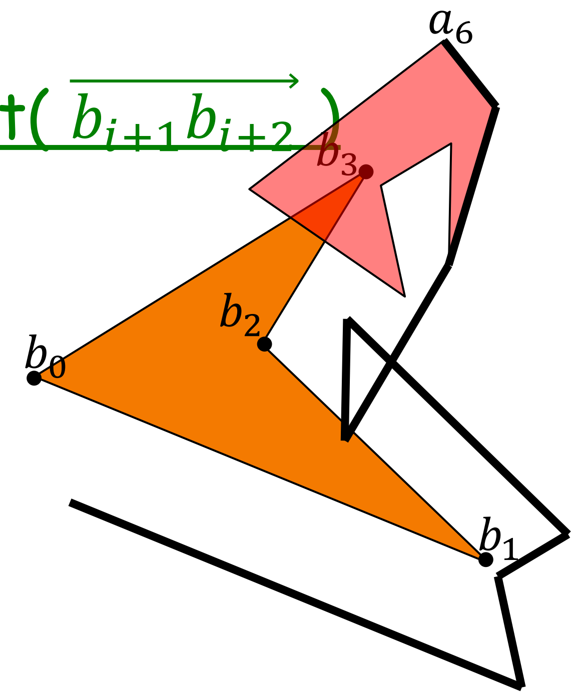
# Motion Planning

MinkowskiTrace( $A$ , $B$ )

- Place $-A$ at vertex $b_0 \in B$.
- for each $a \in$ LocallyRMost( $\overrightarrow{b_0 b_1}$ )
  - for $i \in [0, |B|)$:
    - Trace $a$ along edge( $\overrightarrow{b_i b_{i+1}}$ )
    - <u>$a \leftarrow$ TraceNextLocallyRMost( $\overrightarrow{b_{i+1} b_{i+2}}$ )</u>



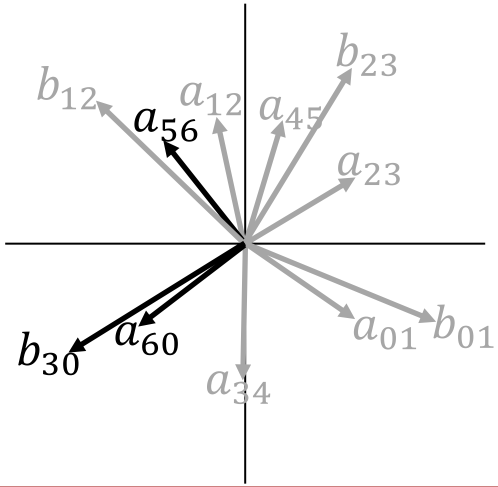



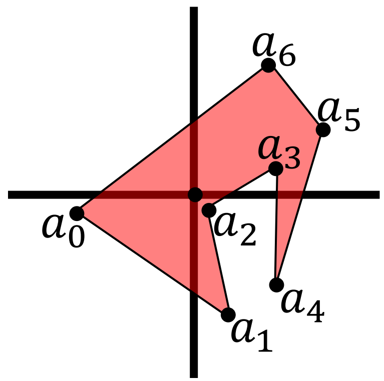



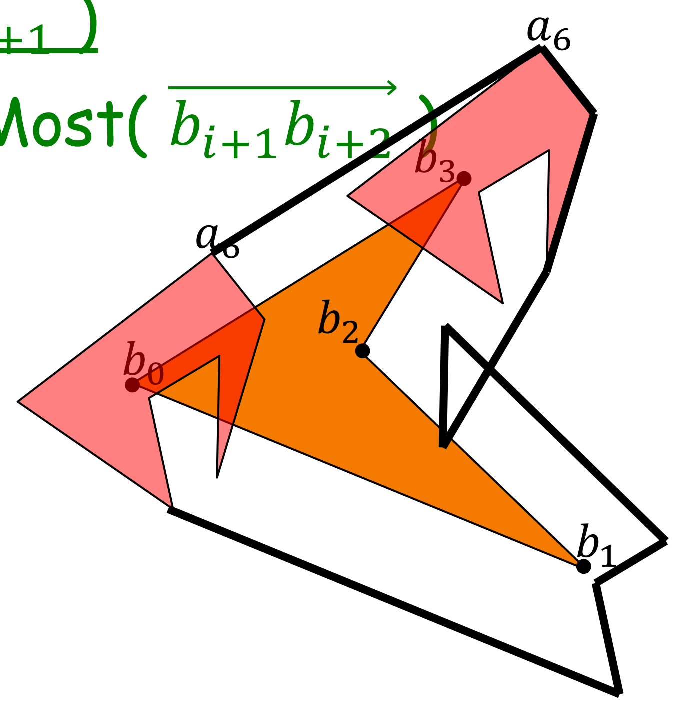
# Motion Planning

MinkowskiTrace( $A$ , $B$ )

- Place $-A$ at vertex $b_0 \in B$.
- for each $a \in$ LocallyRMost( $\overrightarrow{b_0 b_1}$ )
  - for $i \in [0, |B|)$:
    - <u>Trace $a$ along edge( $\overrightarrow{b_i b_{i+1}}$ )</u>
    - $a \leftarrow$ TraceNextLocallyRMost( $\overrightarrow{b_{i+1} b_{i+2}}$ )



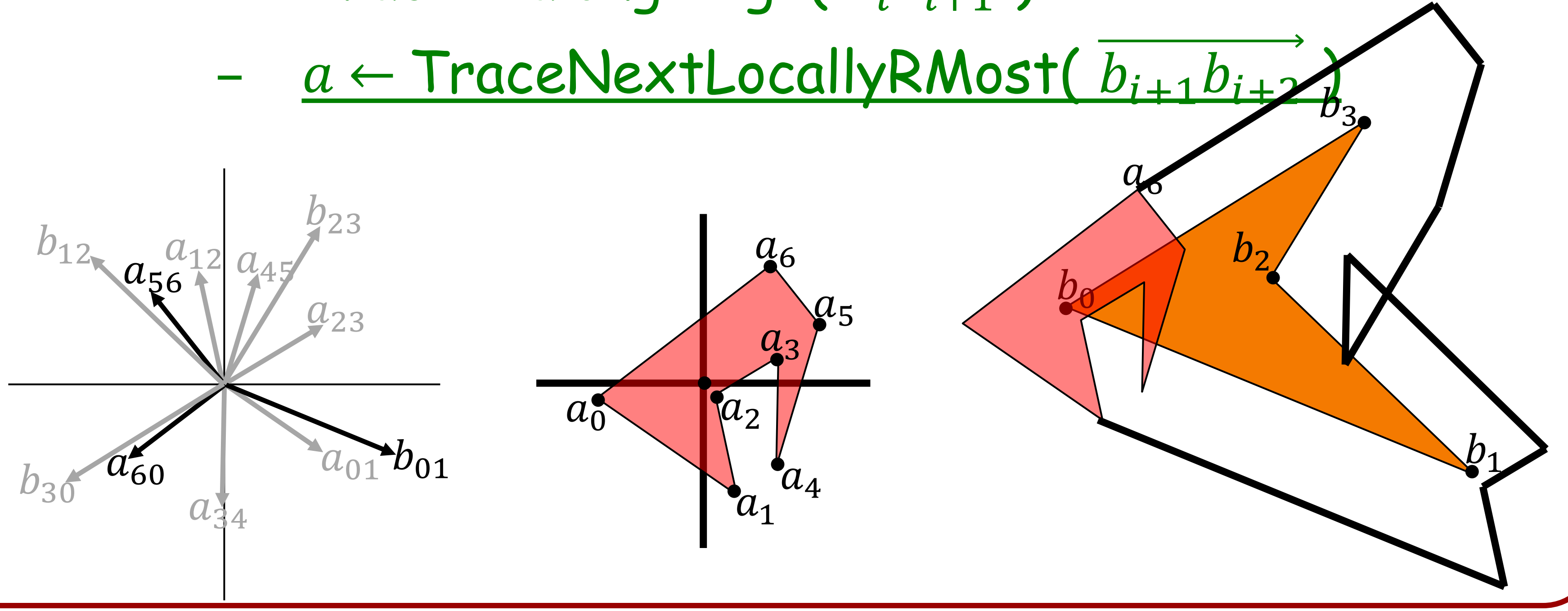



# Motion Planning

MinkowskiTrace( $A$ , $B$ )

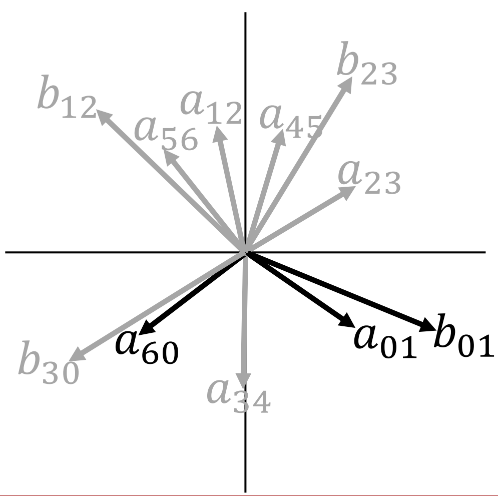
- Place $-A$ at vertex $b_0 \in B$.
- for each $a \in$ LocallyRMost( $\overrightarrow{b_0 b_1}$ )
  - for $i \in [0, |B|)$:
    - Trace $a$ along edge( $\overrightarrow{b_i b_{i+1}}$ )
    - $a \leftarrow$ TraceNextLocallyRMost( $\overrightarrow{b_{i+1} b_{i+2}}$ )

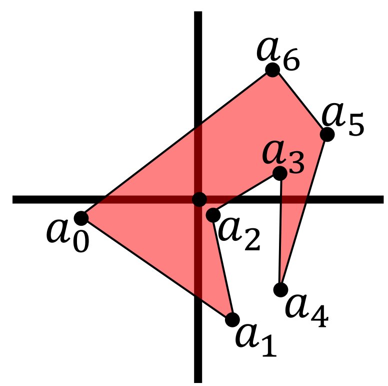


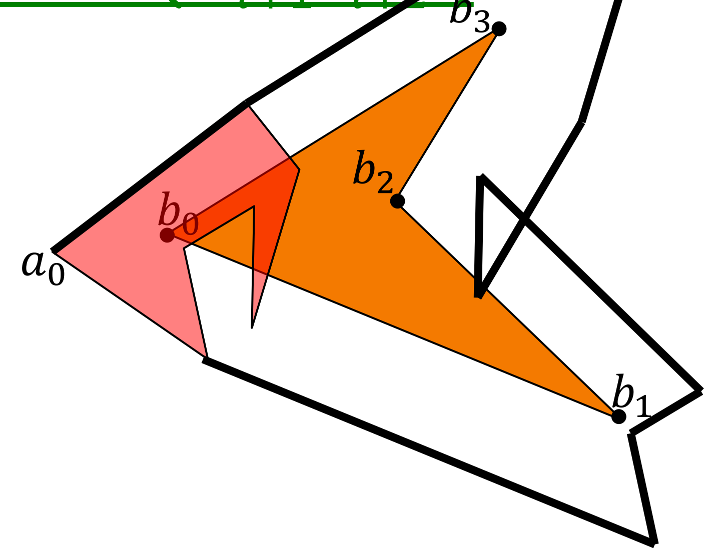
# Motion Planning

MinkowskiTrace( $A$ , $B$ )

- Place $-A$ at vertex $b_0 \in B$.
- for each $a \in$ LocallyRMost( $\overrightarrow{b_0 b_1}$ )
  - for $i \in [0, |B|)$:
    - Trace $a$ along edge( $\overrightarrow{b_i b_{i+1}}$ )
    - <u>$a \leftarrow$ TraceNextLocallyRMost( $\overrightarrow{b_{i+1} b_{i+2}}$ )</u>



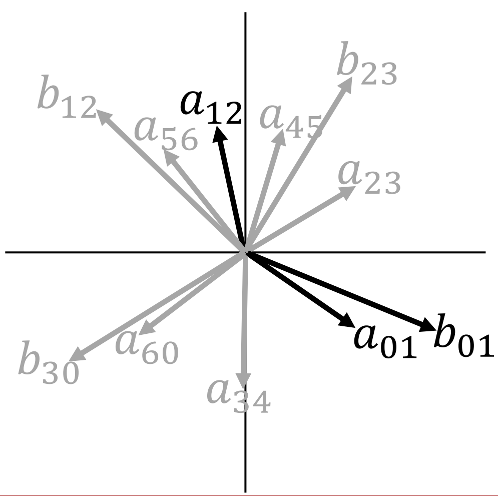



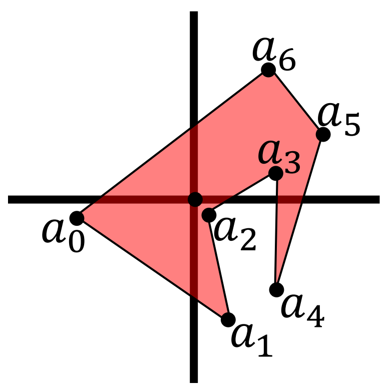



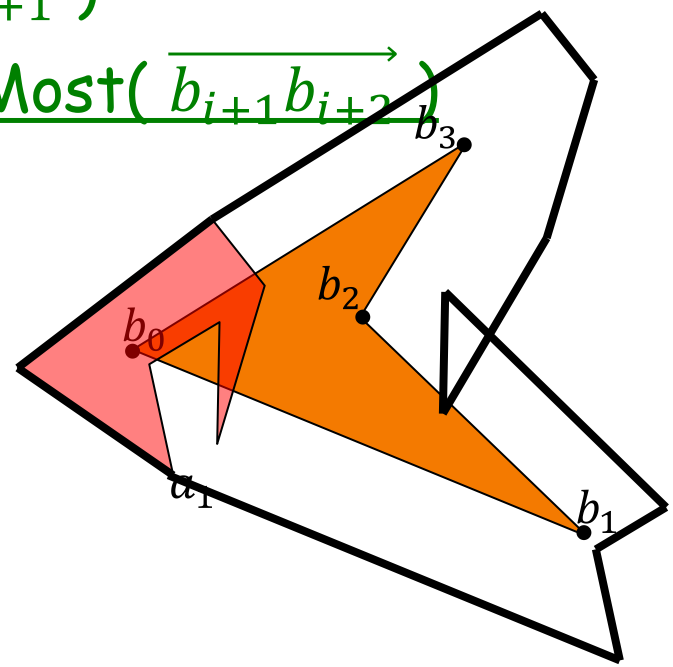
# Motion Planning

MinkowskiTrace( $A$ , $B$ )

- Place $-A$ at vertex $b_0 \in B$.
- for each $a \in$ LocallyRMost( $\overrightarrow{b_0 b_1}$ )
  - » for $i \in [0, |B|)$:
    - – Trace $a$ along edge( $\overrightarrow{b_i b_{i+1}}$ )
    - – <u>$a \leftarrow$ TraceNextLocallyRMost( $\overrightarrow{b_{i+1} b_{i+2}}$ )</u>



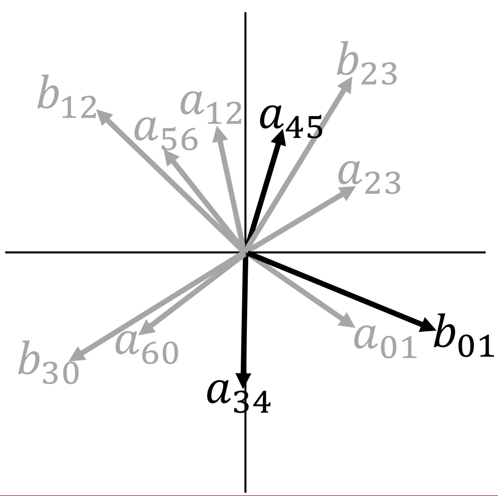



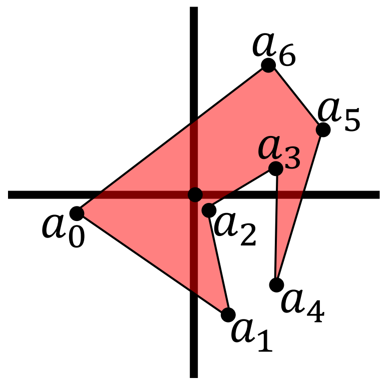



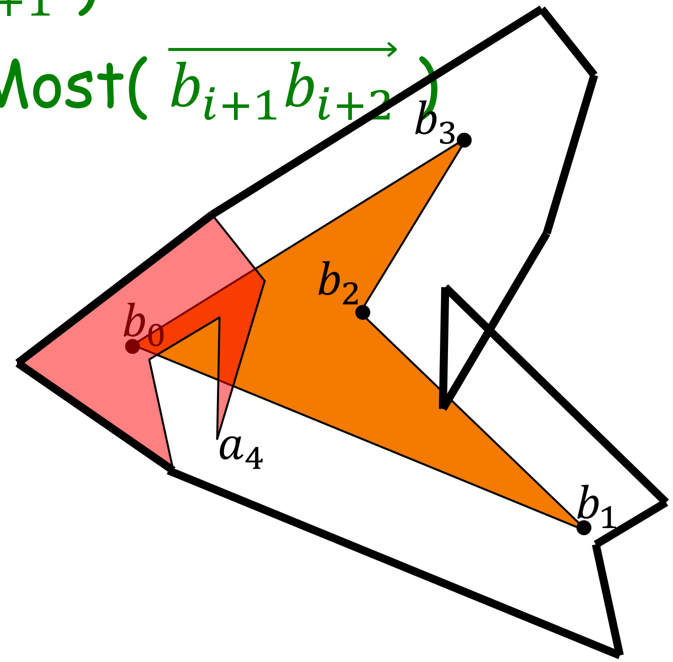
# Motion Planning

MinkowskiTrace( $A$ , $B$ )

- Place $-A$ at vertex $b_0 \in B$.
- <u>for each $a \in$ LocallyRMost( $\overrightarrow{b_0 b_1}$ )</u>
  - for $i \in [0, |B|)$:
    - Trace $a$ along edge( $\overrightarrow{b_i b_{i+1}}$ )
    - $a \leftarrow$ TraceNextLocallyRMost( $\overrightarrow{b_{i+1} b_{i+2}}$ )



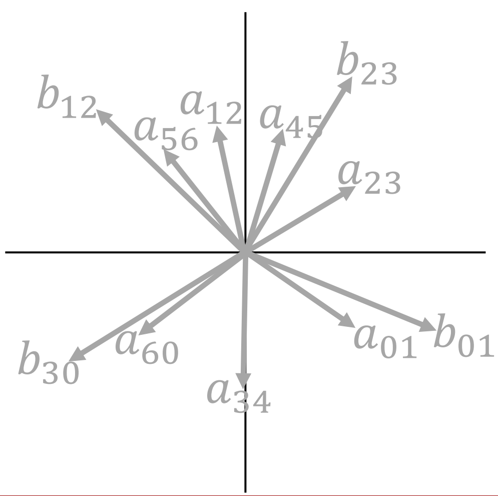



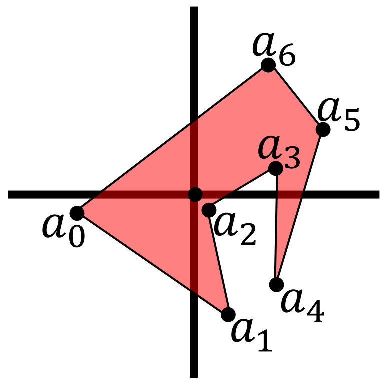



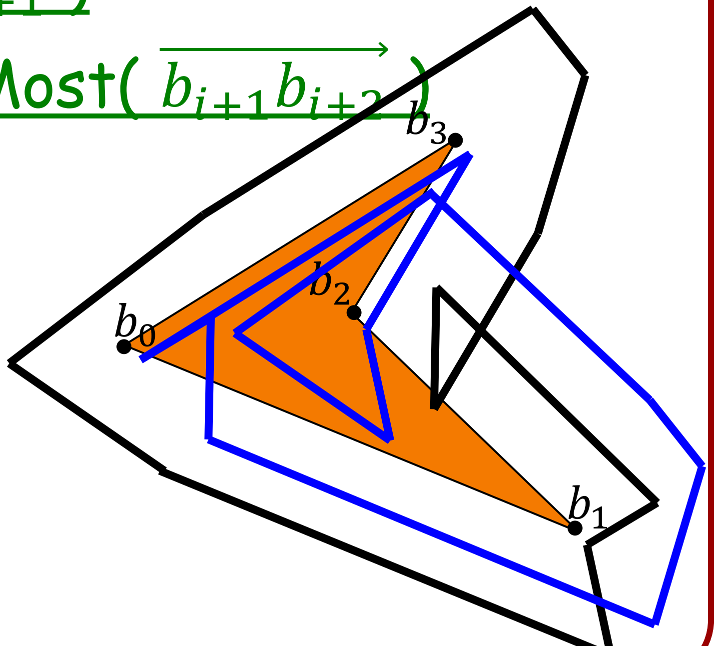
# Motion Planning

MinkowskiTrace( $A$ , $B$ )

- Place $-A$ at vertex $b_0 \in B$.
- for each $a \in$ LocallyRMost( $\overrightarrow{b_0 b_1}$ )
  - for $i \in [0, |B|)$:
    - Trace $a$ along edge( $\overrightarrow{b_i b_{i+1}}$ )
    - $a \leftarrow$ TraceNextLocallyRMost( $\overrightarrow{b_{i+1} b_{i+2}}$ )



# Motion Planning

MinkowskiTrace( $A$ , $B$ )

- Place $-A$ at vertex $b_0 \in B$.
- <u>for each $a \in$ LocallyRMost( $\overrightarrow{b_0 b_1}$ )</u>
  - for $i \in [0, |B|)$:
    - Trace $a$ along edge( $\overrightarrow{b_i b_{i+1}}$ )
    - $a \leftarrow$ TraceNextLocallyRMost( $\overrightarrow{b_{i+1} b_{i+2}}$ )

<u>Note</u>:

- If $A$ is convex, the outer `for` loop cycles around the polygon $B$ once.
- Otherwise we can cycle as many times as there are different extremal vertices in $A$ -- $O(|A|)$.

# Motion Planning

MinkowskiTrace( $A$ , $B$ )

- Place $-A$ at vertex $b_0 \in B$.
- for each $a \in$ LocallyRMost( $\overrightarrow{b_0 b_1}$ )
  - for $i \in [0, |B|)$:
    - Trace $a$ along edge( $\overrightarrow{b_i b_{i+1}}$ )
    - $a \leftarrow$ TraceNextLocallyRMost( $\overrightarrow{b_{i+1} b_{i+2}}$ )

Note:

- In the worst case, tracing the locally maximal vertex can take $O(|A|)$ time.

# Motion Planning

MinkowskiTrace( $A$ , $B$ )

- Place $-A$ at vertex $b_0 \in B$.
- for each $a \in$ LocallyRMost( $\overrightarrow{b_0 b_1}$ )
  - for $i \in [0, |B|)$:
    - Trace $a$ along edge( $\overrightarrow{b_i b_{i+1}}$ )
    - $a \leftarrow$ TraceNextLocallyRMost( $\overrightarrow{b_{i+1} b_{i+2}}$ )

Note:

- If $A$ and $B$ are convex, then in all these tracings, we cycle around $A$ once -- $O(1)$.

# Motion Planning

MinkowskiTrace( $A$ , $B$ )

- Place $-A$ at vertex $b_0 \in B$.
- for each $a \in$ LocallyRMost( $\overrightarrow{b_0 b_1}$ )
  - for $i \in [0, |B|)$:
    - Trace $a$ along edge( $\overrightarrow{b_i b_{i+1}}$ )
    - $a \leftarrow$ TraceNextLocallyRMost( $\overrightarrow{b_{i+1} b_{i+2}}$ )

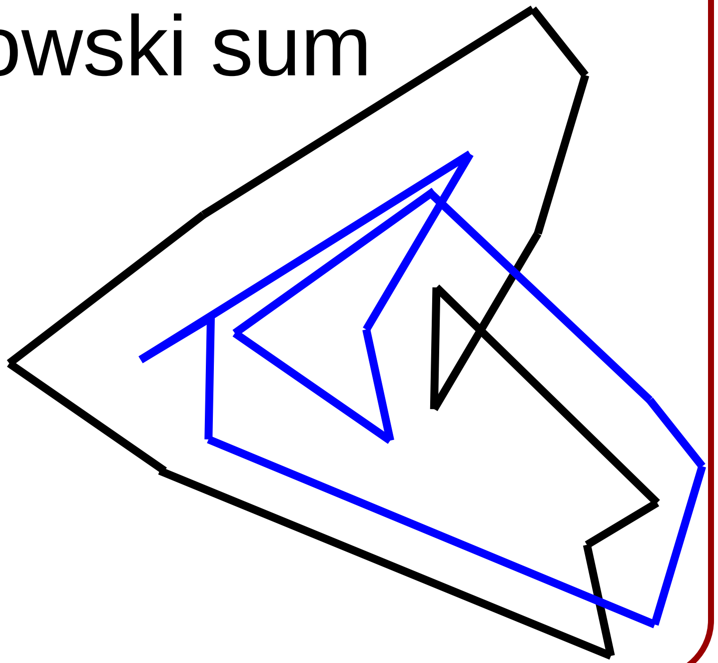
Complexity:

- If $A$ and $B$ convex: $O(|A| + |B|)$
- Else if $A$ convex: $O(|A| \cdot |B|)$
- Else: $O(|A|^2 \cdot |B|^2)$

# Motion Planning

- This only gives the *convolution of $A$ with $B$*.
- If $A$ or $B$ is not convex, this is not the same as the Minkowski sum.

Q: How do we get the Minkowski sum from the convolution?

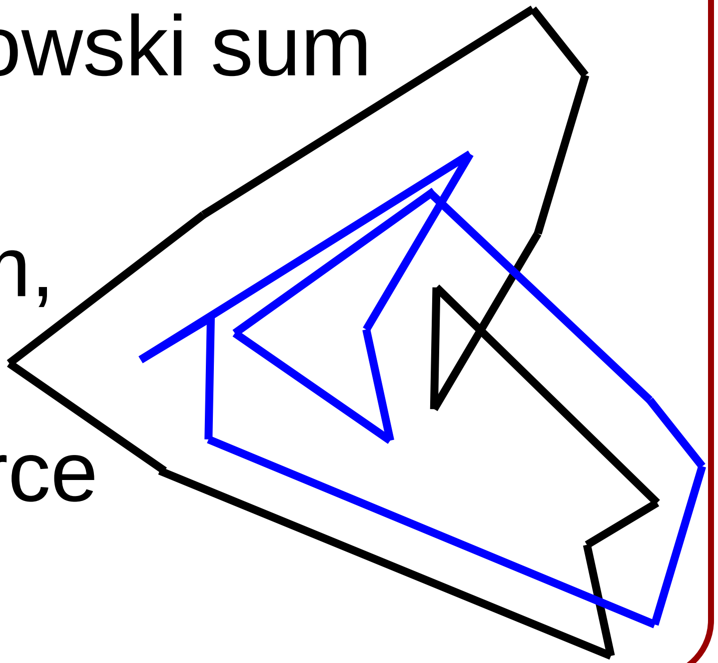


# Motion Planning

- This only gives the *convolution of $A$ with $B$*.
- If $A$ or $B$ is not convex, this is not the same as the Minkowski sum.

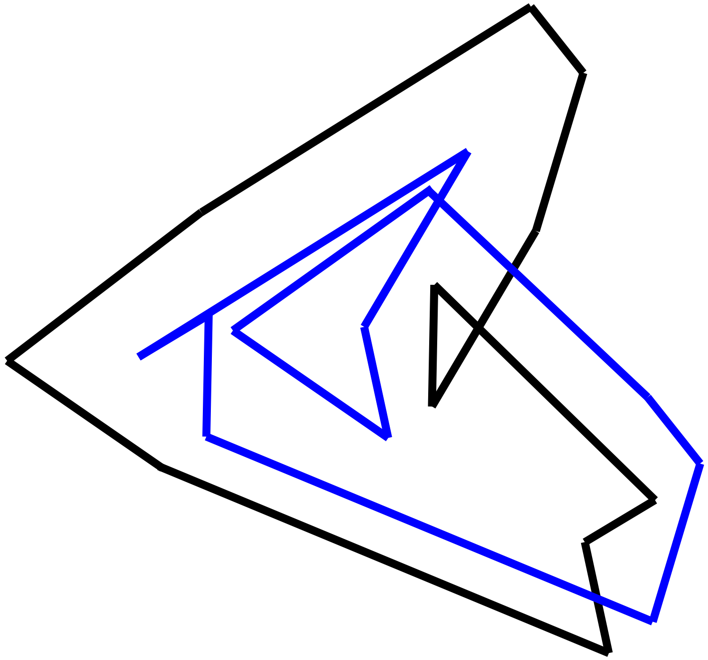
Q: How do we get the Minkowski sum from the convolution?

Q: Given the Minkowski sum, how do we determine if we can get from the source to the target?



# Convolution to Minkowski Sum

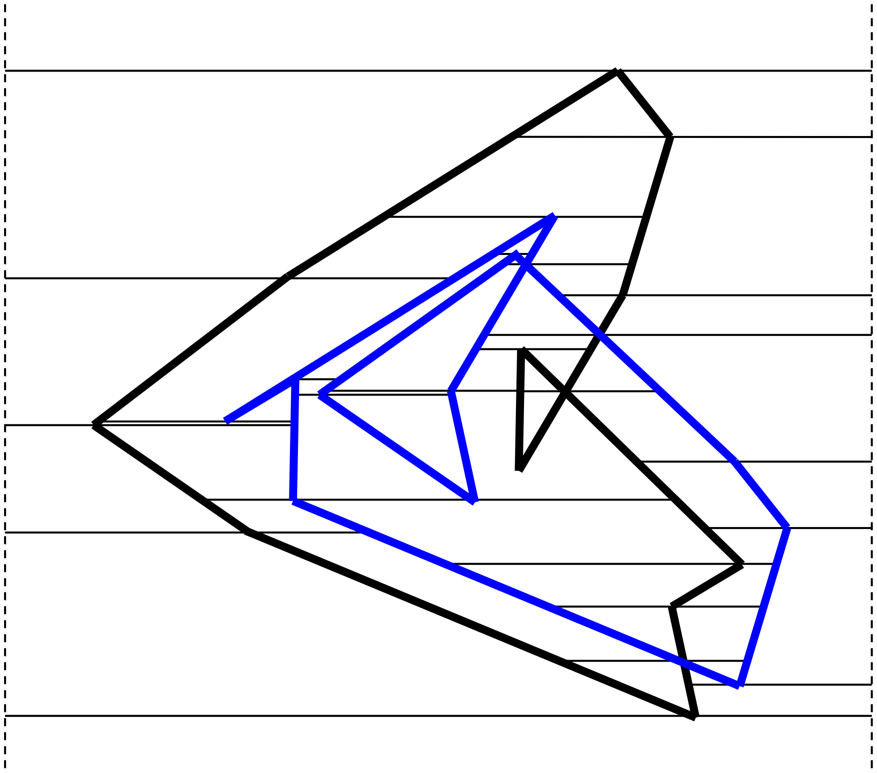
Approach:



# Convolution to Minkowski Sum

Approach:

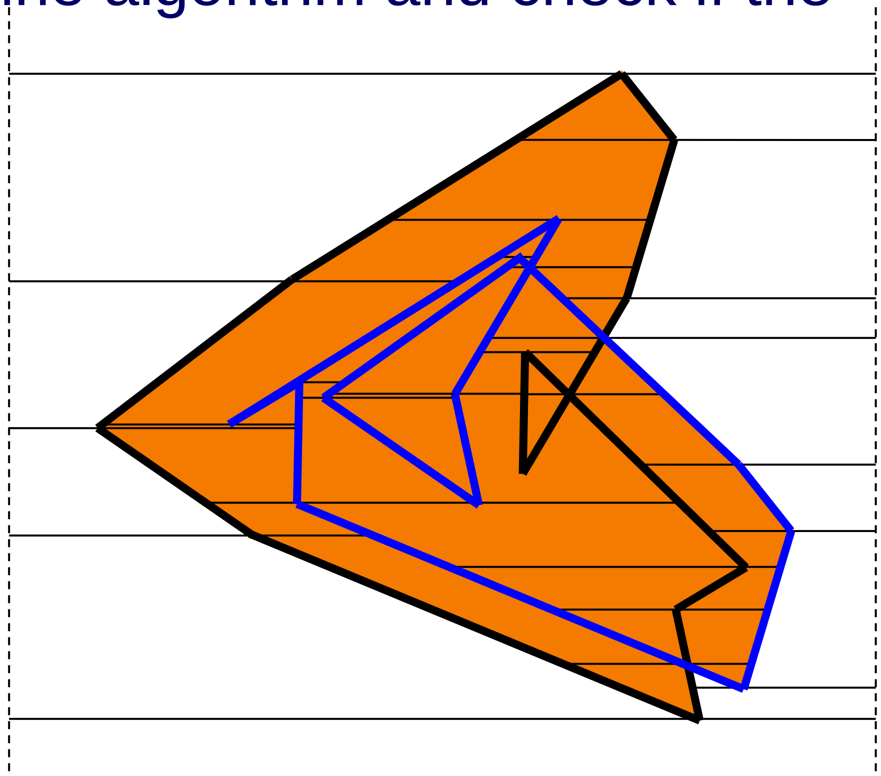
1. Compute the trapezoidalization



# Convolution to Minkowski Sum

Approach:

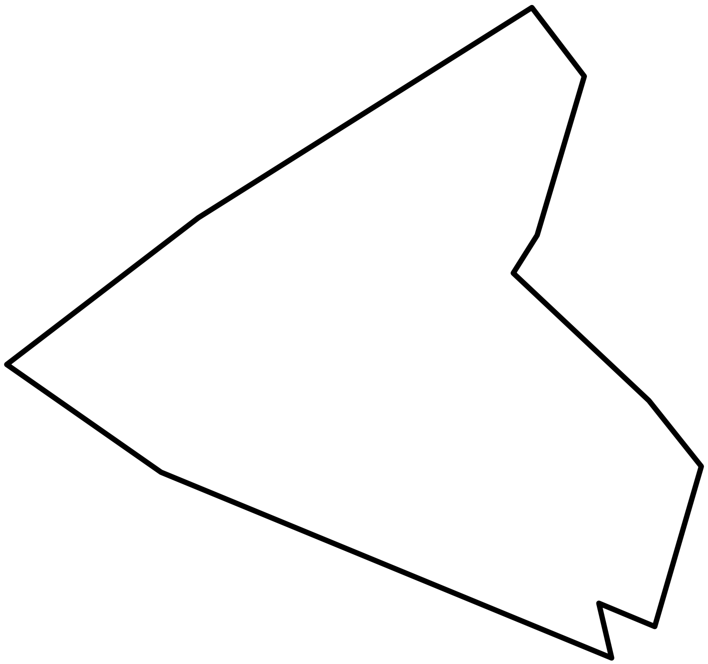
1. Compute the trapezoidalization
2. Use the winding number to identify interior trapezoids
   E.g. Run the sweep- line algorithm and check if the difference between the number of left and right edges crossed on one side of the sweep-line is non-zero.



# Convolution to Minkowski Sum

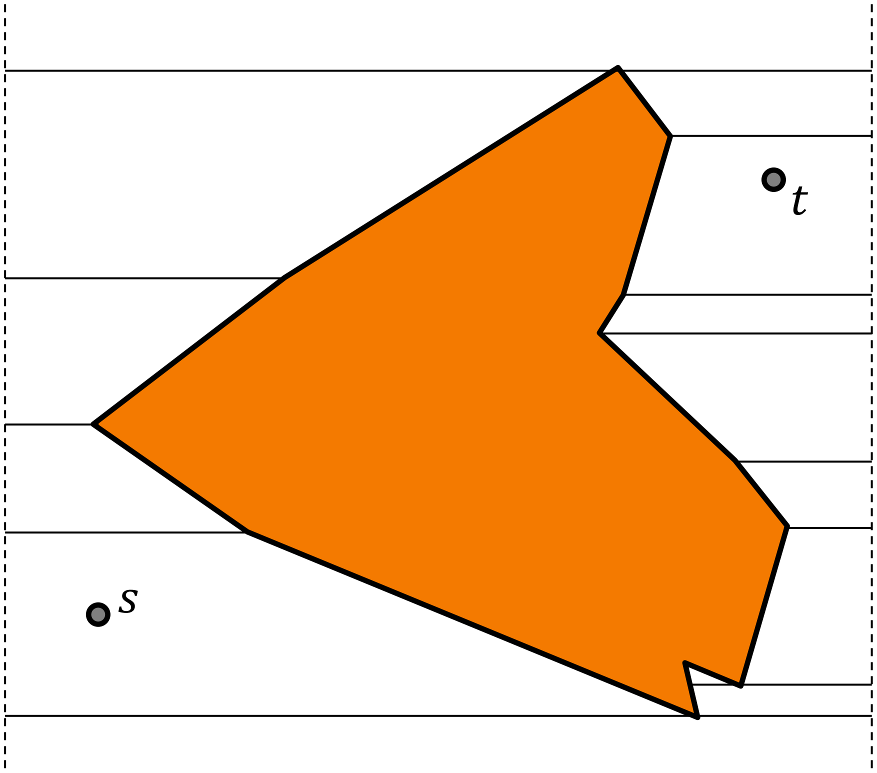
Approach:

1. Compute the trapezoidalization
2. Use the winding number to identify interior trapezoids
3. Compute the edges separating the interior trapezoids from the exterior ones



# Computing Accessibility

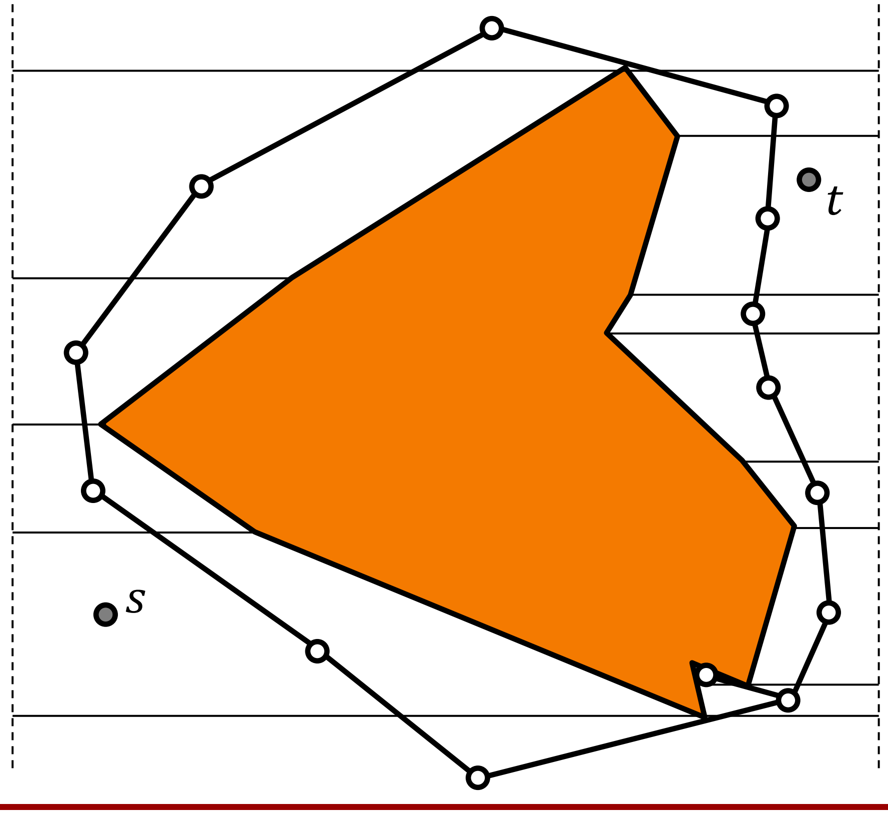
Given the trapezoidalization of the exterior of the convolution, and given source and target positions:



# Computing Accessibility

Given the trapezoidalization of the exterior of the convolution, and given source and target positions:

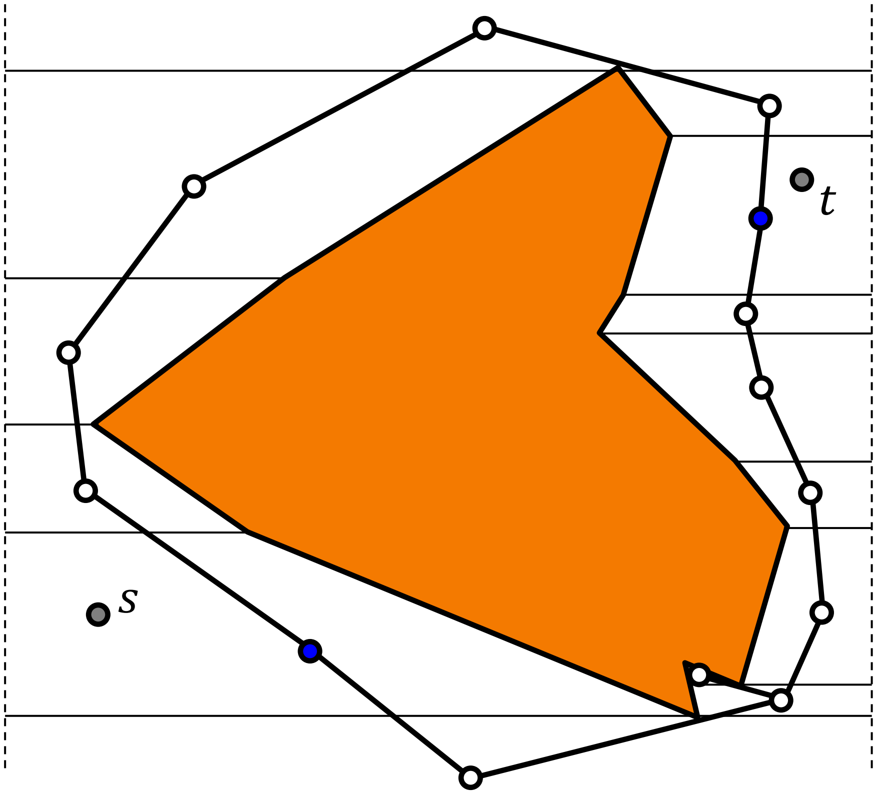
1. Compute the dual graph of the exterior trapezoids



# Computing Accessibility

Given the trapezoidalization of the exterior of the convolution, and given source and target positions:

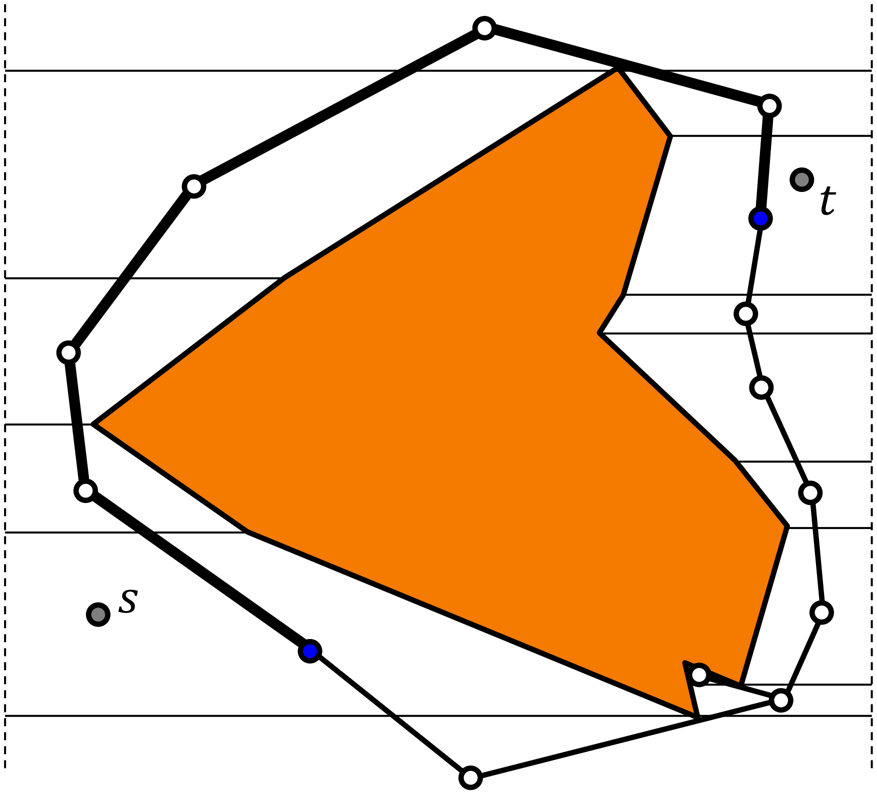
1. Compute the dual graph of the exterior trapezoids
2. Identify (dual) nodes corresponding to the trapezoids containing the source and target



# Computing Accessibility

Given the trapezoidalization of the exterior of the convolution, and given source and target positions:

1. Compute the dual graph of the exterior trapezoids
2. Identify (dual) nodes corresponding to the trapezoids containing the source and target
3. Check if there is a path in the dual graph between them



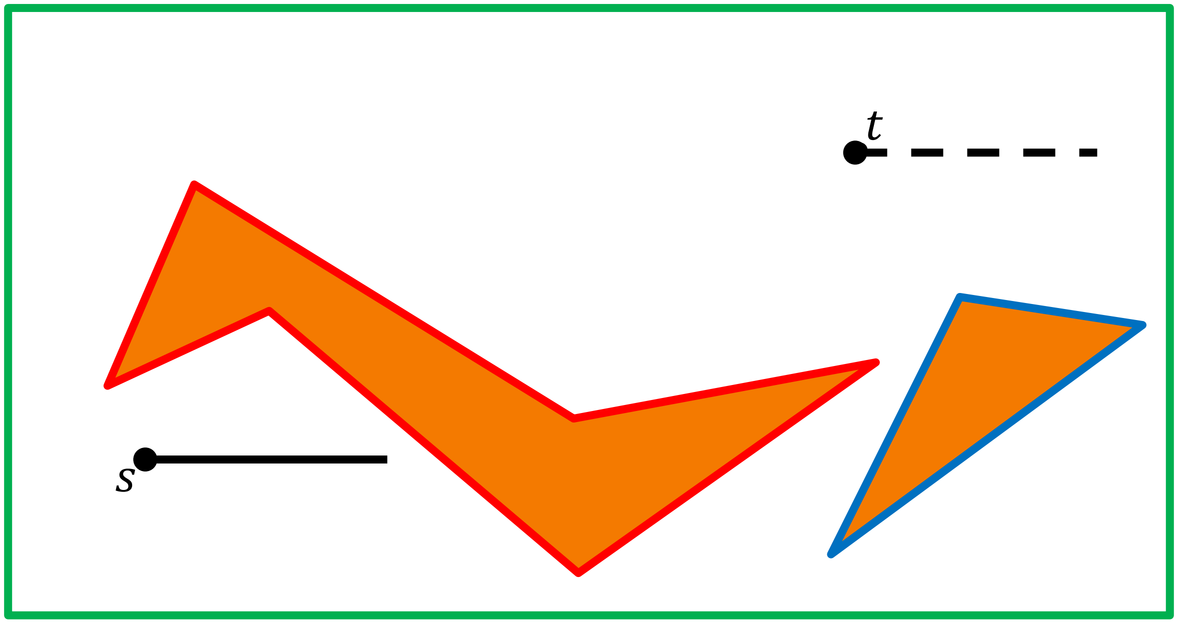
# Outline

- Translating a polygon
- Moving a ladder

# Moving a Ladder

<u>Goal</u>:

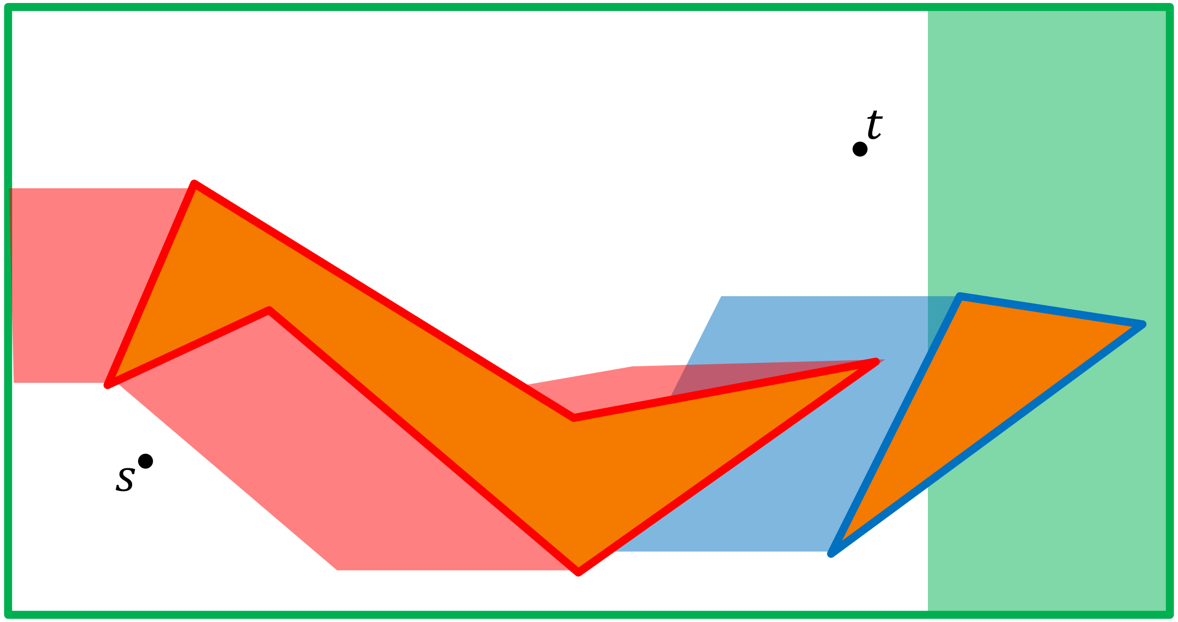
Given a line segment at position $s$, a set of polygons $P$, and a target position $t$, determine if the ladder can be translated <u>and rotated</u> to $t$ while avoiding the polygonal obstacles.



# Moving a Ladder

General Approach:

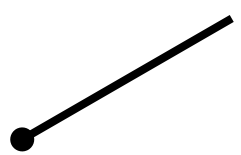
- Compute the Minkowski Sum of $P$ with $l$.

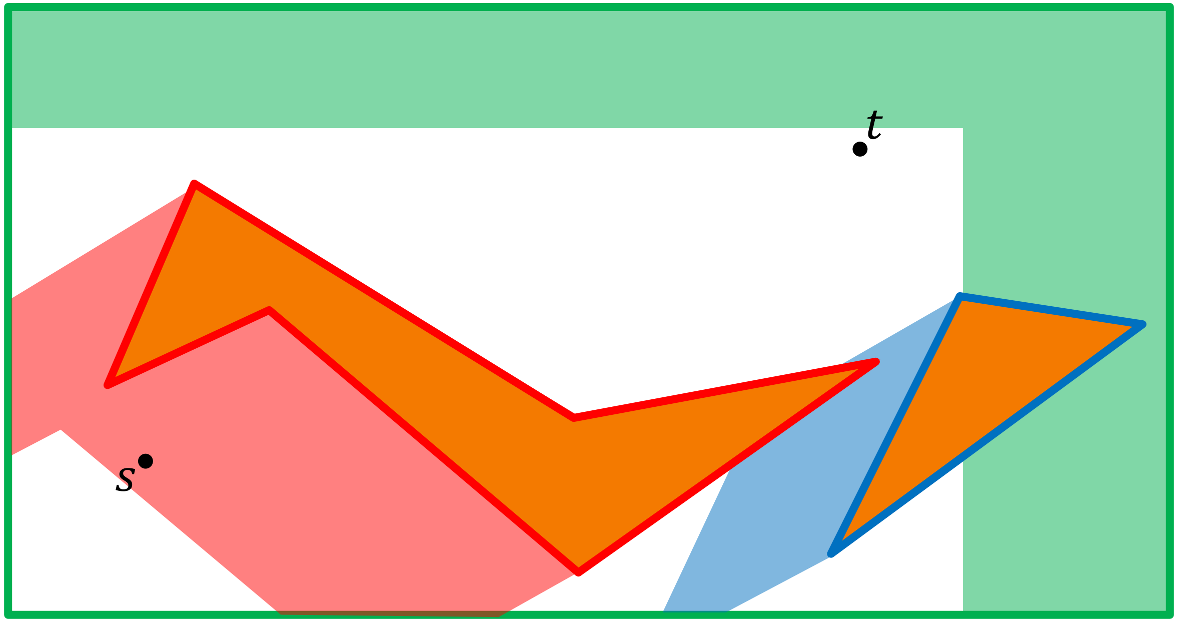


# Moving a Ladder

General Approach:

- Compute the Minkowski Sum of $P$ with the ladder.
- Do this for every rotation of the ladder.

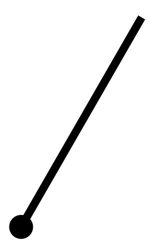


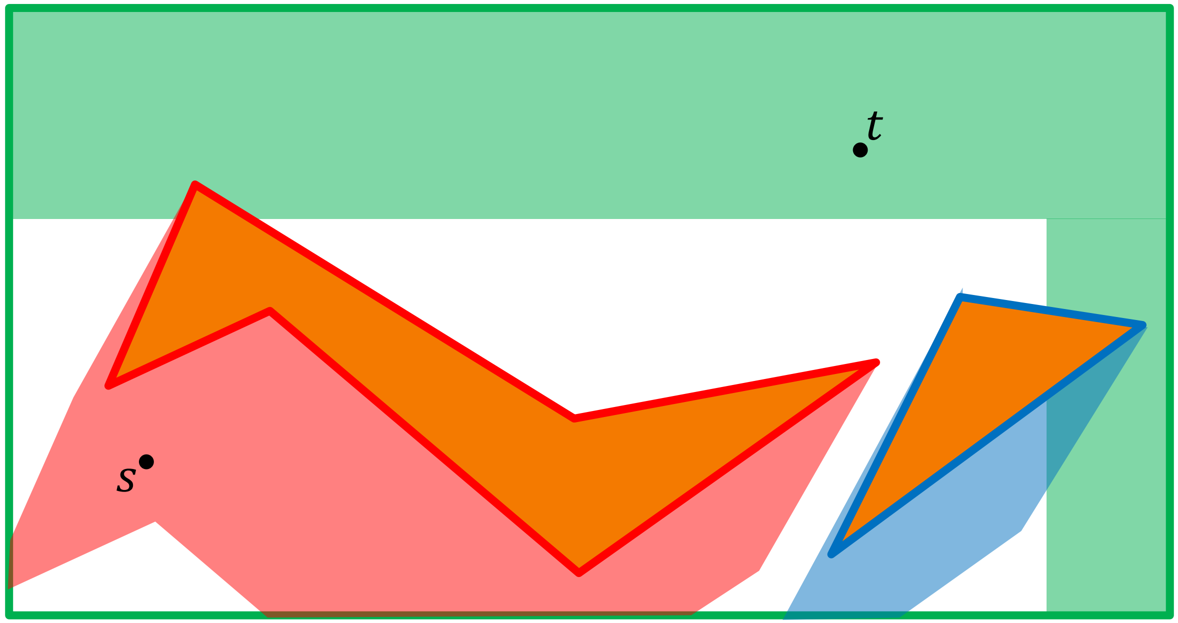


# Moving a Ladder

<u>General Approach</u>:

- Compute the Minkowski Sum of $P$ with the ladder.
- Do this for every rotation of the ladder.

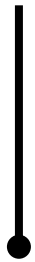


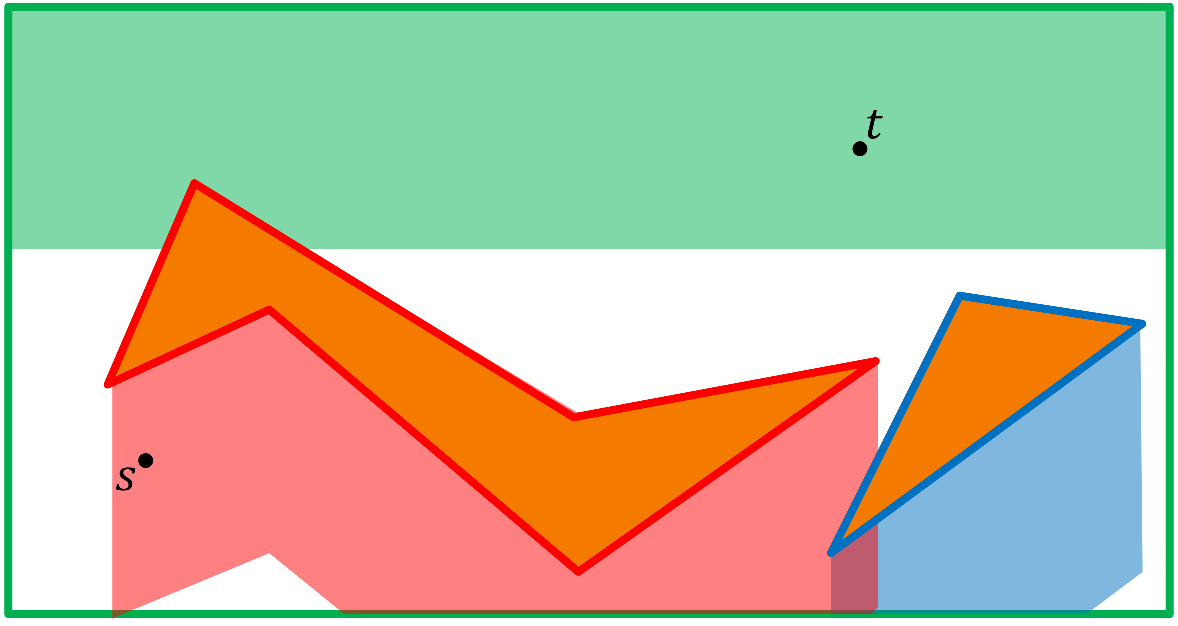


# Moving a Ladder

General Approach:

- Compute the Minkowski Sum of $P$ with the ladder.
- Do this for every rotation of the ladder.

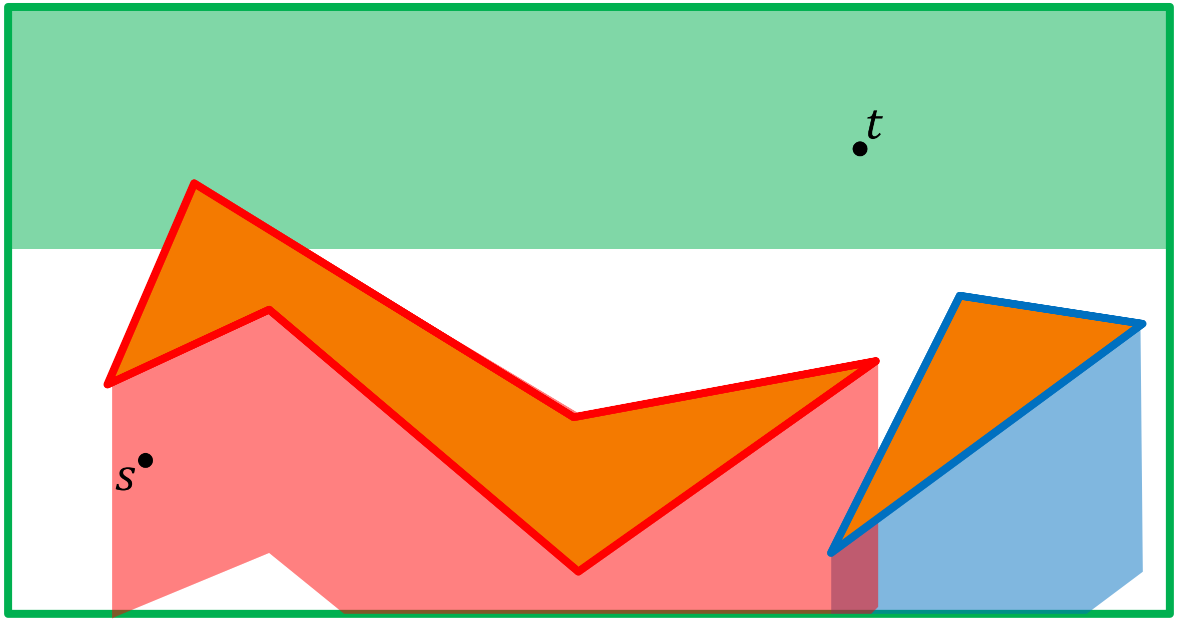




# Moving a Ladder

General Approach:

- Compute the Minkowski Sum of $P$ with the ladder.
- Do this for every rotation of the ladder.
- Stack into a 3D volume and test if there is a path from $s$ to $t$, in the 3D space, that avoids all the polygons.



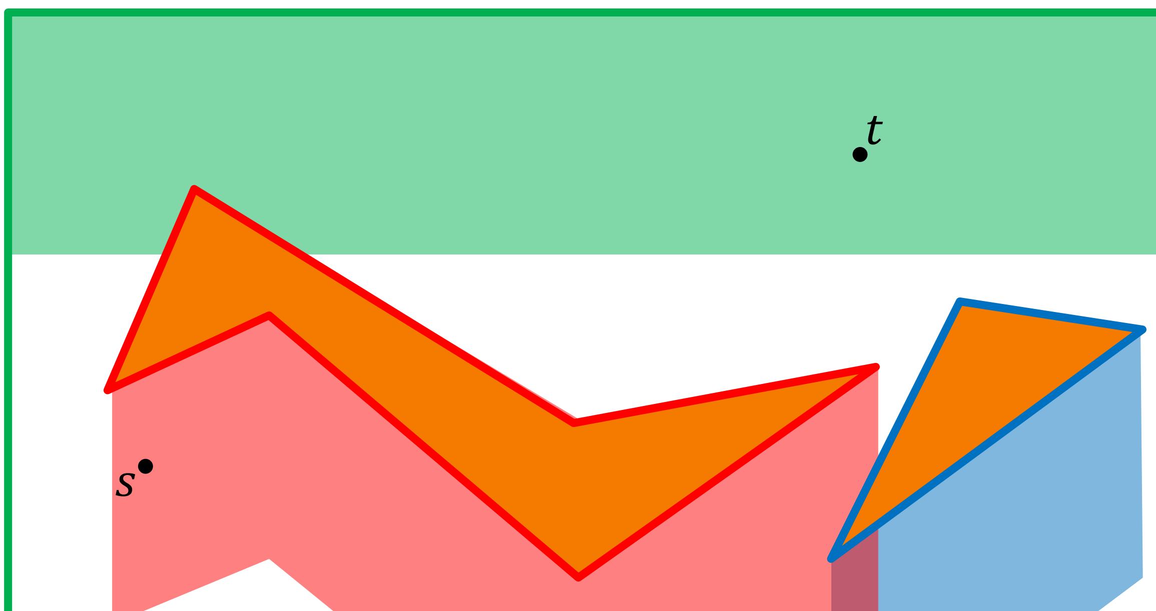
# Moving a Ladder

General Approach:

- Compute the Minkowski Sum of $P$ with the ladder.
- Do t
- Stac … a path from $s$ to $t$, in the 3D space, that avoids all the polygons.

Challenges:

- There are infinitely many angles.
- The boundaries of the 3D sums are not planar.



# Moving a Ladder

Implementation:

Partition empty region into trapezoids:

- Base aligned with the ladder direction.
- Sides defined by the edges of the polygons.

# Moving a Ladder

<u>Implementation</u>:

Partition empty region into trapezoids:

- Base aligned with the ladder direction.
- Sides defined by the edges of the polygons.

In general, as we rotate the ladder, the shape of the cells change but the topology of the partition (i.e. dual graph) does not.

# Moving a Ladder

Implementation:

Partition empty region into trapezoids:

- Base aligned with the ladder direction.
- Sides defined by the edges of the polygons.

In general, as we rotate the ladder, the shape of the cells change but the topology of the partition (i.e. dual graph) does not.

At discrete events, cells may be inserted, deleted, or merged.

# Moving a Ladder

<u>Implementation</u>:

Partition empty region into trapezoids:

These events can only occur when the rotated ladder aligns with a segment between two vertices of the polygons.

In ge … of the

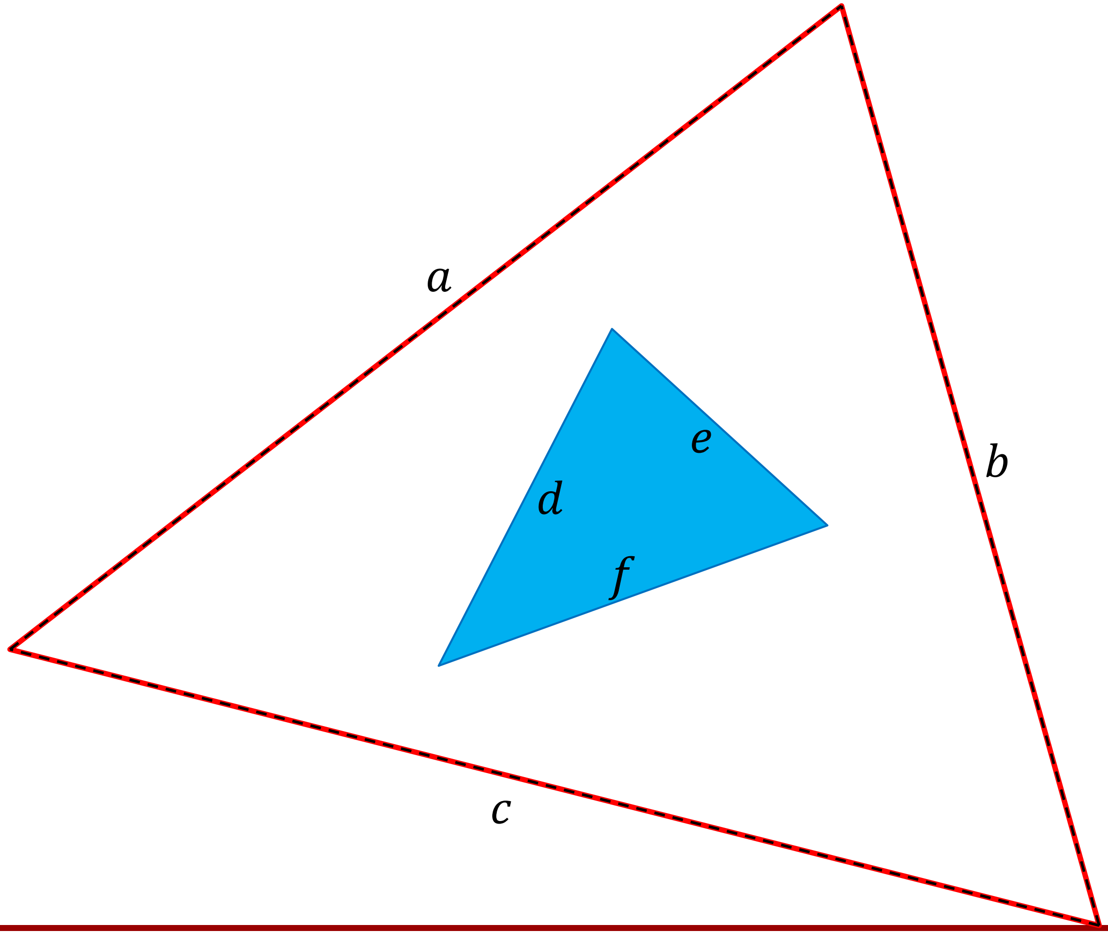
cells … .e.

dual graph) does not.

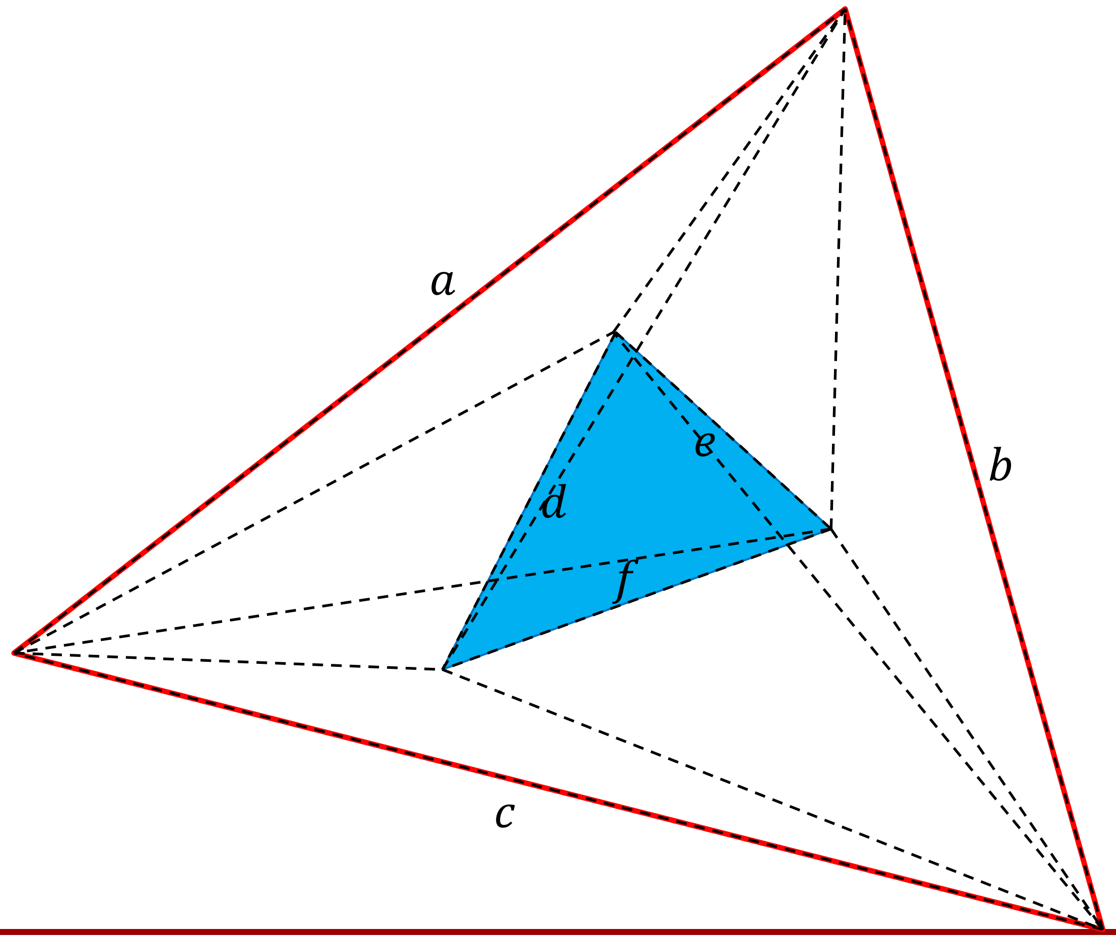
Instead of trying all possible rotations, we only need to consider the $O(|P|^2)$ cases.

At discrete events, cells may be inserted, deleted, or merged.

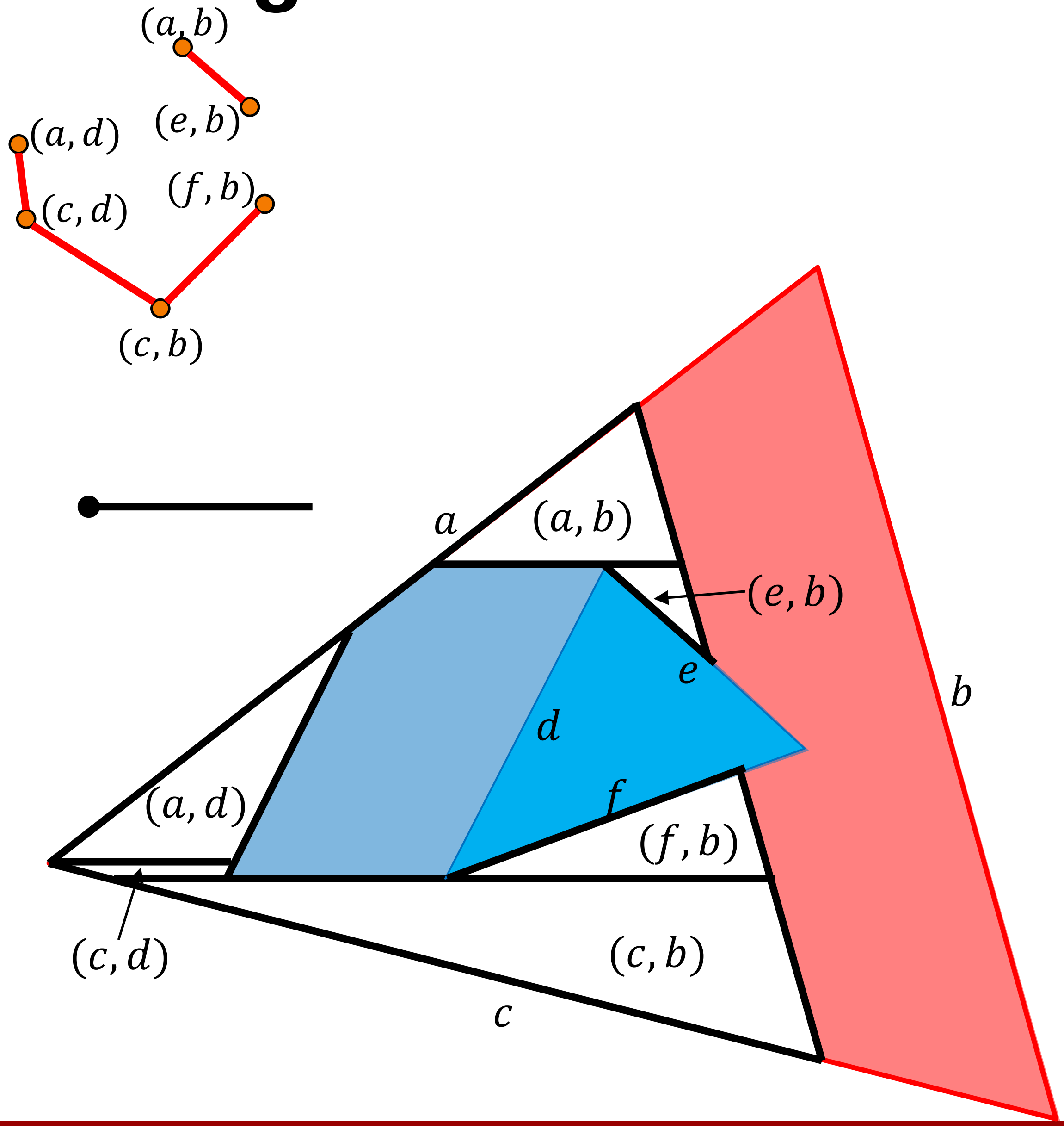
# Moving a Ladder



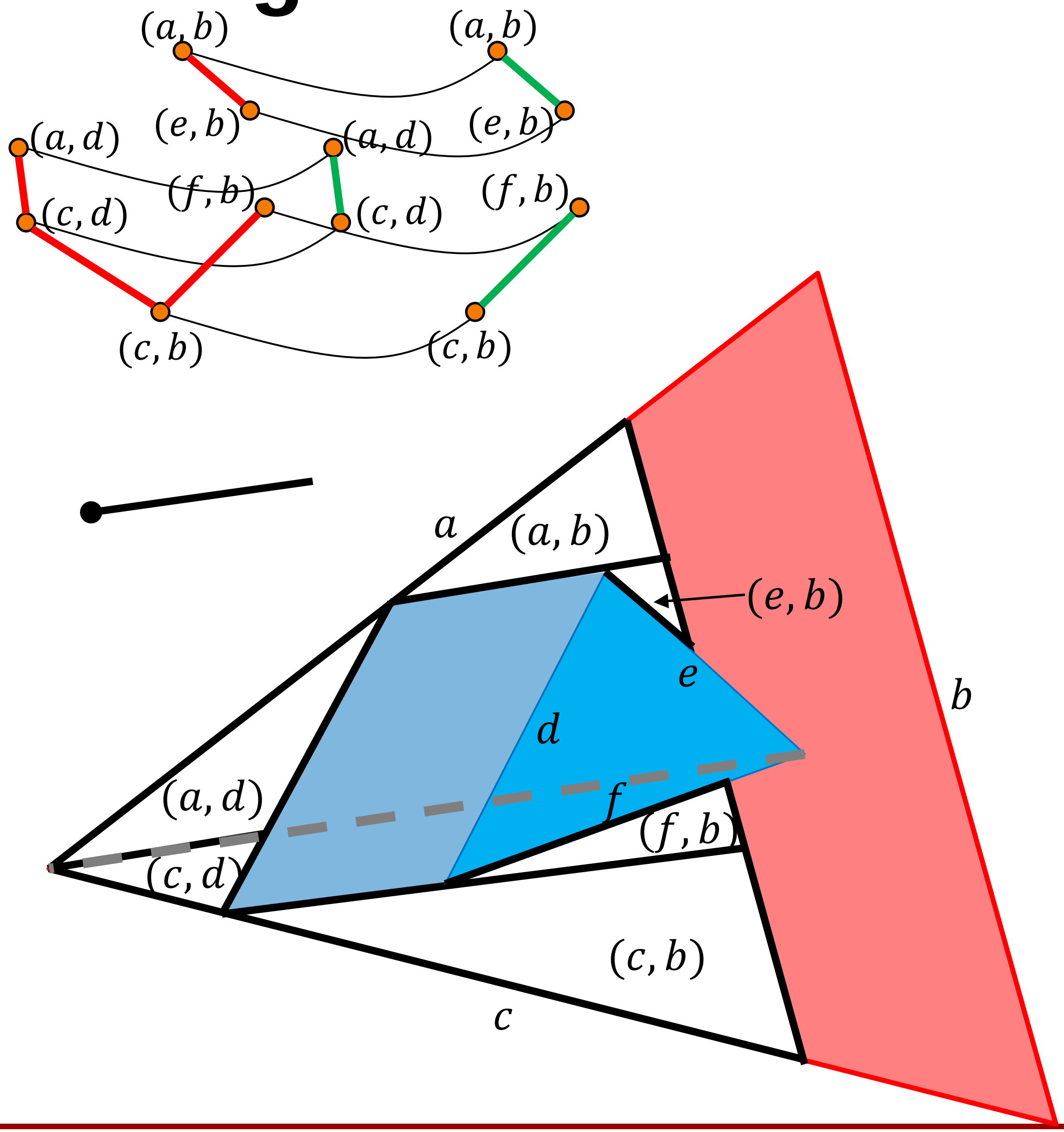
# Moving a Ladder



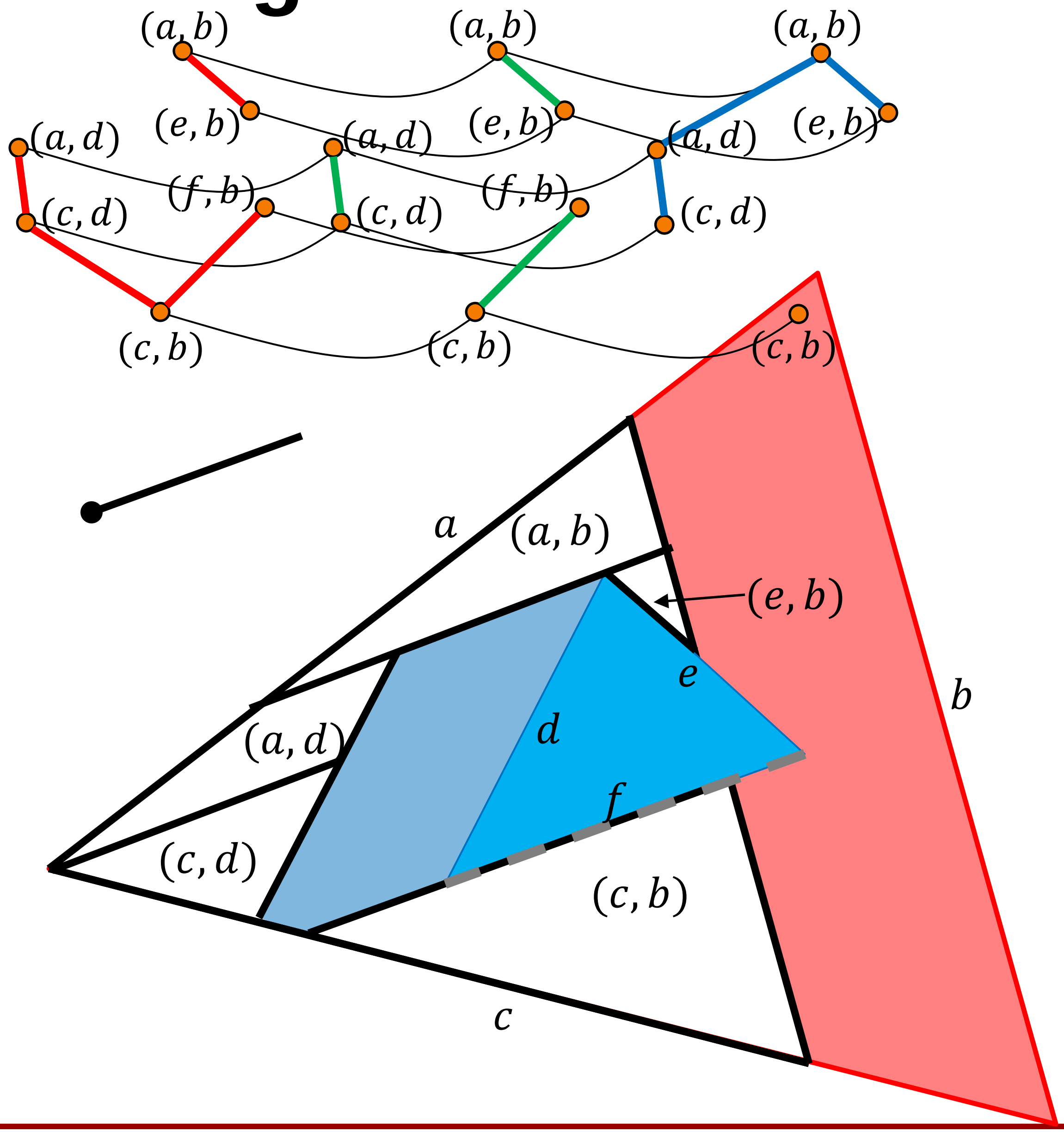
# Moving a Ladder



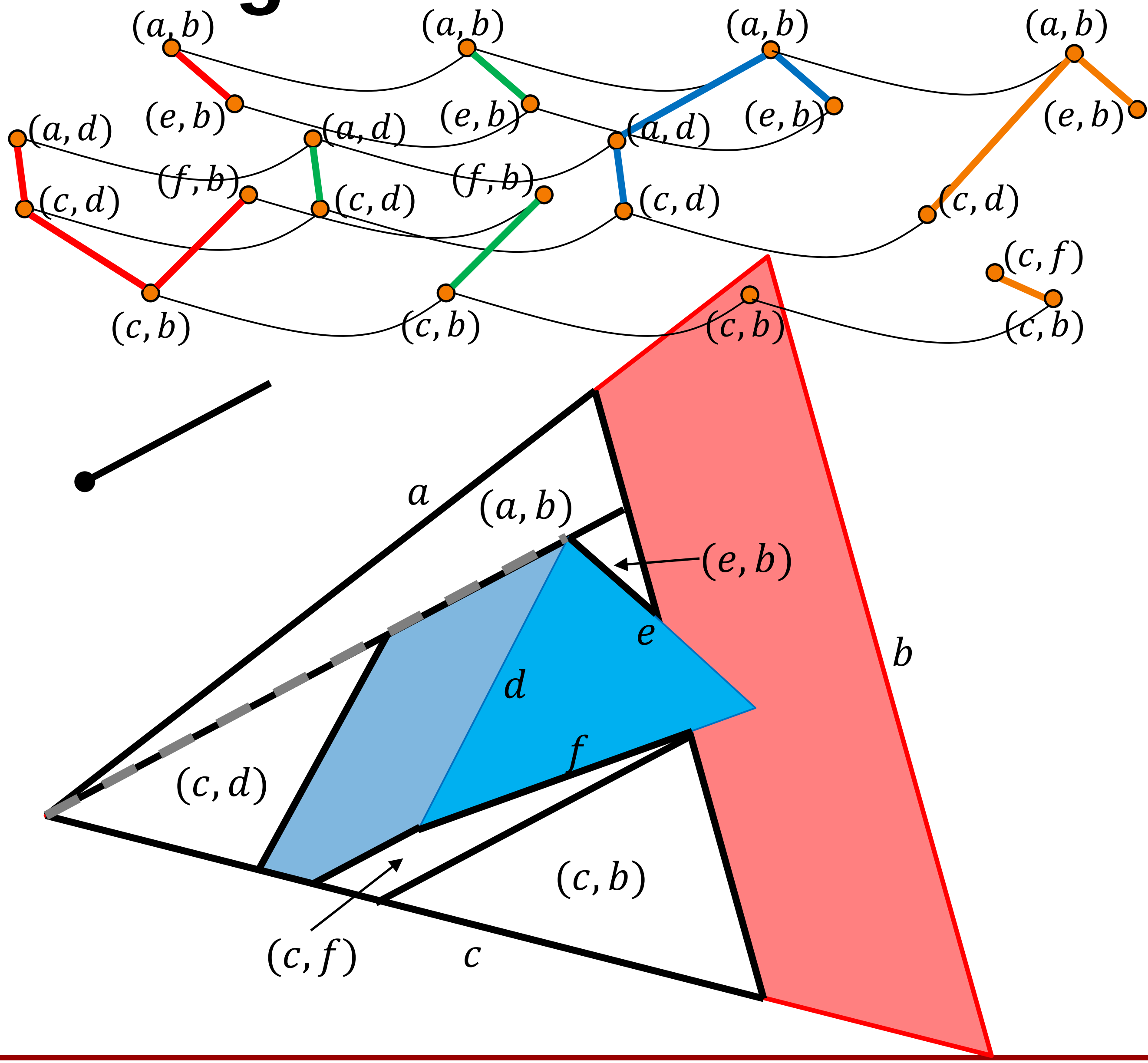
# Moving a Ladder



# Moving a Ladder



# Moving a Ladder



# Moving a Ladder

Approach:

- Set $\Theta = \{\theta_0 = 0, \theta_1, \ldots, \theta_n\}$ to be sorted angles, with $\theta_i$ the angle of the $i$-th segment.
- For $0 \leq i \leq n$:
  - » Compute the Minkowski Sum of $P$ with the rotation of the ladder by angle $\theta_i + \varepsilon$.
  - » Compute the dual graph of the trapezoidalization.
  - » Link the $(i-1)$-st and $i$-th dual graphs.
- Test if there is a path between the cells containing $s$ (at $\theta = 0$) and $t$ in the linked graph.