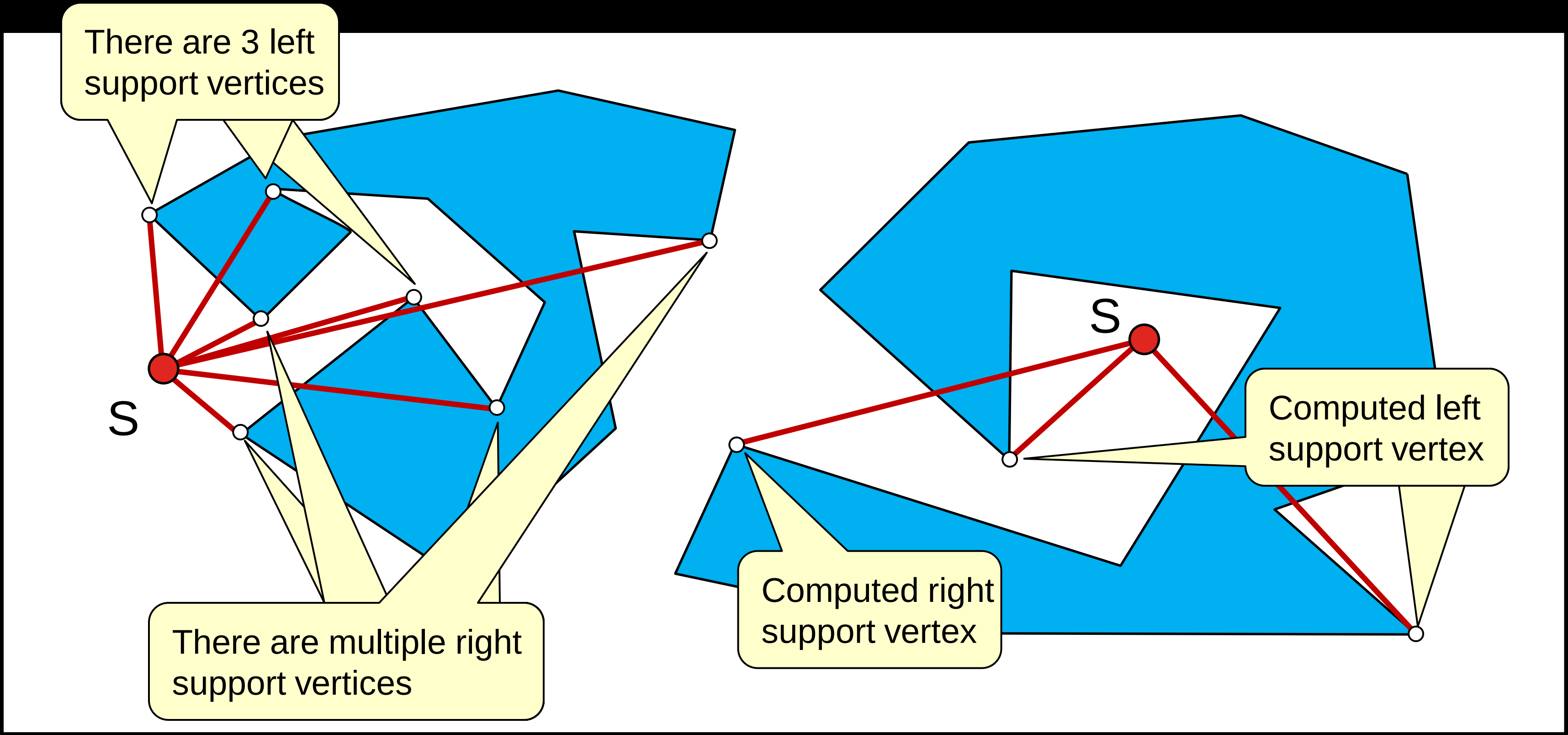
Path Planning
(Non-Convex Obstacles)

Non–Convex Obstacle Supports

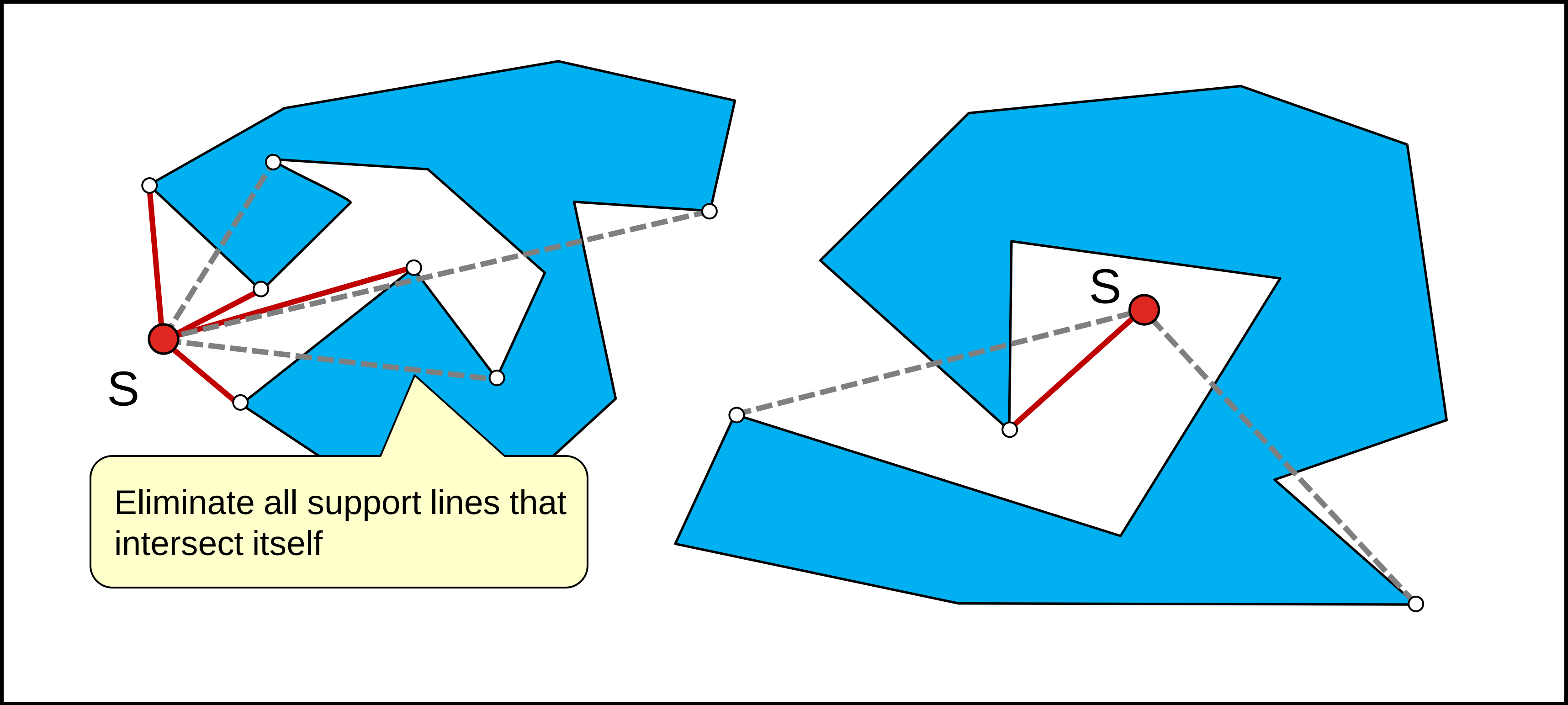
- Our support-line algorithm can produce multiple support lines per polygon if it is non-convex:



There are 3 left support vertices
S
There are multiple right support vertices
S
Computed left support vertex
Computed right support vertex

Non–Convex Obstacle Supports

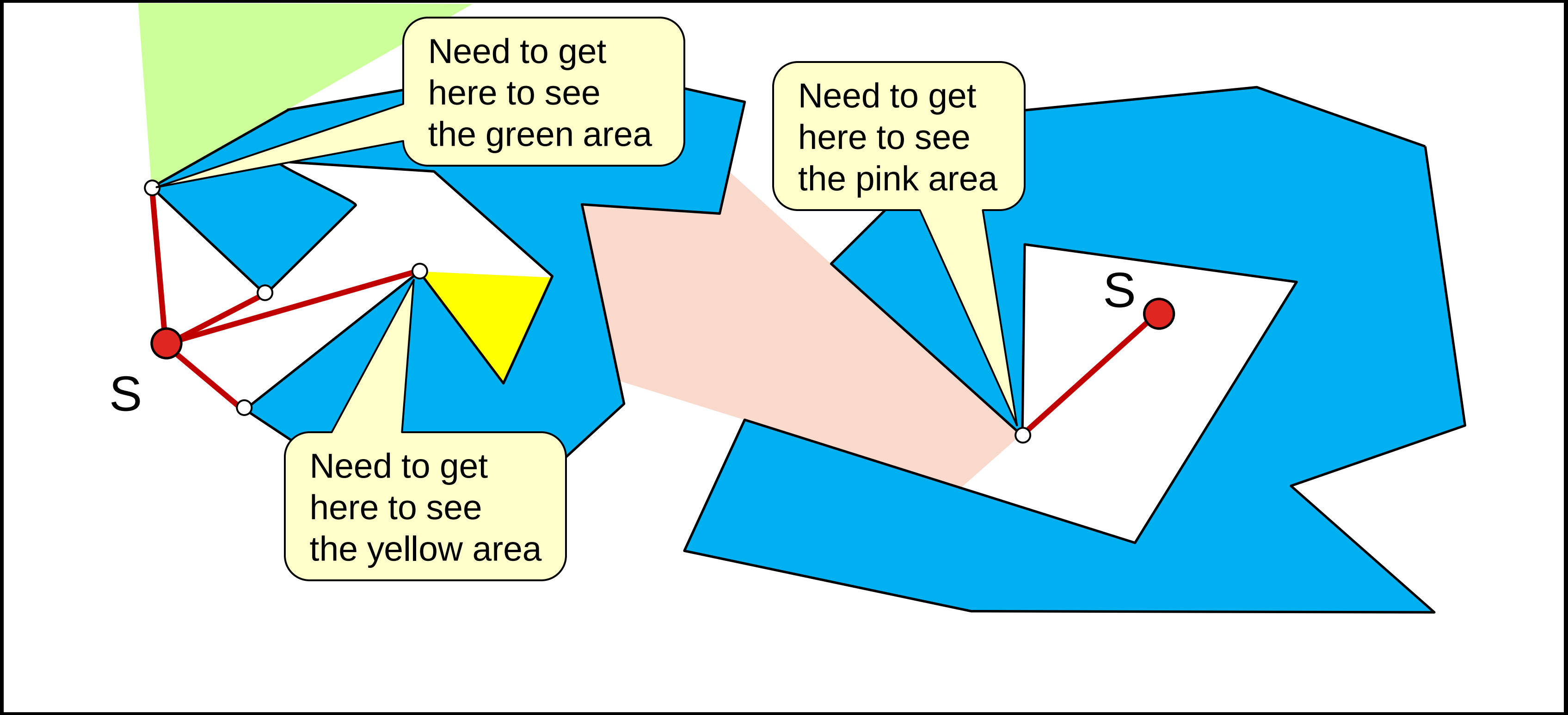
- We can eliminate those that intersect itself as well as those that intersect other polygons.



Eliminate all support lines that intersect itself

Non–Convex Visibility

- A support vertex represents an “observation point” that robot must be at in order to “see” (i.e., have visibility) around a corner. They are all necessary.



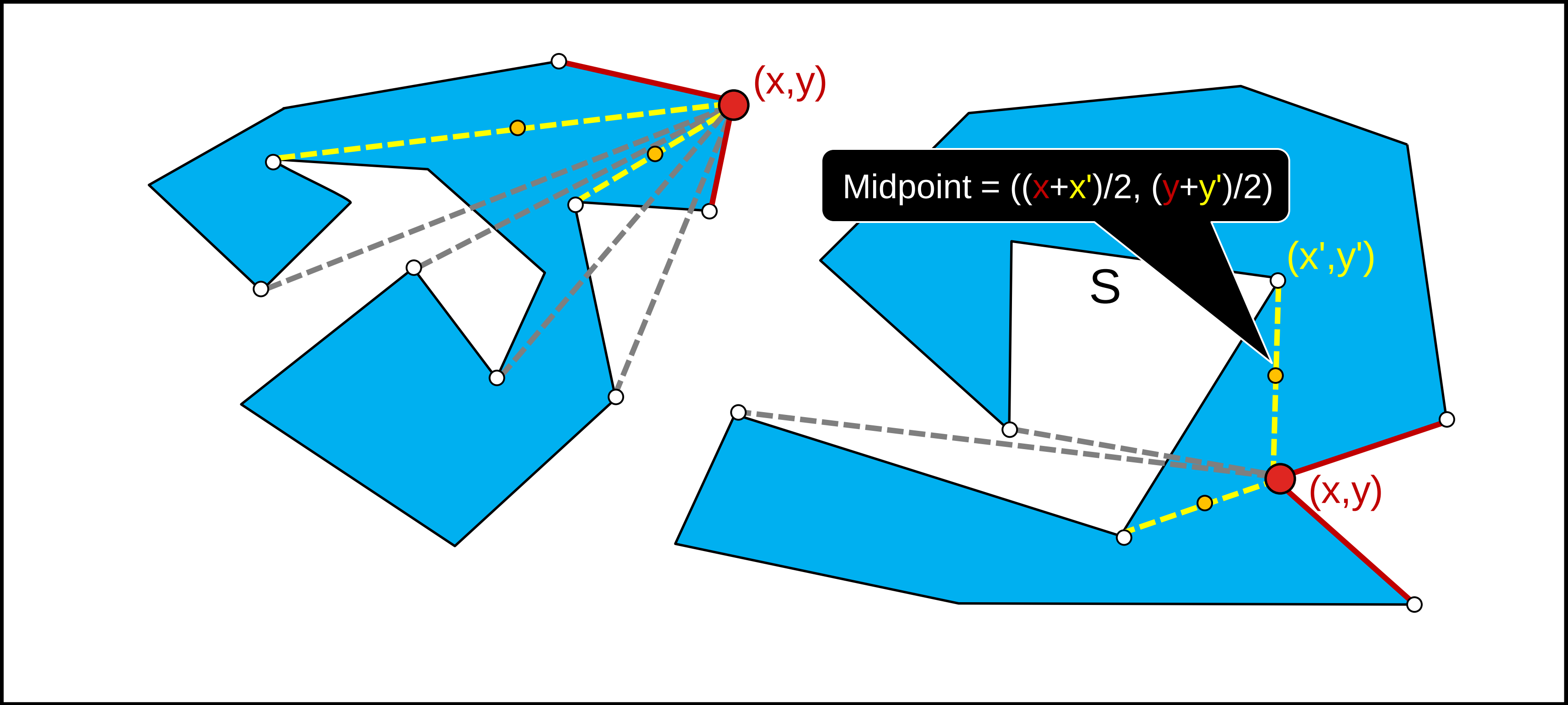
Need to get here to see the green area

Need to get here to see the pink area

Need to get here to see the yellow area

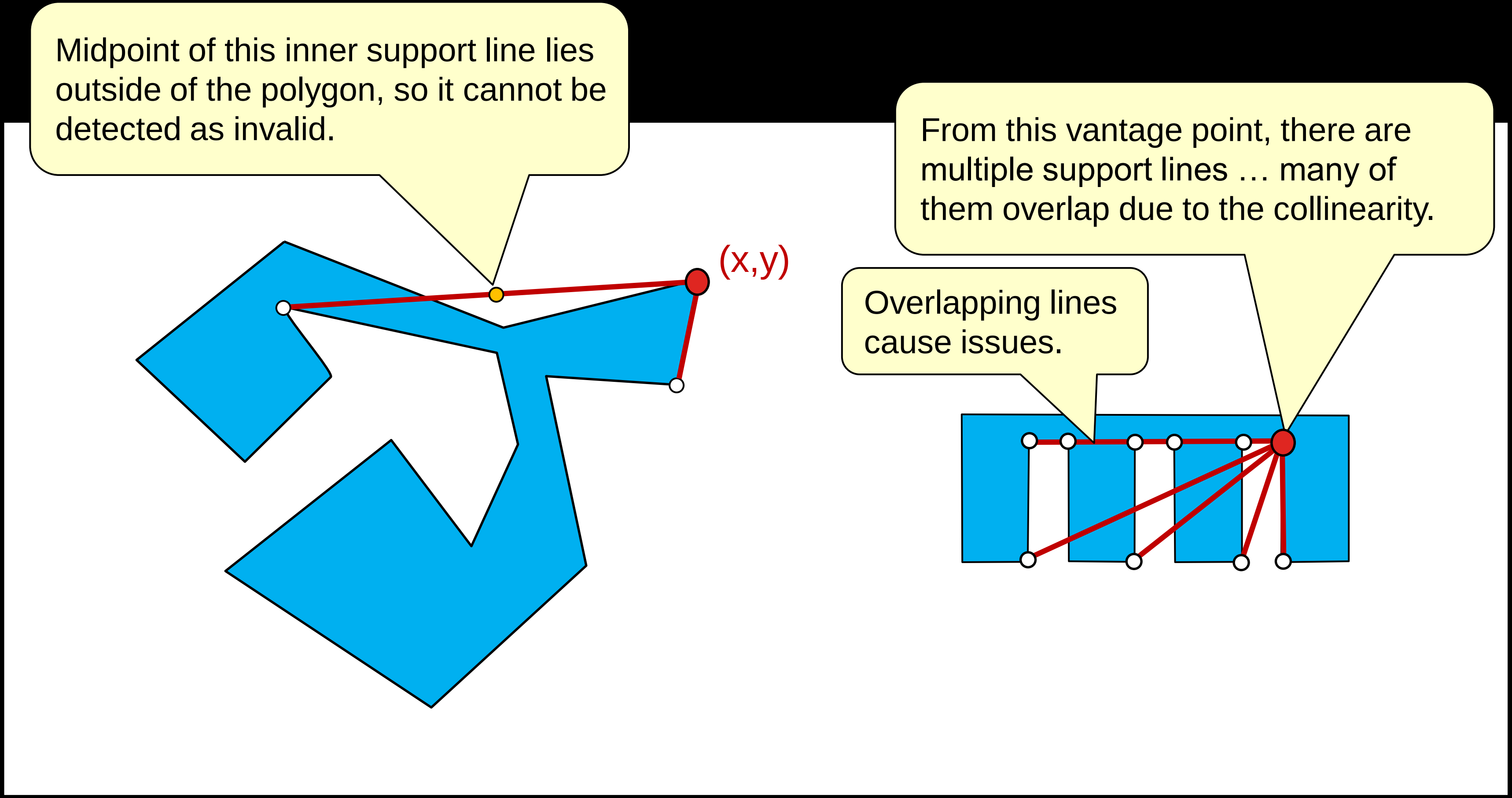
Non–Convex Problems

- Some “inner” support lines intersect edges only at vertices (see yellow below) and these are invalid.
- Perhaps we could check if midpoint of line lies within obstacle



Non–Convex Problems

- But there are cases where this will not work:



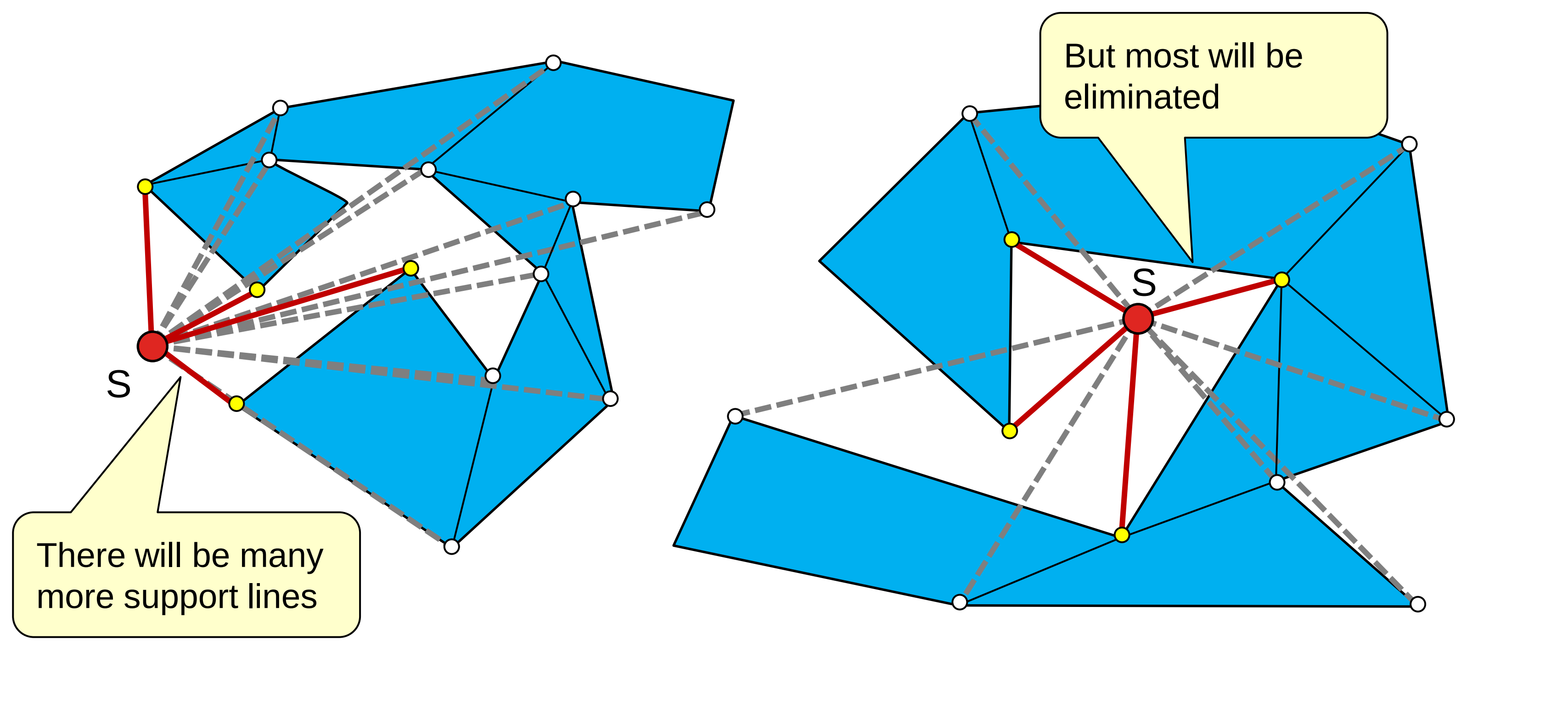
Midpoint of this inner support line lies outside of the polygon, so it cannot be detected as invalid.

From this vantage point, there are multiple support lines … many of them overlap due to the collinearity.

Overlapping lines cause issues.

Non–Convex Easy Solution

- Easiest solution is to break up the non-convex polygons into convex pieces. Then the algorithm from before should work.

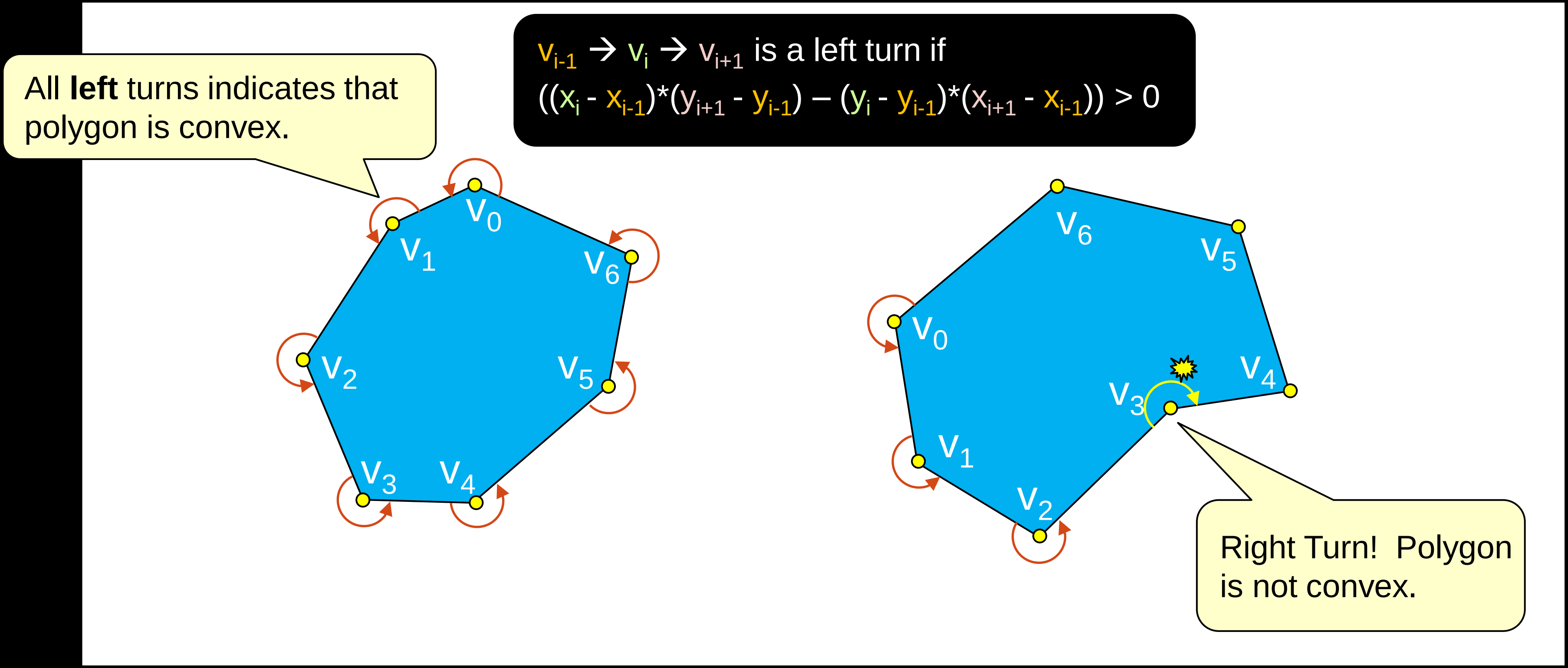


There will be many more support lines

But most will be eliminated

Convexity Checking

- How do we know if a polygon is convex or not ?
  - traverse vertices CCW … all must make **left** turns:



$v_{i-1} \rightarrow v_i \rightarrow v_{i+1}$ is a left turn if

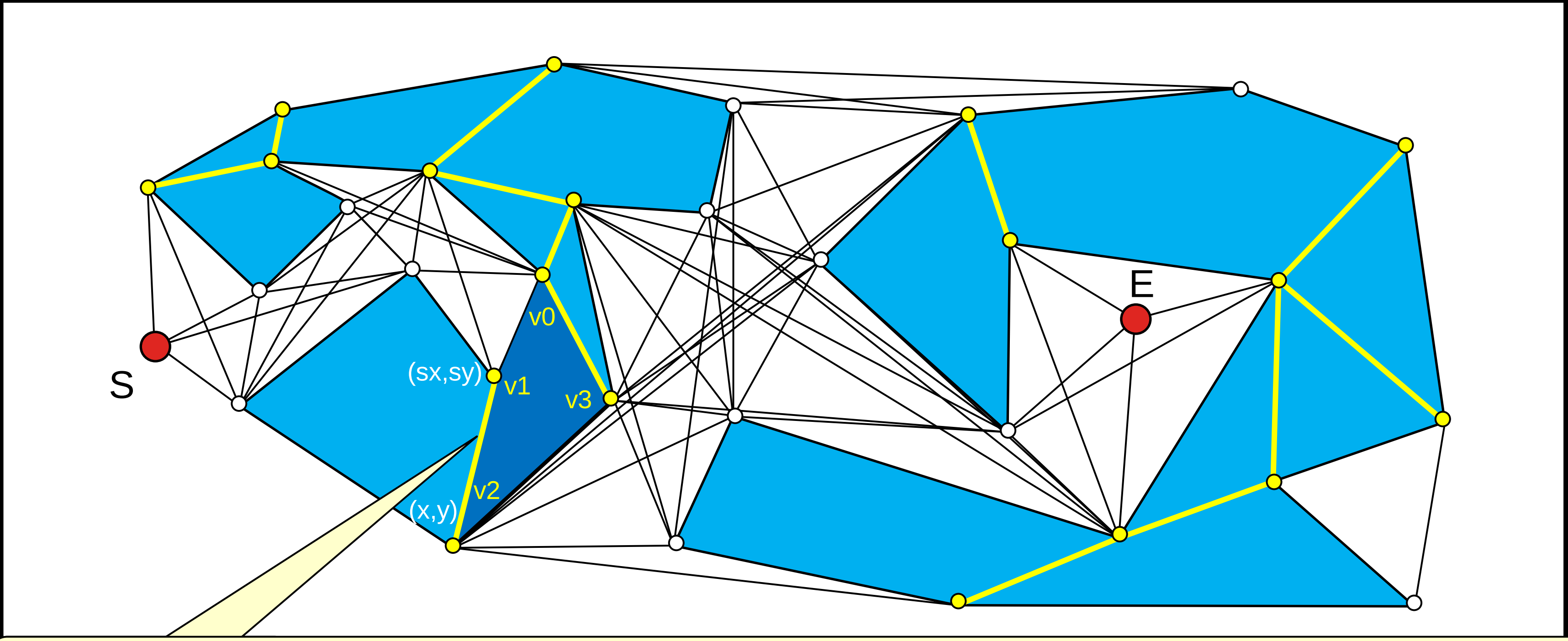
$((x_i - x_{i-1})*(y_{i+1} - y_{i-1}) – (y_i - y_{i-1})*(x_{i+1} - x_{i-1})) > 0$

All **left** turns indicates that polygon is convex.

Right Turn! Polygon is not convex.

Non–Convex Easy Solution

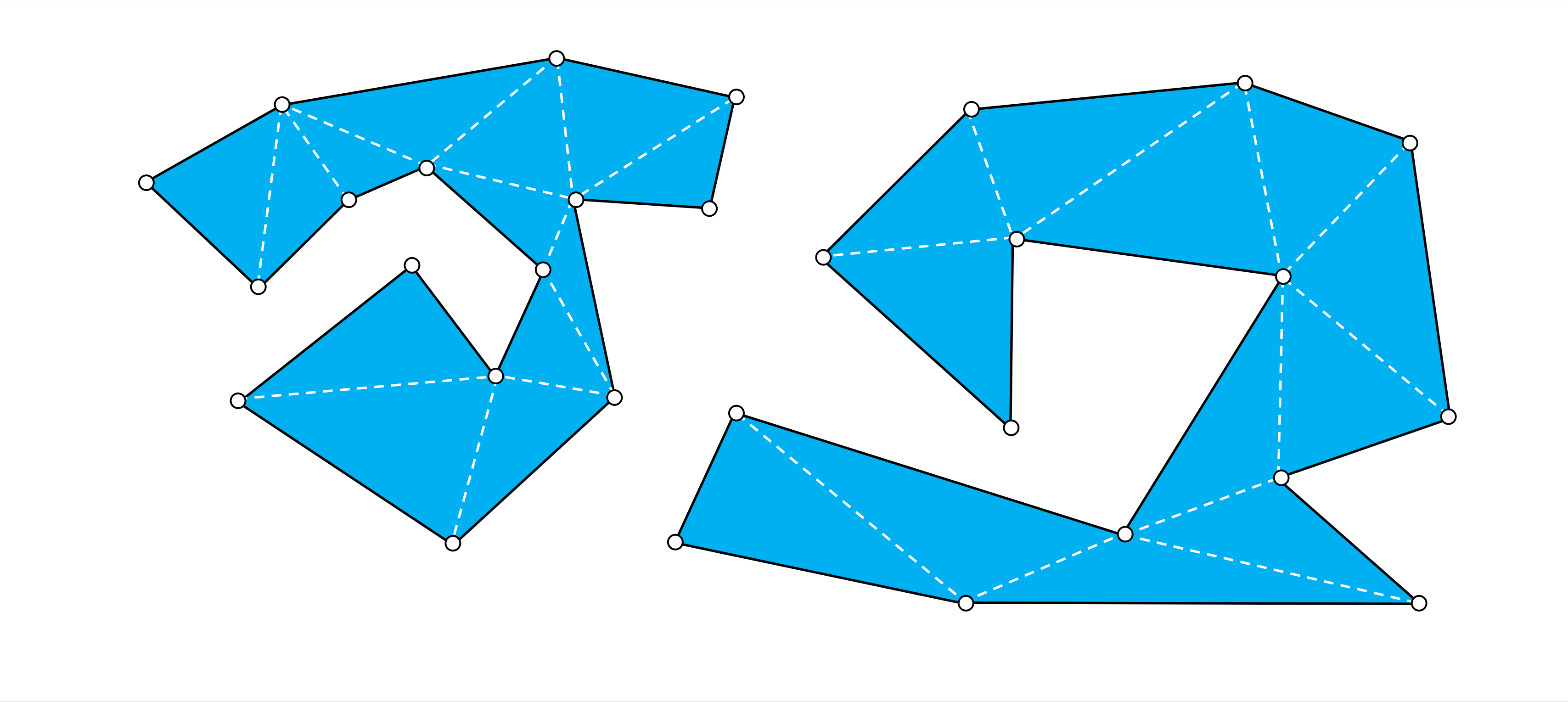
- There will be special cases for shared edges between convex pieces that need to be eliminated when computing the visibility graph:



Must make sure that none of these (yellow) *interior* bordering support lines are in the graph. Just need to check if support line (x,y)→(sx,sy) has same vertex coordinates as edge(v1,v2) … or edge (v2,v1).

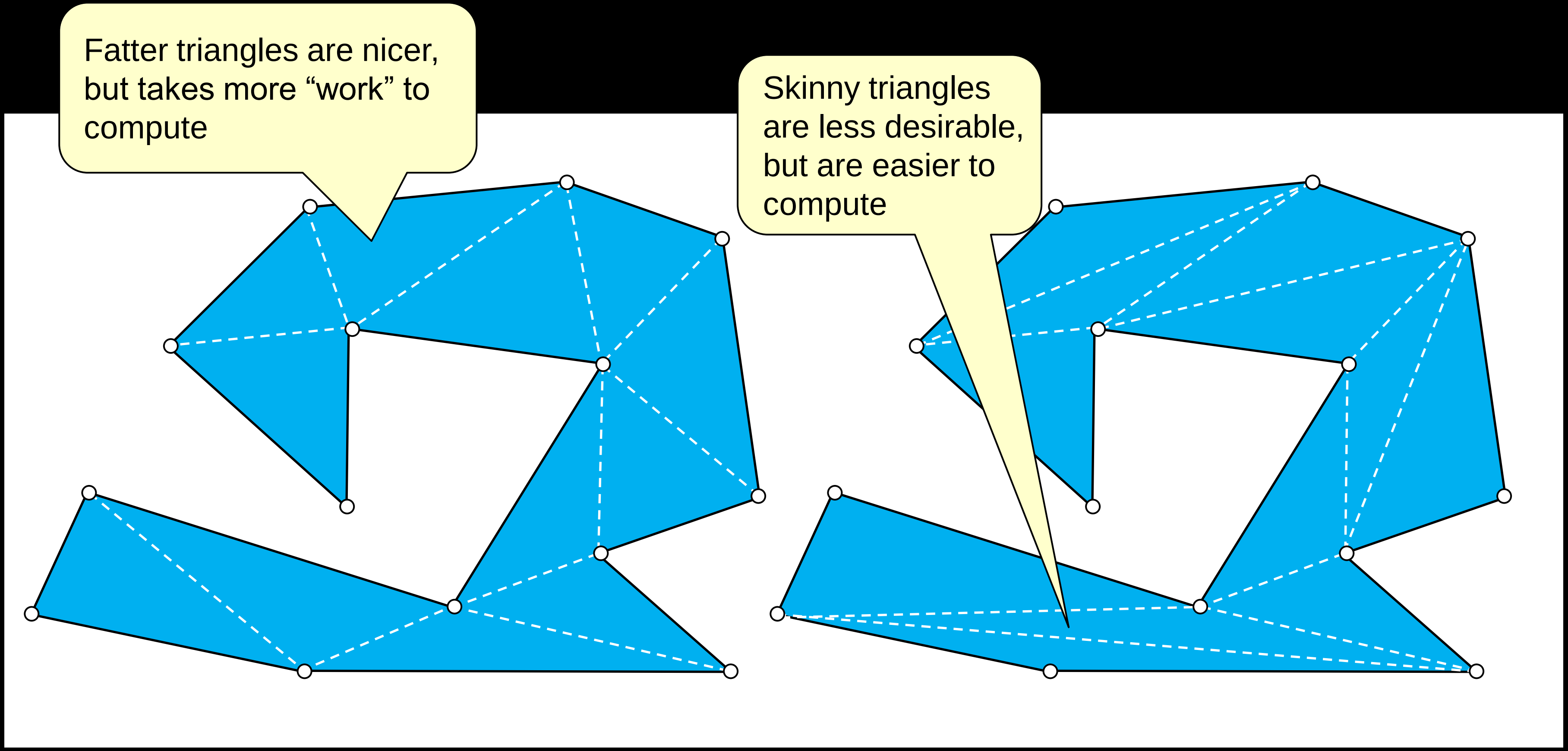
Convert Non–Convex To Convex

- How do we break a polygon into convex pieces ?
  - There are many algorithms.   The most popular is to break into triangles. This is known as *triangulation*.



Triangulation

- How do we triangulate a polygon ?
  - There are many ways/algorithms to do this as well.

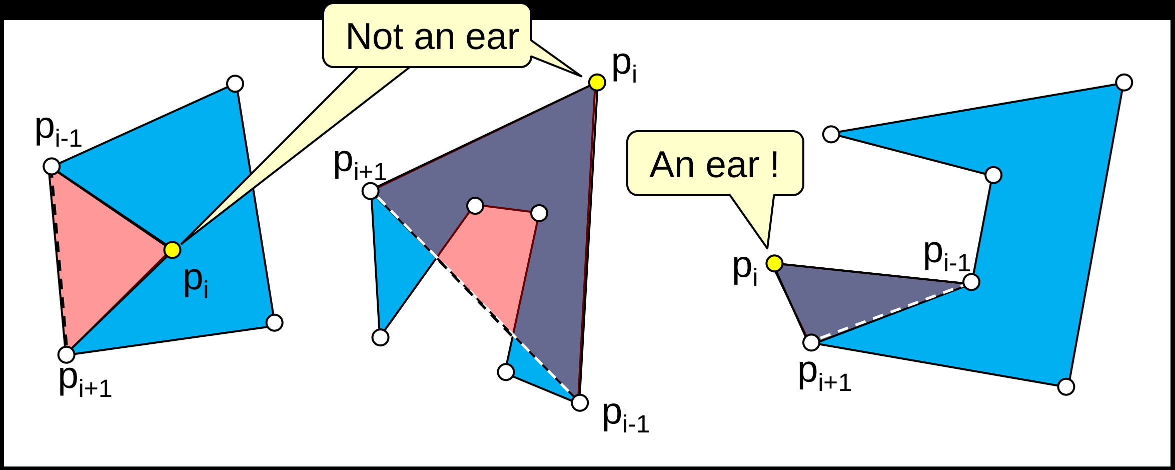


Fatter triangles are nicer, but takes more “work” to compute

Skinny triangles are less desirable, but are easier to compute

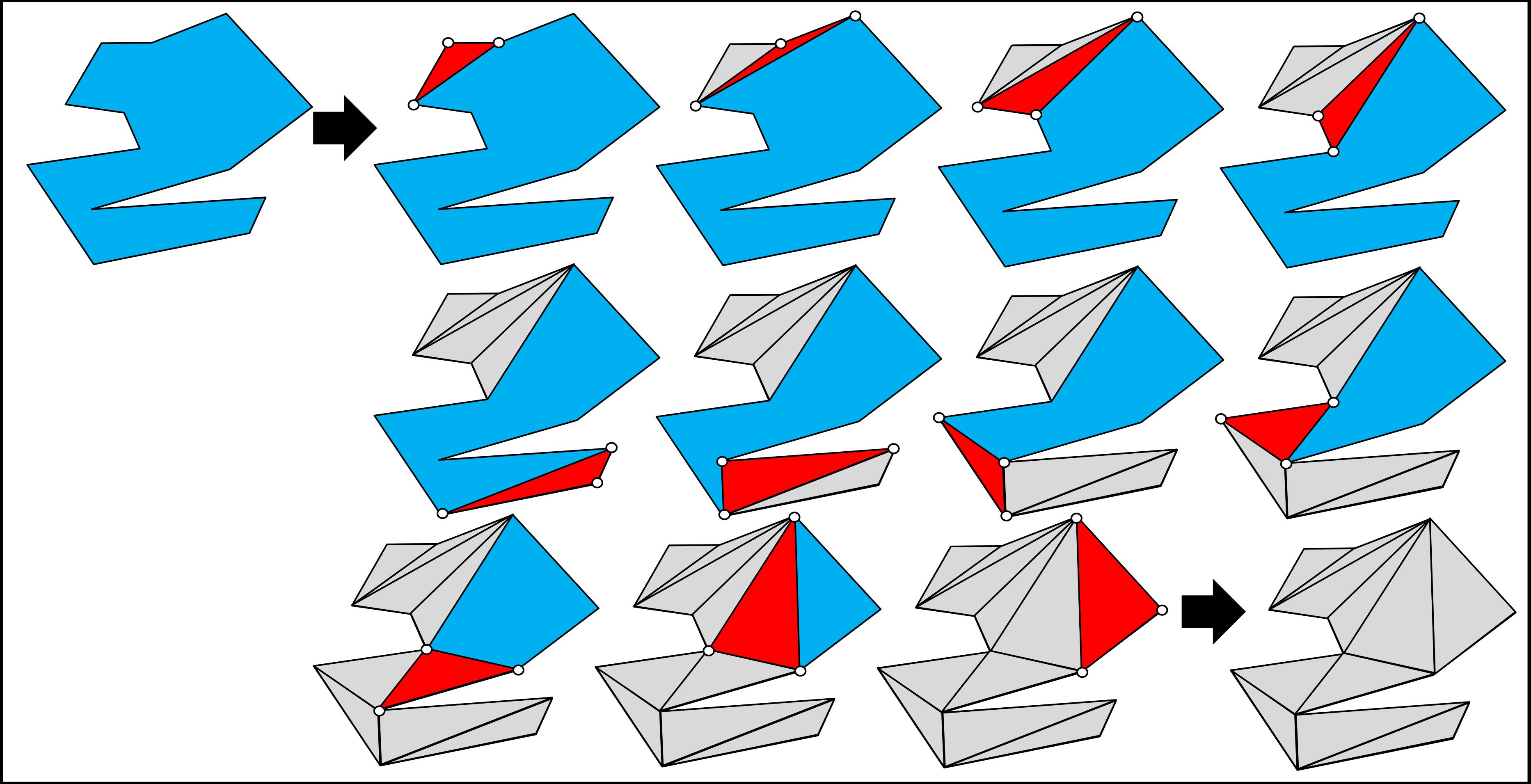
Ear-Cutting Triangulation

- We will use an “ear-cutting” algorithm since it is easy.
  - Idea is to repeatedly cut off “ears” of the polygon, making a set of triangles along the way.
  - An “ear” is a vertex p_i such that Line segment $\overline{p_{i-1}p_{i+1}}$ …
    1. intersects the polygon boundary only at points p_{i-1} and p_{i+1} AND…
    2. it lies entirely inside the polygon.



Ear-Cutting Triangulation

- Repeatedly cut off “ears” of the polygon, making a set of triangles along the way.



Ear-Cutting Algorithm

```
1  function EarCut(Obstacle P)
2      triangles = an empty list
3      ear = null;
4      if (P has only 3 vertices) then
5          Add P to triangles and return triangles
6      Copy all vertices of P into a list of points Q
7      while (Q has more than 3 points) do
8          earIndex = -1
9          for each point $p_i$ in Q do
10             if ($p_{i-1}p_ip_{i+1}$ is a left turn) then
11                 ear = a new obstacle with vertices $p_{i-1}p_i$ and $p_{i+1}$
12                 earIndex = i
13                 for each point $p_k$ in Q (such that $k \neq i-1$, $k \neq i$, $k \neq i+1$) do
14                     if (point $p_k$ lies inside or on boundary of the ear) then
15                         earIndex = -1
16                 if (earIndex != -1) then
17                     Break out of the FOR loop at line 8 … we found an ear
18         if (earIndex == -1) quit and return triangles, since no more ears were found
19         Remove $p_{earIndex}$ from Q
20         Add ear to triangles
21     Add (a new obstacle with vertices $p_0$, $p_1$ and $p_2$) to triangles
22     return triangles
```

Nothing to do if only 3 vertices

Copy obstacle vertices into a new list that we can add/remove from without destroying the original obstacle.

Find an ear

Left turn if CCW ordering and $((x_i - x_{i-1})*(y_{i+1} - y_{i-1}) – (y_i - y_{i-1})*(x_{i+1} - x_{i-1})) > 0$

reject p_i as an ear since it is invalid

Cut the ear off and add to the solution

The last 3 points form an ear, so add it too

Start the Lab …