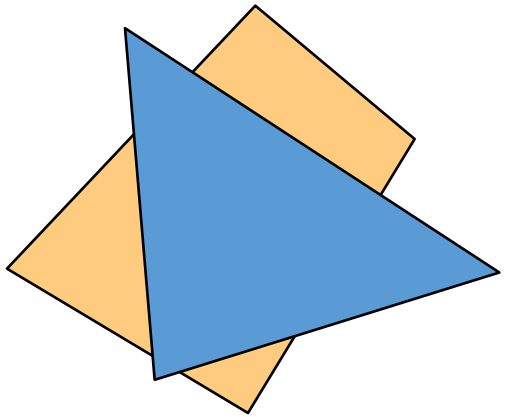
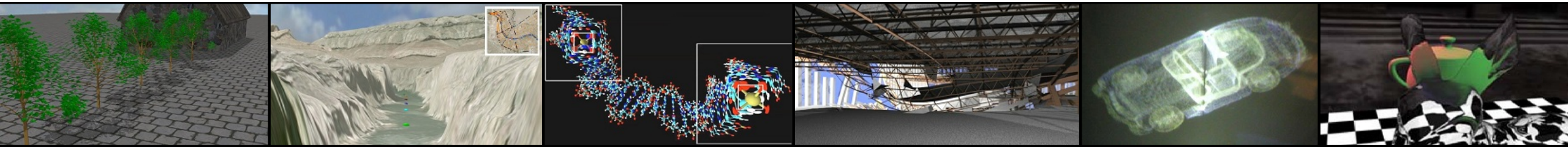
COT 4521: Introduction to Computational Geometry

Convex Polygon Intersection

Paul Rosen
Assistant Professor
University of South Florida

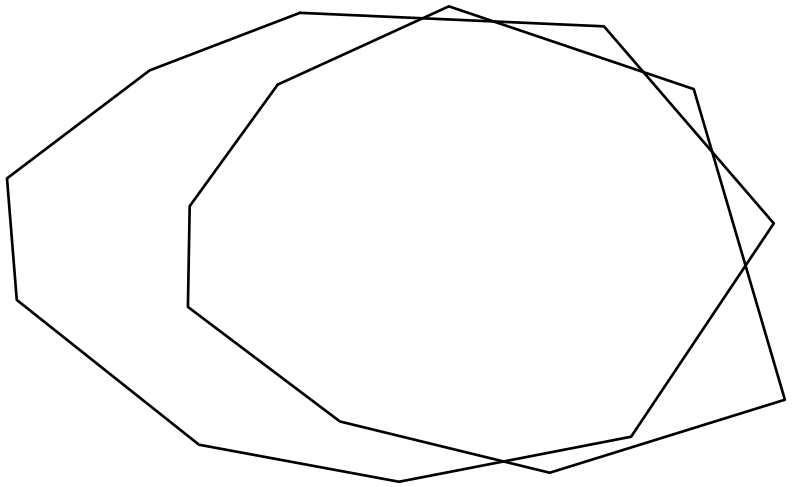
Some slides from Valentina Korzhova

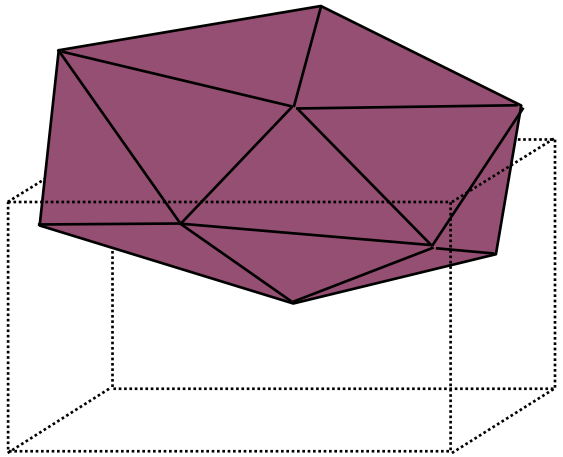




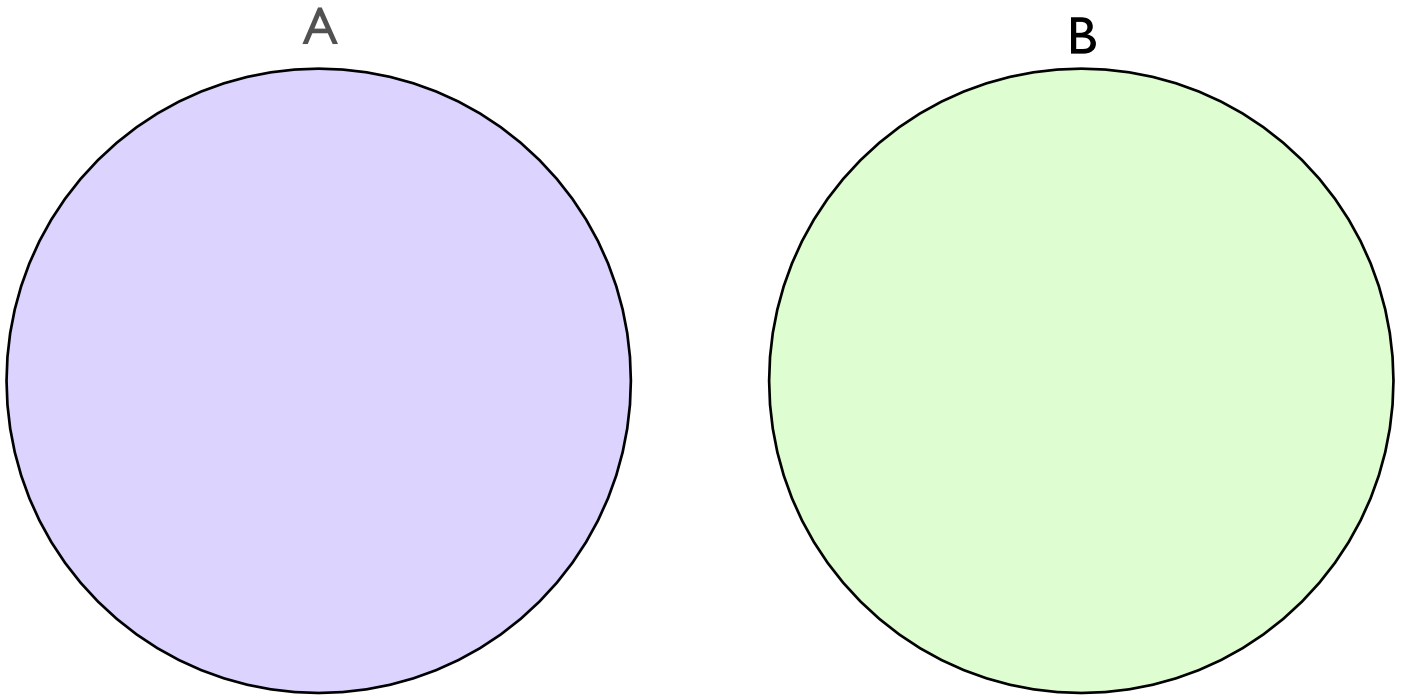
Problem description

- Here we consider a category of related intersection construction problems: Given two (or more) geometric objects, construct a new object which is their intersection.
  - The objects dealt with will be polygons, polyhedra, half-planes, half-spaces, and related types.
  - The objects will generally be specified by their vertices and orientations to identify the interior of the object. (The latter may be given by convention.)

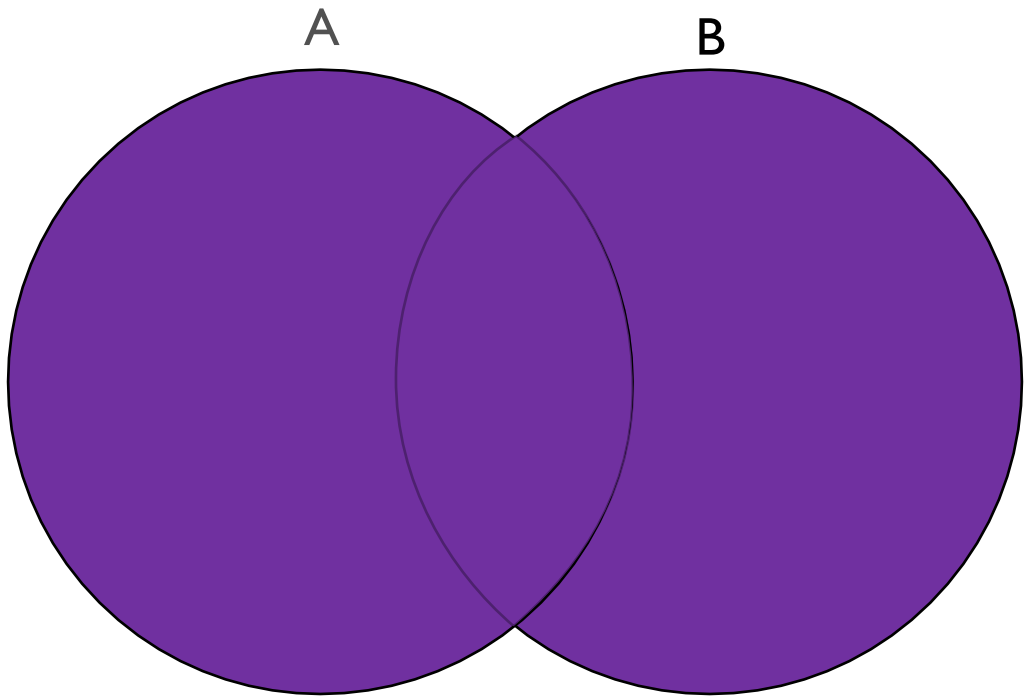




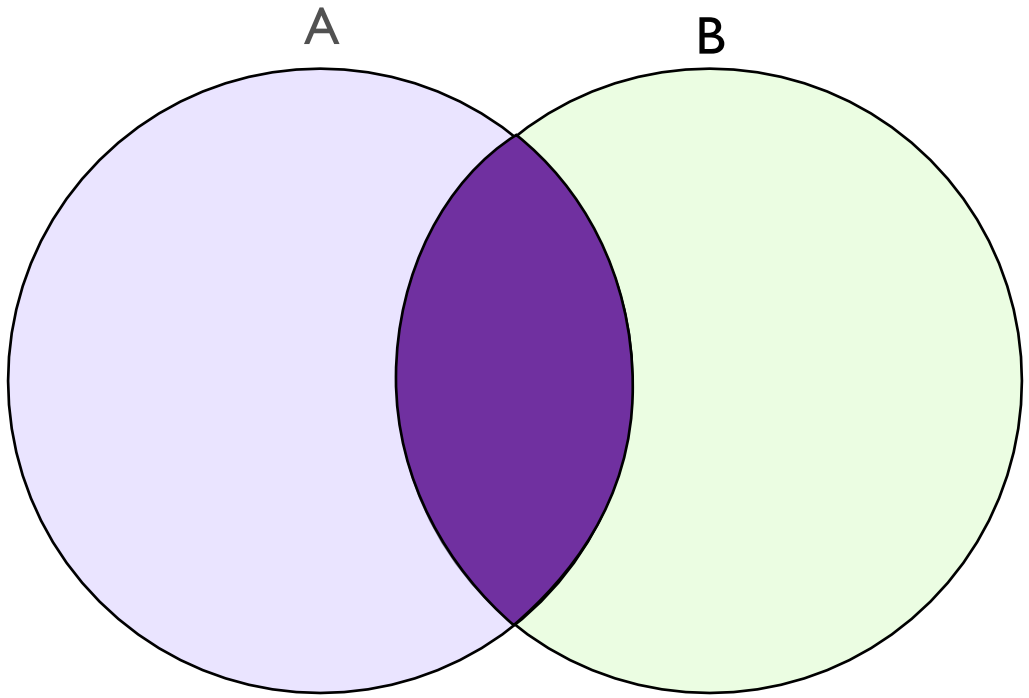
Brief set theory Review



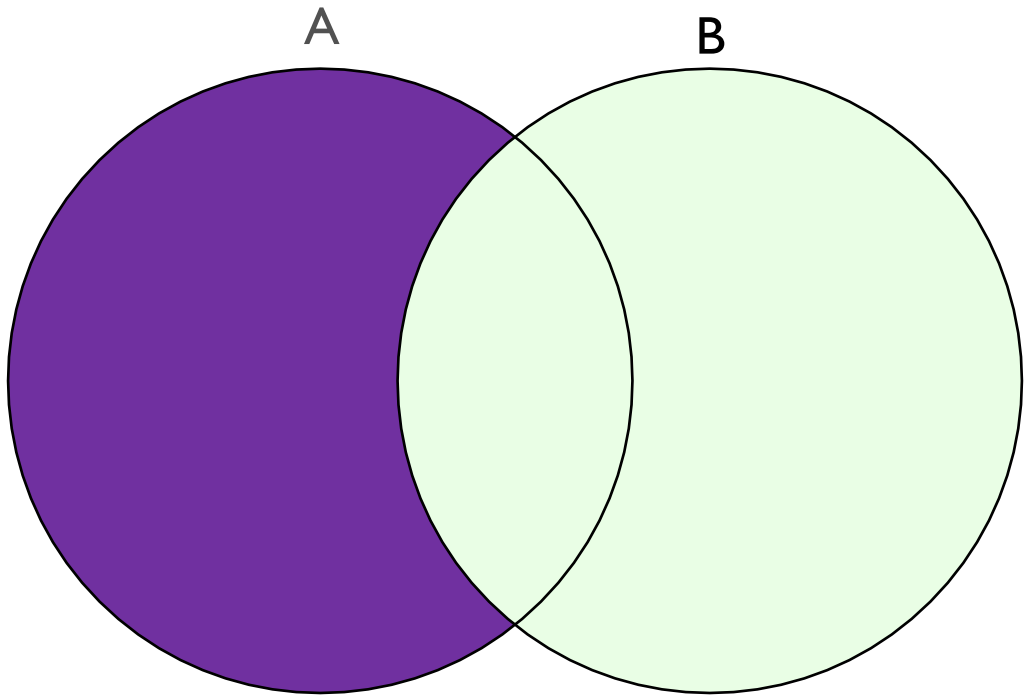
SET THEORY—UNION: A ∪ B



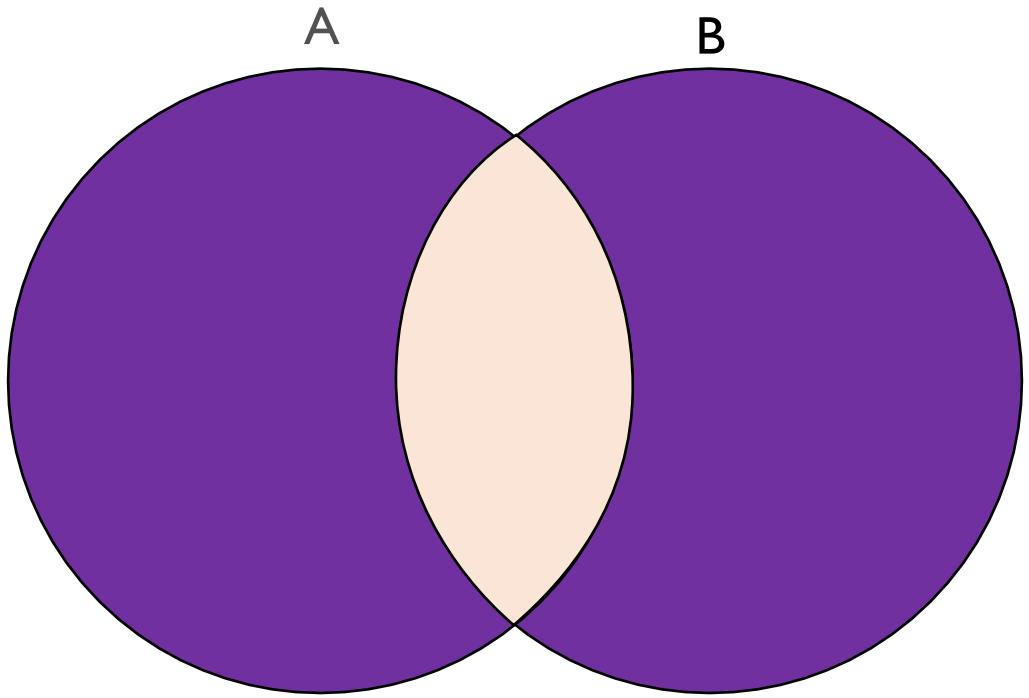
SET THEORY—INTERSECTION: $A \cap B$



SET THEORY—SET DIFFERENCE: $A \setminus B$

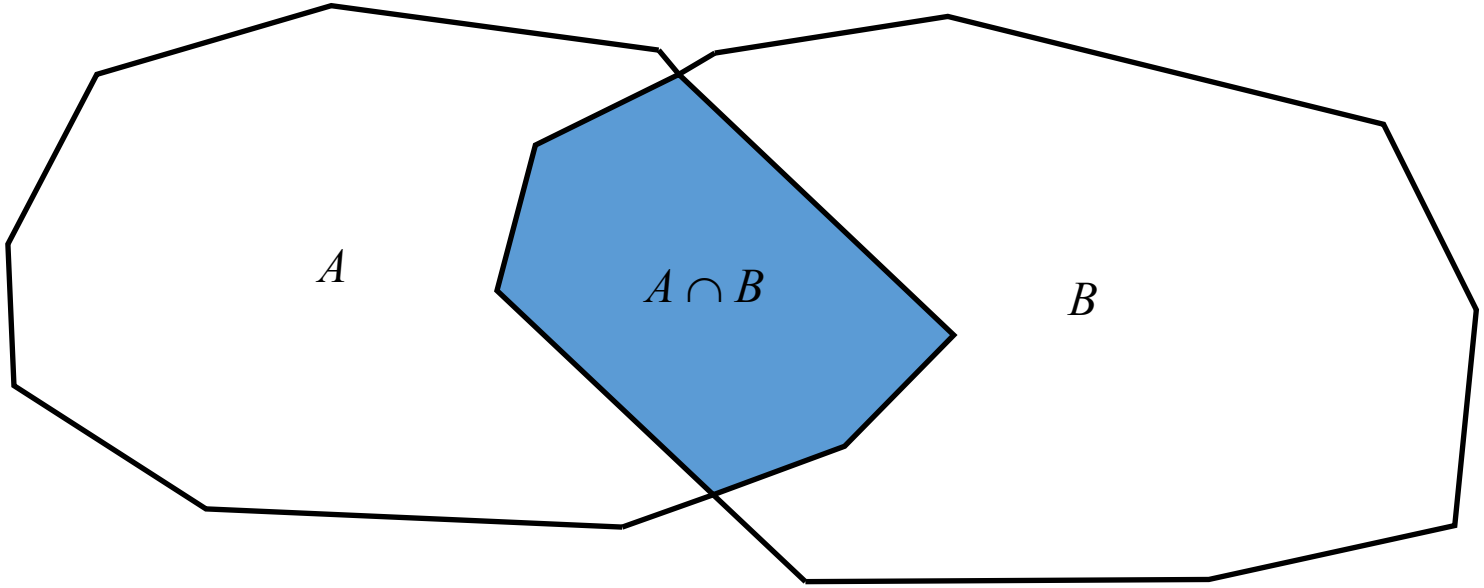


SET THEORY—SYMMETRIC DIFFERENCE: $A \ominus B$



INTERSECTION EXAMPLE

- GIVEN TWO CONVEX POLYGONS, CONSTRUCT THEIR INTERSECTION.
  - (Polygon ≡ boundary and interior, intersection ≡ all points that are members of both polygons.)

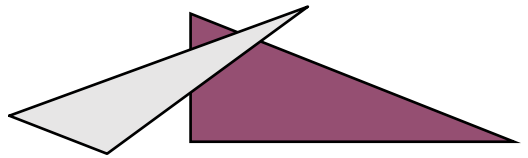


General intersection problems

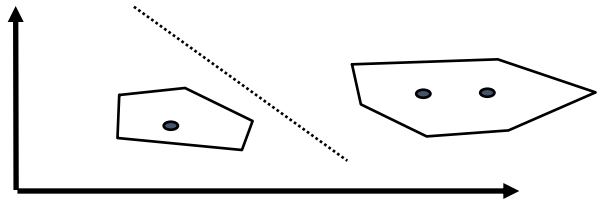
- Test or decision problem
  - Given two geometric objects, determine if they intersect.
- Pairwise counting or reporting problem
  - Given a data set of N geometric objects, count or report their intersections.
- Construction problem
  - Given a data set of N geometric objects, construct a new object which is their intersection.

APPLICATIONS

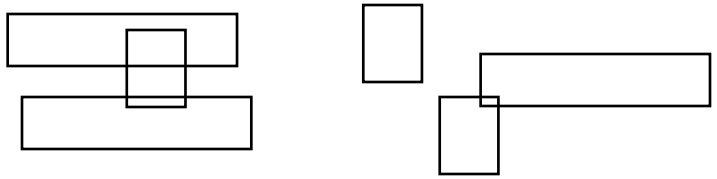
- DOMAIN: GRAPHICS
  - Problem: Hidden-line and hidden surface removal
  - Approach: Intersection of two polygons
  - Type: Construction



- DOMAIN: PATTERN RECOGNITION
  - Problem: Finding a linear classifier between two sets of points
  - Approach: Intersection of convex hulls
  - Type: Test

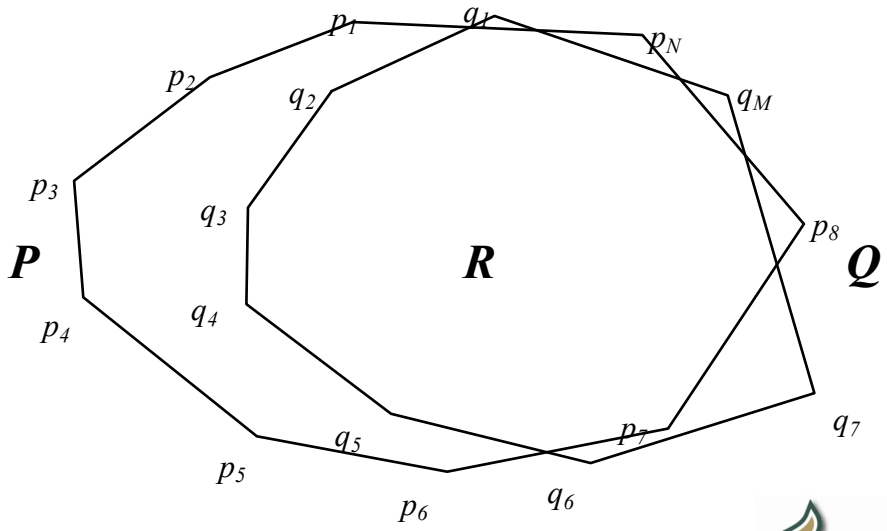


- DOMAIN: VLSI DESIGN
  - Problem: Component overlap detection
  - Approach: Intersection of rectangles
  - Type: Pairwise



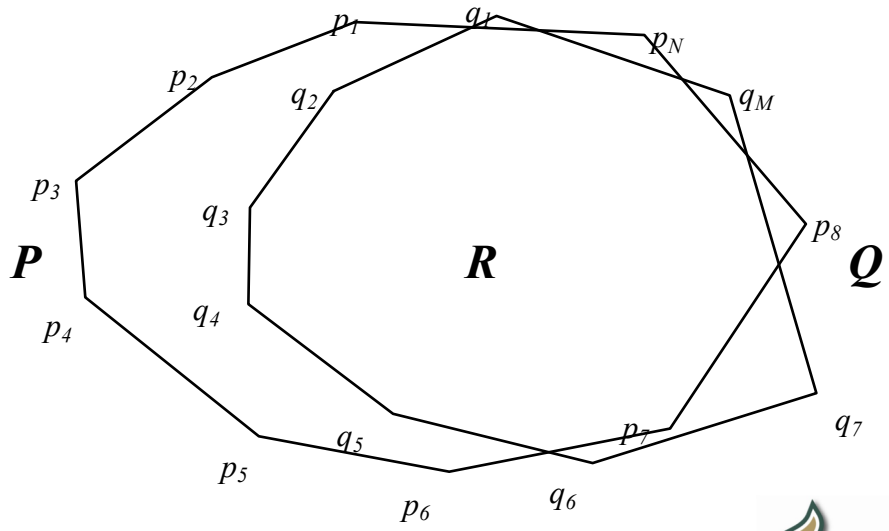
CONVEX POLYGON INTERSECTION

- DEFINITION: For convex (rather than arbitrary) polygons, problem is linear.
- INSTANCE: Convex polygons P and Q, with vertex sets $P = \{p_1, p_2, \ldots, p_N\}$ and $Q = \{q_1, q_2, \ldots, q_M\}$ respectively.
- QUESTION: Construct polygon R which is their intersection.



Intersection of convex polygons

- The intersection of two convex polygons will have linear $O(N + M)$ complexity (both the object and the construction process).
- By convention, P, Q, and R will be oriented counterclockwise, so that the interiors of the polygons lie to the left of their edges



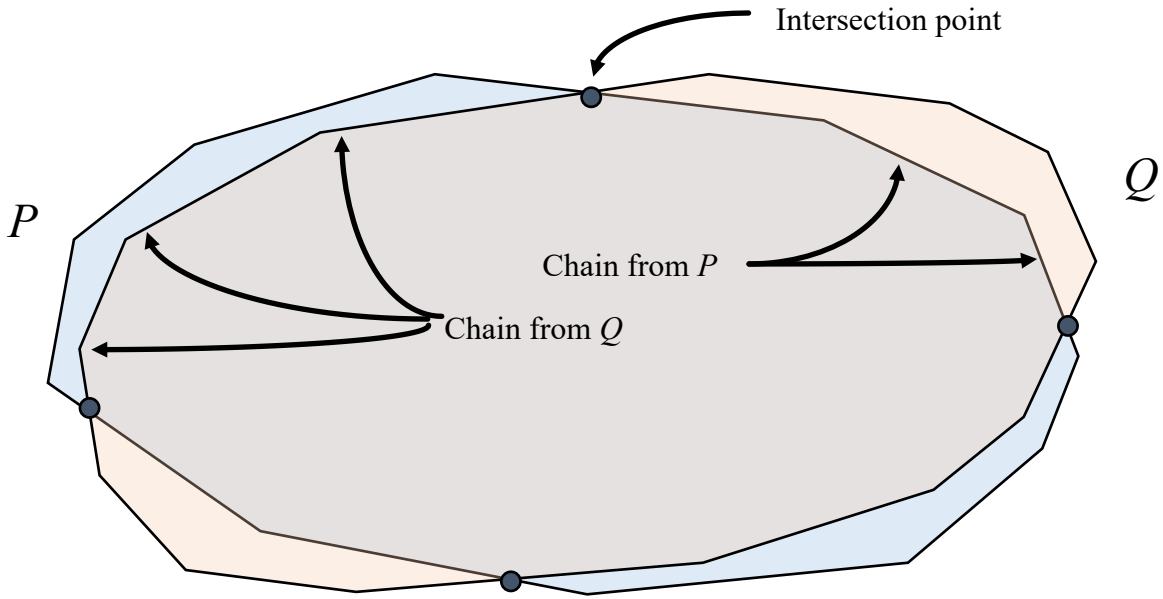
O'Rourke-Chien-Olson-Naddor algorithm

- This algorithm described in three sources:
  1. O'Rourke with C code.
  2. Preparata
  3. Laszlo with C++ code.
     - Presentation is a synthesis of all three with focus on Laszlo.

- The algorithm is based on three undergraduates' homework assignment solutions in 1982.
  - Simpler than previously existing linear O(N + M) algorithm, e.g., Shamos' 1978 algorithm.

Intersection of convex polygons

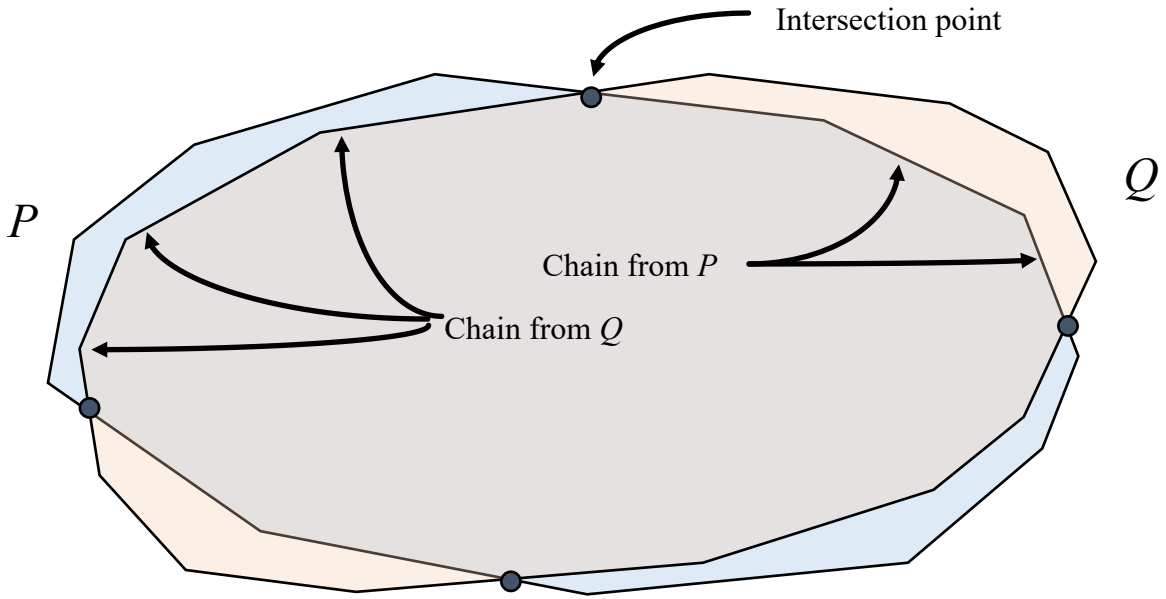
- Intersection structure
  - Form the intersection polygon R of two convex polygons P and Q
  - For the moment, assume that the edges of P and Q intersect non-degenerately* (when two edges intersect, they do so at a single point)



*In fact, the algorithm will handle degenerate intersections.

INTERSECTION OF CONVEX POLYGONS

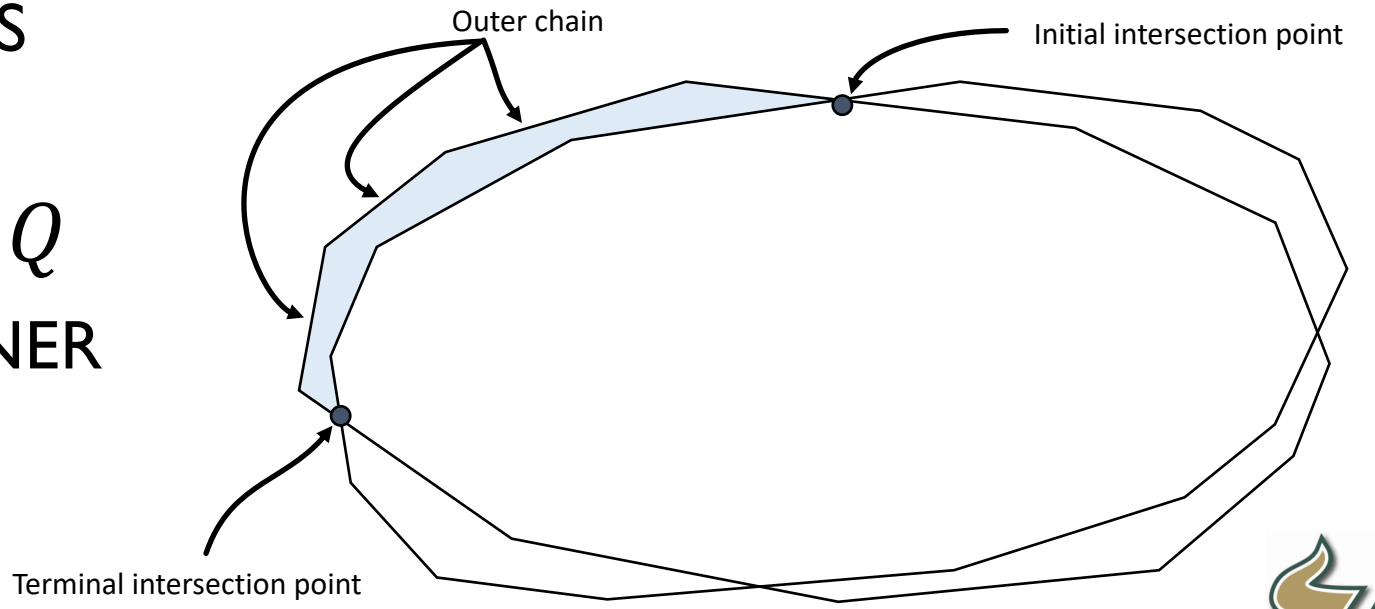
- GIVEN THIS ASSUMPTION
  - The boundary of $R = P \cap Q$ consists of alternating chains of vertices from P and Q
  - Consecutive chains are joined at intersection points, where an edge of P intersects an edge of Q



*In fact, the algorithm will handle degenerate intersections.

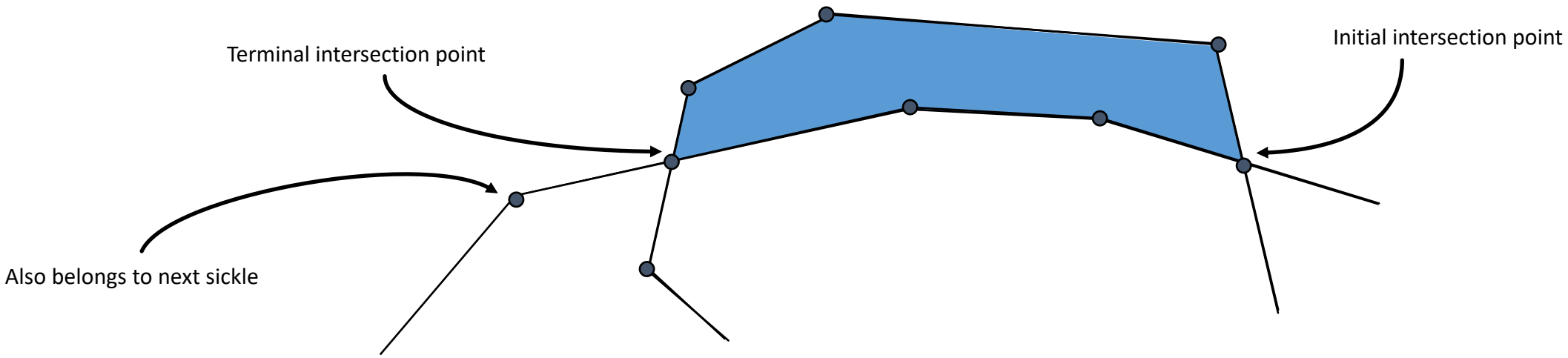
Intersection of convex polygons

- Regions that are within one polygon, but not both, are called **sickles**.
- Sickles are bounded by two chains (inner and outer) of vertices, one from each polygon, and terminated by intersection points at either end
- The boundary of $P \cap Q$ is formed from the inner chains of the sickles



SICKLES

- A VERTEX (OF P OR Q) BELONGS TO A SICKLE EITHER IF:
  1. It lies between the initial and terminal intersection points on either chain.
  2. It is the destination of the edge of the internal chain containing the terminal intersection point of the sickle.
     - Note that some vertices may belong to > 1 sickles.



Algorithm overview

- Initially random starting edges are selected from each polygon

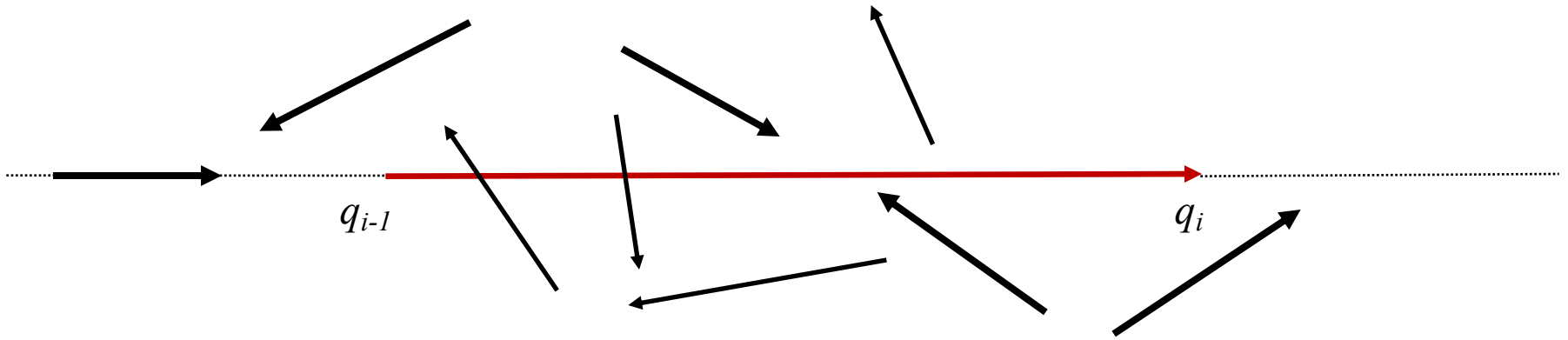
- The algorithm then has two phases, both handled by same advance rules:
  1. Get current edges $edge(p_{i-1}p_i)$ and $edge(q_{j-1}q_j)$ into same sickle
  2. Advance current edges $edge(p_{i-1}p_i)$ and $edge(q_{j-1}q_j)$ together through each sickle
     - Before either leaves the current sickle for the next sickle the terminal intersection point will be found
     - Add inner chain vertices to R

PSEUDOCODE

```
1.  Define current edges of P and Q:
2.    edge(p_{i-1}p_i) and edge(q_{j-1}q_j), with destinations p_i and q_j,
    respectively.
3.    p_0 = p_N and q_0 = q_M, by definition.

4.  IntersectConvexPolygons(P, Q)  /* Informal */

5.    i = j = 1  /* Arbitrarily start with vertex 1 of each polygon. */

6.    repeat
7.      if  ((w = edge(p_{i-1}p_i) ∩ edge(q_{j-1}q_j)) ≠ ∅) then
8.        add w to R

9.      Select current edge to advance based on configuration.

10.     if (selected edge is on inner chain) then
11.       add selected edge destination to R
12.       increment index of selected edge
13.     end if

14.   until iterations > 3(N + M)

15.   if (R = ∅) then  /* Boundaries of P and Q don't intersect */
16.     resolve special cases P ⊆ Q, Q ⊆ P, P ∩ Q = ∅
17. end function
```

AIMING AT

- ADVANCE RULES ARE USED TO SELECT THE CURRENT EDGE TO ADVANCE.
  - The advance rules depend on the notion of "aiming at".
  - The bold arrows aim at $edge(q_{j-1}q_j)$.



- IF THEY ARE NOT COLLINEAR,
  $edge(p_{i-1}p_i)$ AIMS AT $edge(q_{j-1}q_j)$ IF EITHER OF THESE CONDITIONS HOLD:
  1. $edge(p_{i-1}p_i) \times edge(q_{j-1}q_j) > 0$ and p_i is in the left *half plane* of $edge(q_{j-1}q_j)$.
  2. $edge(p_{i-1}p_i) \times edge(q_{j-1}q_j) < 0$ and p_i is in the right *half plane* of $edge(q_{j-1}q_j)$.
- IF THEY ARE COLLINEAR,
  $edge(p_{i-1}p_i)$ AIMS AT $edge(q_{j-1}q_j)$ IF p_i DOES NOT LIE BEYOND q_{j-1}.

Half-plane

- A half-space is either of the two parts into which a hyperplane divides an affine space
- A half-space can be either open or closed
- In 2D, a half-space is called a half-plane

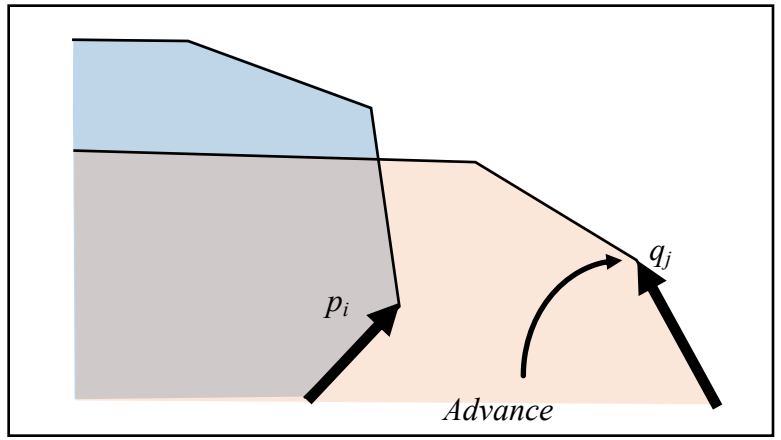
ADVANCE RULES, OVERALL INTENT

- ADVANCE RULES ARE USED TO SELECT THE CURRENT EDGE TO ADVANCE
- O'ROURKE:
  - 'If $edge(p_{i-1}p_i)$ aims at the line containing $edge(q_{j-1}q_j)$ but does not cross it, we want to advance $edge(p_{i-1}p_i)$ to close in on a possible intersection point with $edge(q_{j-1}q_j)$.'
- LASZLO:
  - 'The advance rules are designed so that the intersection point which should be found next is not skipped over. They distinguish between the current edge which may contain the next intersection point and the current edge which cannot possibly contain the next intersection point; the latter edge is advanced.'

Advance rules (by Laszlo)

- Case 1 – $edge(p_{i-1}p_i)$ and $edge(q_{j-1}q_j)$ aim at each other:
  - Advance whichever edge is to the right of the other

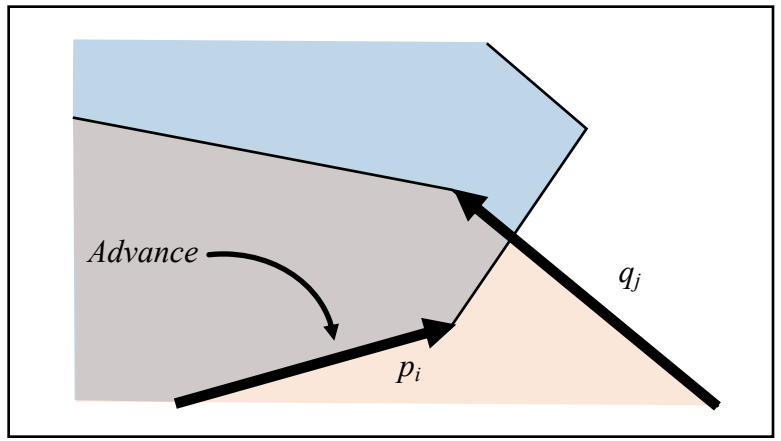
- Assume $edge(q_{j-1}q_j)$ is to the right; then the next intersection point cannot lie on $edge(q_{j-1}q_j)$ because q_j is outside the intersection polygon R.



ADVANCE RULES (BY LASZLO)

- CASE 2 – $edge(p_{i-1}p_i)$ AIMS AT $edge(q_{j-1}q_j)$ BUT NOT VICE VERSA
  - Add p_i to R if p_i is left of $edge(q_{j-1}q_j)$, then advance p_i

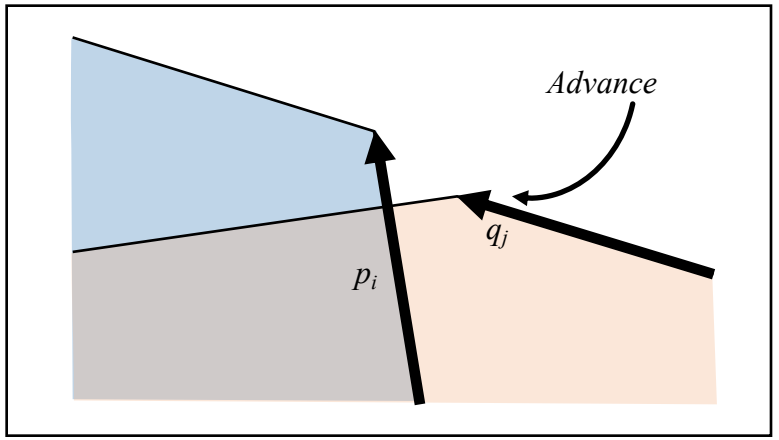
- $edge(p_{i-1}p_i)$ CANNOT CONTAIN THE NEXT INTERSECTION POINT. IN THE FIGURE, p_i **IS NOT** RIGHT OF $edge(q_{j-1}q_j)$, AND **IS** ADDED TO R



ADVANCE RULES (BY LASZLO)

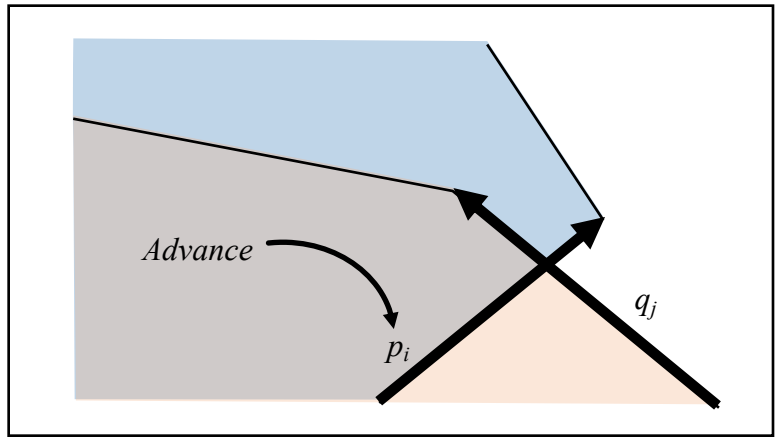
- CASE 3 – $edge(q_{j-1}q_j)$ AIMS AT $edge(p_{i-1}p_i)$ BUT NOT VICE VERSA:
  - Add q_j to R if q_j is left of $edge(p_{i-1}p_i)$, then advance q_j

- $edge(q_{j-1}q_j)$ CANNOT CONTAIN THE NEXT INTERSECTION POINT. IN THE FIGURE, q_j **IS** RIGHT OF $edge(p_{i-1}p_i)$, AND **IS NOT** ADDED TO R.

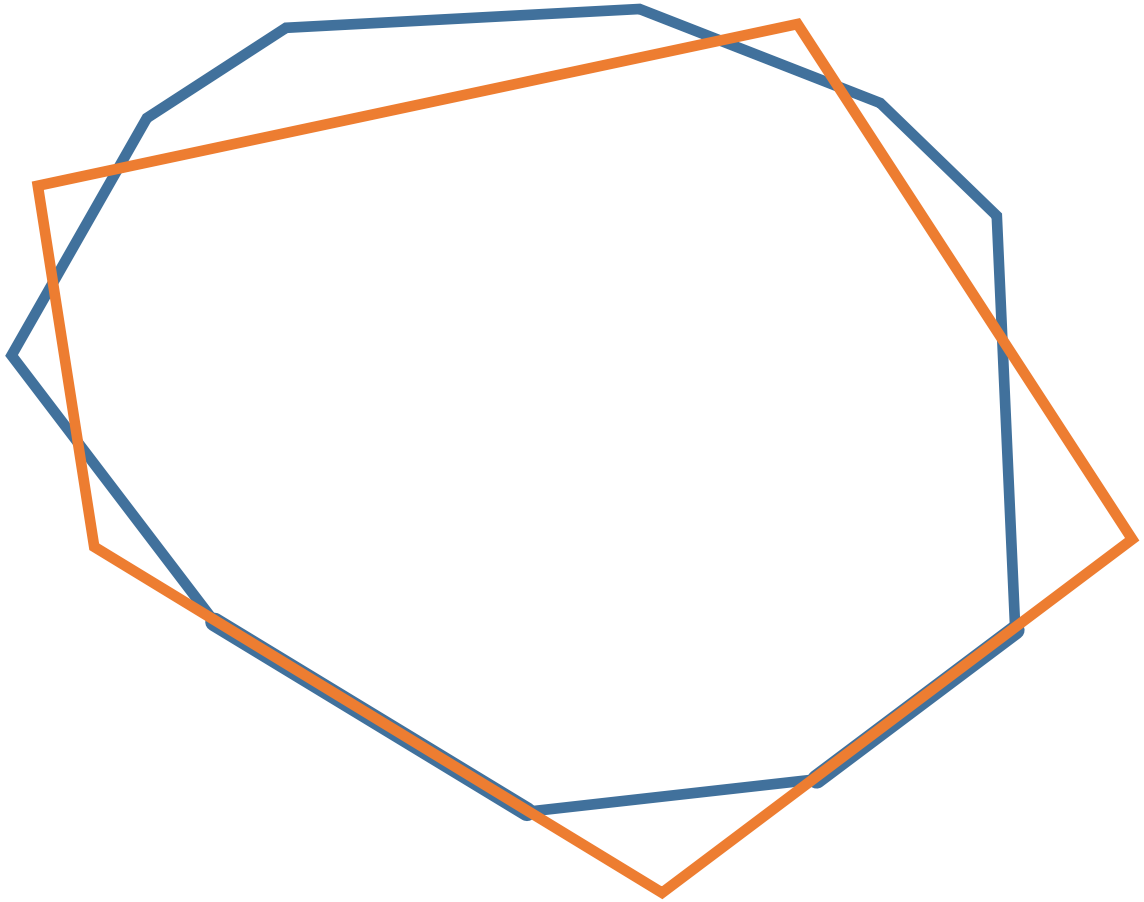


ADVANCE RULES (BY LASZLO)

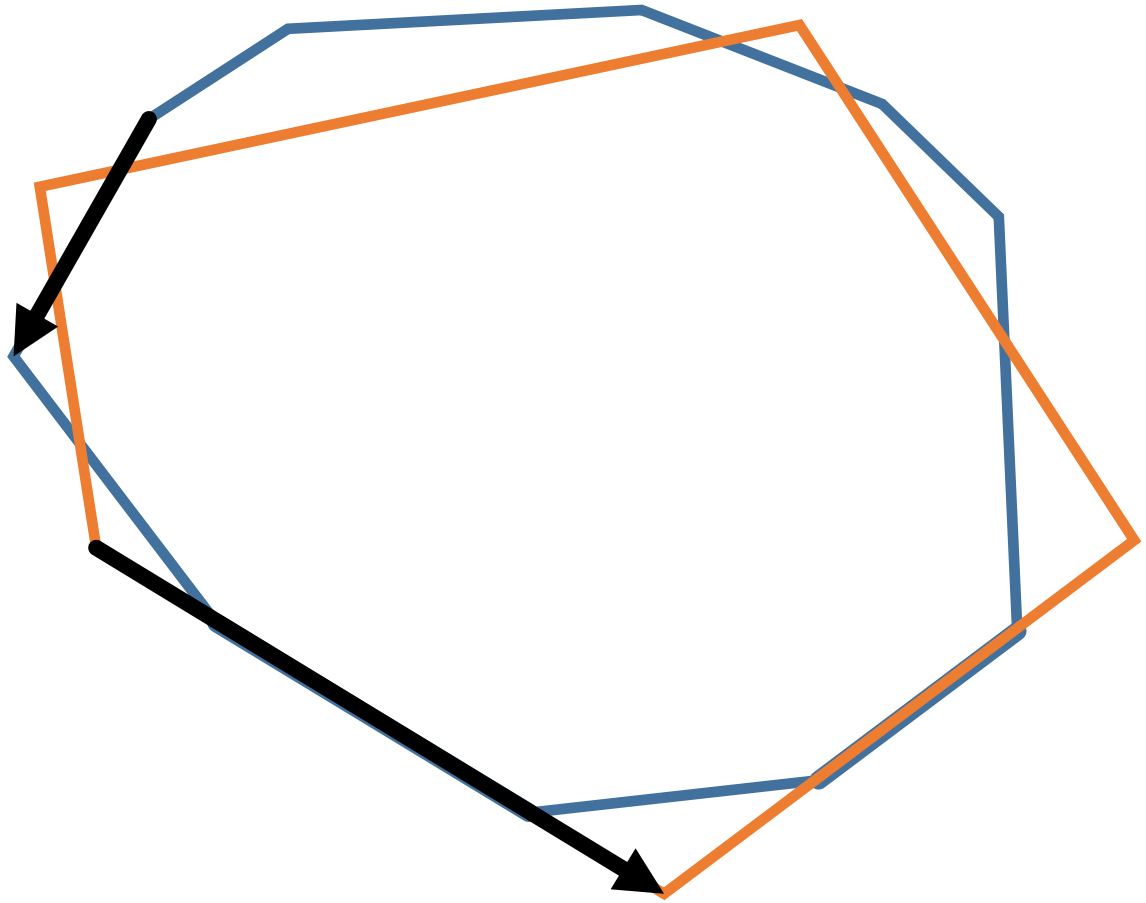
- CASE 4 – $edge(p_{i-1}p_i)$ AND $edge(q_{j-1}q_j)$ DO NOT AIM AT EACH OTHER:
  - Advance whichever edge ends to the right of the other.



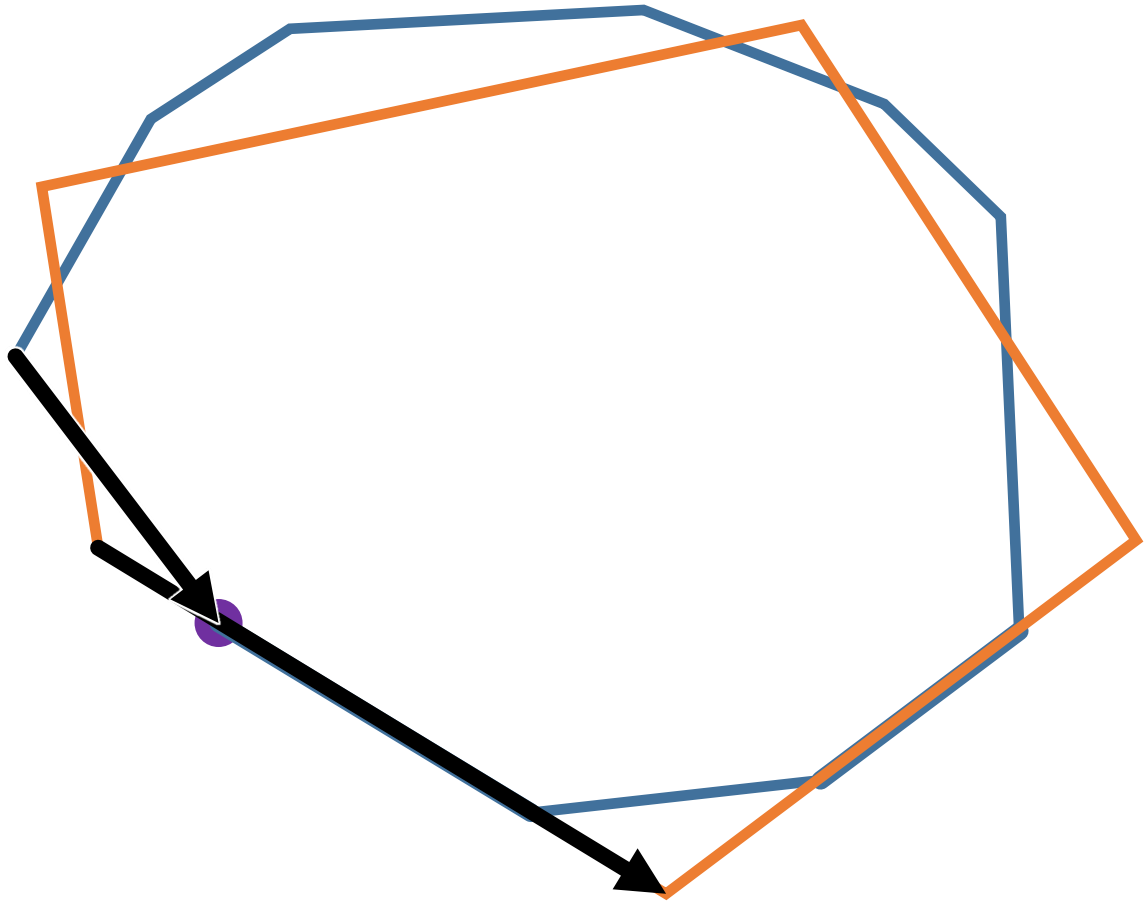
Intersection Example



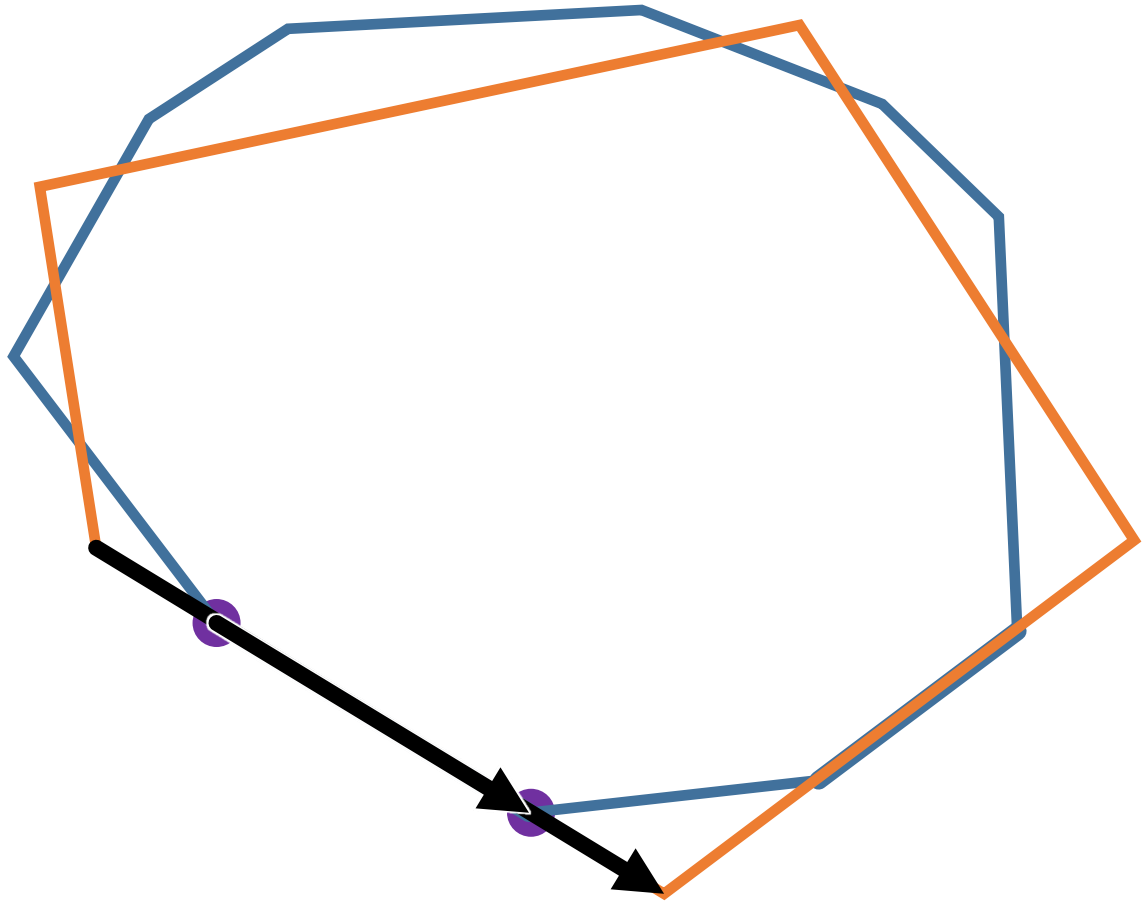
Intersection Example



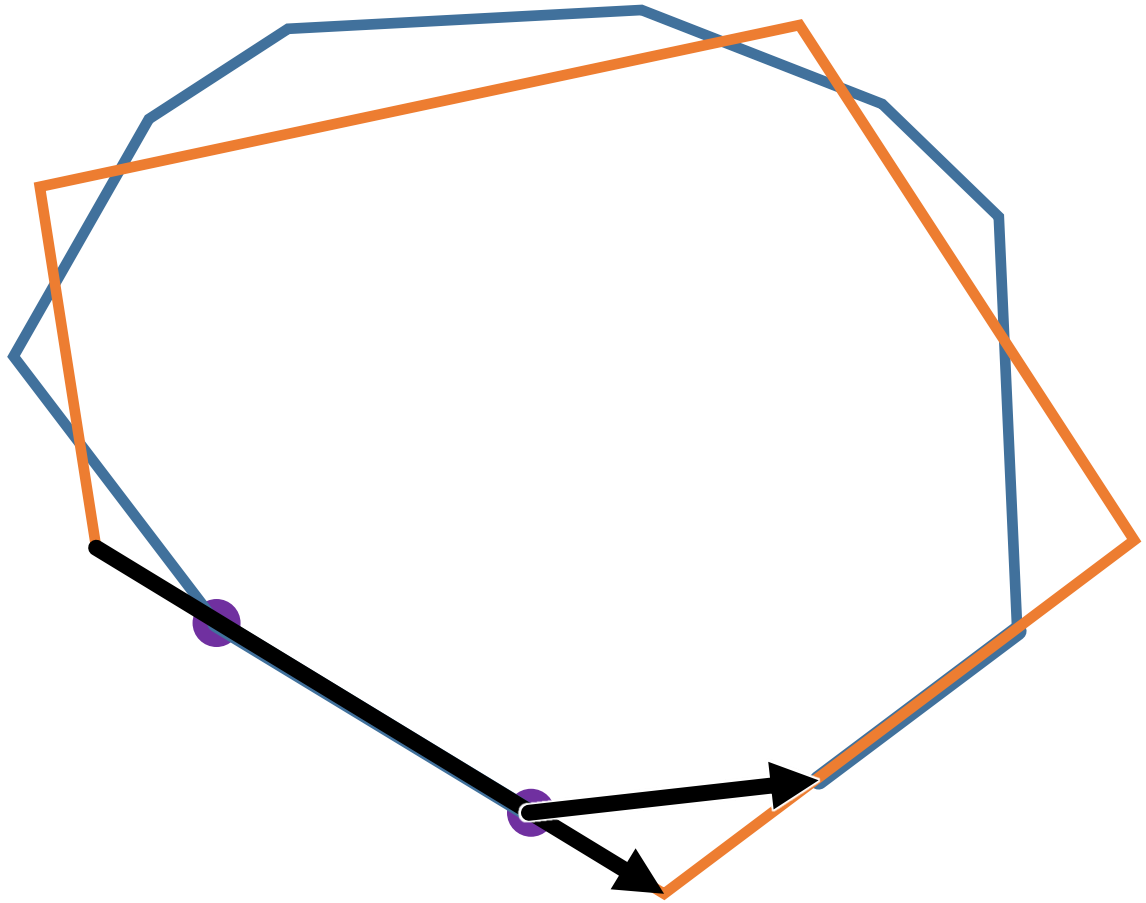
Intersection Example



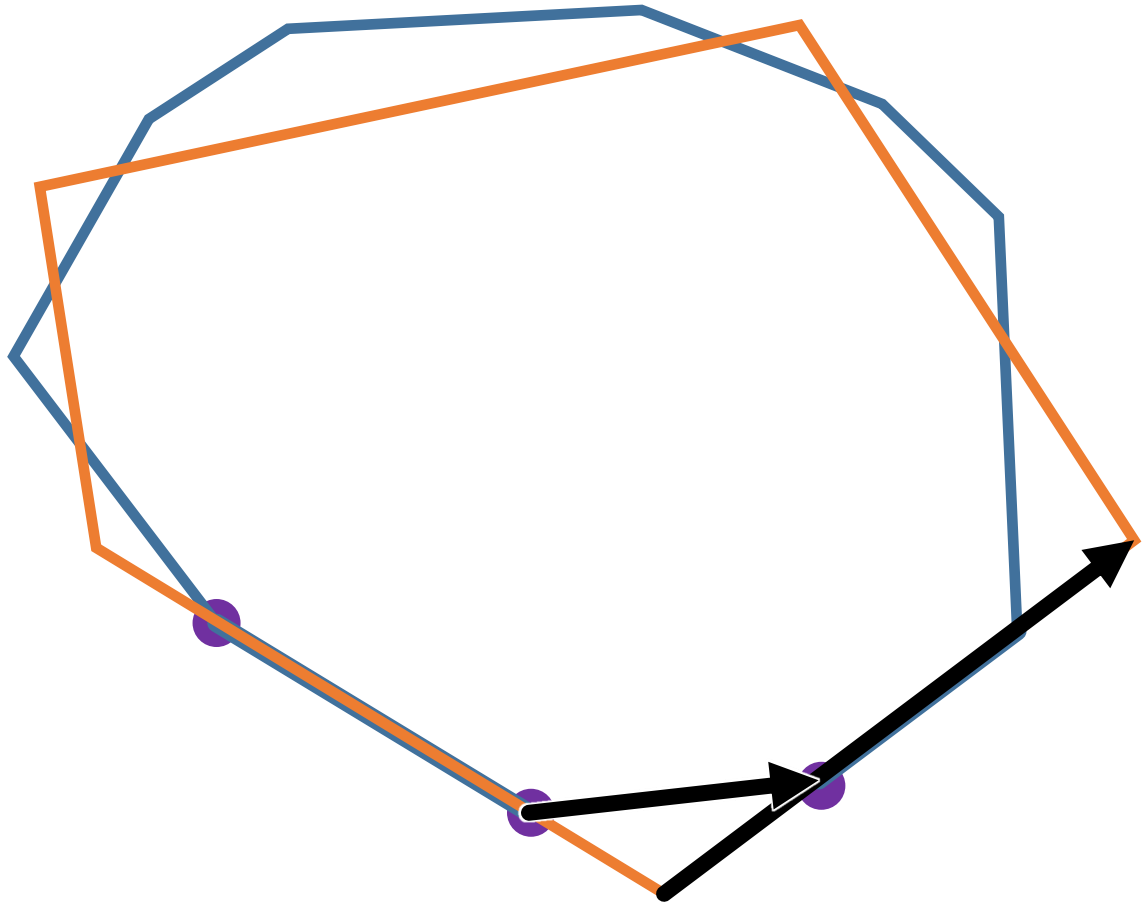
Intersection Example



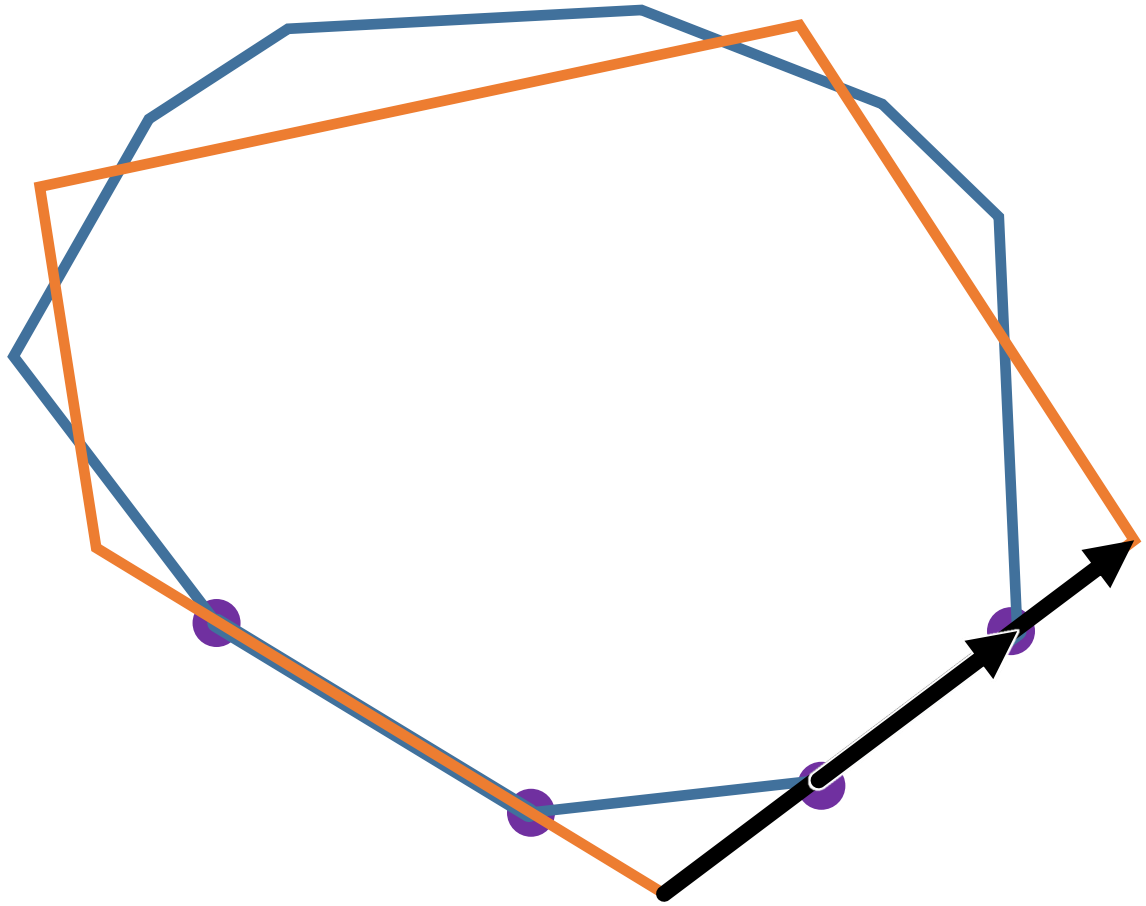
Intersection Example



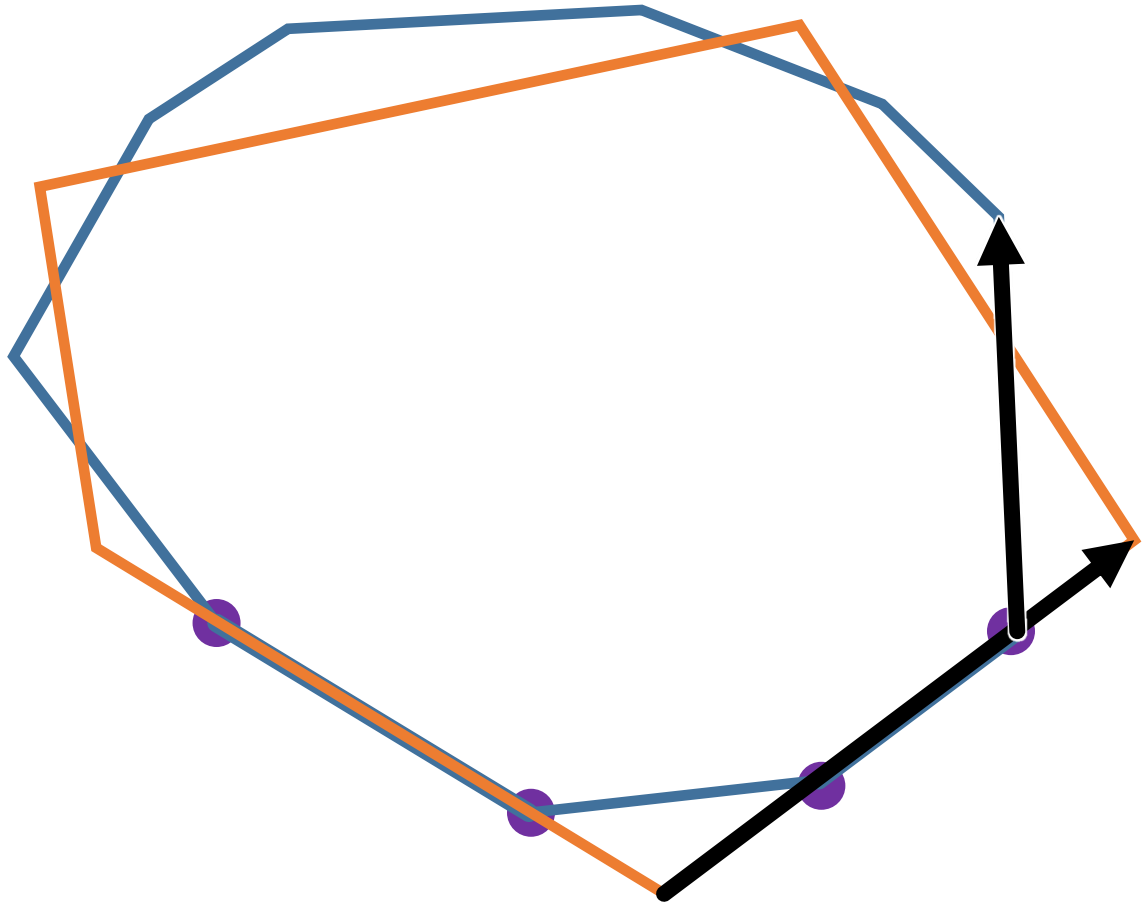
Intersection Example



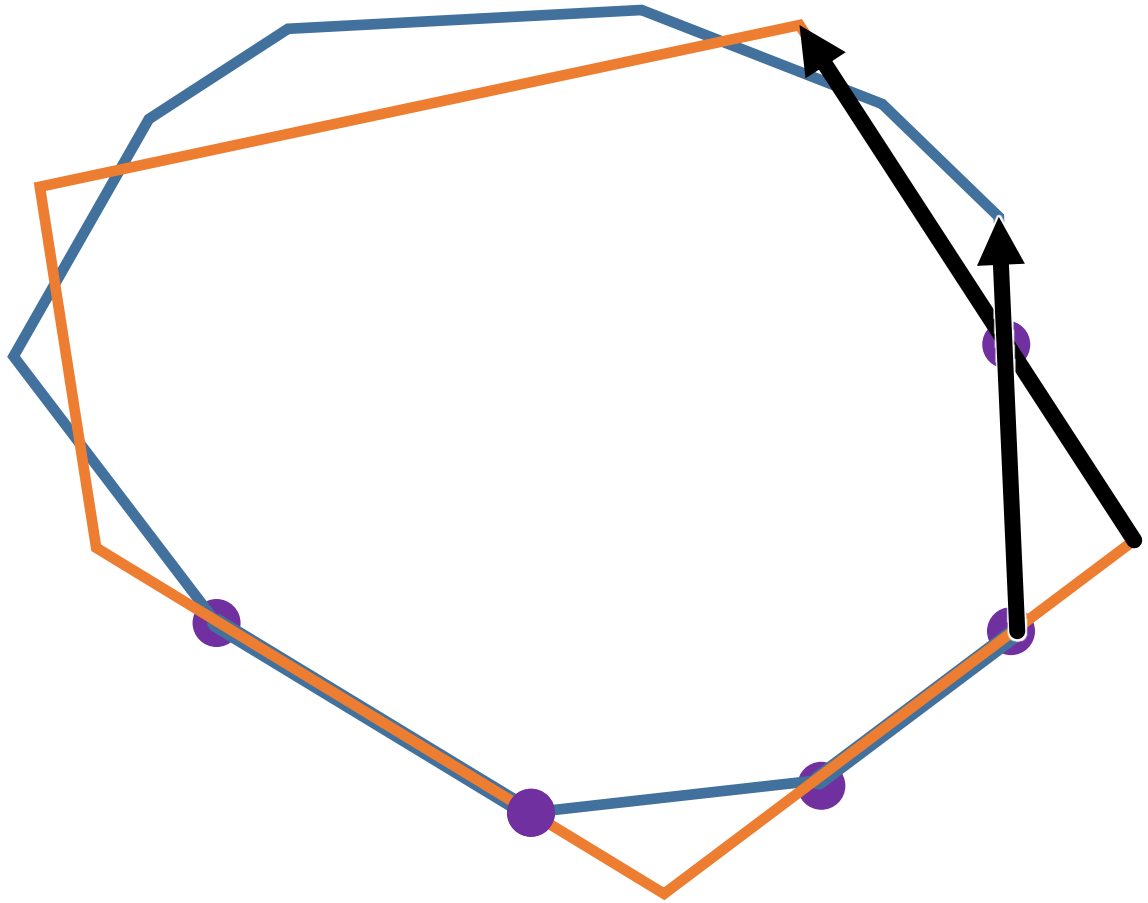
Intersection Example



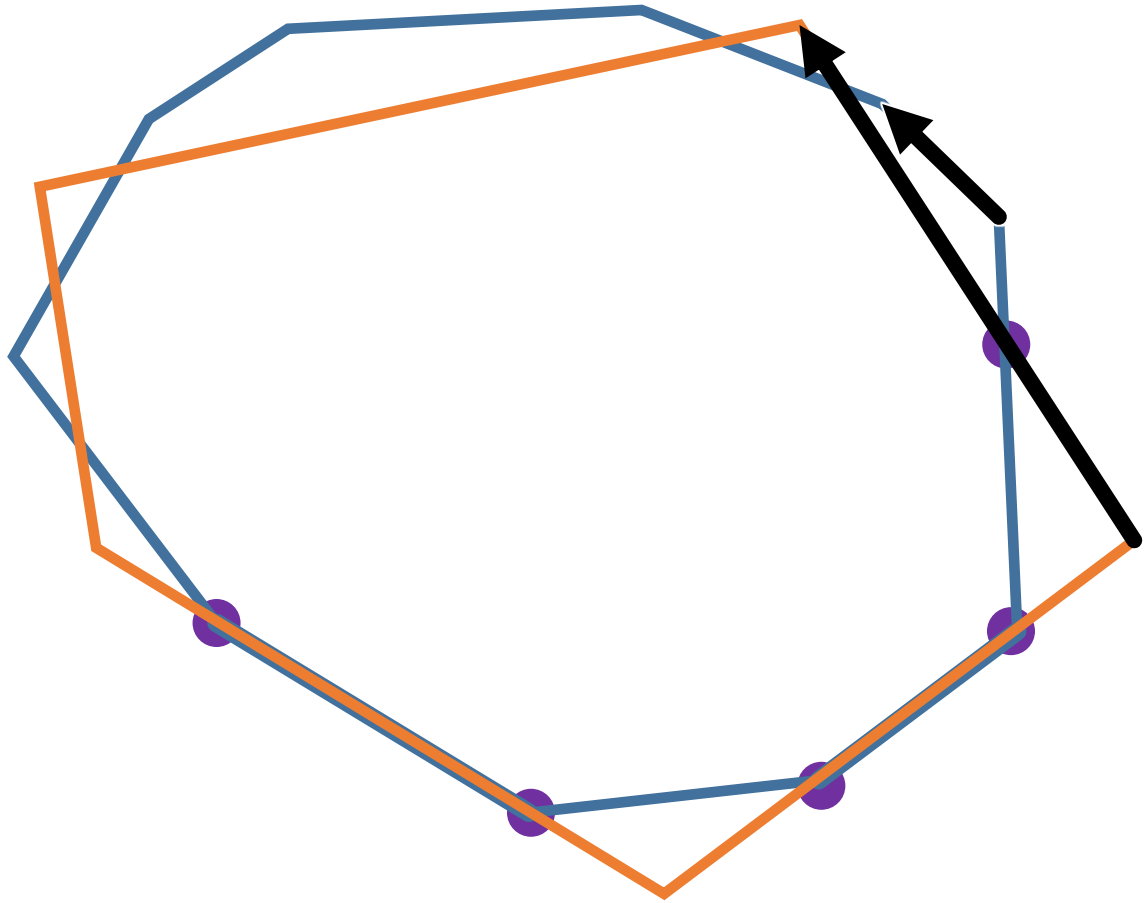
Intersection Example



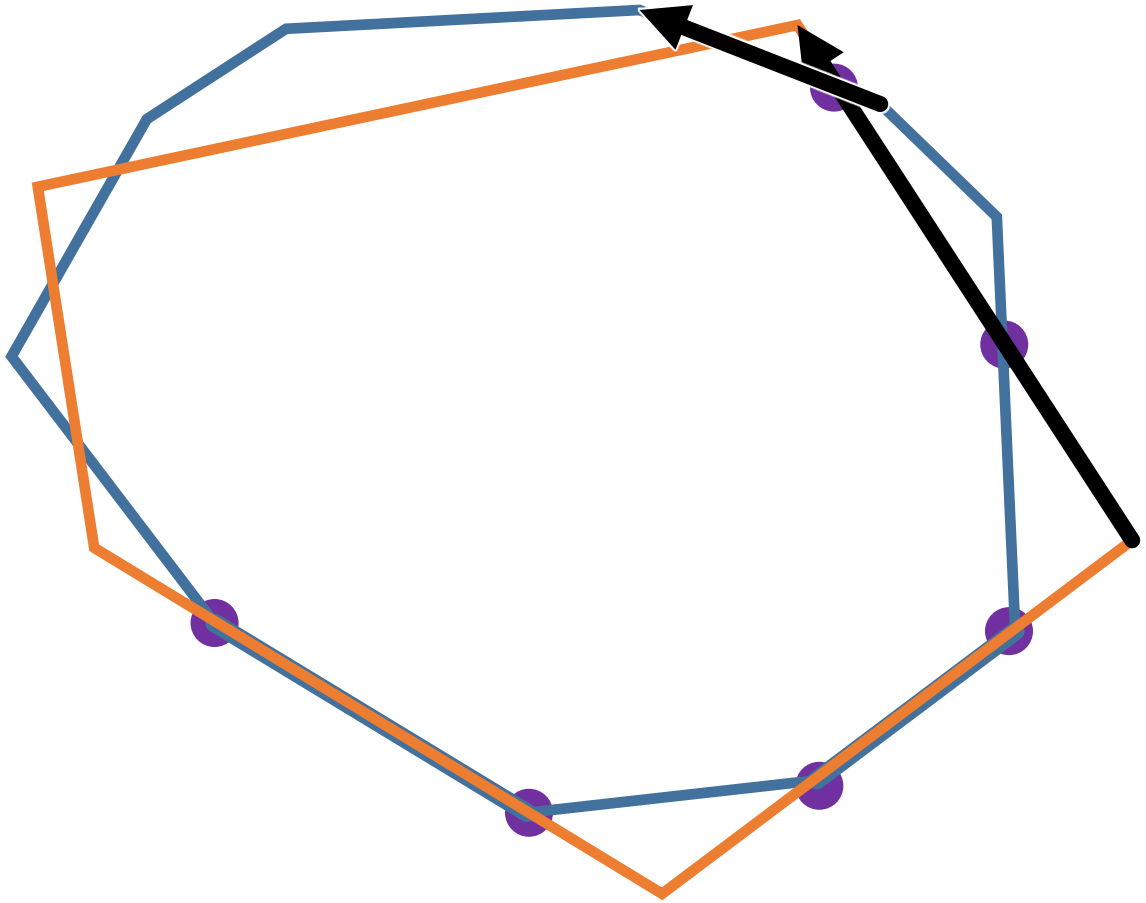
Intersection Example



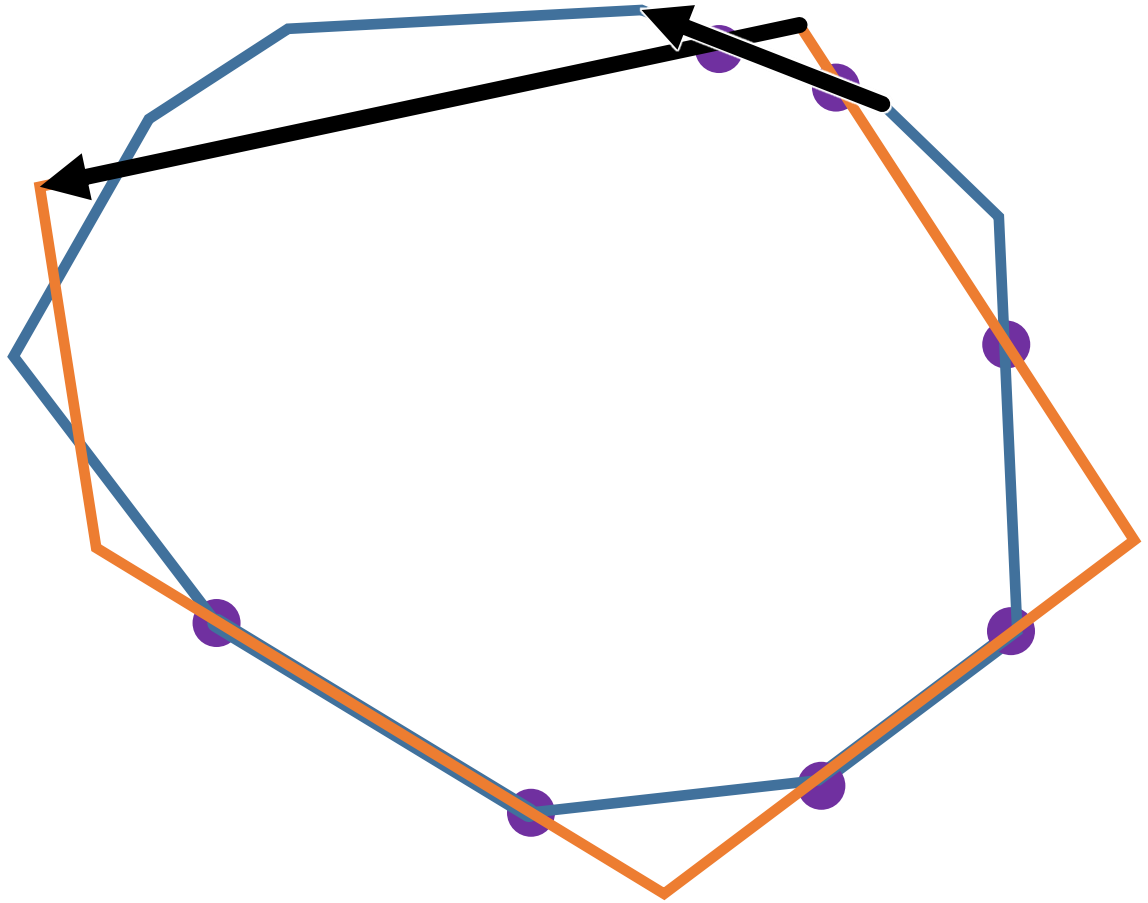
Intersection Example



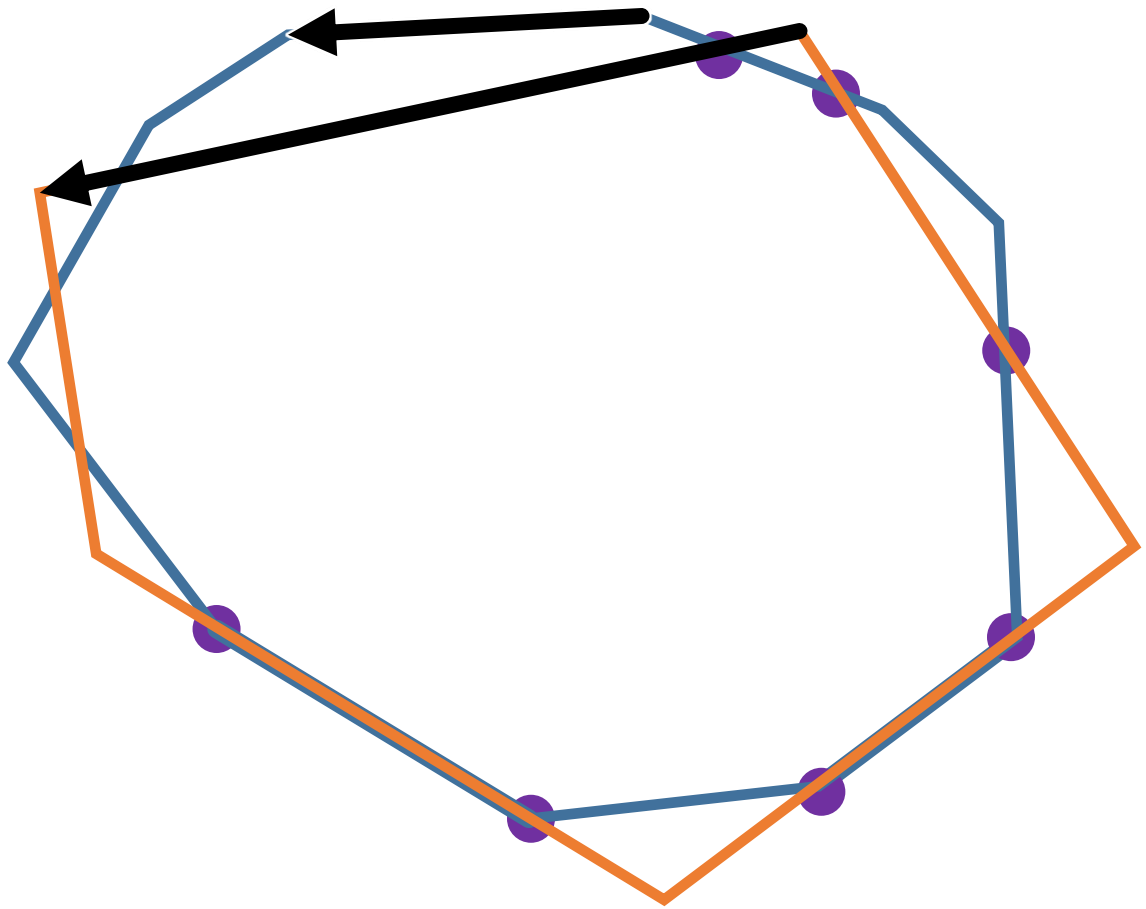
Intersection Example



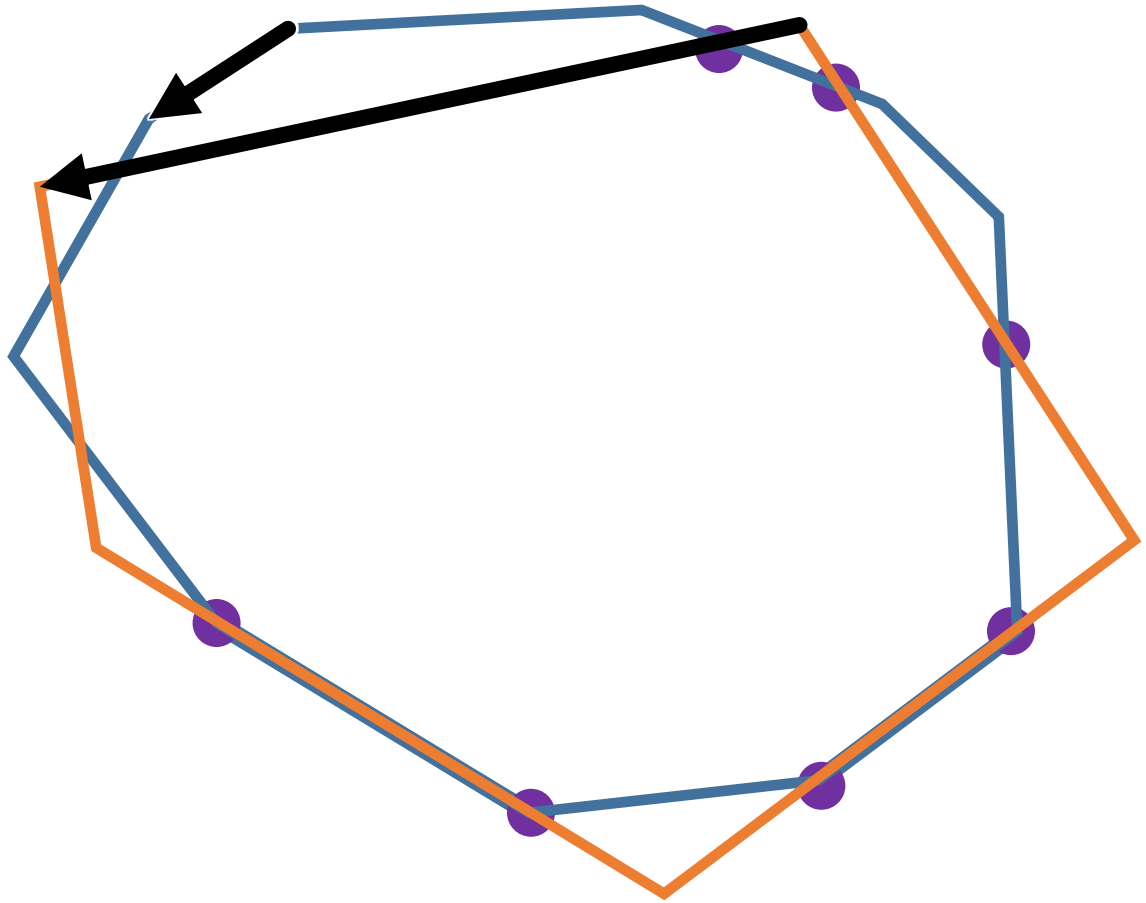
Intersection Example



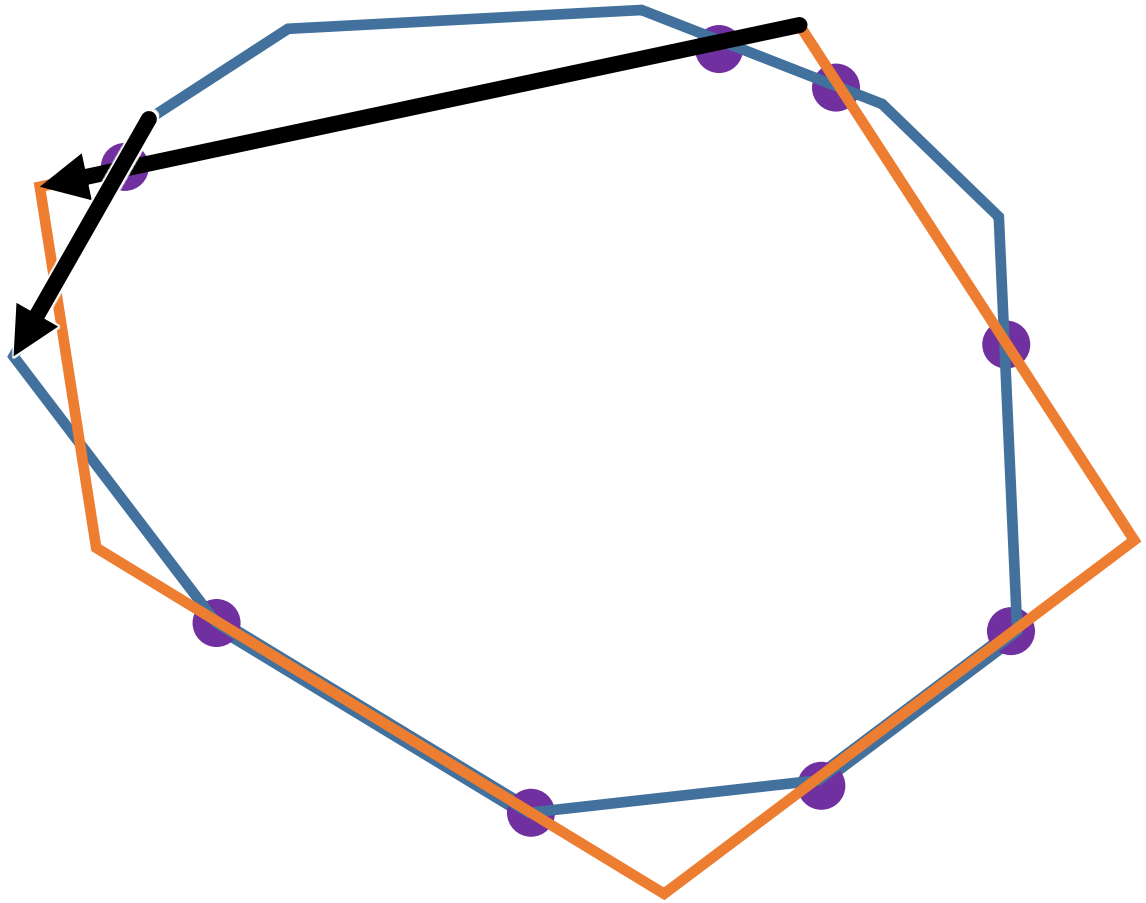
Intersection Example



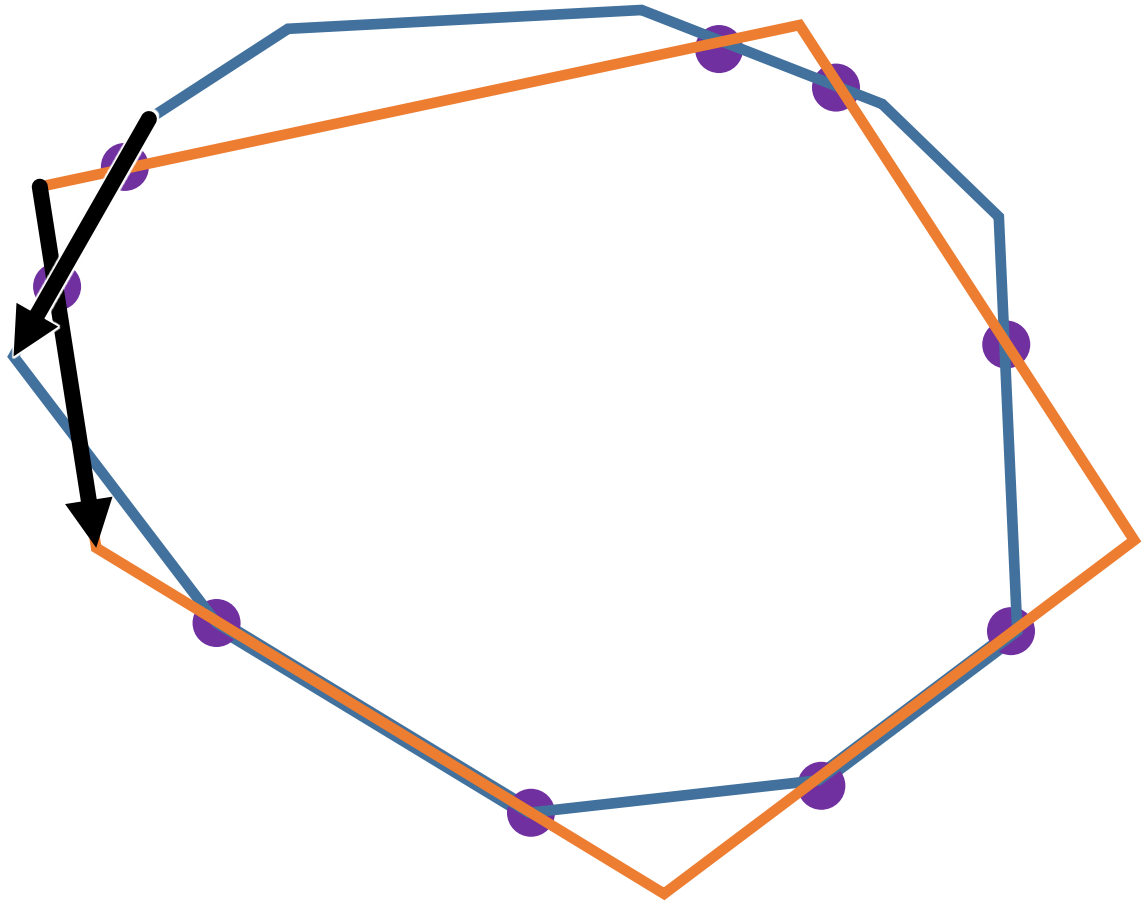
Intersection Example



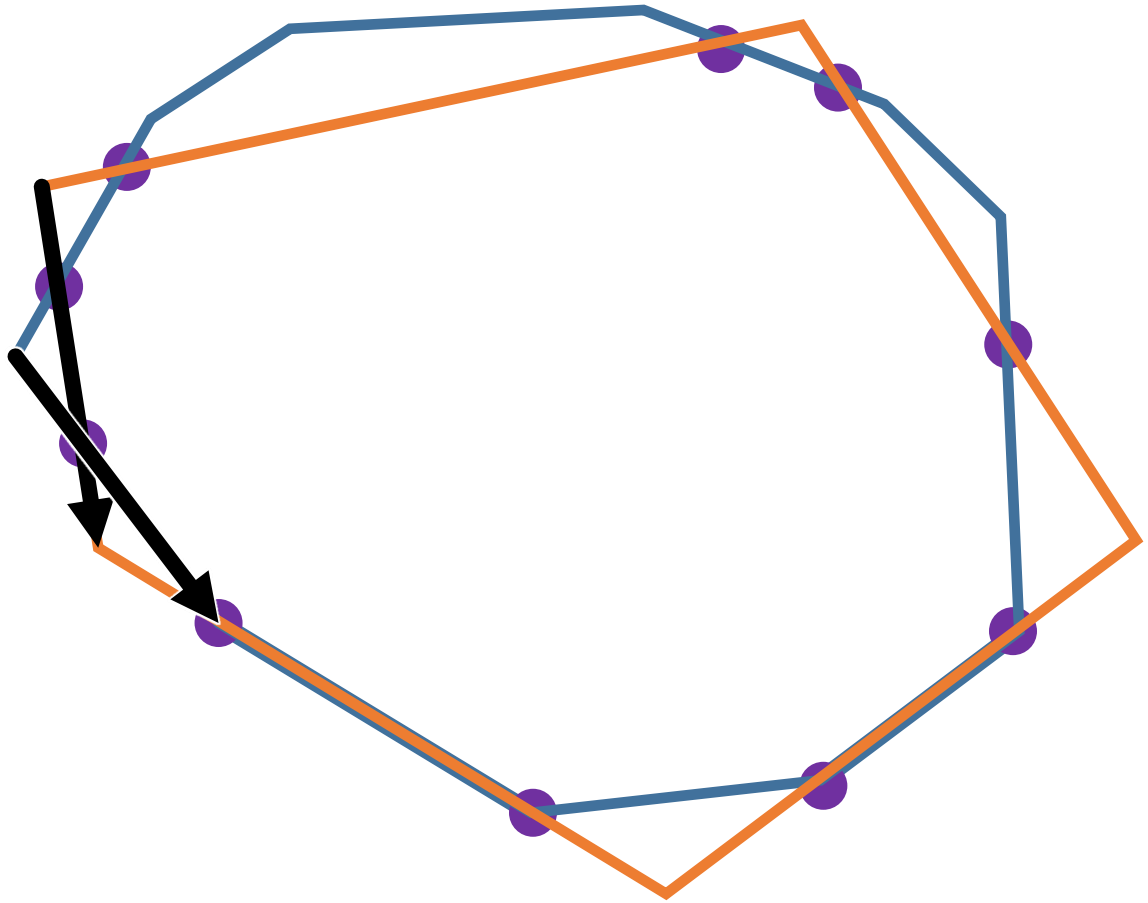
Intersection Example



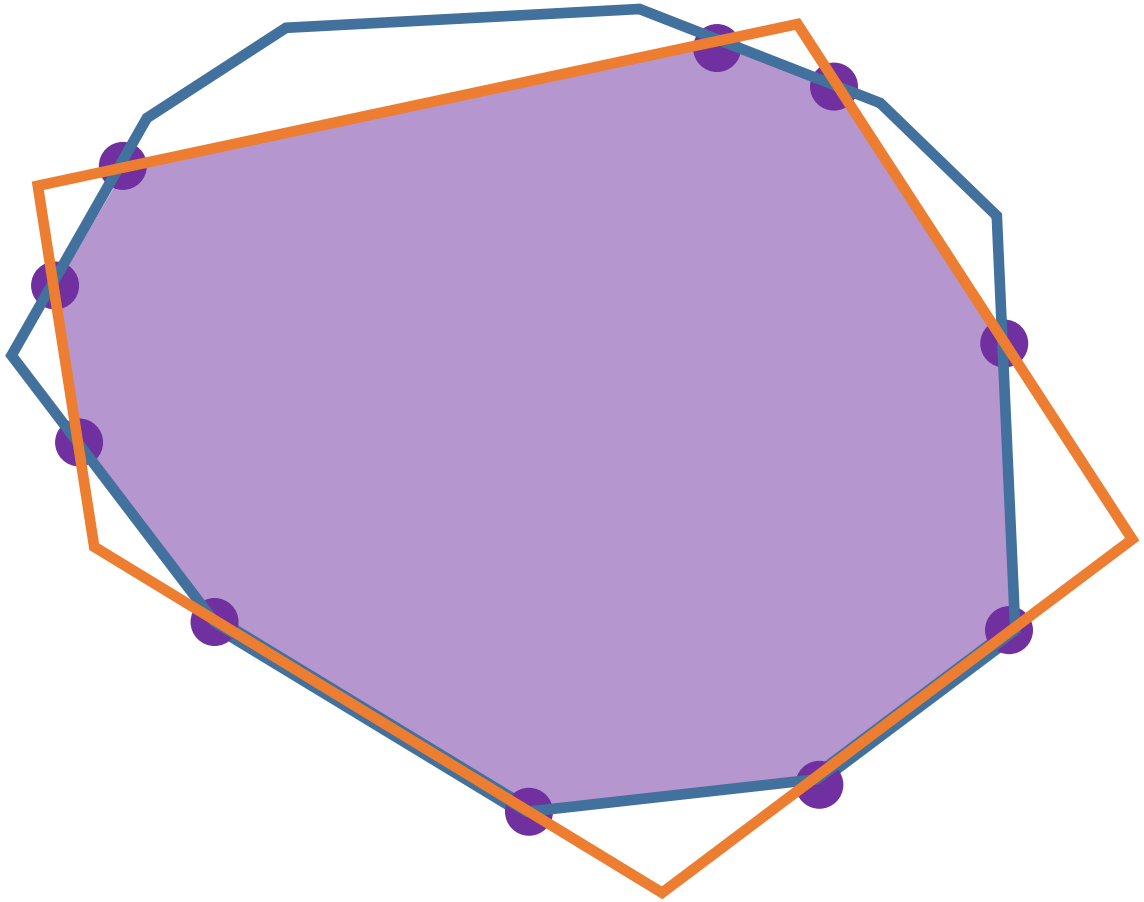
Intersection Example



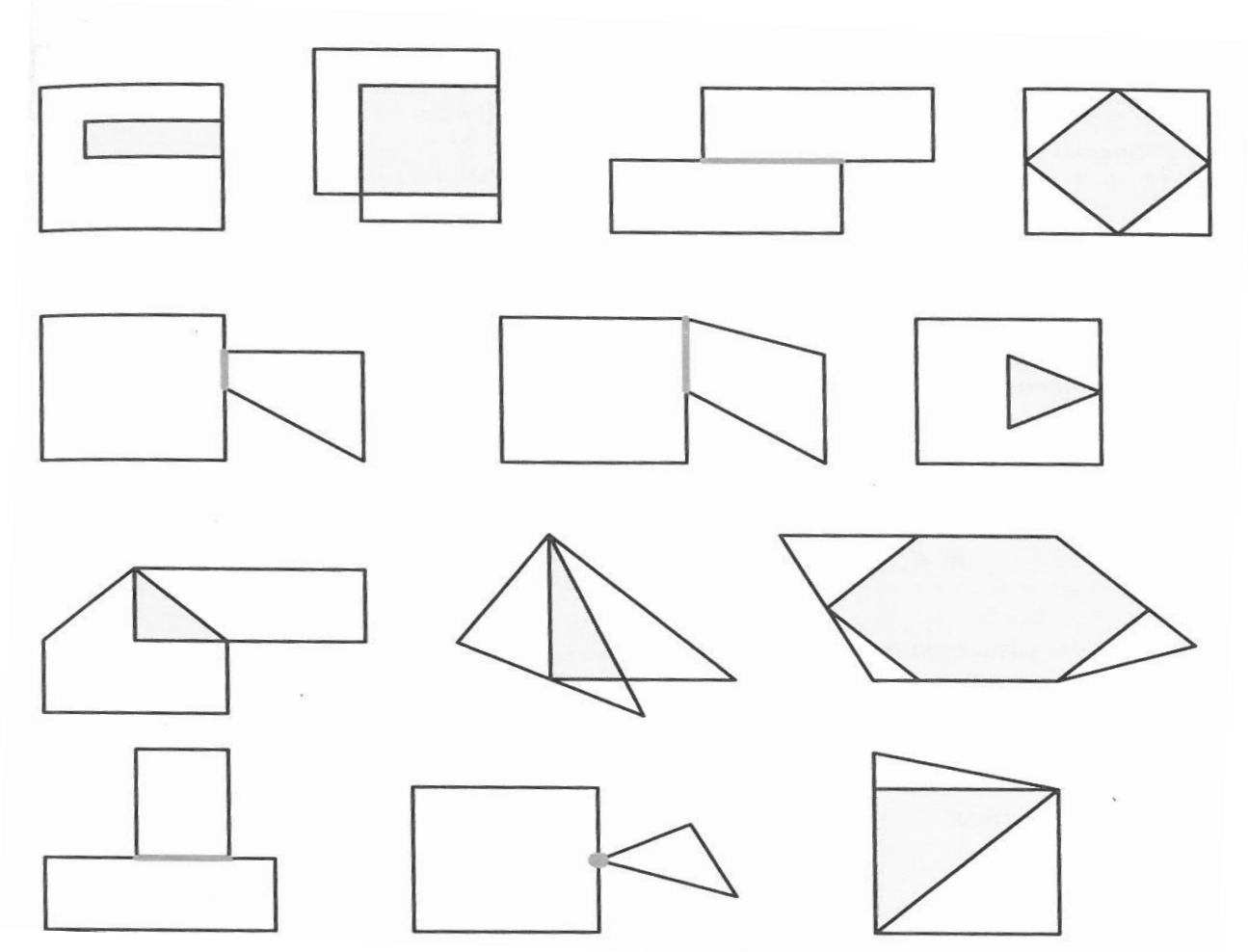
Intersection Example



Intersection Example



Example Degenerate intersections



Proof of correctness

- Correctness follows from two assertions:
  1. If current edges p and q belong to the same sickle, then the next intersection point, at which the sickle terminates, will be found next.
  2. If the boundaries of P and Q intersect, current edges p and q will cross at some intersection point after no more than 2(N + M) iterations (advances).

Analysis

- $\leq 2(N + M)$ iterations (advances) suffice to find the first intersection point and get P and Q into the same sickle.
- Thereafter, $\leq (N + M)$ additional iterations will produce R.

$\Longrightarrow$The algorithm requires $O(N + M)$, i.e. linear time.