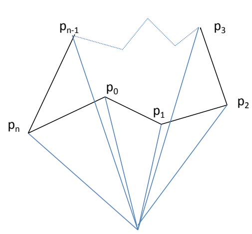
ALGORITHMS FOR SPATIAL DBMS

Basic Geometry (in 2D)

- cross-product u x v:
  - $(x_1, y_1) \times (x_2, y_2) := x_1 y_2 - y_1 x_2$
  - $u \times v = |u|\ |v| \sin \Theta$
    where Θ is angle between u and v
- can use to get angle between two vectors
  - how to test whether p is to the left/right of segment uv ?
  - how to test whether two line segments intersect?
- can use to calculate area of parallelogram/triangle

Basic Polygon Tasks

- calculate area of polygon

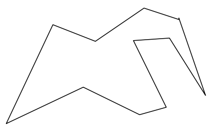


Formula:

$$1/2\sum_{i=1}^{n}(x_i y_{i+1} - x_{i+1} y_i)$$

Point in Polygon

- horizontal stabbing: O(n)

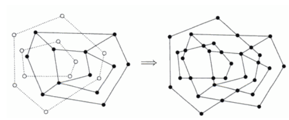


- if pre-processing is possible:
  point location query: O(log n)

Overlays, etc.

Points of interests in overlays:
intersection points

Also useful for
- intersection
- union
- difference



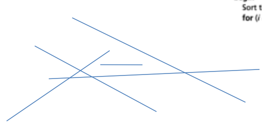
Line Segment Intersection

Given: $\{s_1, \ldots, s_n\}$ a set of n line segments
Output (*detect*): do any two of them intersect
or
Output (*compute*): list all intersections between the line segments

Line Segment Intersection

- trivial (detect/compute): $O(n^2)$
- plane sweep approach
  - natural: before two segments intersect, they are next to each other (assuming no three lines intersect in a point)
  - events: endpoints
  - active list: segments intersecting the current sweep line in order

detect Line Segment Intersection



```
ALGORITHM    SegmentIntersectionTest (S: set of segments): Boolean

begin
  Sort the 2n endpoints of segments in S and place them in ℰ
  for (i = 1 to 2n) do
      begin
        p = ℰ[i]
        if (p is left of a segment s) then
        begin
          Insert (s, ℒ)
          if (Above (s, ℒ) intersects s), return true
          if (Below (s, ℒ) intersects s), return true
        end if
        if (p is right of a segment s) then
        begin
          if (Above (s, ℒ) intersects Below (s, ℒ)) return true
          Delete (s, ℒ)
        end if
      end for
    end
```

- L needs to be dynamic binary search tree
- time: $O(n \log n)$, space: $O(n)$
- problem: does not report all intersections;
  - as it is, it can’t, why?

compute Line Segment Intersection

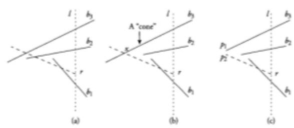
```
ALGORITHM    SegmentIntersection (S: set of segments): set of points

begin
  Sort the 2n endpoints of segments in S and place them in ℰ
  ℒ = ∅
  while (ℰ ≠ ∅) do
  begin
    p Min (ℰ) // extract from ℰ
    if (p is left of a segment s) then
    begin
      Insert (s, ℒ); s1 = Above (s, ℒ); s2 = Below (s, ℒ)
      if (s1 intersects s), AddInter(s ∩ s1, ℰ)
      if (s2 intersects s), AddInter(s ∩ s2, ℰ)
    end
    if (p is right of a segment s) then
    begin
      s1 = Above (s, ℒ); s2 = Below (s, ℒ); Delete (s, ℒ);
      if (s1 intersects s2 to the right of p), AddInter(s1 ∩ s2, ℰ)
    end
    if (p is the intersection of s1 and s2) then
    begin
      s3 = Above (Max(s1, s2), ℒ); s4 = Below (Min(s1, s2), ℒ)
      if (s3 intersects Min(s1, s2)), AddInter(s3 ∩ Min(s1, s2), ℰ)
      if (s4 intersects Max(s1, s2)), AddInter(s3 ∩ Max(s1, s2), ℰ)
      Swap s1 and s2 in ℒ
    end
      if (s3 intersects s2), AddInter(s3 ∩ s2, ℰ)
      if (s1 intersects s4), AddInter(s4 ∩ s1, ℰ)
      Swap s1 and s2 in ℒ
    end
end
```

- what do we need to implement L ?
- time/space analysis ?

Red/Blue Intersection

For our applications:

- can assume two intersection-free sets (e.g. two polygons), think of them as red/blue
- simplifies general problem
- can be solved in time O(n log n + k)
- algorithm is a bit tricky though; main problem:



Polyline Intersection

The algorithms can be adapted to work for intersections of polylines (i.e. allowing common endpoints).

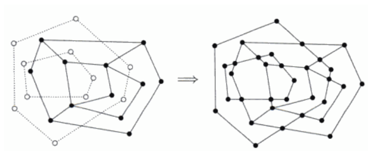
Polygon Intersection (detect)

Given: two polygons P, Q
Output: do P and Q intersect ?

Algorithm:
```
run line segment intersection test on edges of P (red) and Q (blue)
if intersect, then yes,
else
      p := point of P;
      if p in Q, then yes
      else
         q := point of Q
         if q in P, then yes
         else no.
```

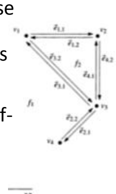
Polygon Operations

General approach:
- calculate common faces
- orient edges along boundary (face is to the left of an edge), leads to doubly connected edge lists, each edge has two sides
- run line segment intersection and update edge lists
- recalculate faces
- get union, difference, intersection, etc.



Doubly edge connected lists

Record of
- vertices
- half-edges (sides)
- faces (boundary traversal in clockwise order)
- faces can have holes (traversed counter-clockwise: face is always to left of half-edge)

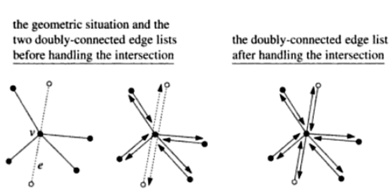


Vertex	Coordinates	IncidentEdge
v_1	(0, 4)	$\vec{e}_{1,1}$
v_2	(2, 4)	$\vec{e}_{4,2}$
v_3	(2, 2)	$\vec{e}_{2,1}$
v_4	(1, 1)	$\vec{e}_{2,2}$

Face	OuterComponent	InnerComponents
f_1	**nil**	$\vec{e}_{1,1}$
f_2	$\vec{e}_{4,1}$	**nil**

Half-edge	Origin	Twin	IncidentFace	Next	Prev
$\vec{e}_{1,1}$	v_1	$\vec{e}_{1,2}$	f_1	$\vec{e}_{4,2}$	$\vec{e}_{3,1}$
$\vec{e}_{1,2}$	v_2	$\vec{e}_{1,1}$	f_2	$\vec{e}_{3,2}$	$\vec{e}_{4,1}$
$\vec{e}_{2,1}$	v_3	$\vec{e}_{2,2}$	f_1	$\vec{e}_{2,2}$	$\vec{e}_{4,2}$
$\vec{e}_{2,2}$	v_4	$\vec{e}_{2,1}$	f_1	$\vec{e}_{3,1}$	$\vec{e}_{2,1}$
$\vec{e}_{3,1}$	v_3	$\vec{e}_{3,2}$	f_1	$\vec{e}_{1,1}$	$\vec{e}_{2,2}$
$\vec{e}_{3,2}$	v_1	$\vec{e}_{3,1}$	f_2	$\vec{e}_{4,1}$	$\vec{e}_{1,2}$
$\vec{e}_{4,1}$	v_3	$\vec{e}_{4,2}$	f_2	$\vec{e}_{1,2}$	$\vec{e}_{3,2}$
$\vec{e}_{4,2}$	v_2	$\vec{e}_{4,1}$	f_1	$\vec{e}_{2,1}$	$\vec{e}_{1,1}$

Updating edge lists

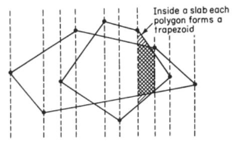


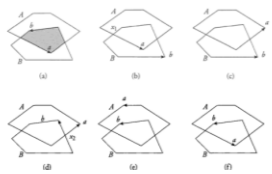
Convex Polygon Intersection

Given: convex polygons P, Q
Output: intersection

Naïve algorithm: |P| |Q|

Can be done in time O(|P|+|Q|):
- Shamos-Hoey
  split into slabs, do line swe
- O'Rourke
  follow boundary





Clipping