An algorithm for point-in-polygon

Leila De Floriani

Point-in-polygon

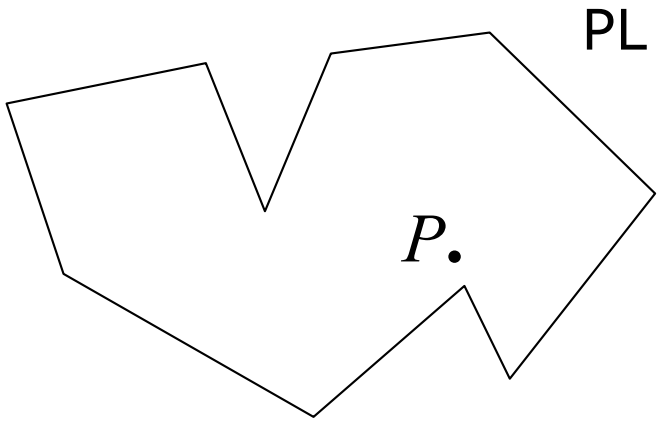
- **Point-in-polygon (region):** basic geometric operation on polygonal regions.

- **Problem:**

  Let PL be a simple polygon and P a point in the plane.

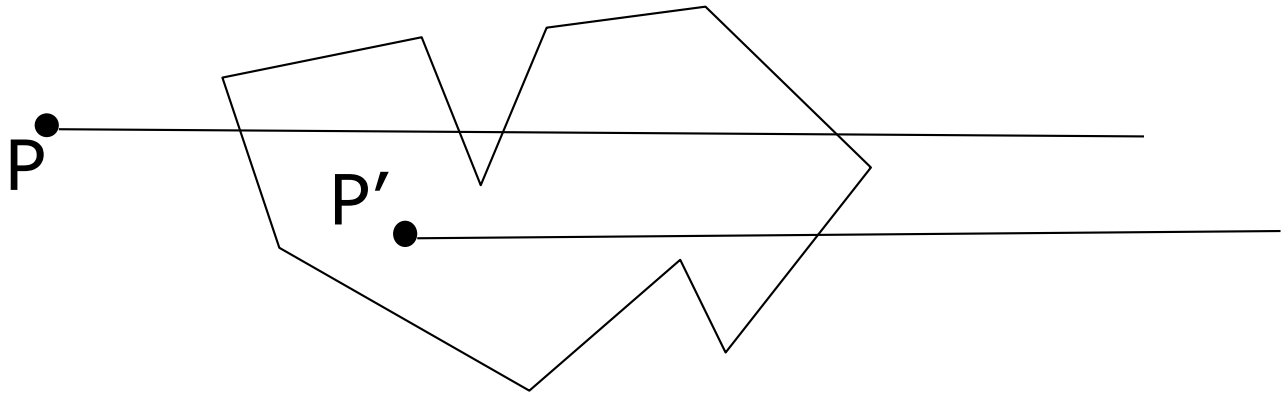
  Find the position of P with respect to PL.

- **Possible outcomes:**
  - P is in the interior of PL
  - P is in the exterior of PL
  - P is on PL



Point-in-polygon

- The general solution is based on **Jordan theorem.**
- Consider a half-line L with origin in P and consider the crossings of L with polygon PL.
- Each time line L crosses polygon PL, L will either
  - enter the polygonal region bounded by PL (interior of PL), or
  - exit from the interior region and enter the exterior region of PL



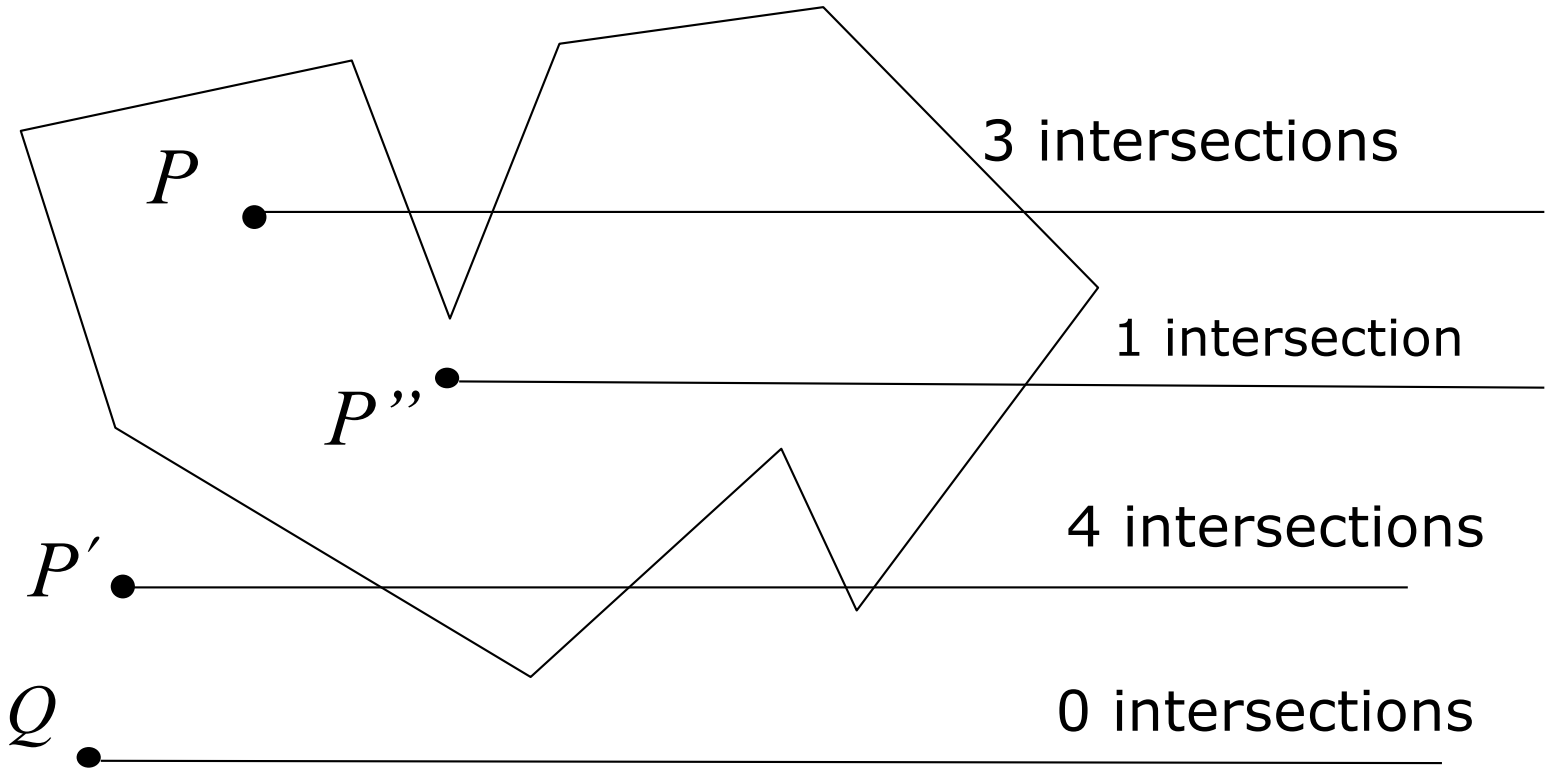
Point-in-polygon

- Let us consider (for simplicity of computation) a horizontal half-line L with origin in the query point

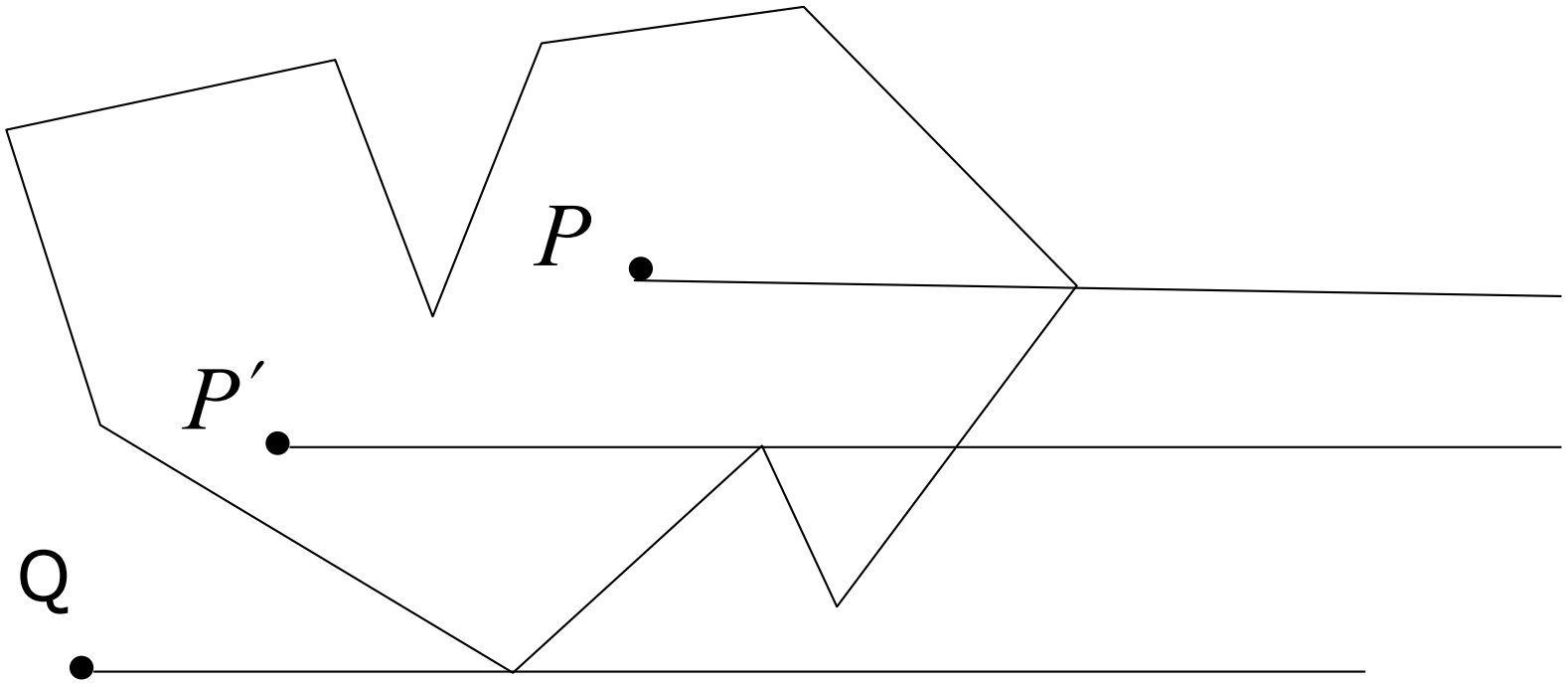
- **General idea of the algorithm:** *count the number of intersections **k** between line L and polygon PL*
  - If one of the intersections between PL and L is the same as P, then P is on polygon PL

  else
  - if k is **even**, then P is **outside** PL
  - if k is **odd**, P is **inside** PL

Point-in-polygon: examples



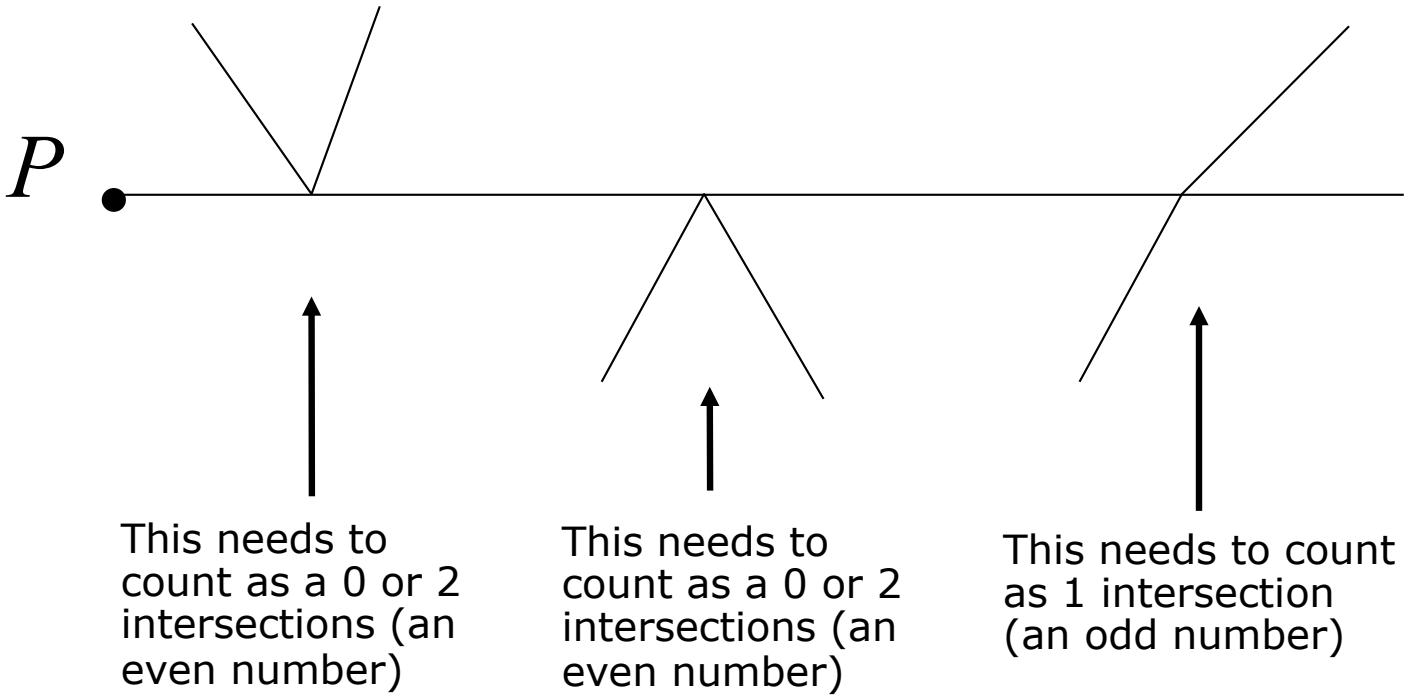
Point-in-polygon: examples



Point-in-polygon: special cases

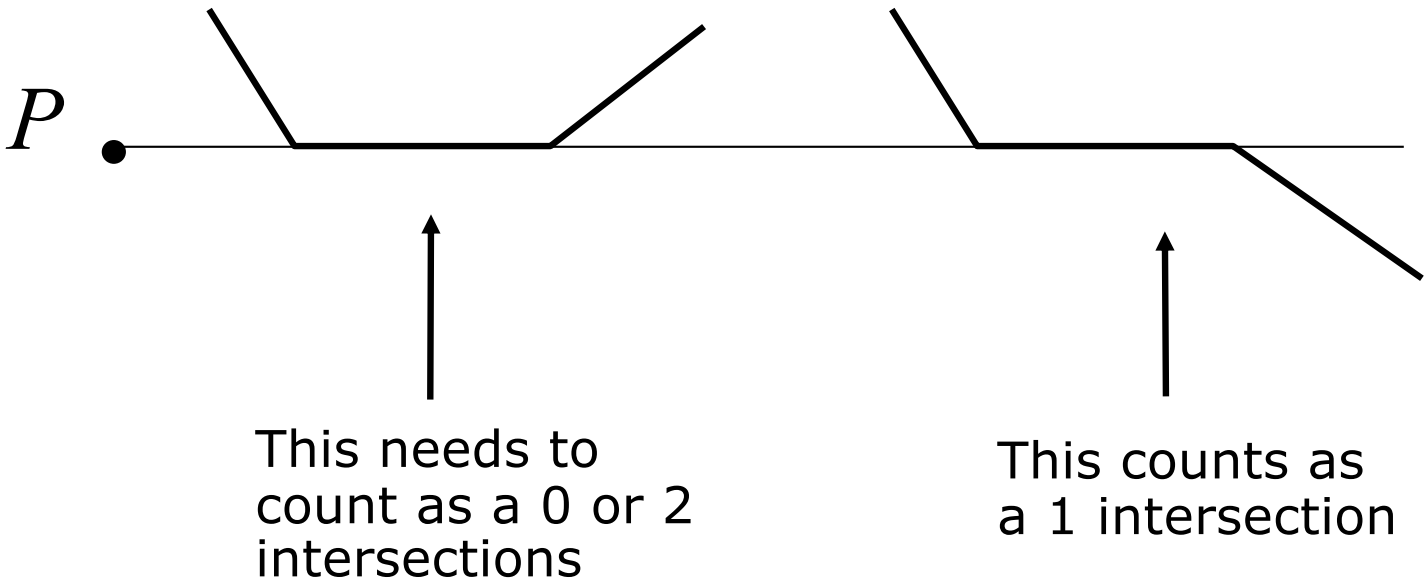
Intersection at a vertex

- Notice that we will find the intersection at a vertex always twice (one for each of the two edges sharing the vertex)



Point-in-polygon: special cases

Intersection along an edge

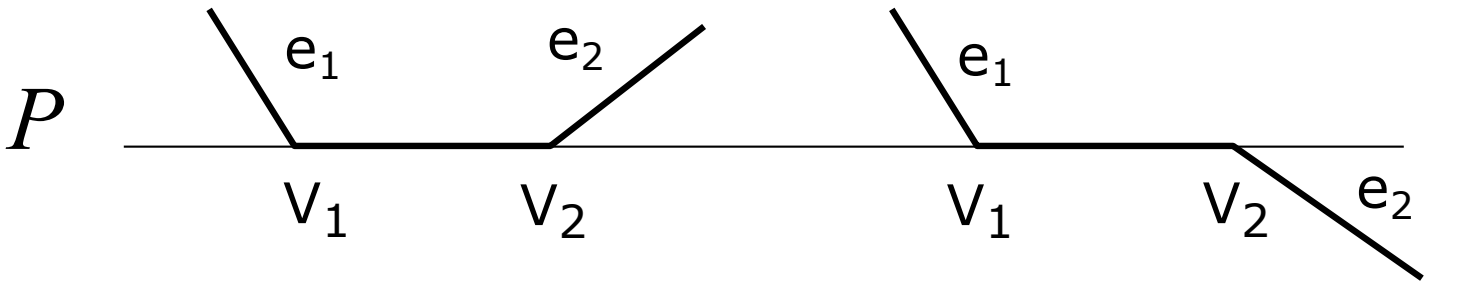


Point-in-polygon: rules for special cases

1. If an intersection corresponds to a polygon vertex V, it counts as
   - 1 if V is the lowest vertex (with the lowest y-coordinate) of the edge
   - 0 if V is the highest vertex (with the highest y-coordinate) of the edge

Point-in-polygon: rules for special cases

2. The intersection with a horizontal edge counts as 0 (we can avoid computing it) except when p is on the edge (see detailed description)



V_1 counts as 1 when we consider edge e_1 and V_2 as 1 when we consider edge e_2

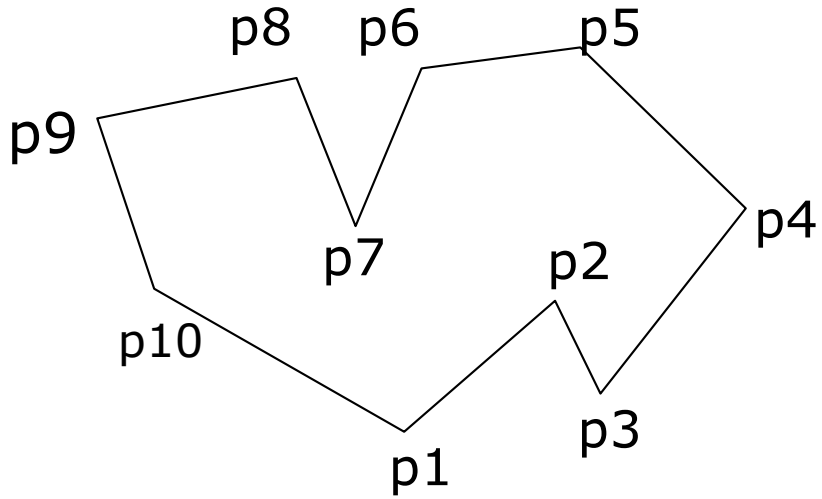
V_1 counts as 1 when we consider edge e_1 and V_2 as 0 when we consider edge e_2

Point-in-polygon algorithm: detailed description

Point-in-polygon

- The algorithm takes as **input**
  - a polygon PL
  - a point P

- **Output:**
  - P is in the interior of PL
  - P is in the exterior of PL
  - P is on PL



- A point is a pair $P=(x_P,y_P)$, where x_P and y_P are the x- and y-coordinates, respectively, of P.

- Polygon PL is encoded as the list of its *vertices* (points) $[p_1,p_2,\ldots,p_n]$ in counterclockwise order.

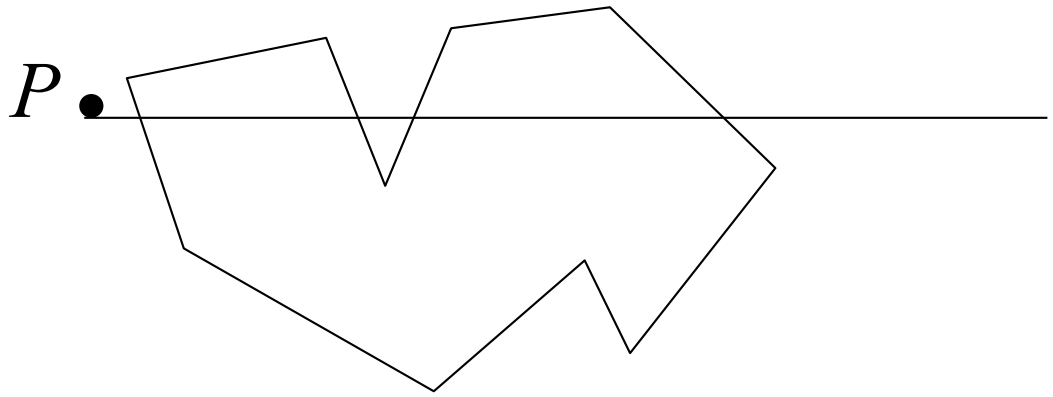
- Each pair of consecutive vertices (p_i,p_{i+1}) defines an edge of PL, and Pair (p_n,p_1) is also an edge.

Point-in-polygon: algorithm

1. Loop on all the edges of PL and intersect each edge with the horizontal half-line L with origin in p:
   Intersect_Edge *algorithm* (see next slides)

2. If the number **k** of intersections between L and PL is even, then then **p** is **OUTSIDE PL.**

3. if **k** is odd, p is either **INSIDE** or **ON PL**:
   - If p is the intersection between an edge of PL and L, then p is **ON PL.**

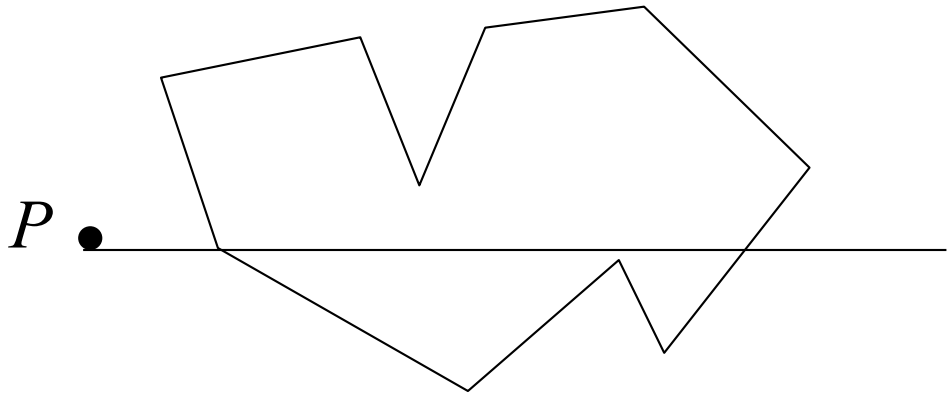


Intersect_Edge algorithm

- **Input:**
  - An edge *e* of the input polygon, represented as a pair of points: p1 =(x1,y1),p2 =(x2,y2)
  - Query point p =(xp,yp)

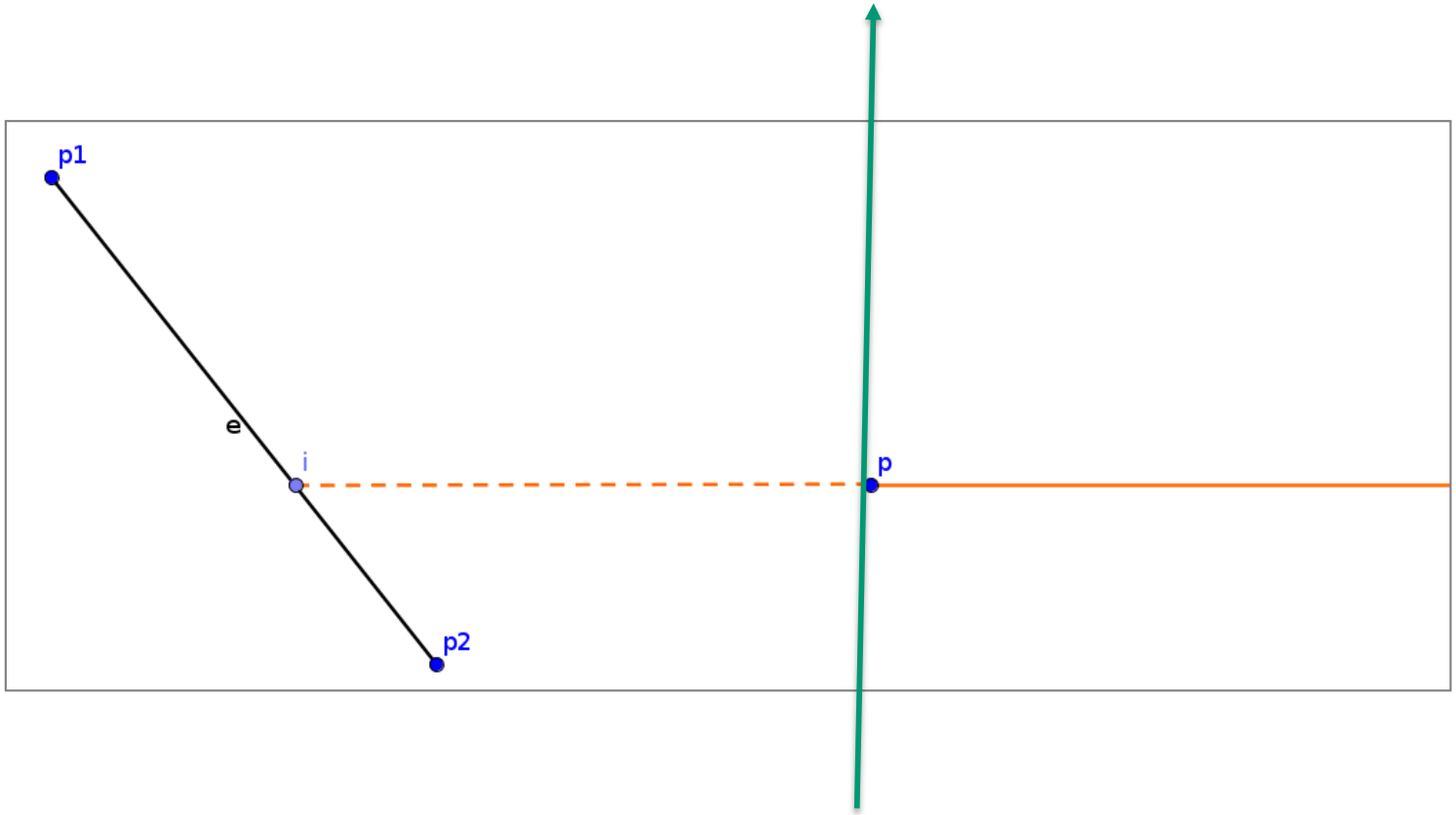
- The algorithm computes the intersection between edge e and the horizontal half-line L originating at *p*

- **Returns:**
  - 0: no intersection
  - 1: intersection
  - 2: p is on edge e



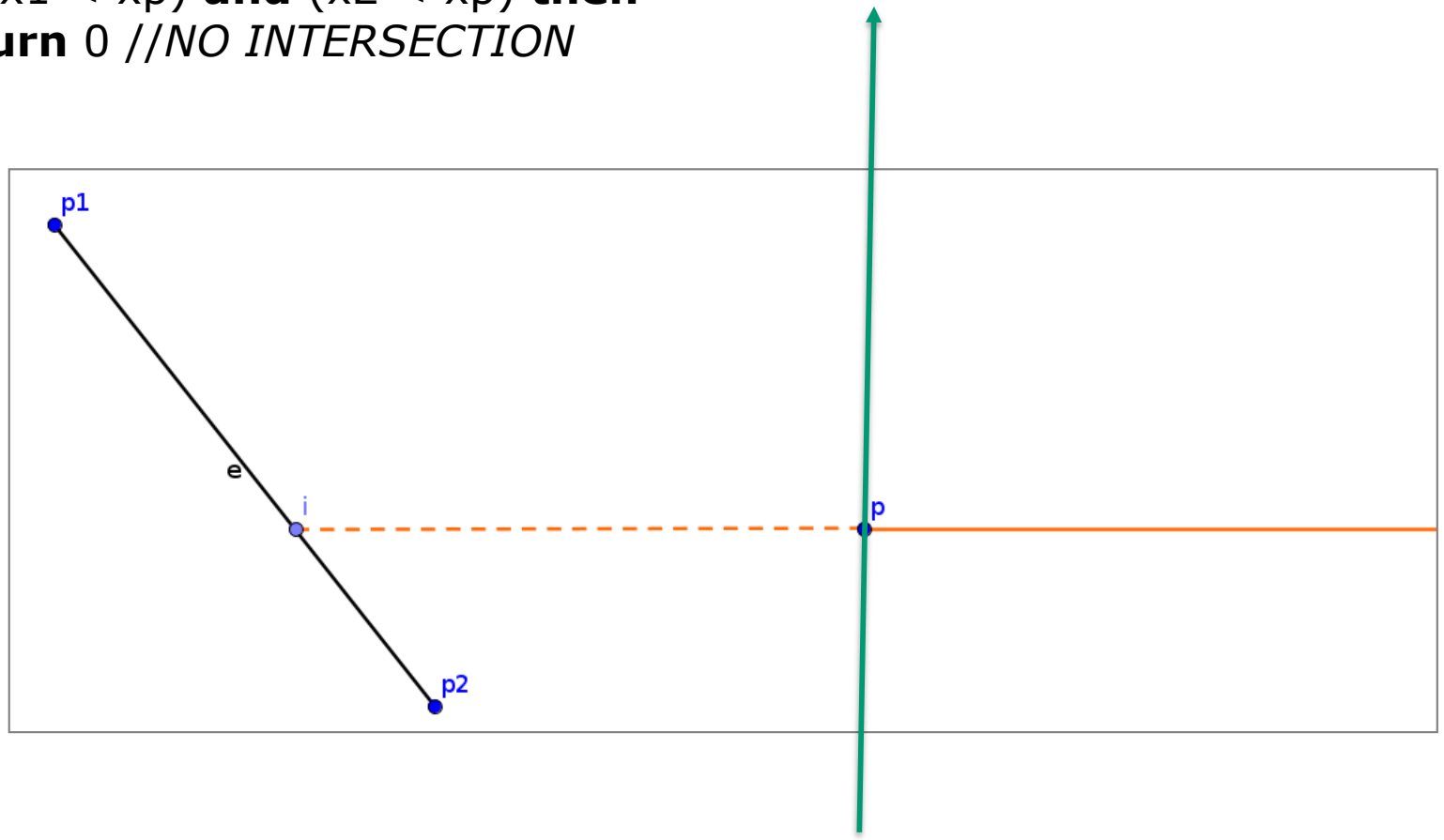
Intersect_Edge: case 1

Case 1: e is on the **left** of the vertical oriented line through *p* (the vertical line is oriented upwards)



Intersect_Edge: case 1

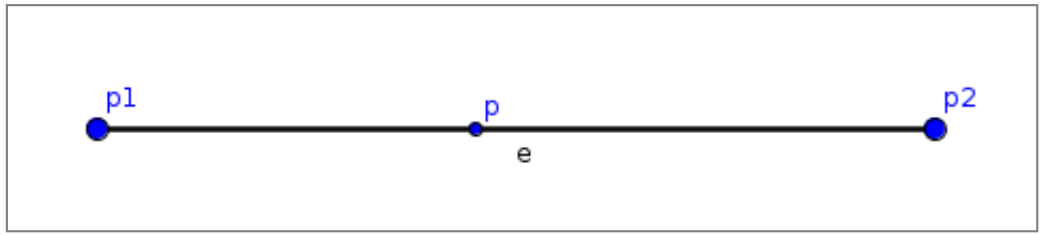
// **Case 1:** *edge e is on the left of the oriented vertical line passing through p*
if (x1 < xp) **and** (x2 < xp) **then**
return 0 //*NO INTERSECTION*



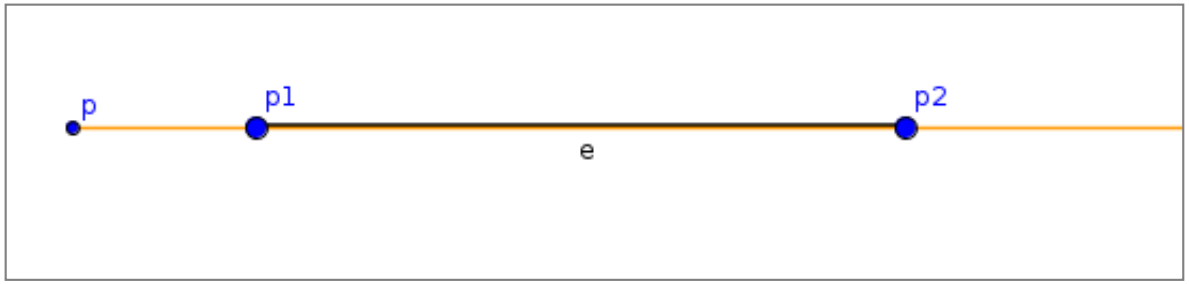
case 1 (e is on the left of the oriented vertical line through p)

Intersect_Edge: case 2

- **Case 2:** e is **horizontal**



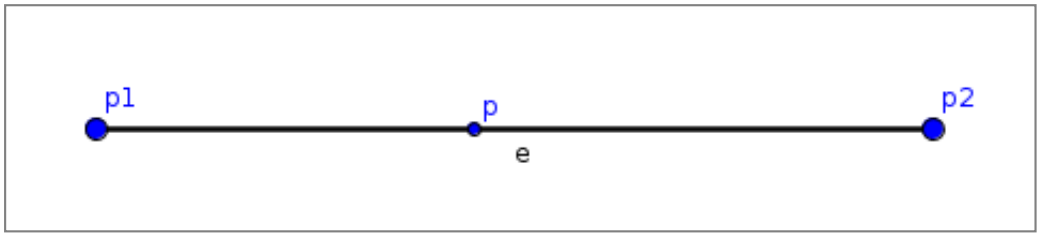
case 2 (p is on e)



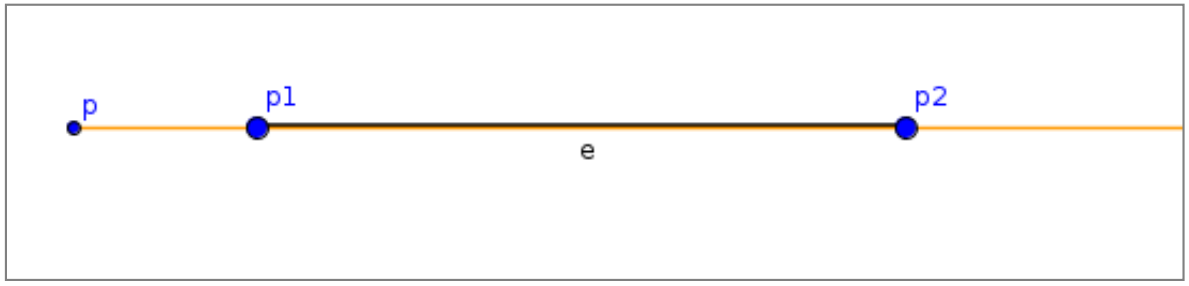
case 2 (horizontal edge is irrelevant)

Intersect_Edge: case 2

```
// Case 2: horizontal  edge
if (y1 = y2) // edge e is horizontal
  xmin ← min(x1, x2)
  xmax ← max(x1, x2)
      // xp is in the [xmin,xmax] range and  yp is equal to y1
   if (xmin <= xp <= xmax) and (yp = y1) then
        return 2 // p is ON  edge e
   else
return 0 // NO INTERSECTION, the horizontal edge is irrelevant
```



case 2 (p is on e)

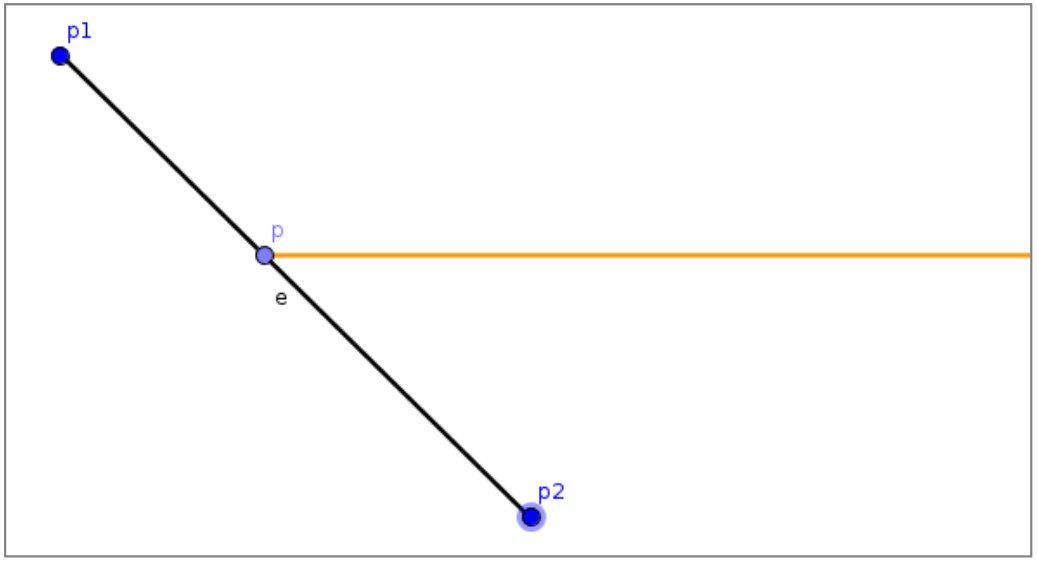


case 2 (horizontal segment is irrelevant)

Intersect_Edge: case 3

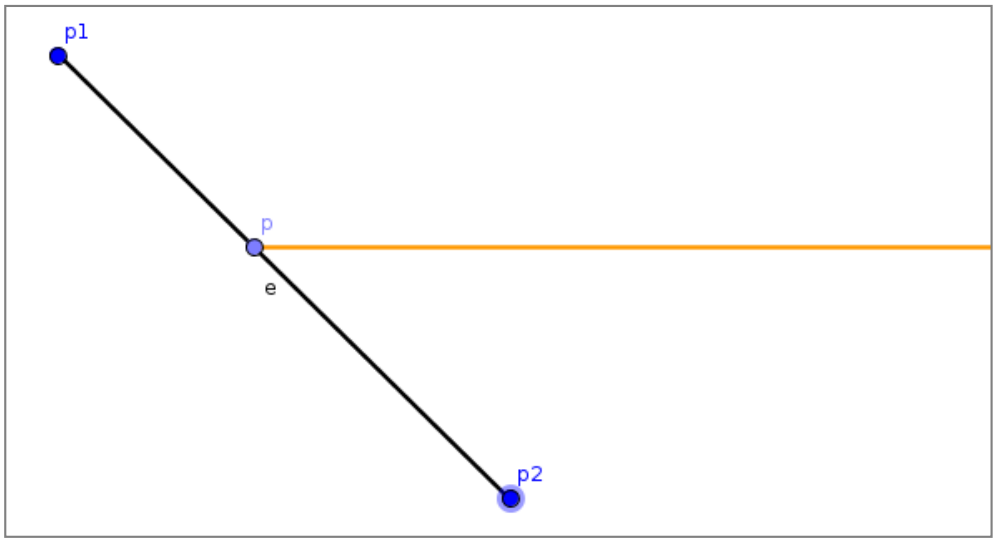
- *Compute the intersection point between the line passing through edge e and the half-line originating from p: (x,yp)*

- It is a valid intersection if:
  3. **it is the same as p**
  4. it is one of the two endpoints of e
  5. it is a point inside edge e



Intersect_Edge: case 3

```
// compute the intersection point between line l and edge e
// the intersection point has coordinates  (x, yp)
x ← ((yp - y1) / (y2 - y1)) * (x2 - x1)  + x1
ymin ← min(y1, y2)
ymax ← max(y1, y2)
 // case 3: p is on edge e
if (x = xp) and (ymin <= yp <= ymax)
      then return 2 // p is ON edge e
```

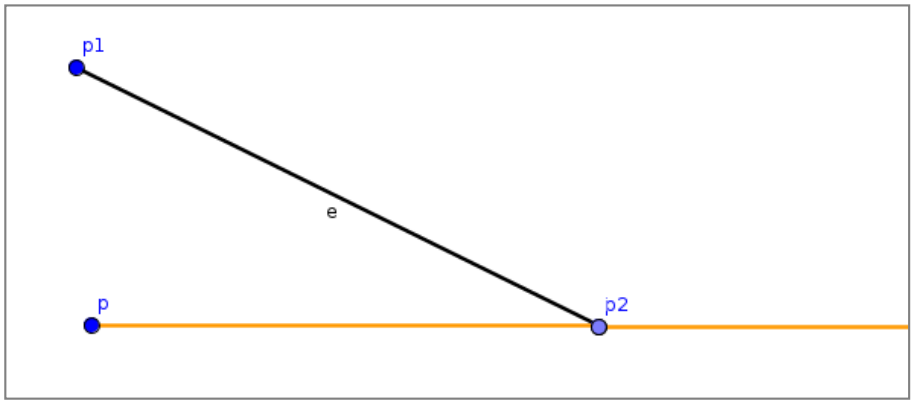


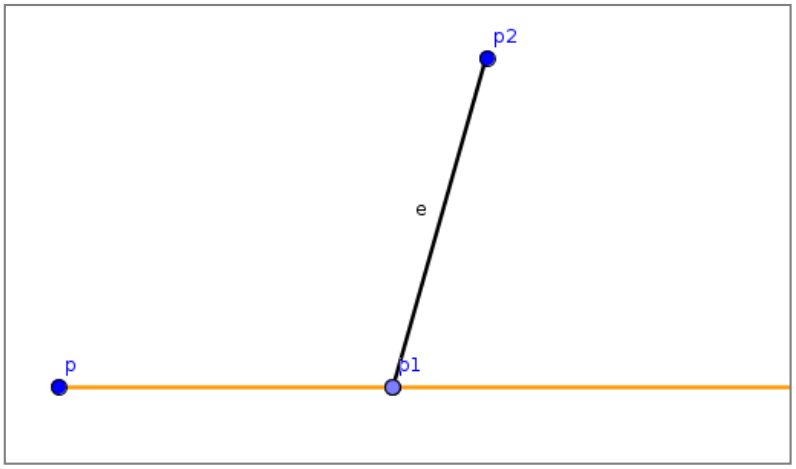
case 3 (*p* is the intersection point)

Intersect_Edge: case 4

- *Compute the intersection point between the line passing through edge e and the half-line originating from p: (x,yp).*

- It is a valid intersection if:
  3. it is the same as p
  4. **it is one of the two endpoints of e** (but it does not coincide with p – covered by case 3)
  5. it is a point inside edge e





Intersect_Edge: case 4

*// **case 4:** the intersection point (x,yp) is either the same as p1 or p2*

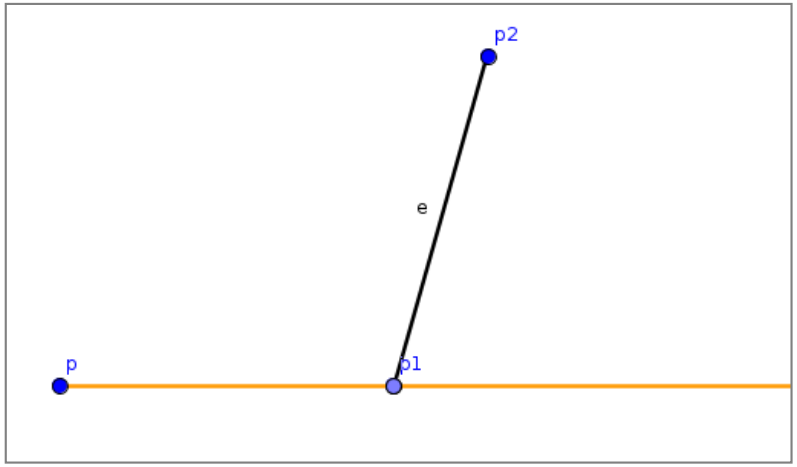
*// **case 4(a):** (x,yp) is p1 =(x1,y1)*

// if xp = x1, already covered by case 3

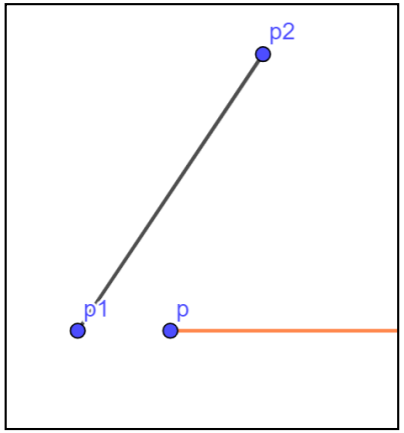
```
if (xp < x1) and (yp = y1) then
   if (y1 < y2) then
    return 1 // p1: endpoint with lowest y
          value
 else
    return 0
```

// if xp > x1, there is NO INTERSECTION and we return 0

case 4(a)



when xp>x1 in case 4(a)



Intersect_Edge: case 4

*// **case 4:** the intersection point (x,yp) is the same as p1 or p2*

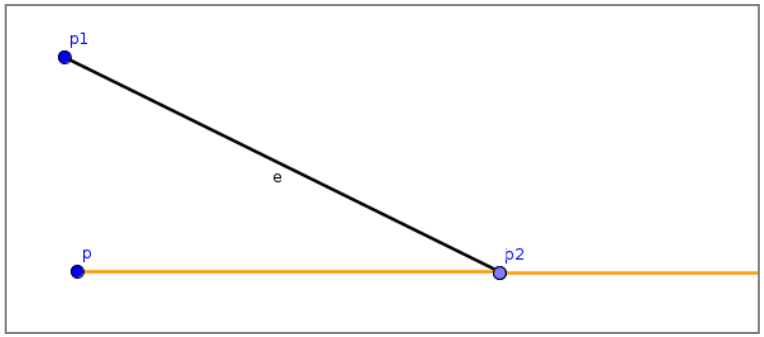
*// **case 4(b):** (x,yp) is p2=(x2,y2)*

// if xp = x2, already covered by case 3

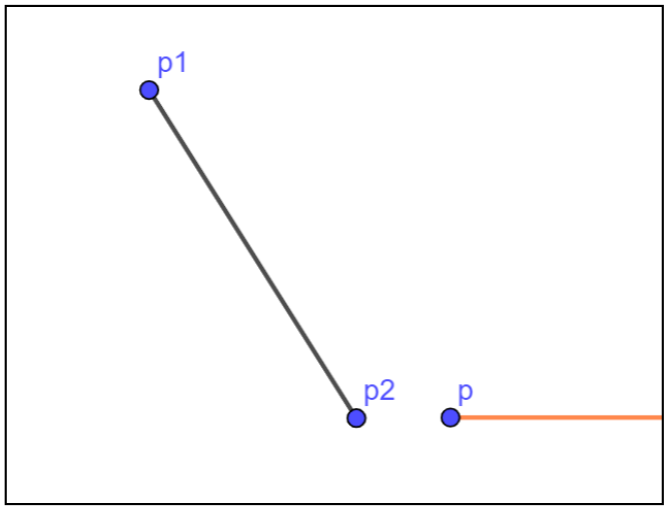
```
if (xp < x2) and (yp = y2) then
  if (y2 < y1) then
     return 1 // p2: endpoint with lowest
        y value
  else
  return 0
```

// if xp > x2, there is NO INTERSECTION and we return 0

case 4(b)



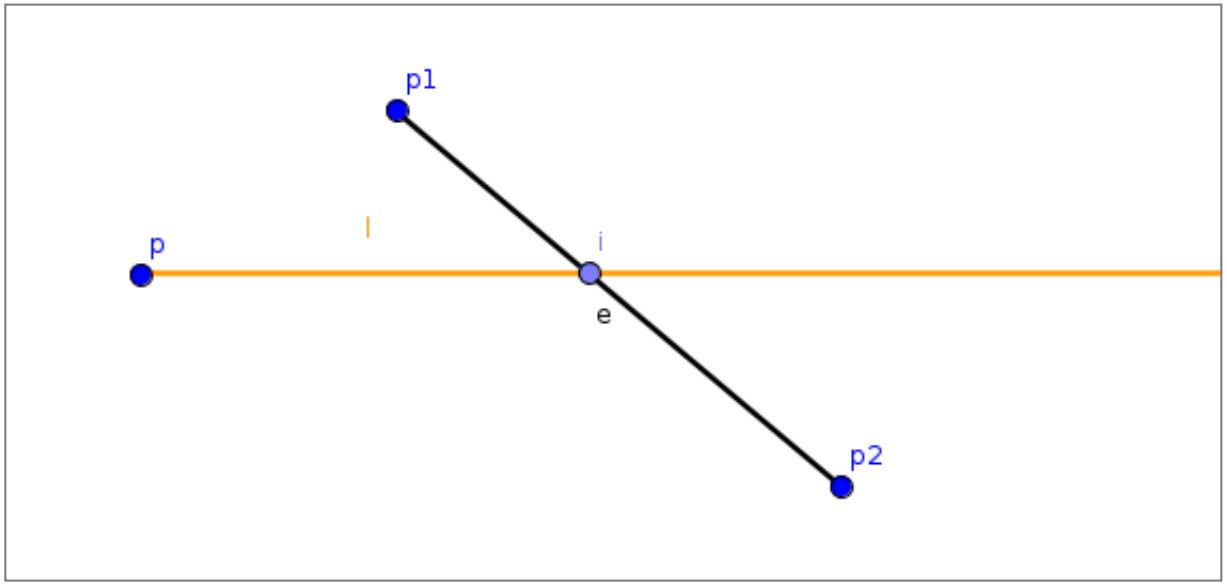
when xp>x1 in case 4(b)



Intersect_Edge: case 5

- *Compute the intersection point between the line passing through edge e and the half-line originating from p: (x,yp).*

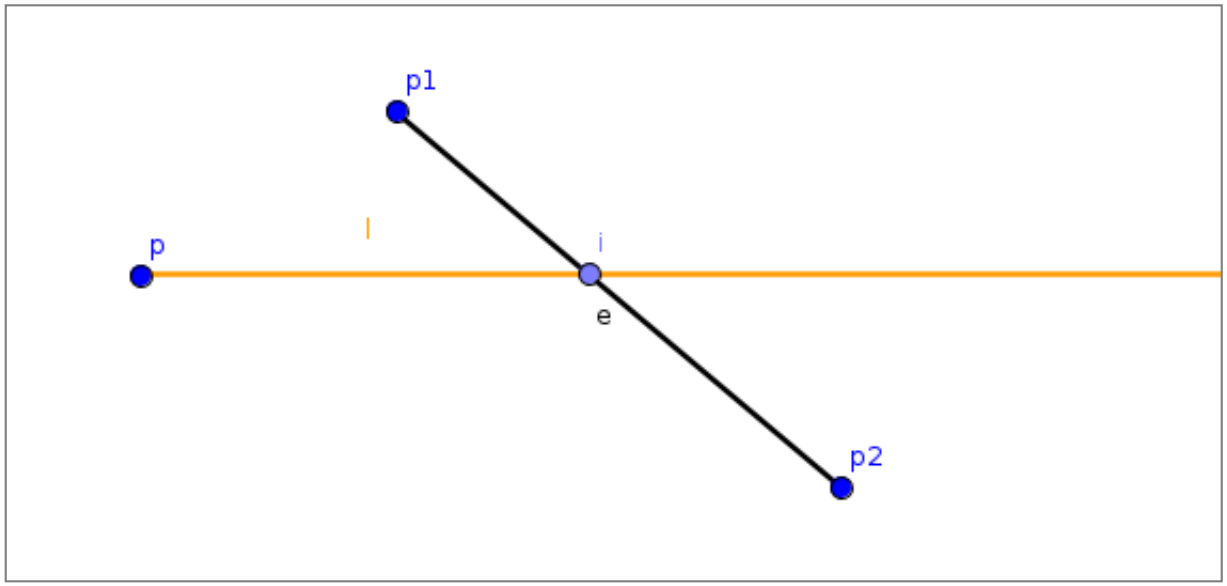
- It is a valid intersection if:
  3. it is the same as p
  4. it is one of the two endpoints of e
  5. **it is a point inside edge e**



Intersect_Edge: case 5

```
// Case 5: intersection point  (x,yp) inside the edge;
      the intersection happens at the right of p and its y-coordinate yp is into
      (ymin,ymax) range

if (x > xp) and (ymin < yp < ymax) then
     return 1 // INTERSECTION
     // otherwise its y-coordinate is outside (ymin,ymax) range
else
     return 0 // NO INTERSECTION
```



case 5 (intersection point inside the edge)