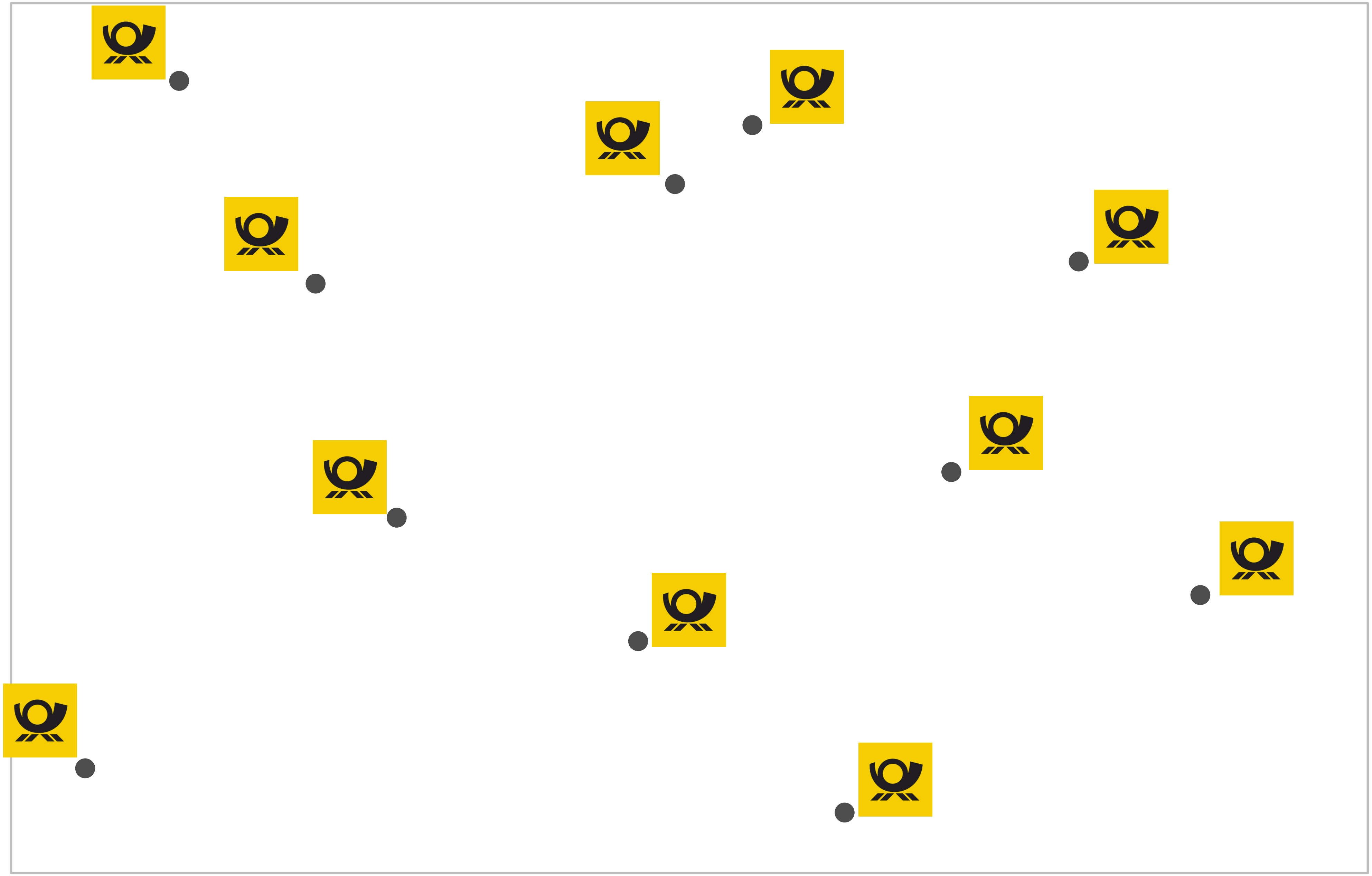
Computational Geometry Lecture
Voronoi Diagrams

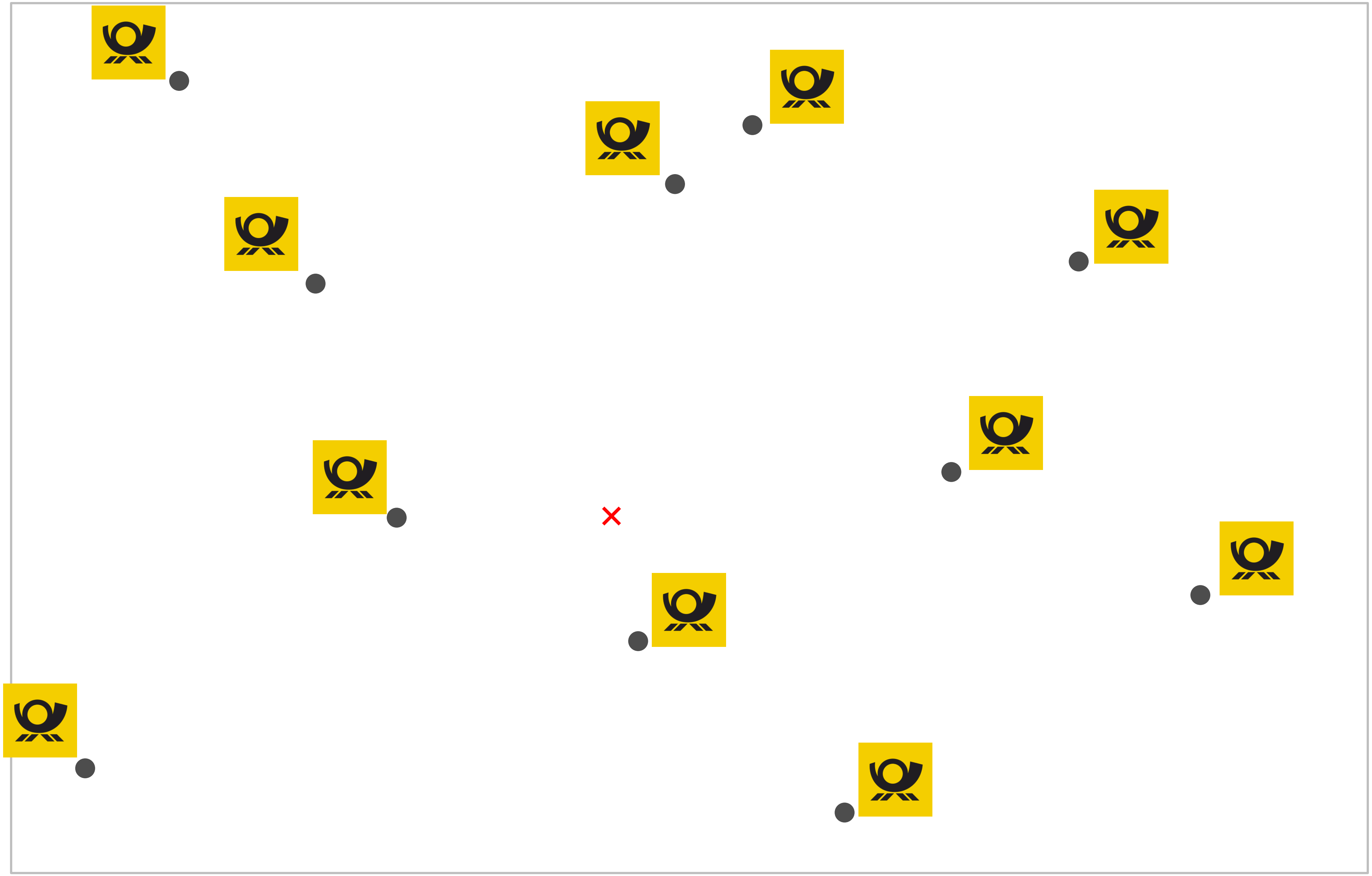
INSTITUT FÜR THEORETISCHE INFORMATIK · FAKULTÄT FÜR INFORMATIK

Tamara Mchledidze · Darren Strash
03.06.2014

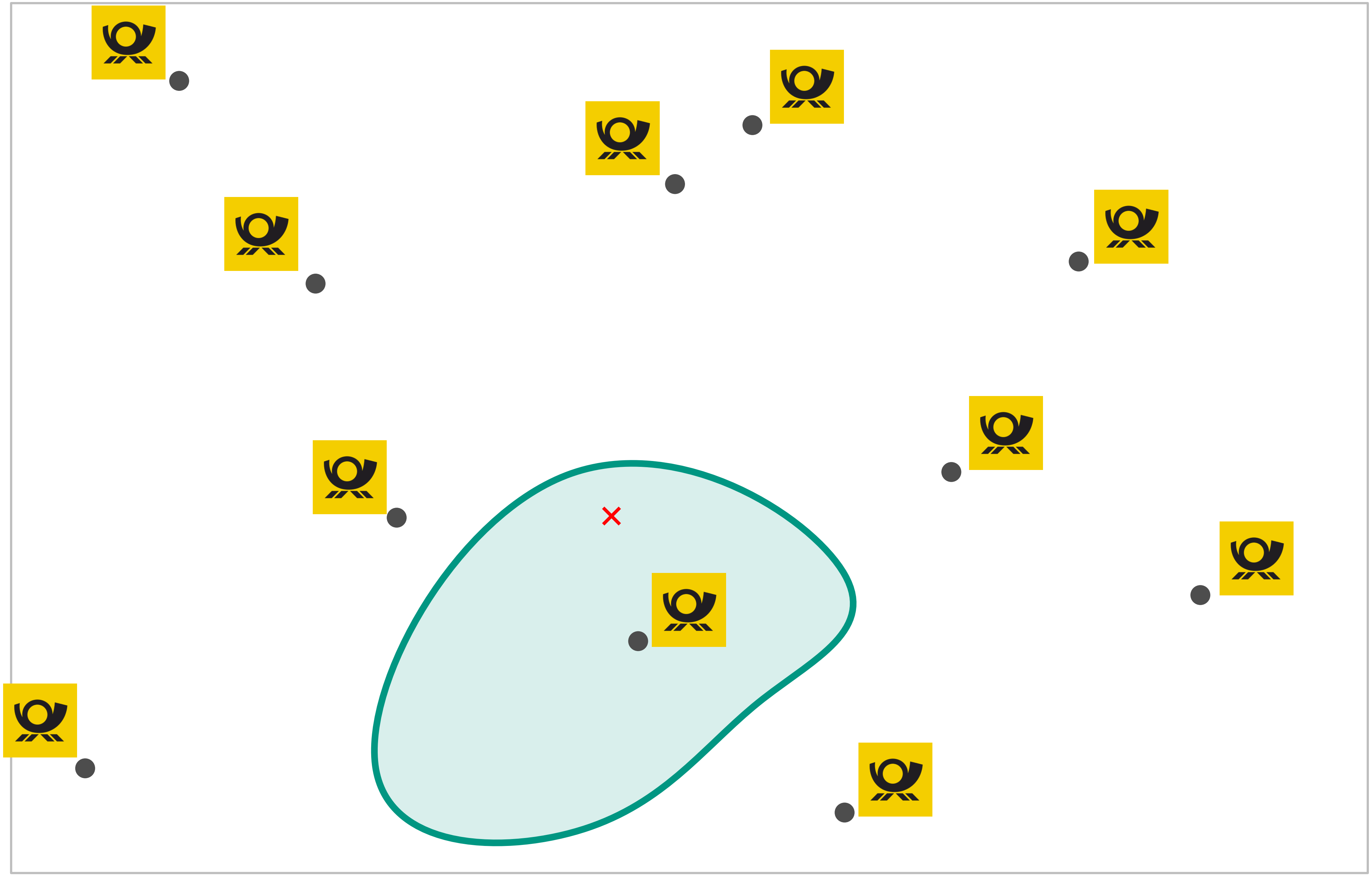
The Post Office Problem



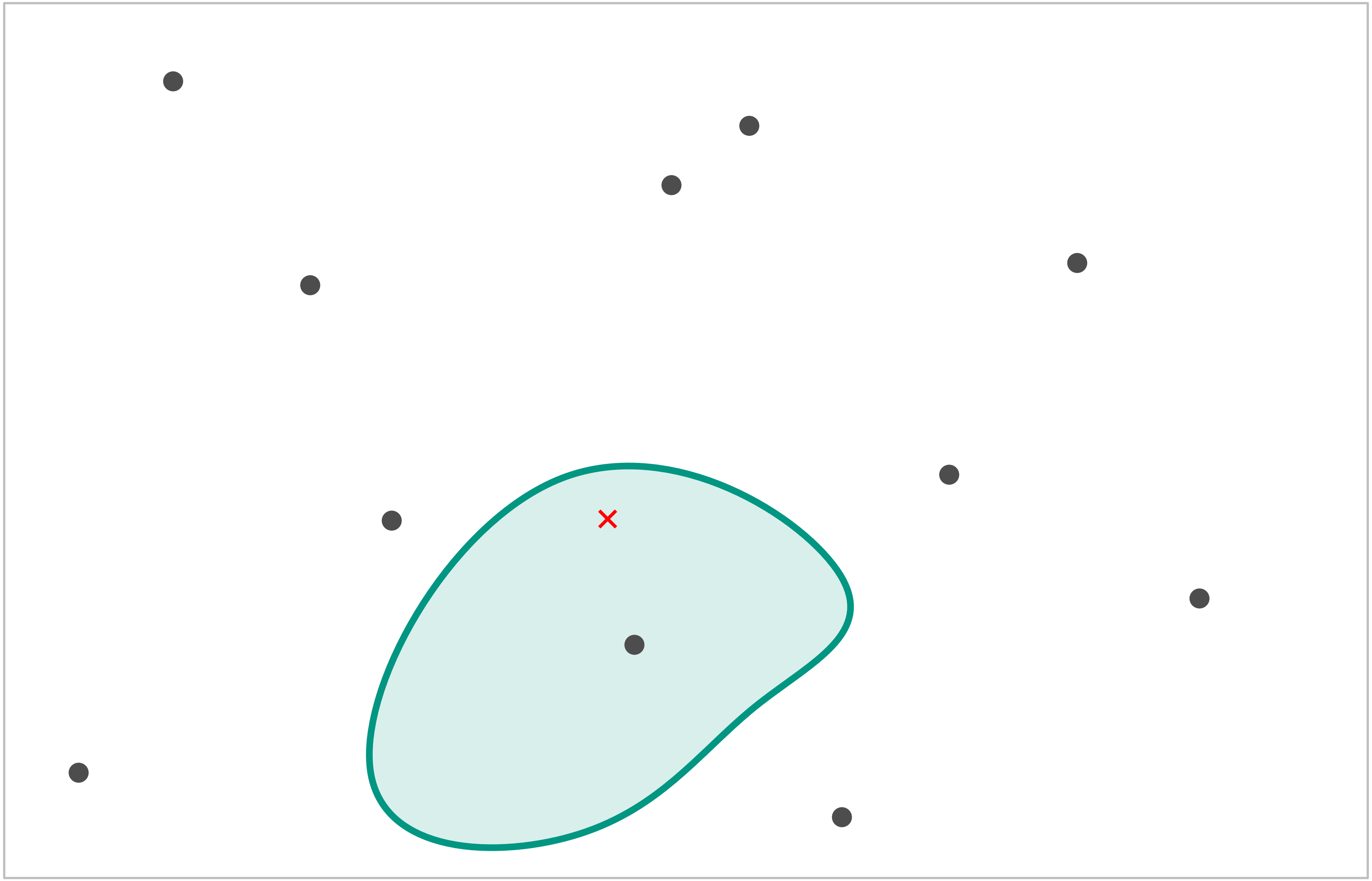
The Post Office Problem



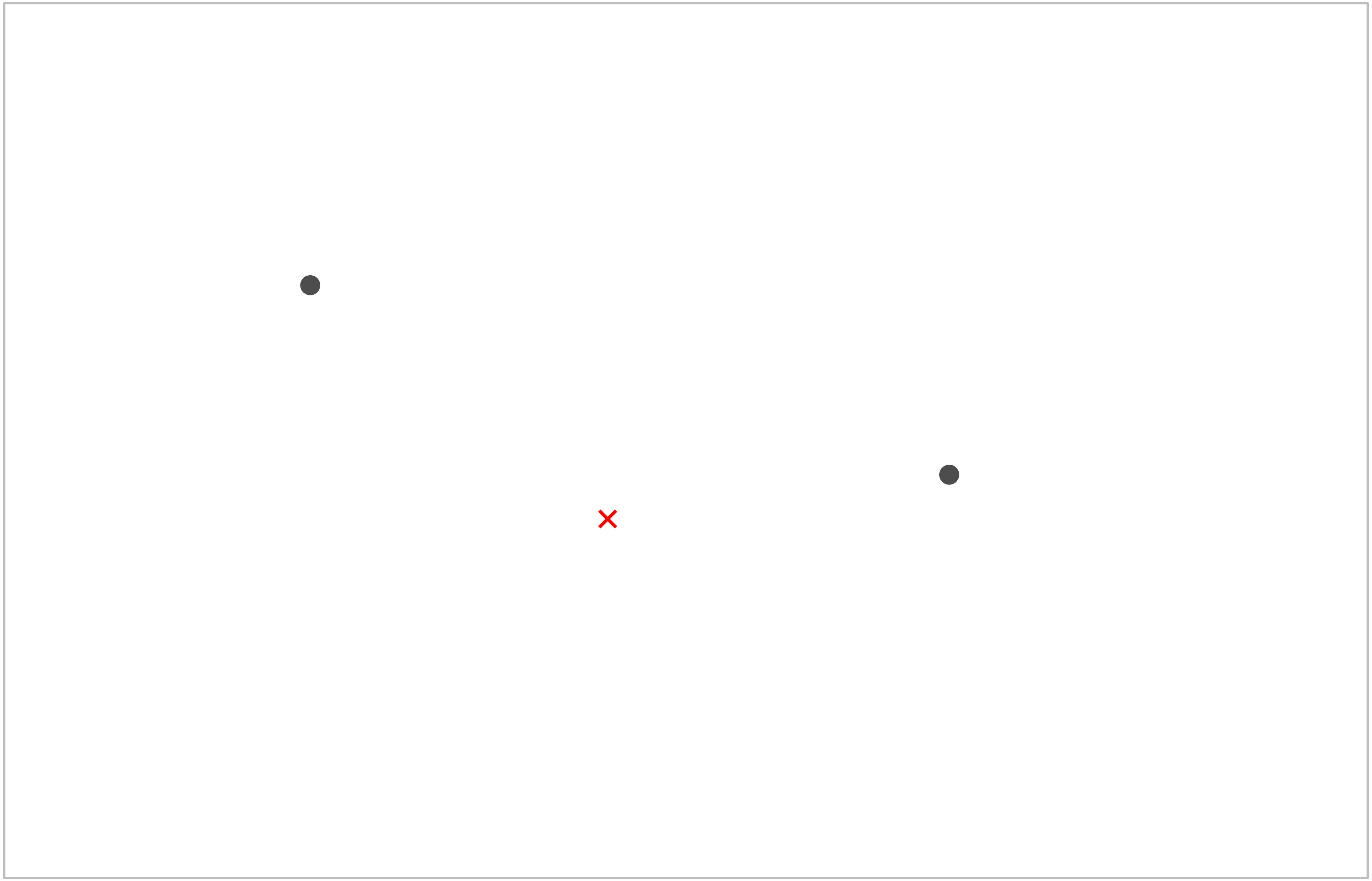
The Post Office Problem



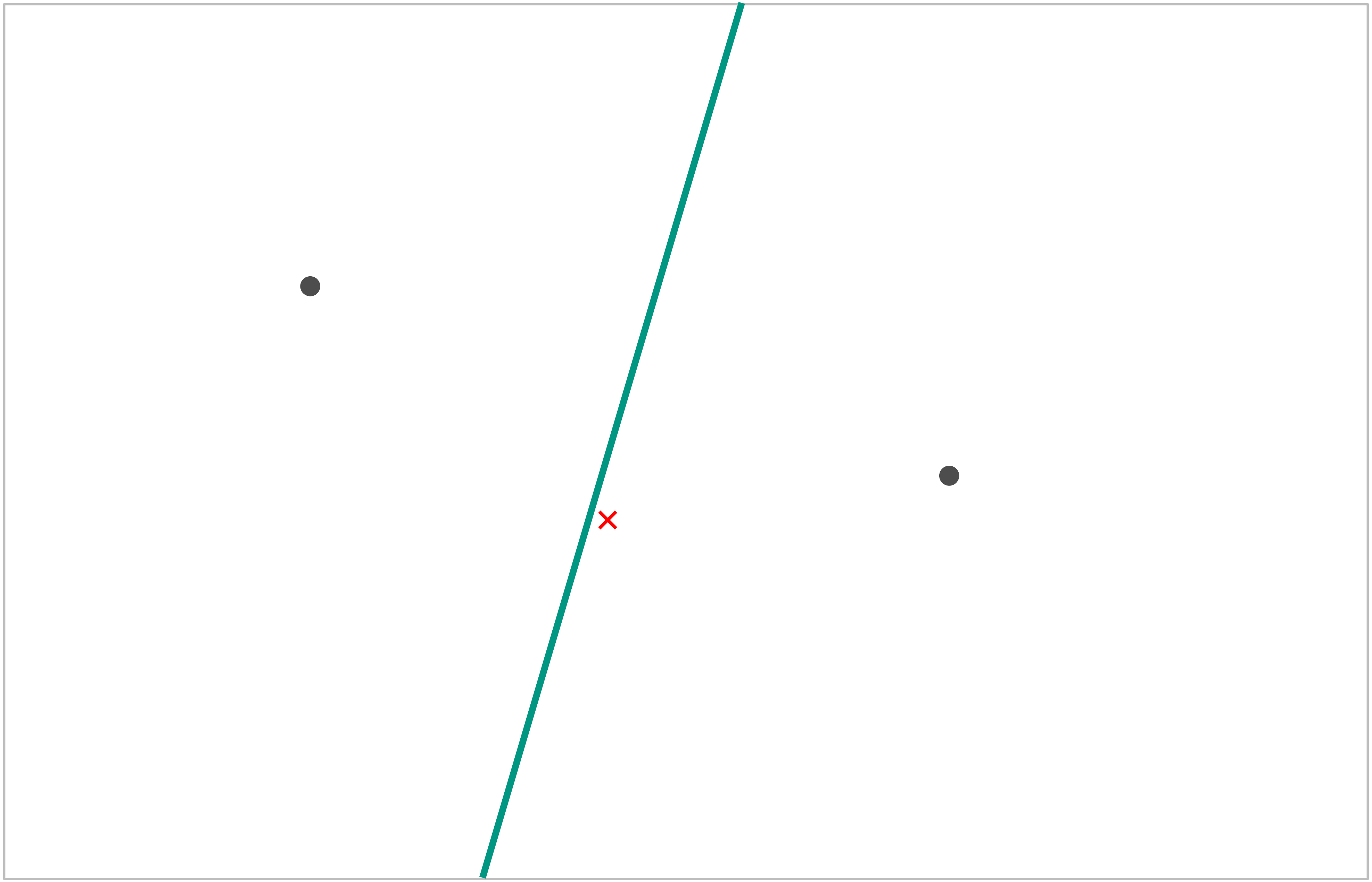
The Post Office Problem



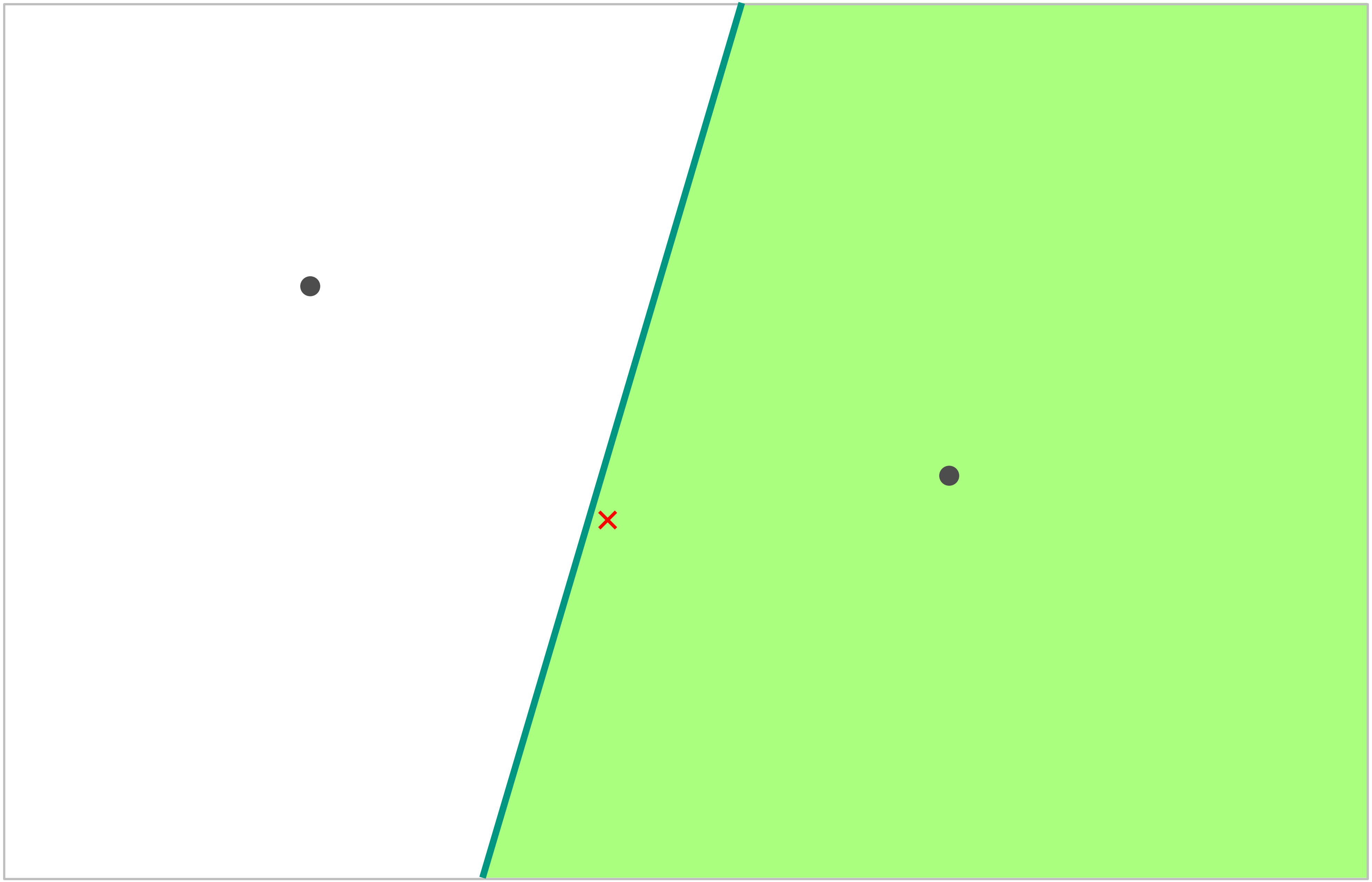
The Post Office Problem



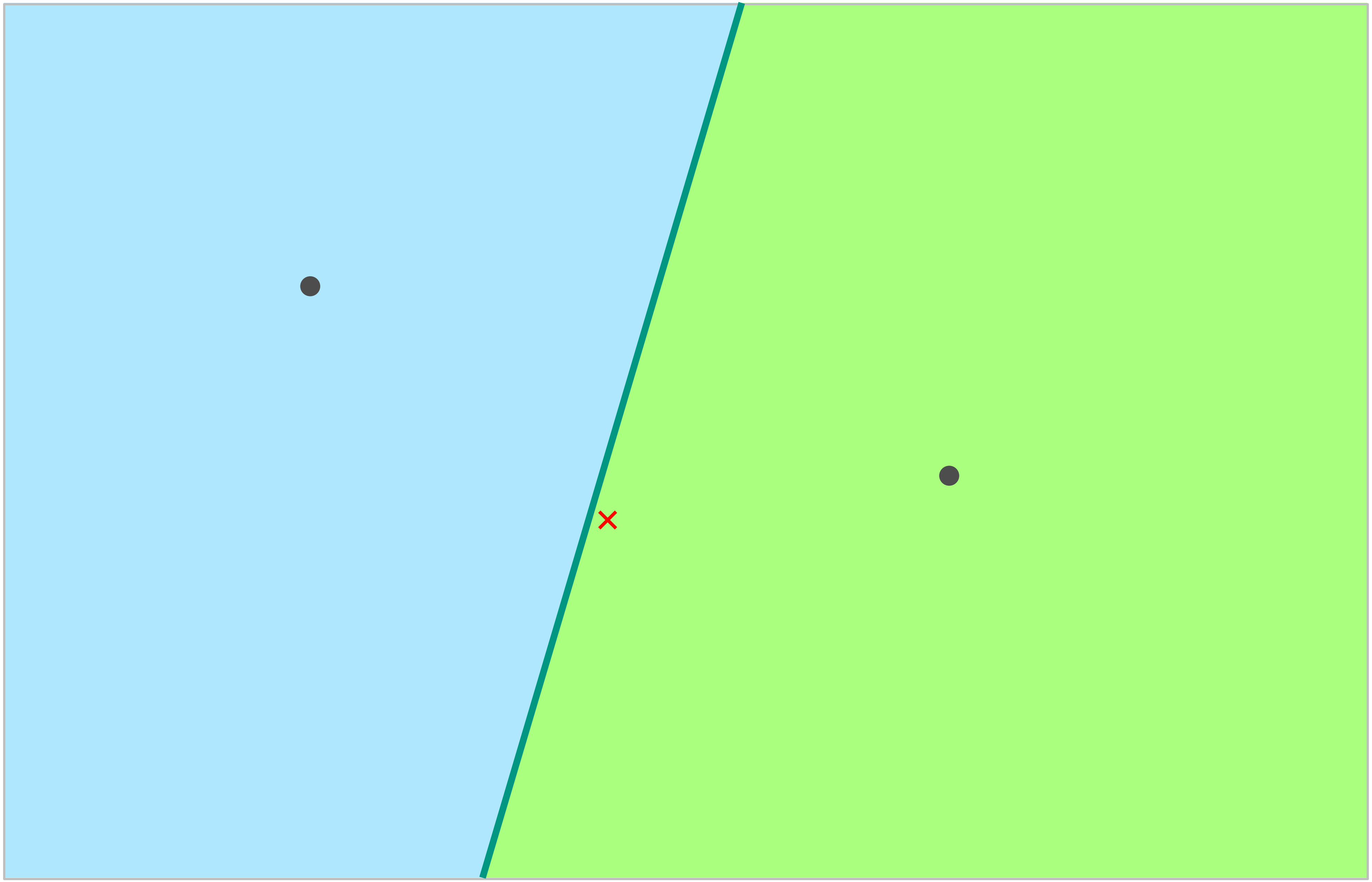
The Post Office Problem



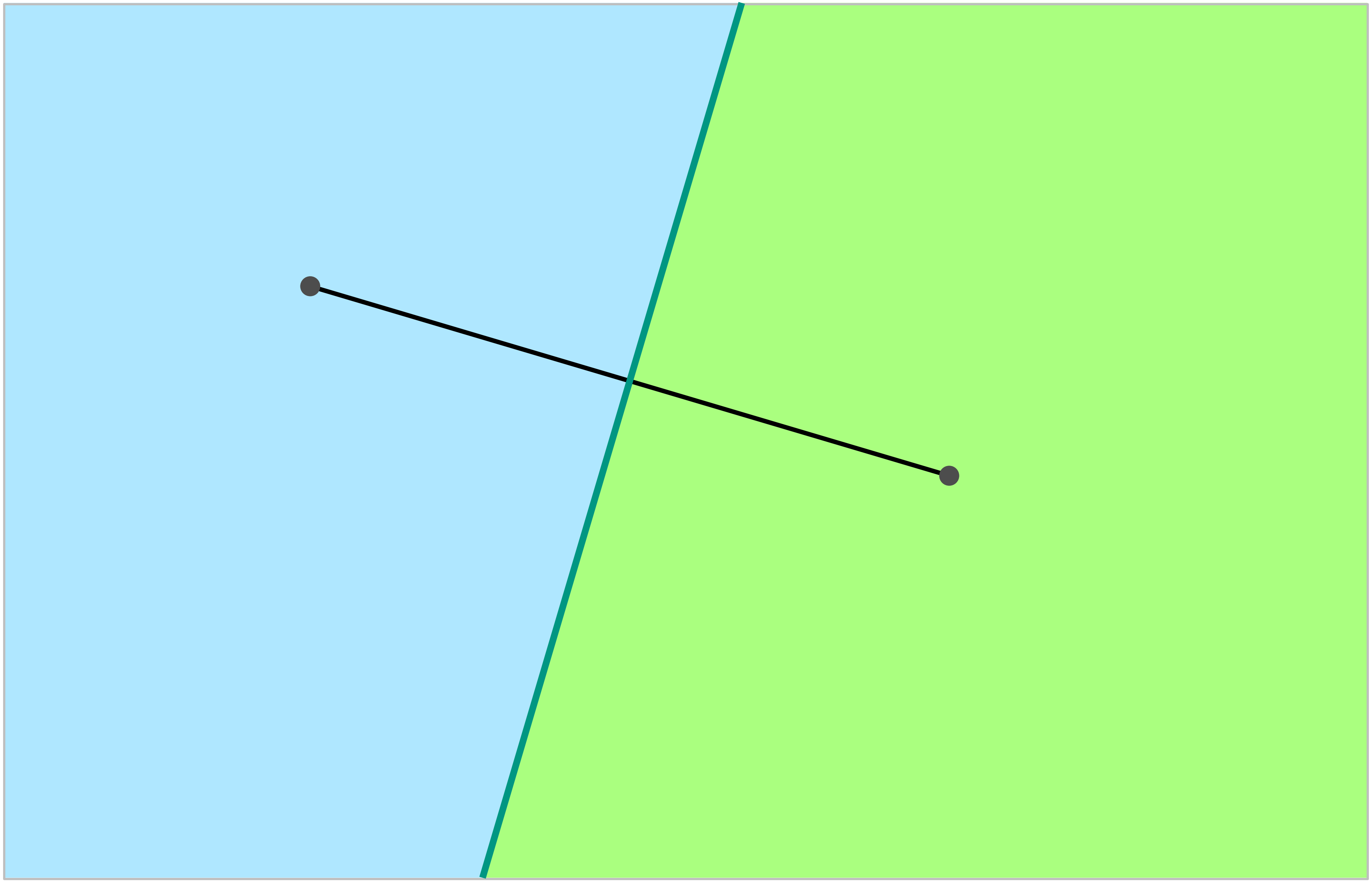
The Post Office Problem



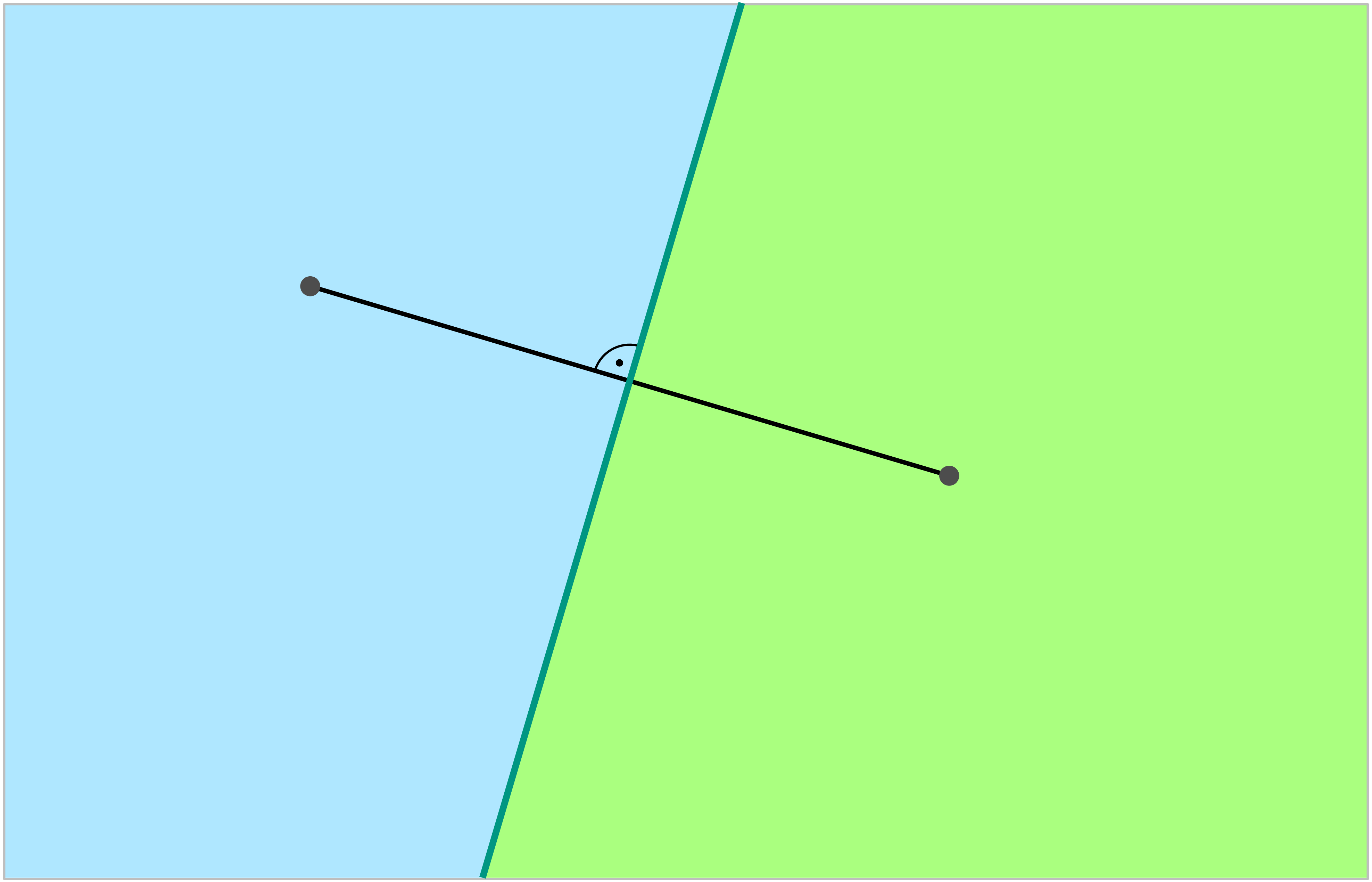
The Post Office Problem



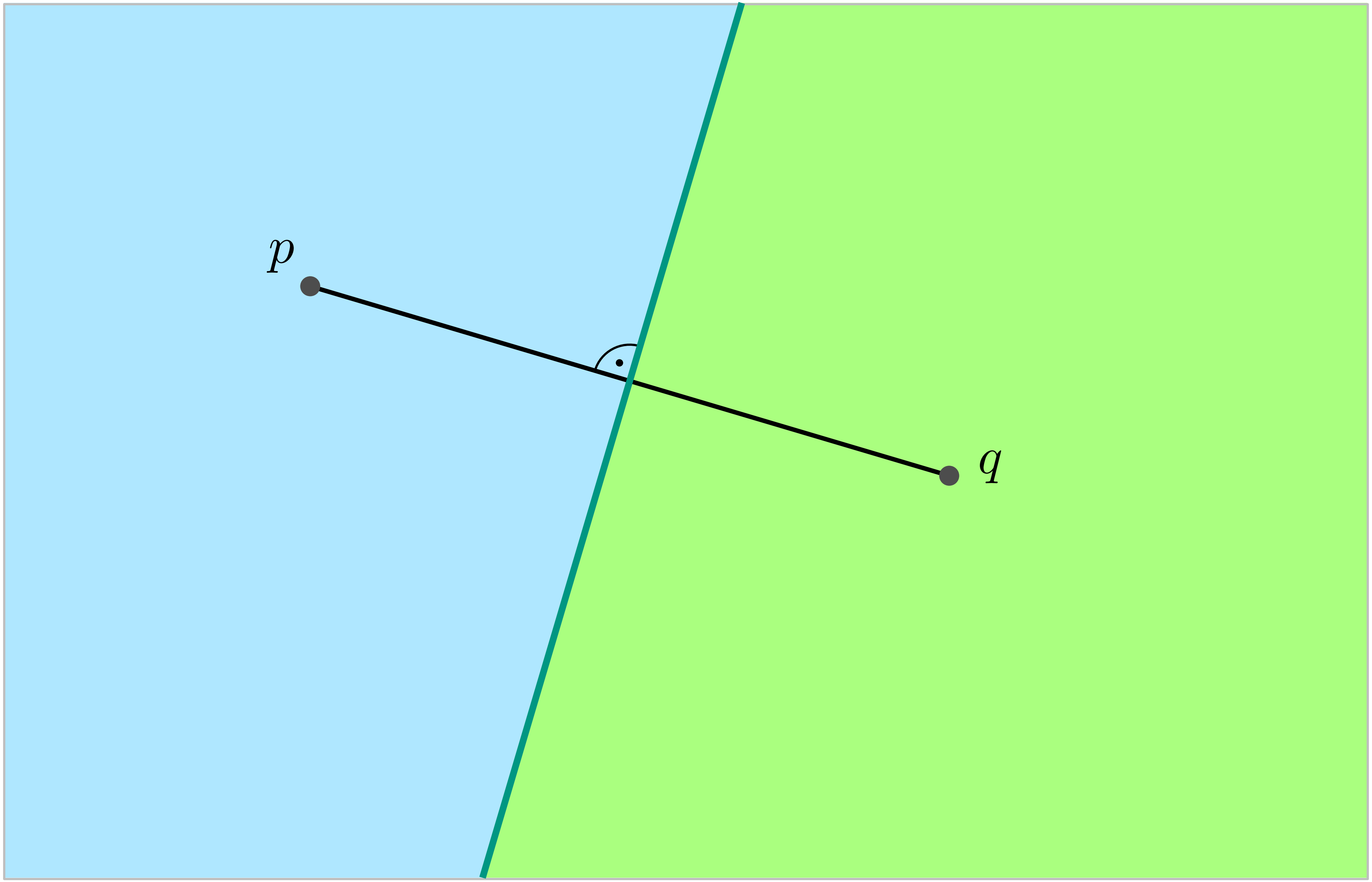
The Post Office Problem



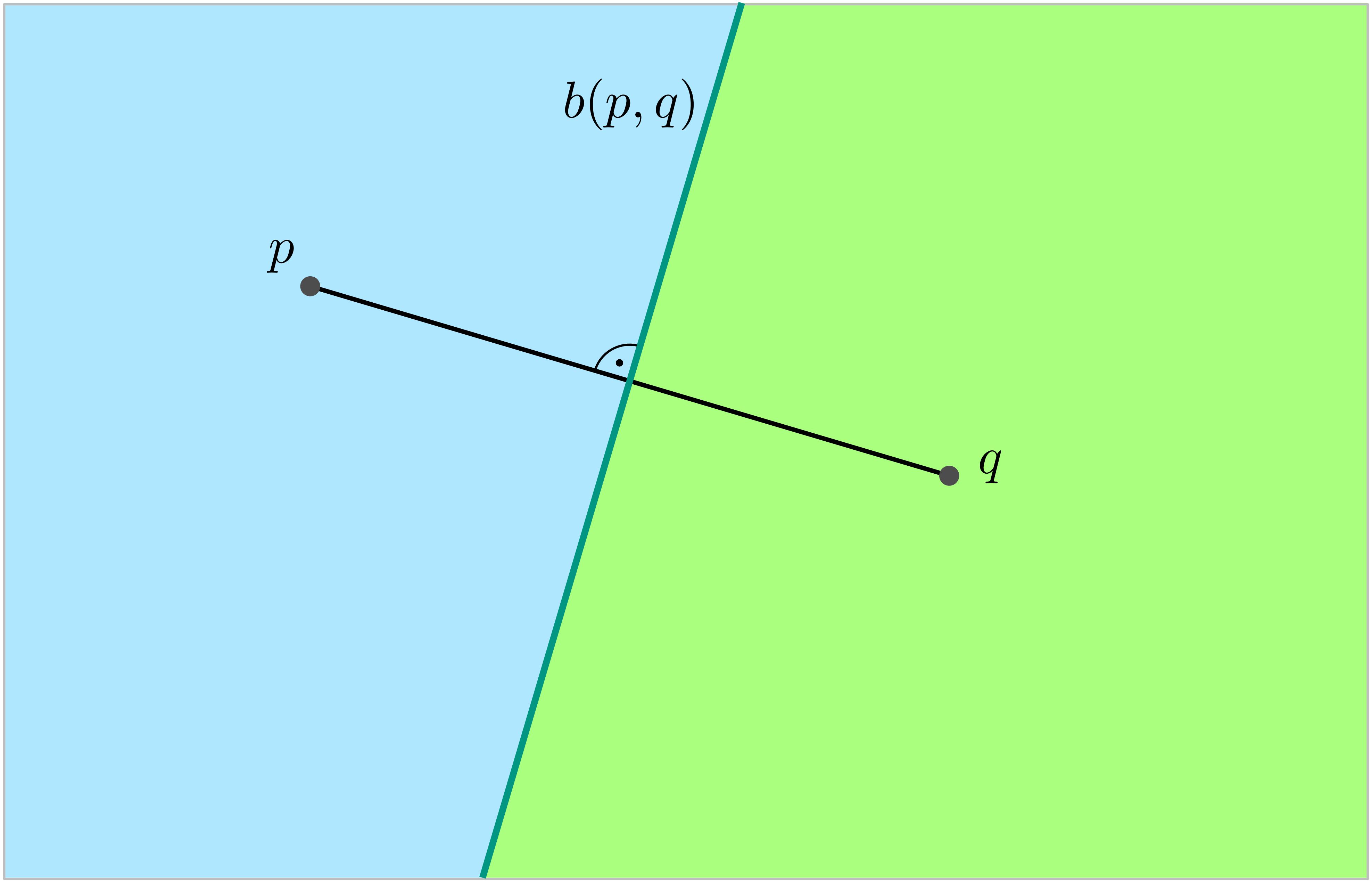
The Post Office Problem



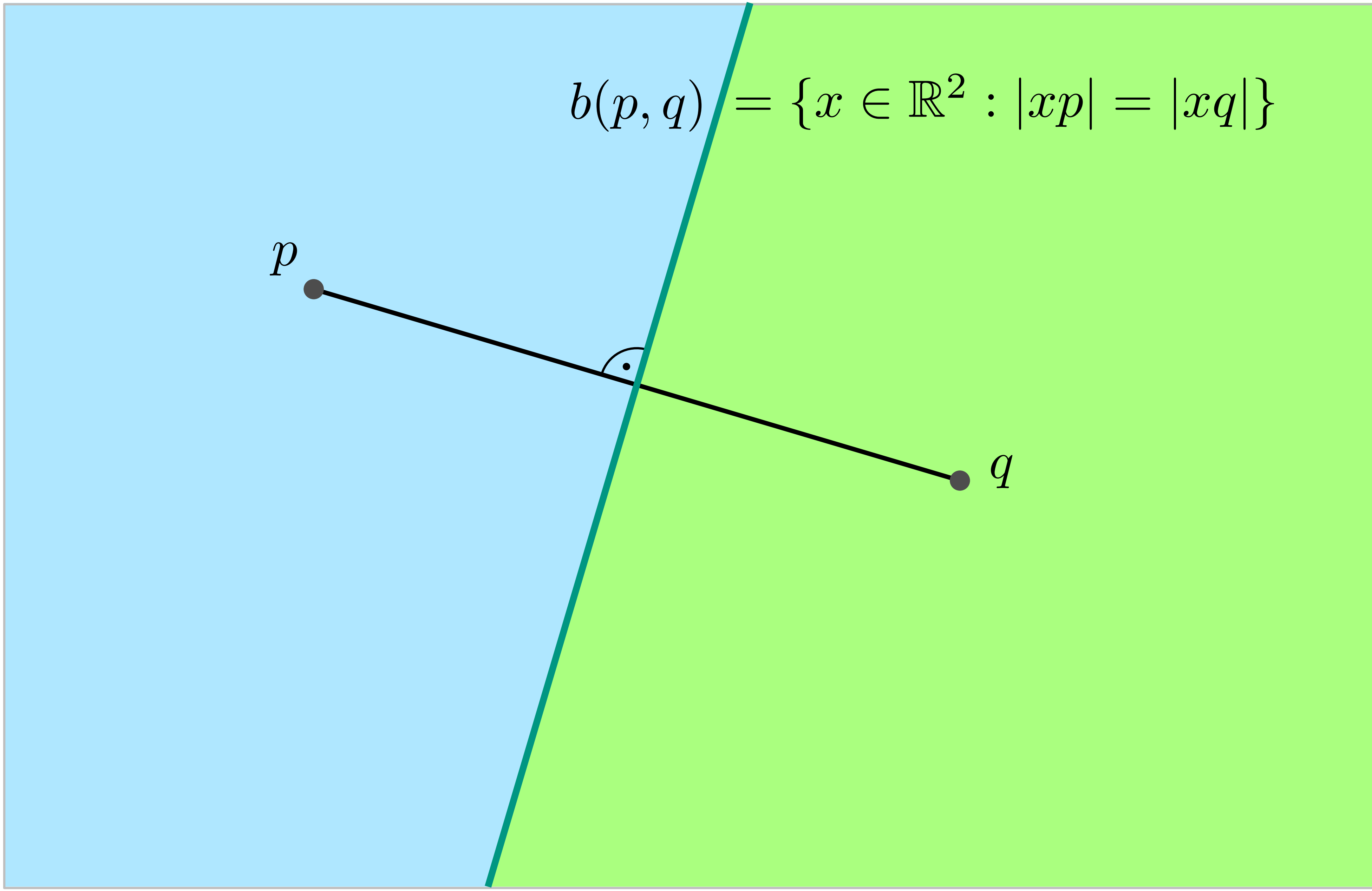
The Post Office Problem



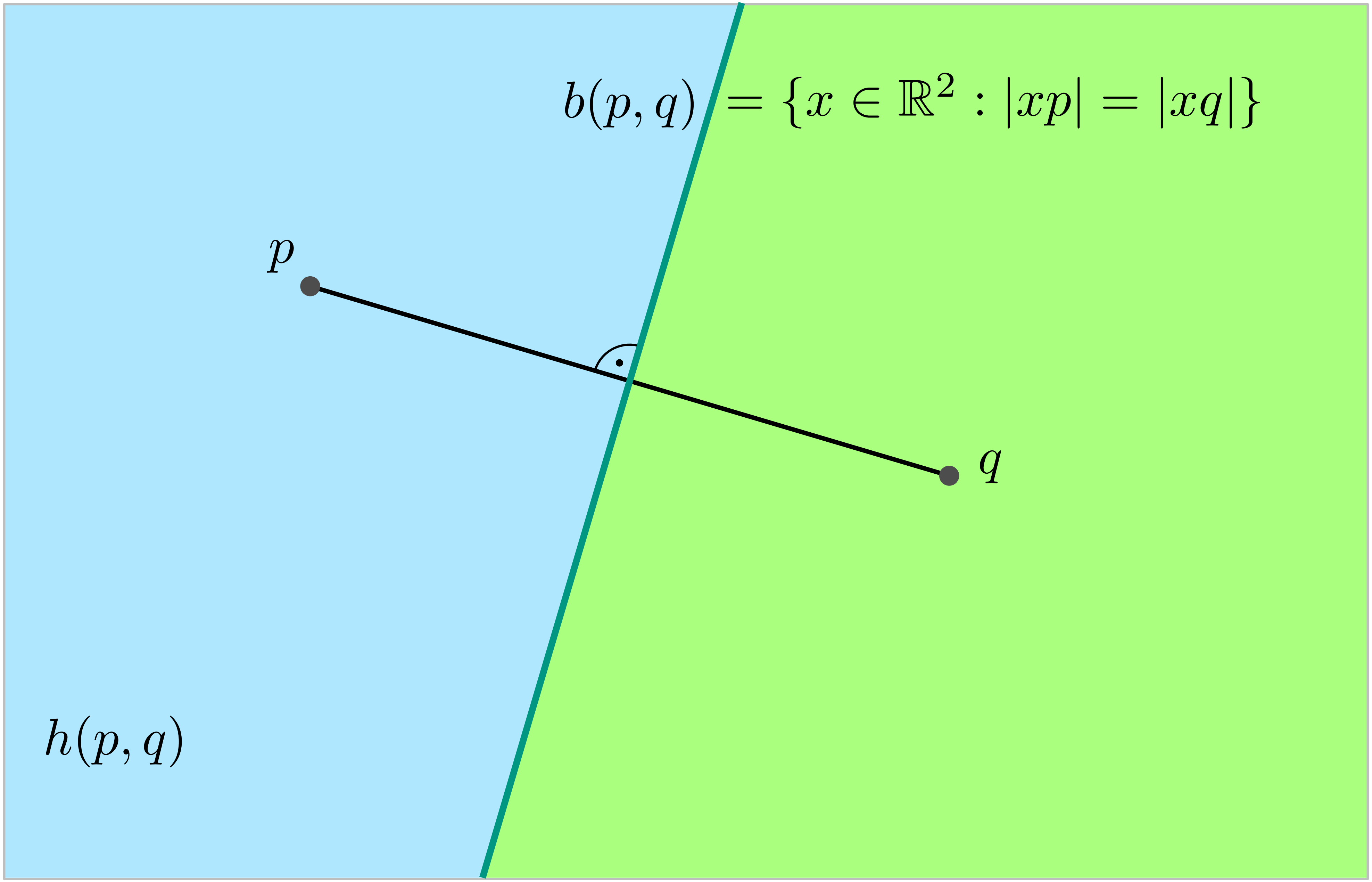
The Post Office Problem



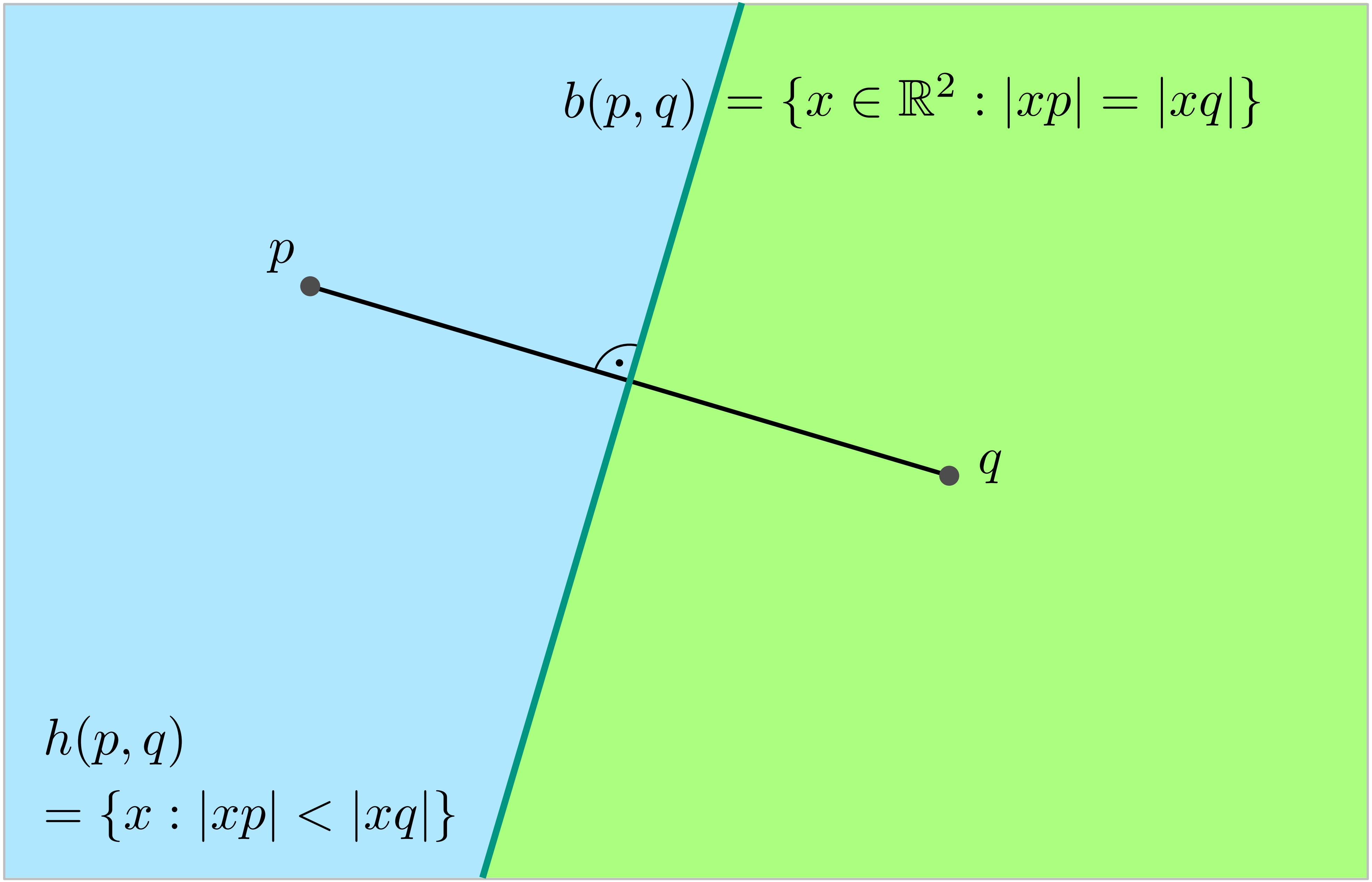
The Post Office Problem



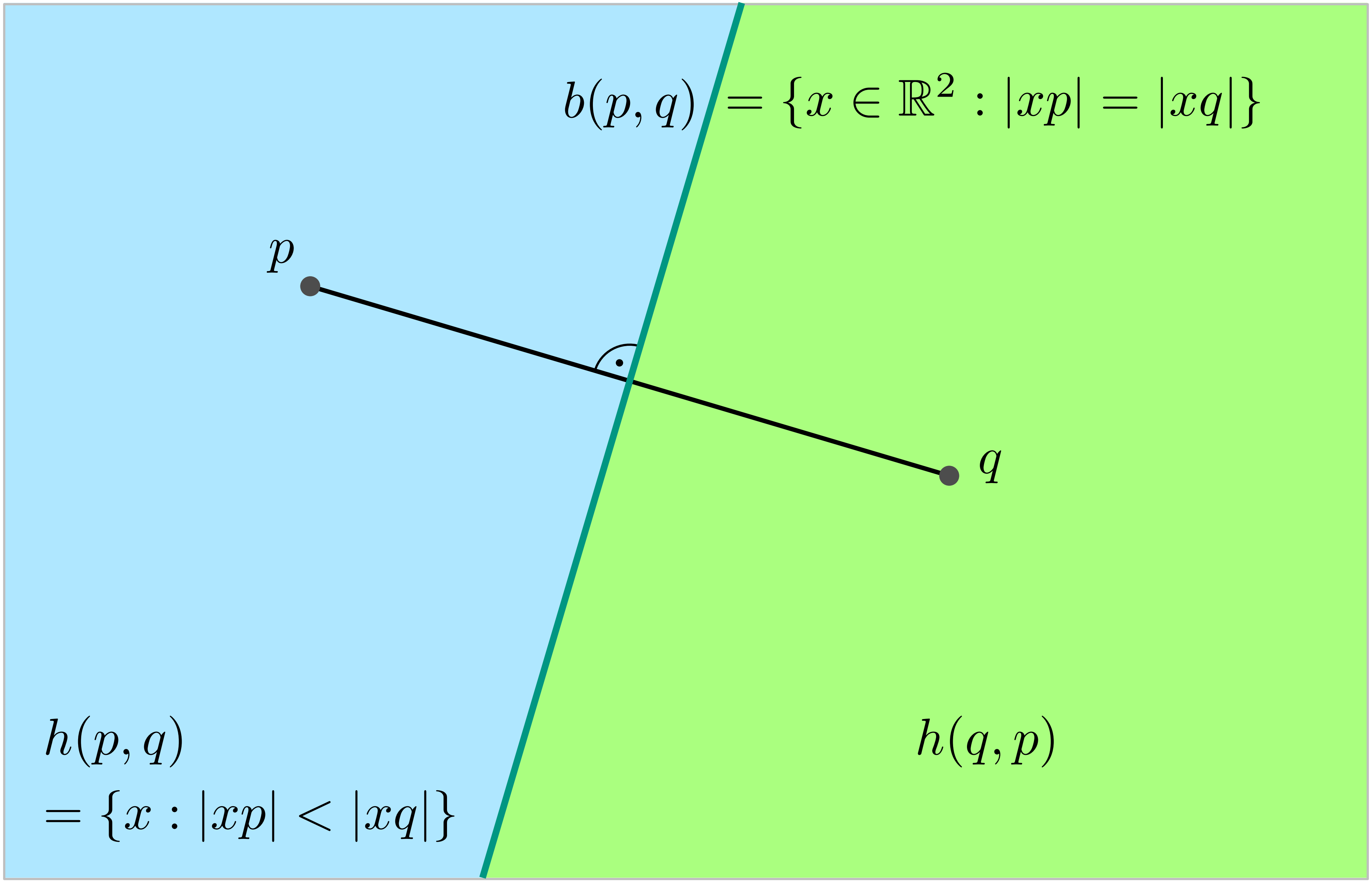
The Post Office Problem



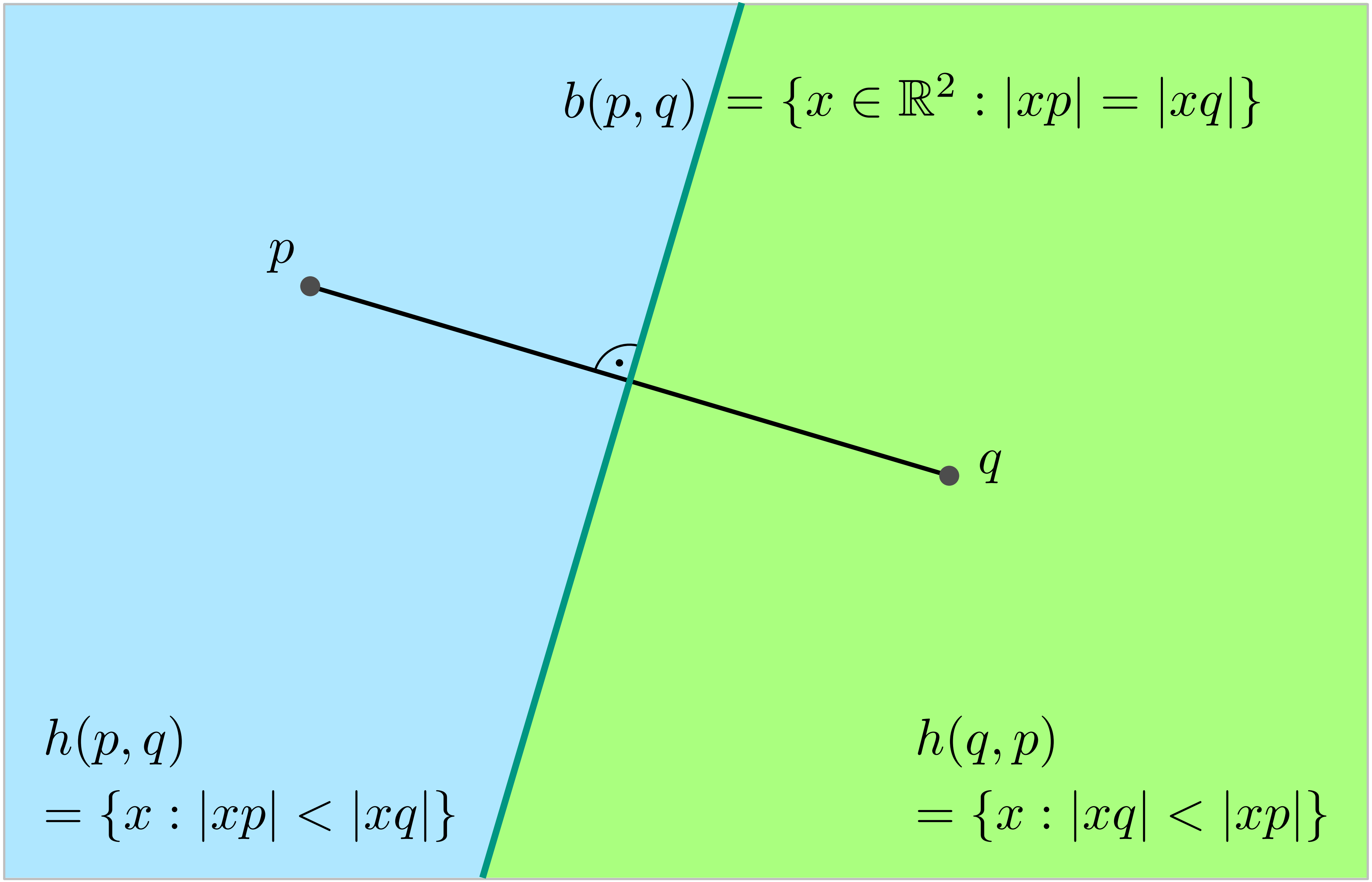
The Post Office Problem



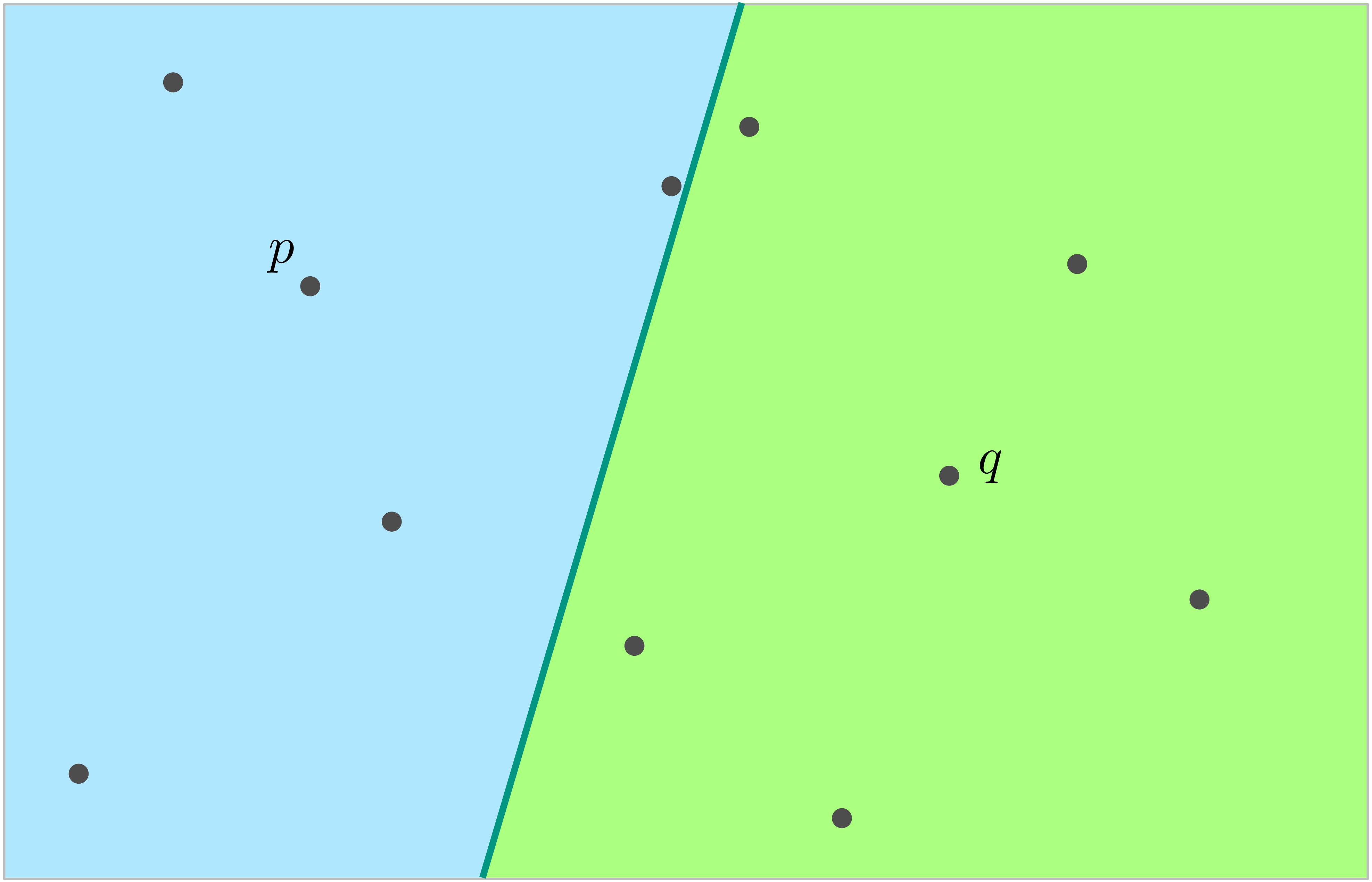
The Post Office Problem



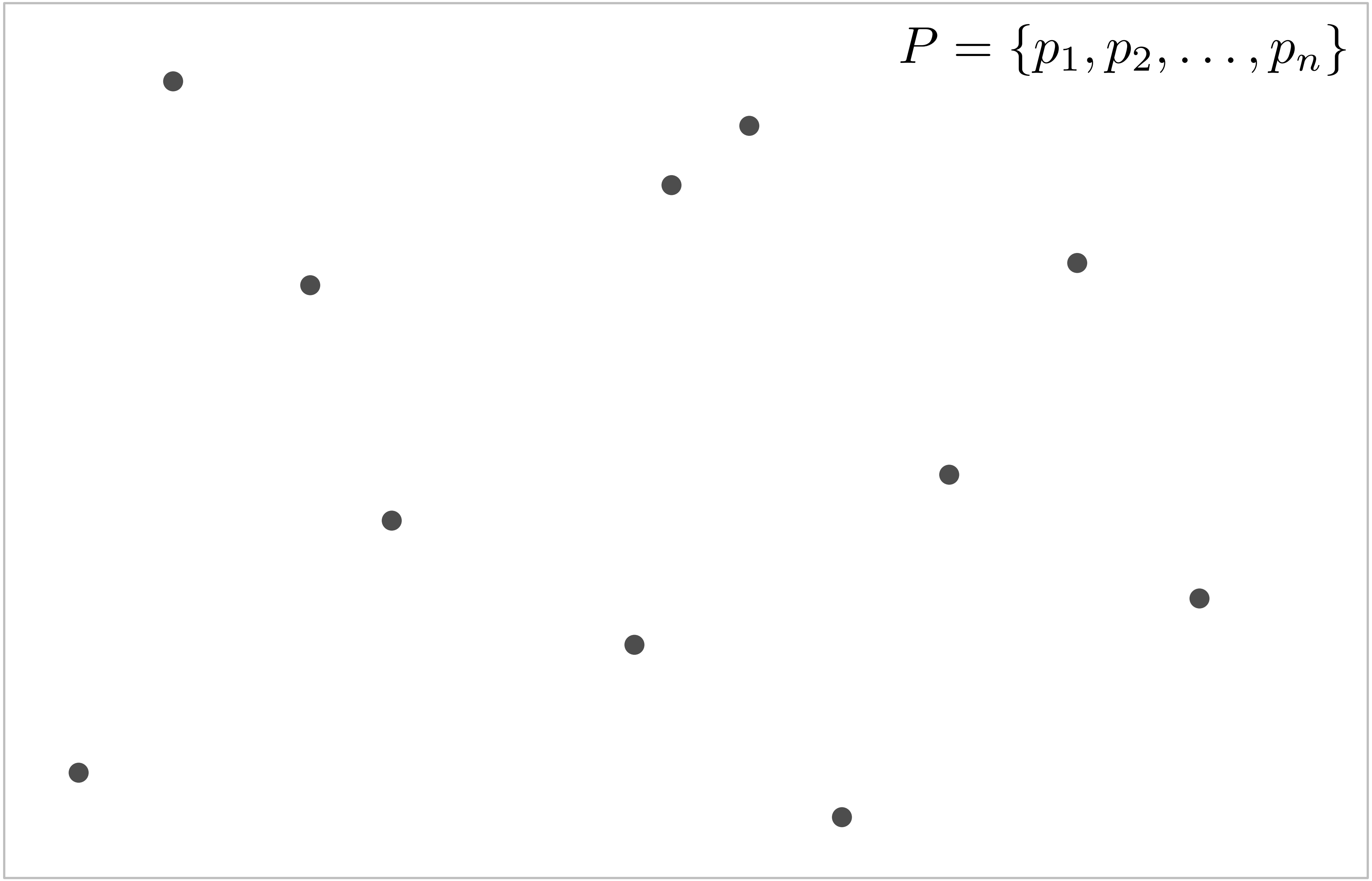
The Post Office Problem



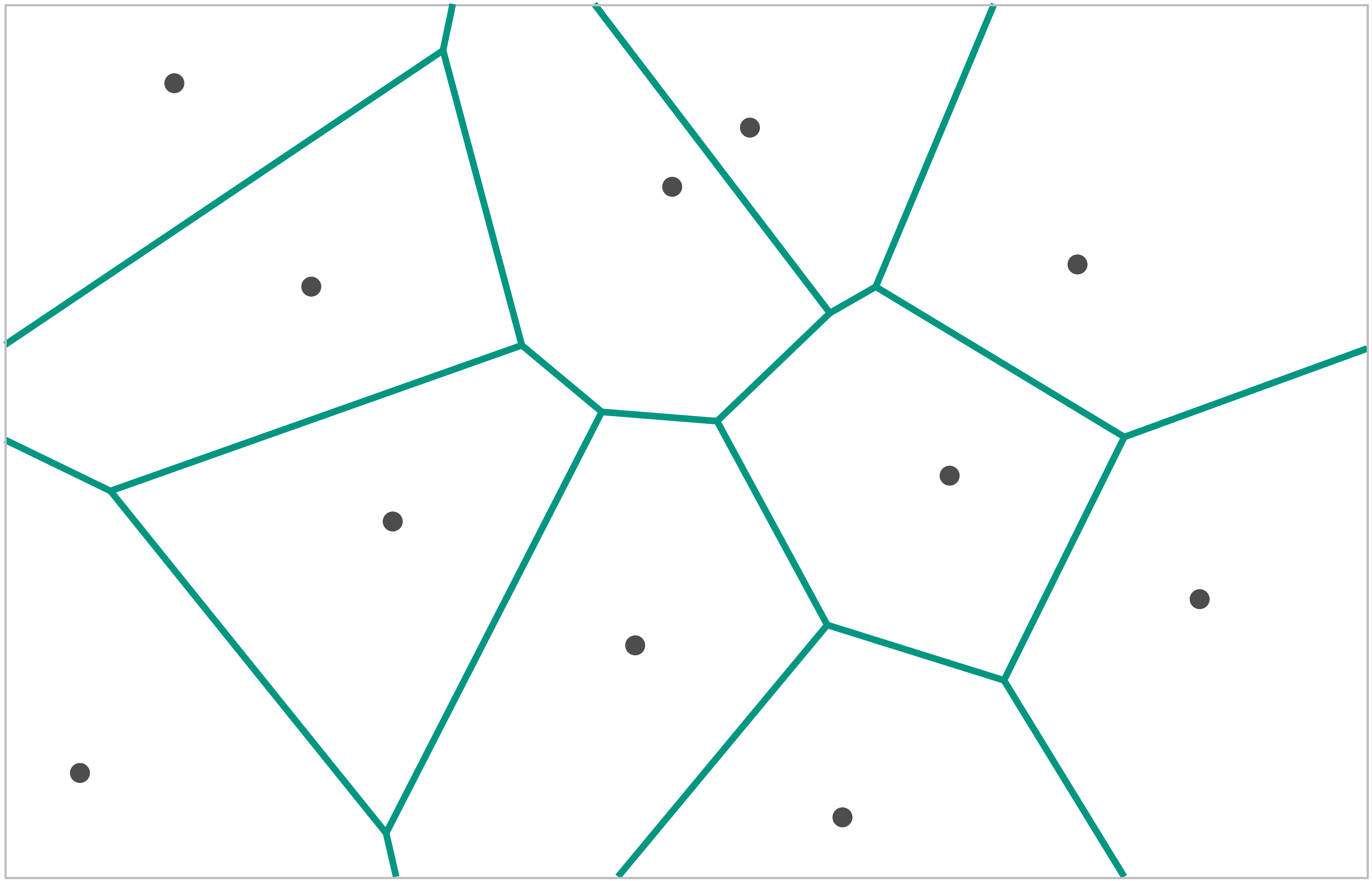
The Post Office Problem



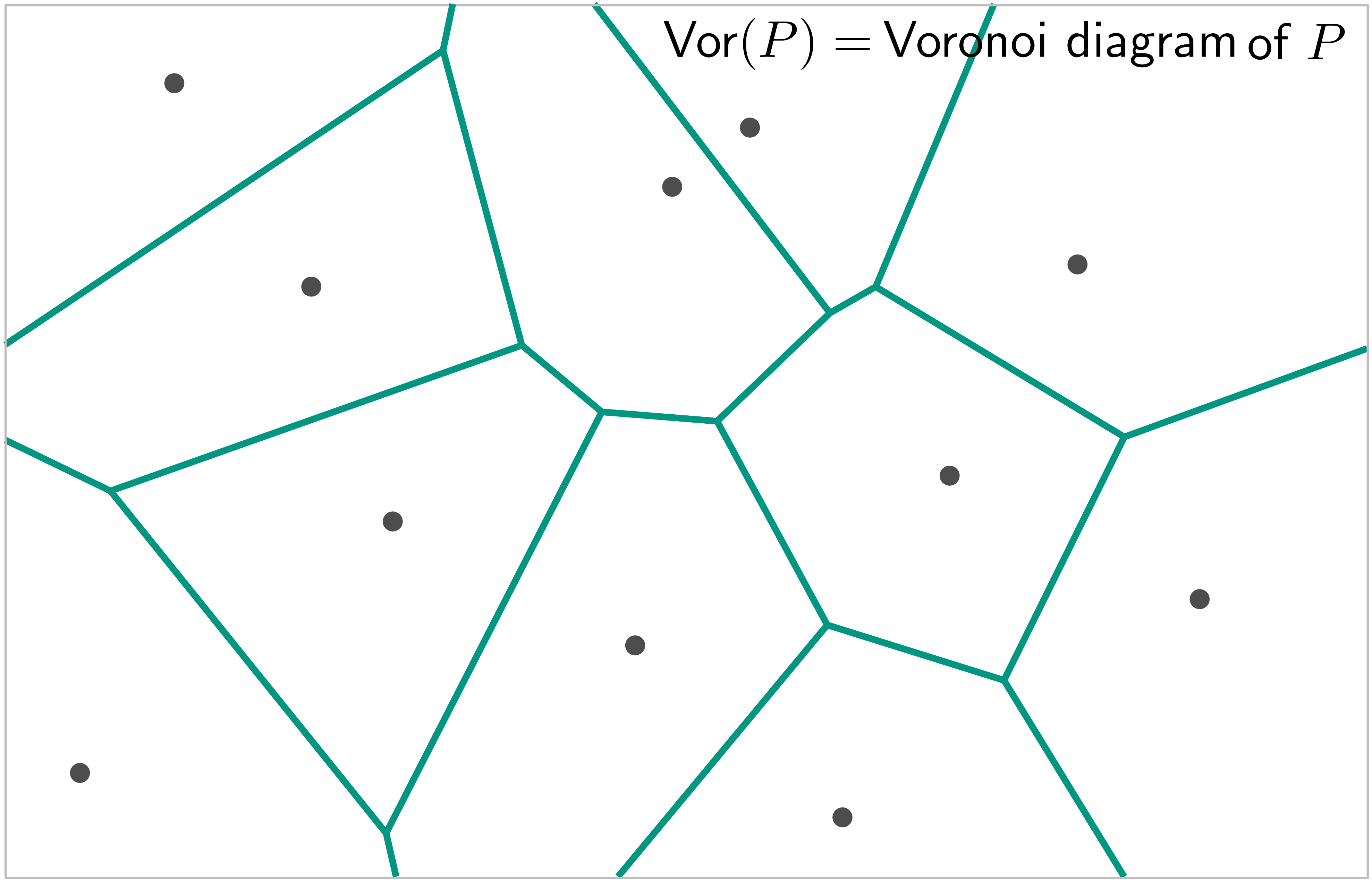
The Post Office Problem



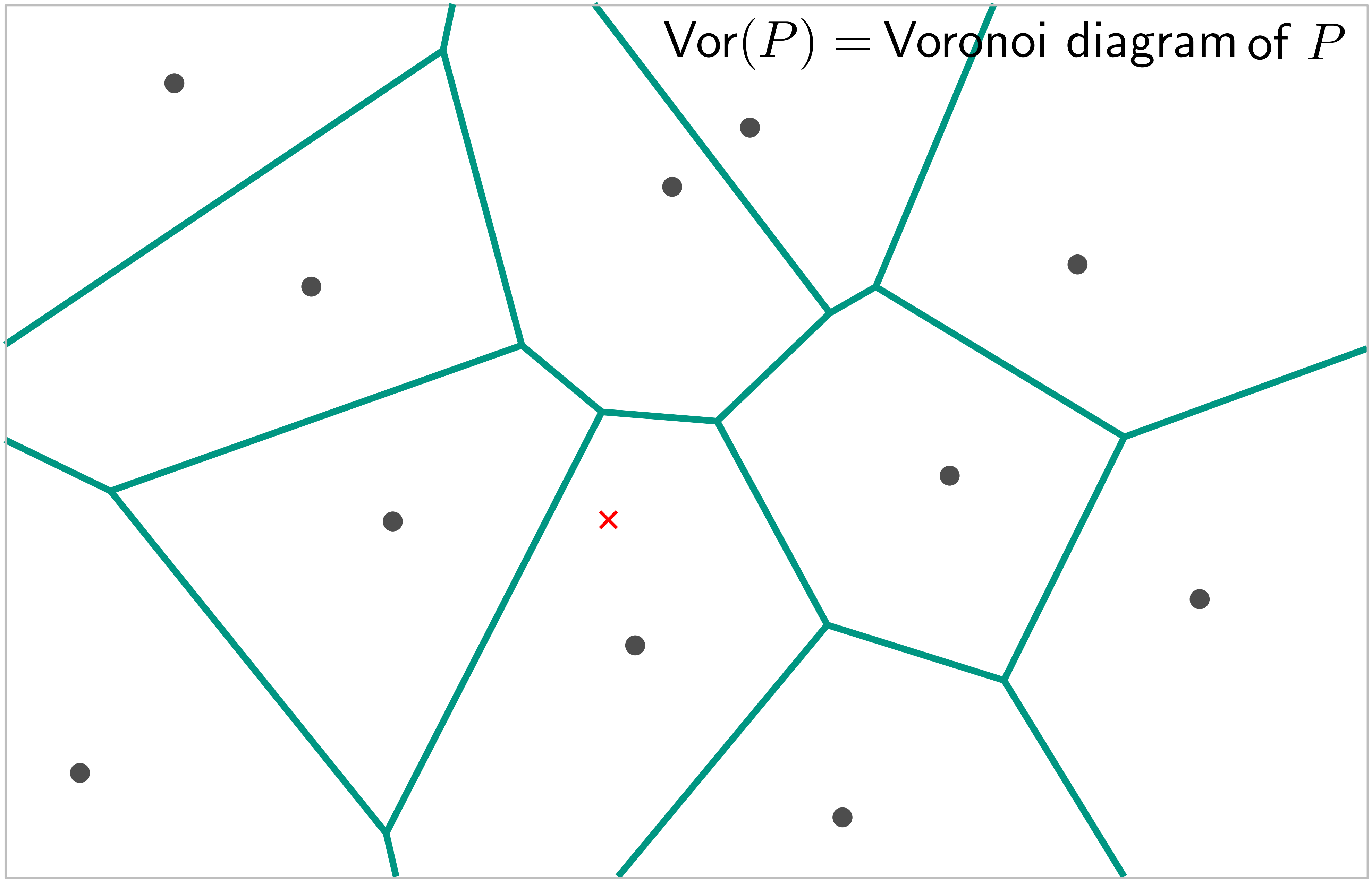
The Post Office Problem



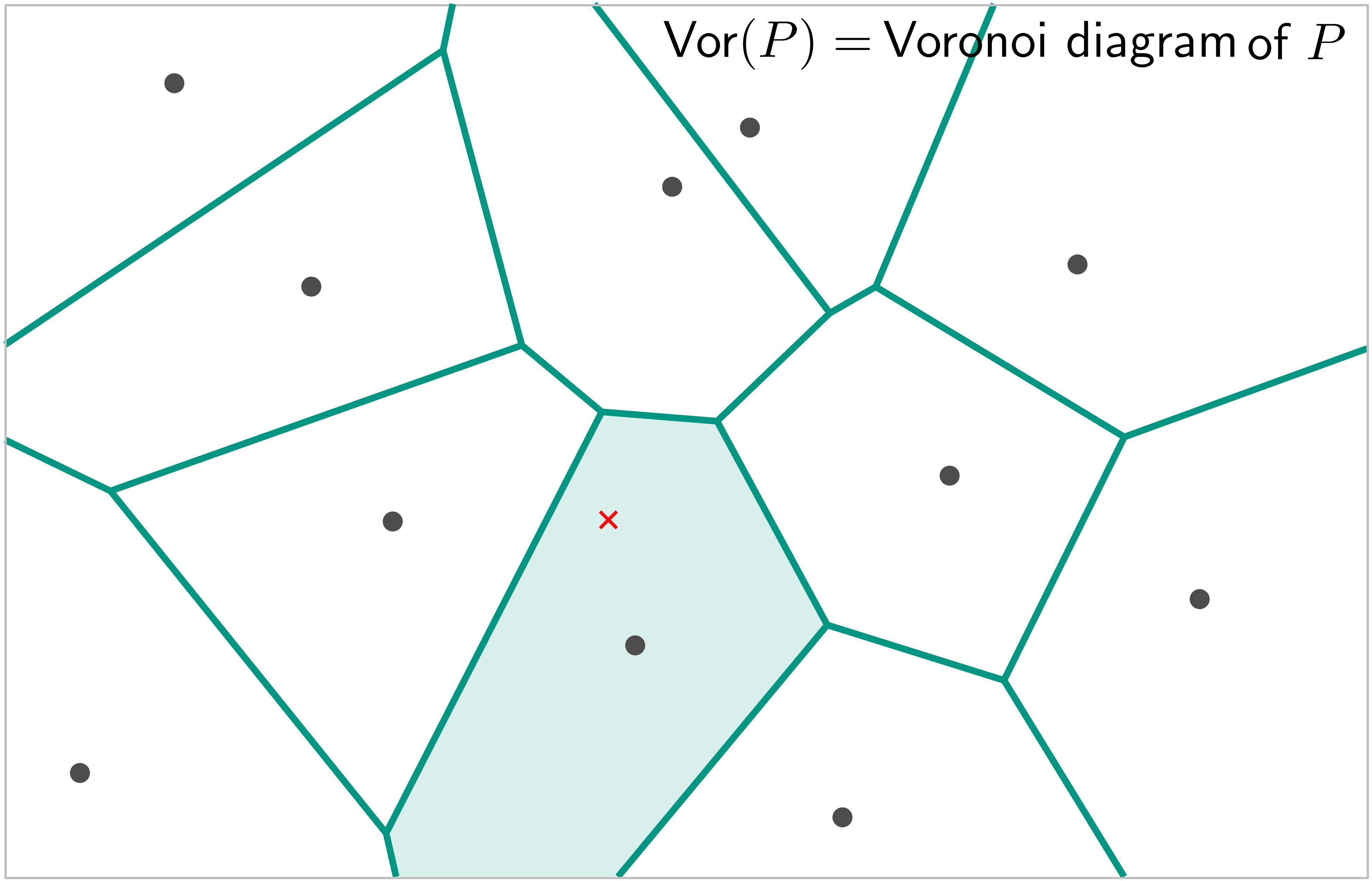
The Post Office Problem



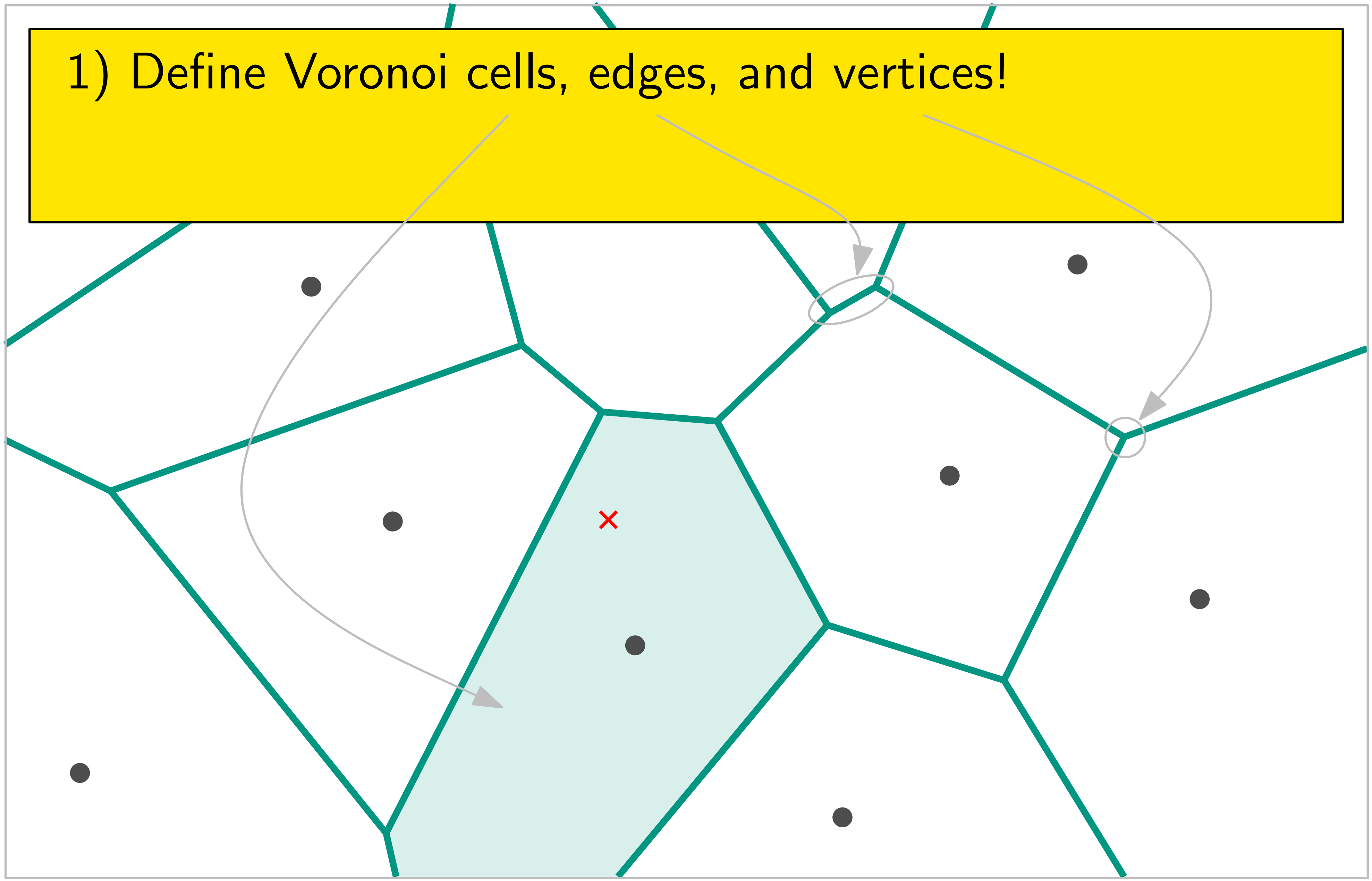
The Post Office Problem



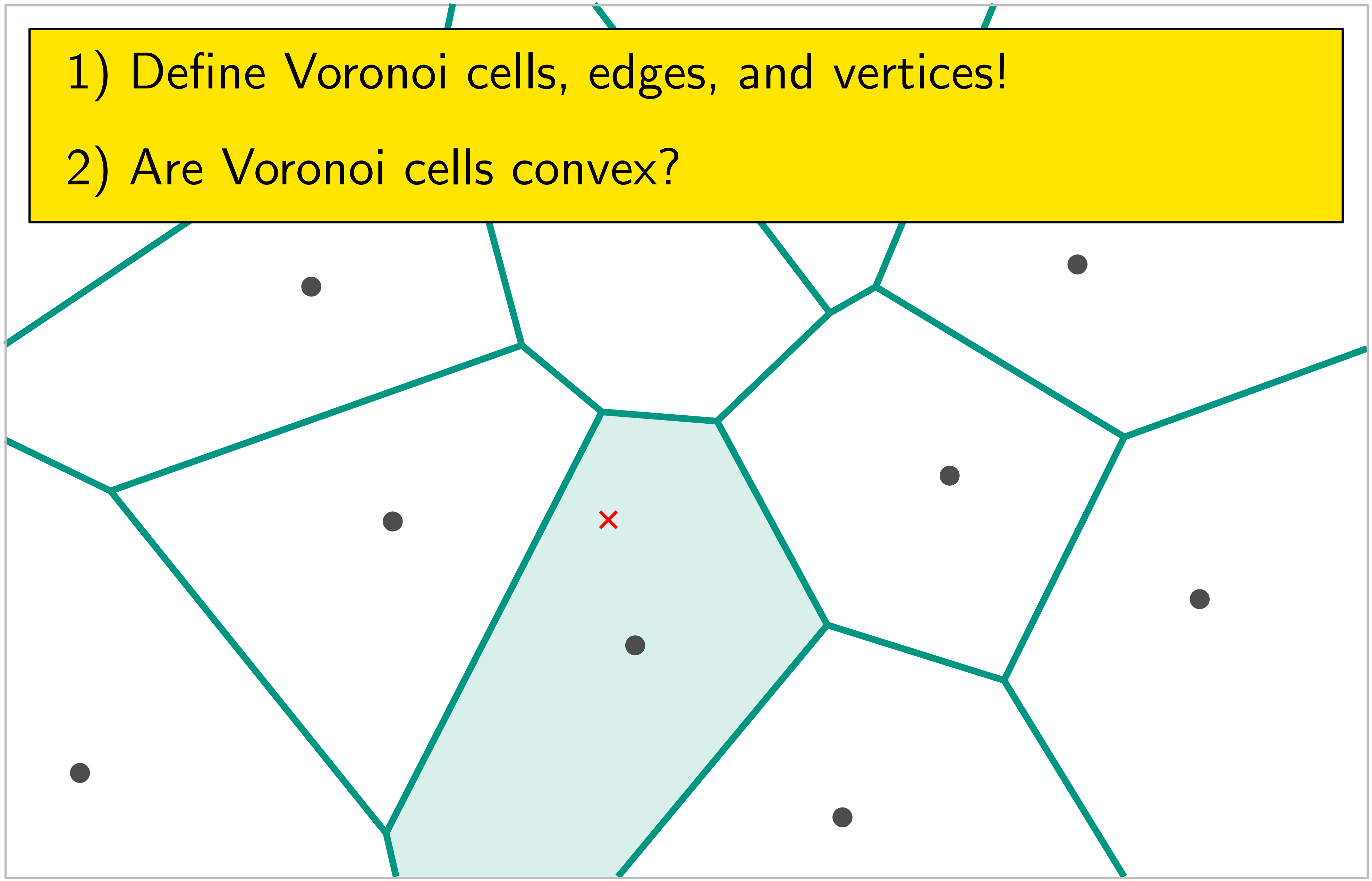
The Post Office Problem



The Post Office Problem



The Post Office Problem



1) Define Voronoi cells, edges, and vertices!

2) Are Voronoi cells convex?

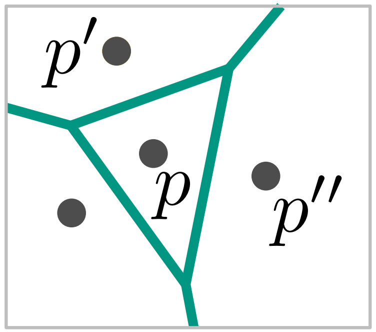
The Voronoi Diagram

Let P be a set of points in the plane and let $p, p', p'' \in P$.

The Voronoi Diagram

Let P be a set of points in the plane and let $p, p', p'' \in P$.

Voronoi Diagram

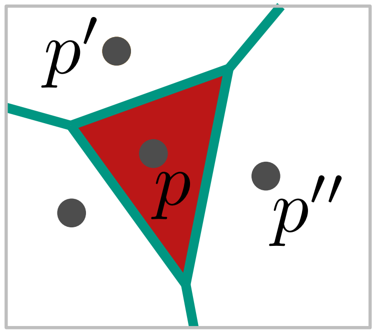


Vor(P)

The Voronoi Diagram

Let P be a set of points in the plane and let $p, p', p'' \in P$.

Voronoi Diagram

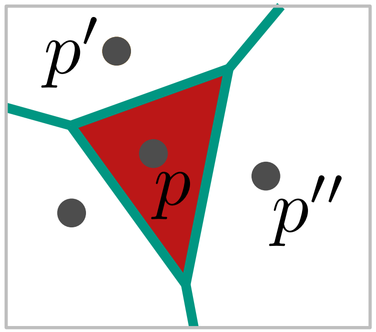


$\text{Vor}(P)$

The Voronoi Diagram

Let P be a set of points in the plane and let $p, p', p'' \in P$.

Voronoi Diagram



$\text{Vor}(P)$

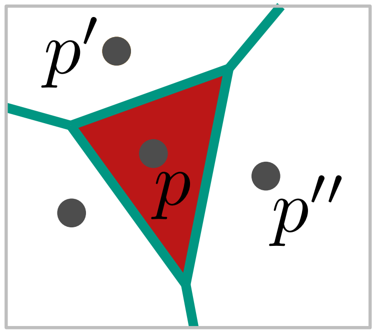
- Voronoi cell

$\mathcal{V}(\{p\}) =$

The Voronoi Diagram

Let P be a set of points in the plane and let $p, p', p'' \in P$.

Voronoi Diagram



$\mathrm{Vor}(P)$

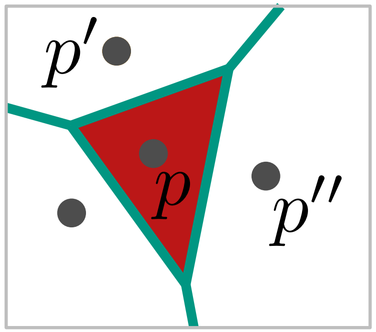
- Voronoi cell

$\mathcal{V}(\{p\}) = \mathcal{V}(p) =$

The Voronoi Diagram

Let P be a set of points in the plane and let $p, p', p'' \in P$.

Voronoi Diagram



$\text{Vor}(P)$

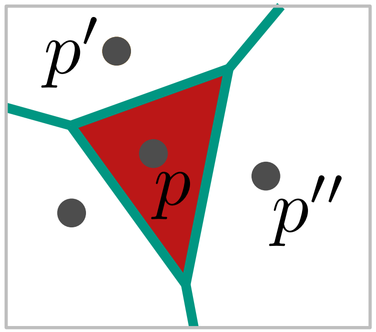
- Voronoi cell

$$\mathcal{V}(\{p\}) = \mathcal{V}(p) = \{x \in \mathbb{R}^2 : |xp| < |xq| \, \forall q \in P \setminus \{p\}\}$$

The Voronoi Diagram

Let P be a set of points in the plane and let $p, p', p'' \in P$.

Voronoi Diagram



$\text{Vor}(P)$

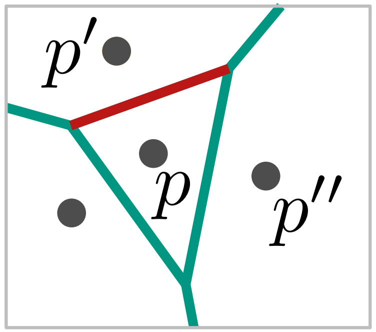
- Voronoi cell

$$\mathcal{V}(\{p\}) = \mathcal{V}(p) = \{x \in \mathbb{R}^2 : |xp| < |xq| \; \forall q \in P \setminus \{p\}\}$$
$$= \bigcap_{q \neq p} h(p, q)$$

The Voronoi Diagram

Let P be a set of points in the plane and let $p, p', p'' \in P$.

Voronoi Diagram



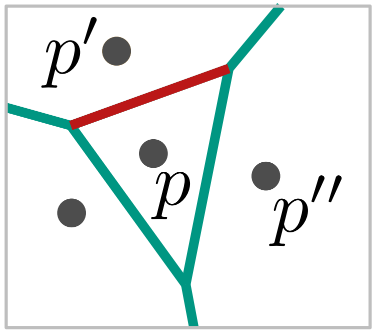
$\text{Vor}(P)$

- Voronoi cell

$$\mathcal{V}(\{p\}) = \mathcal{V}(p) = \{x \in \mathbb{R}^2 : |xp| < |xq| \; \forall q \in P \setminus \{p\}\}$$
$$= \bigcap_{q \neq p} h(p, q)$$

The Voronoi Diagram

Let P be a set of points in the plane and let $p, p', p'' \in P$.

Voronoi Diagram  Vor(P)

- Voronoi cell

$$\mathcal{V}(\{p\}) = \mathcal{V}(p) = \{x \in \mathbb{R}^2 : |xp| < |xq| \, \forall q \in P \setminus \{p\}\}$$
$$= \bigcap_{q \neq p} h(p, q)$$

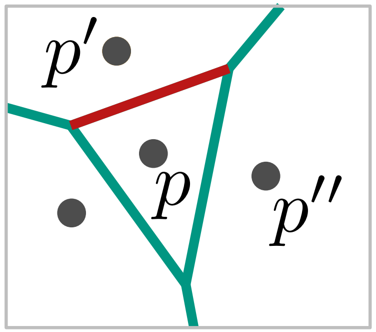
- Voronoi edges

$$\mathcal{V}(\{p, p'\}) =$$

The Voronoi Diagram

Let P be a set of points in the plane and let $p, p', p'' \in P$.

Voronoi Diagram



$\text{Vor}(P)$

- Voronoi cell

$$\mathcal{V}(\{p\}) = \mathcal{V}(p) = \{x \in \mathbb{R}^2 : |xp| < |xq| \, \forall q \in P \setminus \{p\}\}$$
$$= \bigcap_{q \neq p} h(p, q)$$

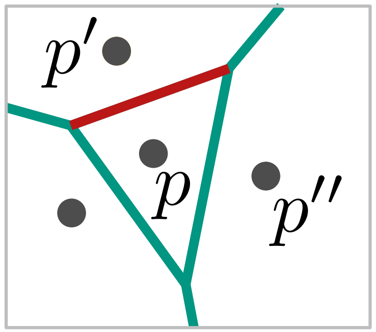
- Voronoi edges

$$\mathcal{V}(\{p, p'\}) = \{x : |xp| = |xp'| \text{ and } |xp| < |xq| \;\; \forall q \neq p, p'\}$$

The Voronoi Diagram

Let P be a set of points in the plane and let $p, p', p'' \in P$.

Voronoi Diagram



$\text{Vor}(P)$

- Voronoi cell

$$\mathcal{V}(\{p\}) = \mathcal{V}(p) = \{x \in \mathbb{R}^2 : |xp| < |xq| \,\forall q \in P \setminus \{p\}\}$$
$$= \bigcap_{q \neq p} h(p, q)$$

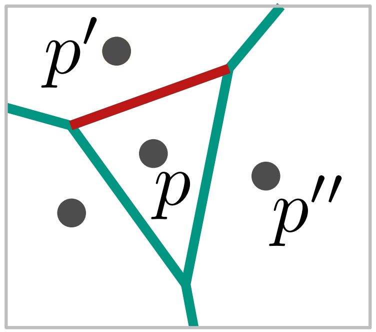
- Voronoi edges

$$\mathcal{V}(\{p, p'\}) = \{x : |xp| = |xp'| \text{ and } |xp| < |xq| \;\; \forall q \neq p, p'\}$$
$$= \partial\mathcal{V}(p) \cap \partial\mathcal{V}(p')$$

The Voronoi Diagram

Let P be a set of points in the plane and let $p, p', p'' \in P$.

Voronoi Diagram



$\text{Vor}(P)$

- Voronoi cell

$$\mathcal{V}(\{p\}) = \mathcal{V}(p) = \{x \in \mathbb{R}^2 : |xp| < |xq| \, \forall q \in P \setminus \{p\}\}$$
$$= \bigcap_{q \neq p} h(p, q)$$

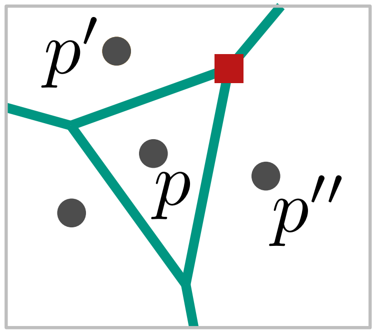
- Voronoi edges

$$\mathcal{V}(\{p, p'\}) = \{x : |xp| = |xp'| \text{ and } |xp| < |xq| \ \ \forall q \neq p, p'\}$$
$$= \text{int}(\,\partial\mathcal{V}(p) \cap \partial\mathcal{V}(p')), \text{ d.h. without endpoints}$$

The Voronoi Diagram

Let P be a set of points in the plane and let $p, p', p'' \in P$.

Voronoi Diagram



$\mathrm{Vor}(P)$

- Voronoi cell

$$\mathcal{V}(\{p\}) = \mathcal{V}(p) = \{x \in \mathbb{R}^2 : |xp| < |xq| \, \forall q \in P \setminus \{p\}\}$$
$$= \bigcap_{q \neq p} h(p,q)$$

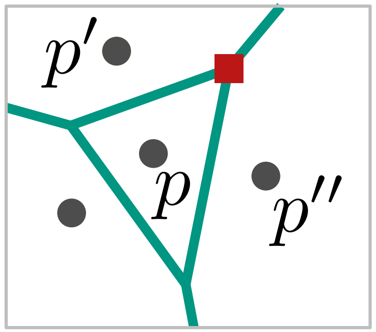
- Voronoi edges

$$\mathcal{V}(\{p,p'\}) = \{x : |xp| = |xp'| \text{ and } |xp| < |xq| \ \ \forall q \neq p, p'\}$$
$$= \text{int}(\partial\mathcal{V}(p) \cap \partial\mathcal{V}(p')), \text{ d.h. without endpoints}$$

The Voronoi Diagram

Let P be a set of points in the plane and let $p, p', p'' \in P$.

Voronoi Diagram



$\text{Vor}(P)$

- Voronoi cell

$$\mathcal{V}(\{p\}) = \mathcal{V}(p) = \{x \in \mathbb{R}^2 : |xp| < |xq| \, \forall q \in P \setminus \{p\}\}$$
$$= \bigcap_{q \neq p} h(p, q)$$

- Voronoi edges

$$\mathcal{V}(\{p, p'\}) = \{x : |xp| = |xp'| \text{ and } |xp| < |xq| \quad \forall q \neq p, p'\}$$
$$= \text{int}(\,\partial\mathcal{V}(p) \cap \partial\mathcal{V}(p')), \text{ d.h. without endpoints}$$

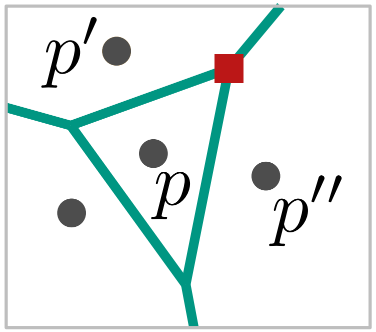
- Voronoi vertices

$\mathcal{V}(\{p, p', p''\})$

The Voronoi Diagram

Let P be a set of points in the plane and let $p, p', p'' \in P$.

Voronoi Diagram



$\mathrm{Vor}(P)$

- Voronoi cell

$$\mathcal{V}(\{p\}) = \mathcal{V}(p) = \{x \in \mathbb{R}^2 : |xp| < |xq| \; \forall q \in P \setminus \{p\}\}$$
$$= \bigcap_{q \neq p} h(p, q)$$

- Voronoi edges

$$\mathcal{V}(\{p, p'\}) = \{x : |xp| = |xp'| \text{ and } |xp| < |xq| \;\; \forall q \neq p, p'\}$$
$$= \text{int}(\,\partial\mathcal{V}(p) \cap \partial\mathcal{V}(p')), \text{ d.h. without endpoints}$$

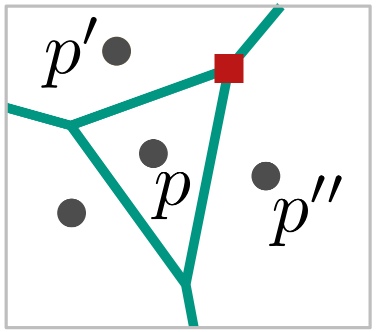
- Voronoi vertices

$$\mathcal{V}(\{p, p', p''\}) = \partial\mathcal{V}(p) \cap \partial\mathcal{V}(p') \cap \partial\mathcal{V}(p'')$$

The Voronoi Diagram

Let P be a set of points in the plane and let $p, p', p'' \in P$.

Voronoi Diagram



$\text{Vor}(P)$

- Voronoi cell

$$\mathcal{V}(\{p\}) = \mathcal{V}(p) = \{x \in \mathbb{R}^2 : |xp| < |xq| \, \forall q \in P \setminus \{p\}\}$$
$$= \bigcap_{q \neq p} h(p,q)$$

- Voronoi edges

$$\mathcal{V}(\{p, p'\}) = \{x : |xp| = |xp'| \text{ and } |xp| < |xq| \;\; \forall q \neq p, p'\}$$
$$= \text{int}(\,\partial\mathcal{V}(p) \cap \partial\mathcal{V}(p')), \text{ d.h. without endpoints}$$

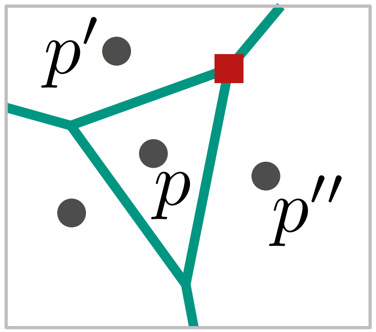
- Voronoi vertices

$$\mathcal{V}(\{p, p', p''\}) = \partial\mathcal{V}(p) \cap \partial\mathcal{V}(p') \cap \partial\mathcal{V}(p'')$$

The Voronoi Diagram

Let P be a set of points in the plane and let $p, p', p'' \in P$.

Voronoi Diagram



Vor(P) → subdivision

- Voronoi cell

$$\mathcal{V}(\{p\}) = \mathcal{V}(p) = \{x \in \mathbb{R}^2 : |xp| < |xq| \, \forall q \in P \setminus \{p\}\}$$
$$= \bigcap_{q \neq p} h(p,q)$$

- Voronoi edges

$$\mathcal{V}(\{p,p'\}) = \{x : |xp| = |xp'| \text{ and } |xp| < |xq| \;\; \forall q \neq p, p'\}$$
$$= \text{int}(\,\partial\mathcal{V}(p) \cap \partial\mathcal{V}(p')), \text{ d.h. without endpoints}$$

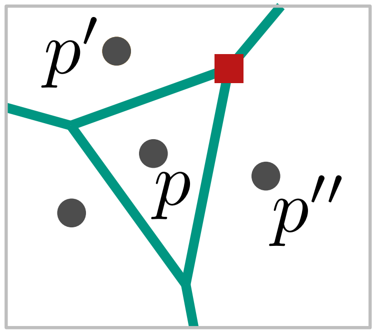
- Voronoi vertices

$$\mathcal{V}(\{p,p',p''\}) = \partial\mathcal{V}(p) \cap \partial\mathcal{V}(p') \cap \partial\mathcal{V}(p'')$$

The Voronoi Diagram

Let P be a set of points in the plane and let $p, p', p'' \in P$.

Voronoi Diagram



Vor(P) → subdivision
→ geometric graph

- Voronoi cell

$$\mathcal{V}(\{p\}) = \mathcal{V}(p) = \{x \in \mathbb{R}^2 : |xp| < |xq| \; \forall q \in P \setminus \{p\}\}$$
$$= \bigcap_{q \neq p} h(p, q)$$

- Voronoi edges

$$\mathcal{V}(\{p, p'\}) = \{x : |xp| = |xp'| \text{ and } |xp| < |xq| \quad \forall q \neq p, p'\}$$
$$= \text{int}\big(\partial\mathcal{V}(p) \cap \partial\mathcal{V}(p')\big), \text{ d.h. without endpoints}$$

- Voronoi vertices

$$\mathcal{V}(\{p, p', p''\}) = \partial\mathcal{V}(p) \cap \partial\mathcal{V}(p') \cap \partial\mathcal{V}(p'')$$

Properties

Theorem 1: Let $P \subset \mathbb{R}^2$ be a set of n points. If all points are collinear, then $\mathrm{Vor}(P)$ consists of $n-1$ parallel lines. Otherwise $\mathrm{Vor}(P)$ is connected and its edges are either segments or half lines.

Properties

Theorem 1: Let $P \subset \mathbb{R}^2$ be a set of n points. If all points are collinear, then $\mathrm{Vor}(P)$ consists of $n-1$ parallel lines. Otherwise $\mathrm{Vor}(P)$ is connected and its edges are either segments or half lines.

How many edges and vertices $\mathrm{Vor}(P)$ has?

Properties

Theorem 1: Let $P \subset \mathbb{R}^2$ be a set of n points. If all points are collinear, then $\mathrm{Vor}(P)$ consists of $n-1$ parallel lines. Otherwise $\mathrm{Vor}(P)$ is connected and its edges are either segments or half lines.

How many edges and vertices $\mathrm{Vor}(P)$ has?

Find a set P so that a cell in $\mathrm{Vor}(P)$ has linear complexity.

Properties

Theorem 1: Let $P \subset \mathbb{R}^2$ be a set of n points. If all points are collinear, then $\mathrm{Vor}(P)$ consists of $n-1$ parallel lines. Otherwise $\mathrm{Vor}(P)$ is connected and its edges are either segments or half lines.

How many edges and vertices $\mathrm{Vor}(P)$ has?

Find a set P so that a cell in $\mathrm{Vor}(P)$ has linear complexity.

Can this happen with (almost) all cells?

Properties

Theorem 1: Let $P \subset \mathbb{R}^2$ be a set of n points. If all points are collinear, then $\mathrm{Vor}(P)$ consists of $n-1$ parallel lines. Otherwise $\mathrm{Vor}(P)$ is connected and its edges are either segments or half lines.

How many edges and vertices $\mathrm{Vor}(P)$ has?

Find a set P so that a cell in $\mathrm{Vor}(P)$ has linear complexity.

Can this happen with (almost) all cells?

Theorem 2: Let $P \subset \mathbb{R}^2$ be a set on n points. $\mathrm{Vor}(P)$ has at most $2n-5$ nodes and $3n-6$ edges.

Properties

Theorem 1: Let $P \subset \mathbb{R}^2$ be a set of n points. If all points are collinear, then $\mathrm{Vor}(P)$ consists of $n-1$ parallel lines. Otherwise $\mathrm{Vor}(P)$ is connected and its edges are either segments or half lines.

How many edges and vertices $\mathrm{Vor}(P)$ has?

Find a set P so that a cell in $\mathrm{Vor}(P)$ has linear complexity.

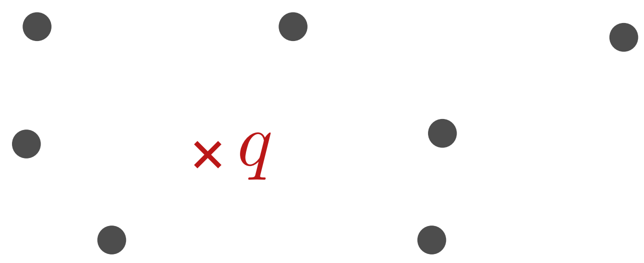
Can this happen with (almost) all cells?

Theorem 2: Let $P \subset \mathbb{R}^2$ be a set on n points. $\mathrm{Vor}(P)$ has at most $2n-5$ nodes and $3n-6$ edges.

Exercise!

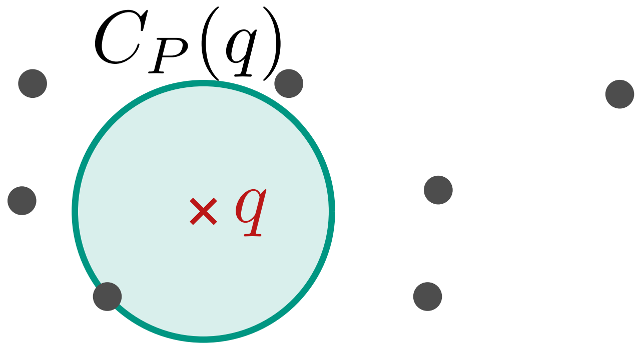
Characterization

Definition: Let q be a point. Define $C_P(q)$ to be the points in P that lie on the empty circle with center q.



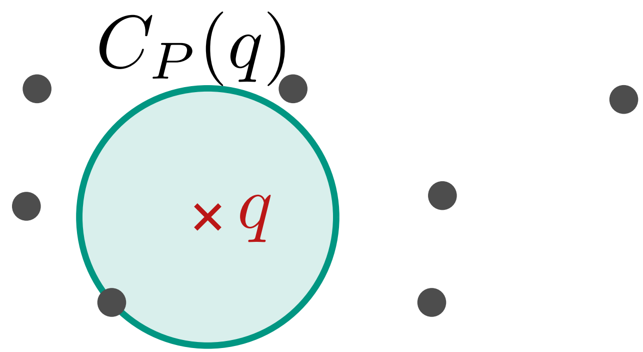
Characterization

Definition: Let q be a point. Define $C_P(q)$ to be the points in P that lie on the empty circle with center q.

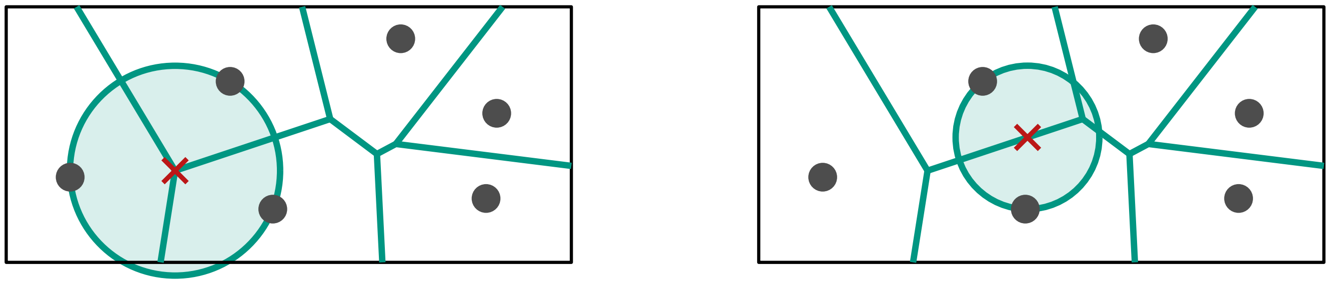


Characterization

Definition: Let q be a point. Define $C_P(q)$ to be the points in P that lie on the empty circle with center q.



Theorem 3:
- A point q is a Voronoi vertex $\Leftrightarrow |C_P(q) \cap P| \geq 3$,
- the bisector $b(p_i, p_j)$ defines a Voronoi edge $\Leftrightarrow \exists q \in b(p_i, p_j)$ with $C_P(q) \cap P = \{p_i, p_j\}$.



Computing $\mathrm{Vor}(P)$

How can we calculate $\mathrm{Vor}(P)$ with methods we already know?

Computing $\mathrm{Vor}(P)$

How can we calculate $\mathrm{Vor}(P)$ with methods we already know?

For each $p \in P$ is $\mathcal{V}(p) = \bigcap_{p' \neq p} h(p, p')$ is the intersection of $n - 1$ half planes.

Computing $\mathrm{Vor}(P)$

How can we calculate $\mathrm{Vor}(P)$ with methods we already know?

For each $p \in P$ is $\mathcal{V}(p) = \bigcap_{p' \neq p} h(p, p')$ is the intersection of $n-1$ half planes.

foreach $p \in P$ **do**
  compute $\mathcal{V}(p) = \bigcap_{p' \neq p} h(p, p')$

Computing $\mathrm{Vor}(P)$

How can we calculate $\mathrm{Vor}(P)$ with methods we already know?

For each $p \in P$ is $\mathcal{V}(p) = \bigcap_{p' \neq p} h(p, p')$ is the intersection of $n - 1$ half planes.

foreach $p \in P$ **do**

compute $\mathcal{V}(p) = \bigcap_{p' \neq p} h(p, p')$ $O(n \log n)$ [Lecture 4]

Computing $\mathrm{Vor}(P)$

How can we calculate $\mathrm{Vor}(P)$ with methods we already know?

For each $p \in P$ is $\mathcal{V}(p) = \bigcap_{p' \neq p} h(p, p')$ is the intersection of $n - 1$ half planes.

foreach $p \in P$ **do** $O(n^2 \log n)$
  compute $\mathcal{V}(p) = \bigcap_{p' \neq p} h(p, p')$ $O(n \log n)$ [Lecture 4]

Computing $\mathrm{Vor}(P)$

How can we calculate $\mathrm{Vor}(P)$ with methods we already know?

For each $p \in P$ is $\mathcal{V}(p) = \bigcap_{p' \neq p} h(p, p')$ is the intersection of $n-1$ half planes.

foreach $p \in P$ **do** $O(n^2 \log n)$
  compute $\mathcal{V}(p) = \bigcap_{p' \neq p} h(p, p')$ $O(n \log n)$ [Lecture 4]

Is $O(n^2 \log n)$ running time for a linear-size object necessary?

Computing $\mathrm{Vor}(P)$

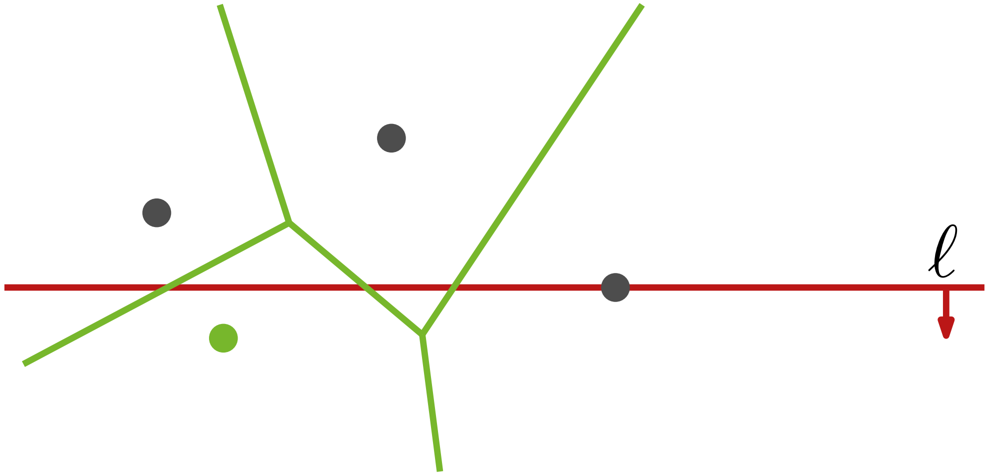
How can we calculate $\mathrm{Vor}(P)$ with methods we already know?

For each $p \in P$ is $\mathcal{V}(p) = \bigcap_{p' \neq p} h(p, p')$ is the intersection of $n - 1$ half planes.

foreach $p \in P$ **do** $O(n^2 \log n)$
compute $\mathcal{V}(p) = \bigcap_{p' \neq p} h(p, p')$ $O(n \log n)$ [Lecture 4]

Is $O(n^2 \log n)$ running time for a linear-size object necessary?

Idea 2: Sweep line



Computing $\mathrm{Vor}(P)$

How can we calculate $\mathrm{Vor}(P)$ with methods we already know?

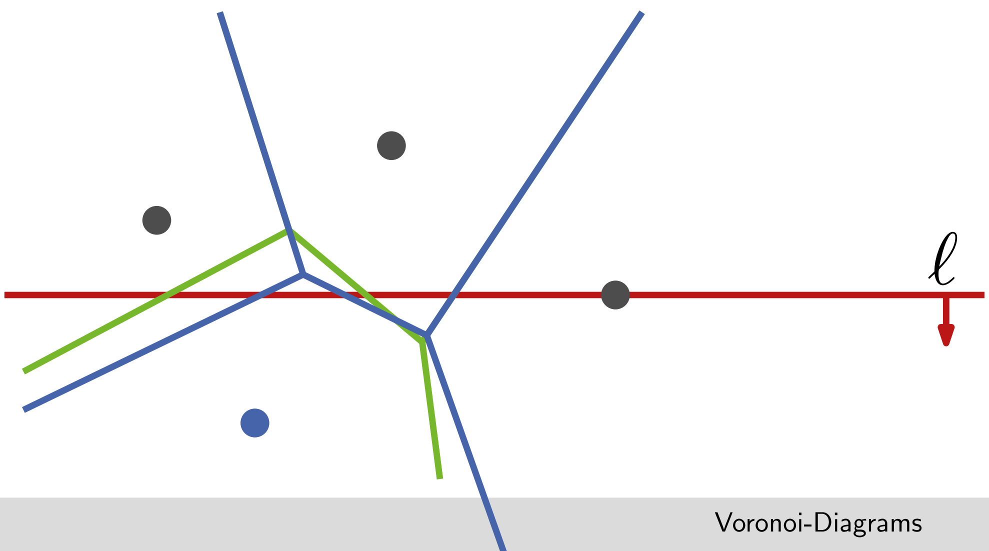
For each $p \in P$ is $\mathcal{V}(p) = \bigcap_{p' \neq p} h(p, p')$ is the intersection of $n-1$ half planes.

foreach $p \in P$ **do** $O(n^2 \log n)$
compute $\mathcal{V}(p) = \bigcap_{p' \neq p} h(p, p')$ $O(n \log n)$ [Lecture 4]

Is $O(n^2 \log n)$ running time for a linear-size object necessary?

Idea 2: Sweep line

Problem:
$\mathrm{Vor}(P)$ over ℓ depends on points under ℓ!



Computing $\mathrm{Vor}(P)$

How can we calculate $\mathrm{Vor}(P)$ with methods we already know?

For each $p \in P$ is $\mathcal{V}(p) = \bigcap_{p' \neq p} h(p, p')$ is the intersection of $n-1$ half planes.

foreach $p \in P$ **do** $O(n^2 \log n)$

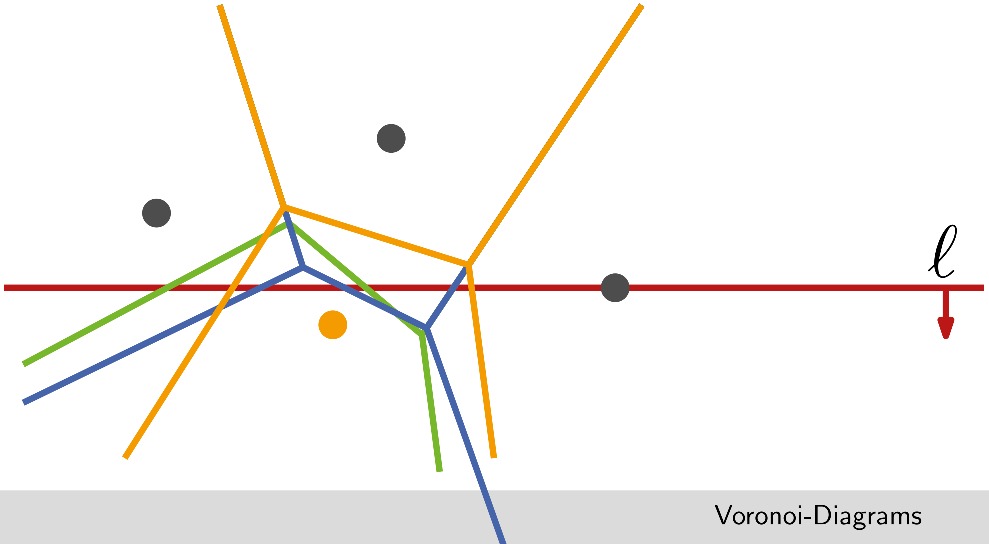
compute $\mathcal{V}(p) = \bigcap_{p' \neq p} h(p, p')$ $O(n \log n)$ [Lecture 4]

Is $O(n^2 \log n)$ running time for a linear-size object necessary?

Idea 2: Sweep line

Problem:

$\mathrm{Vor}(P)$ over ℓ depends on points under ℓ!



In the Direction of the Sweep Line

Obviously the intersection of $\mathrm{Vor}(P)$ and sweep line ℓ at the current time point is not known yet.

In the Direction of the Sweep Line

Obviously the intersection of $\mathrm{Vor}(P)$ and sweep line ℓ at the current time point is not known yet.

Instead, we store the part above ℓ that is already fixed!

In the Direction of the Sweep Line

Obviously the intersection of $\mathrm{Vor}(P)$ and sweep line ℓ at the current time point is not known yet.

Instead, we store the part above ℓ that is already fixed!

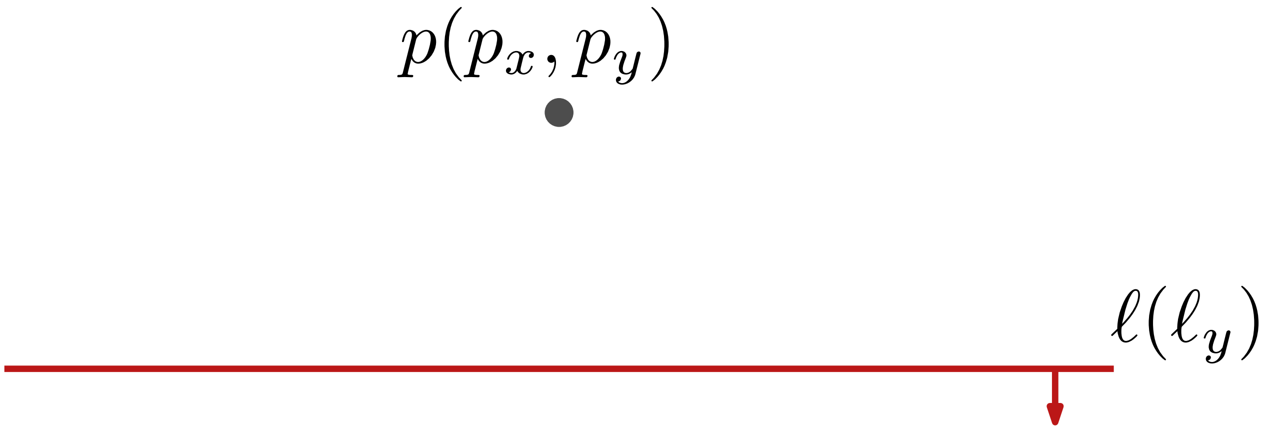
What does it consist of?

In the Direction of the Sweep Line

Obviously the intersection of $\mathrm{Vor}(P)$ and sweep line ℓ at the current time point is not known yet.

Instead, we store the part above ℓ that is already fixed!

What does it consist of?



In the Direction of the Sweep Line

Obviously the intersection of $\mathrm{Vor}(P)$ and sweep line ℓ at the current time point is not known yet.

Instead, we store the part above ℓ that is already fixed!

What does it consist of?

$$p(p_x, p_y)$$

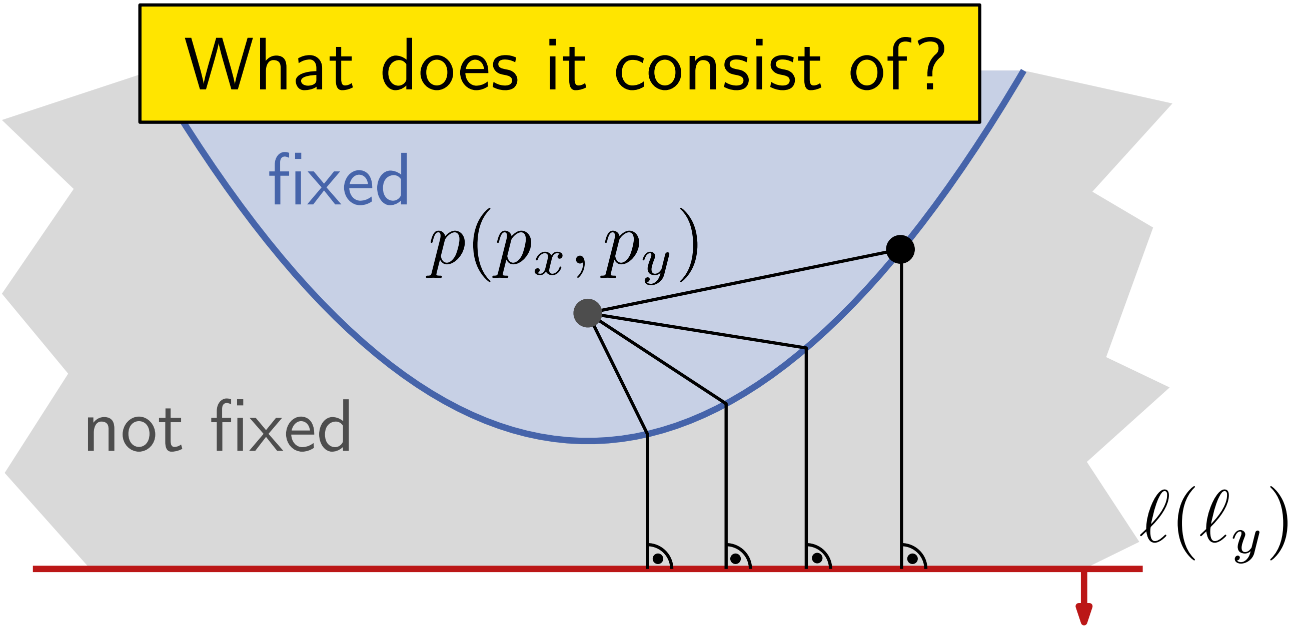
Points closer to p than ℓ are already processed.

$$\ell(\ell_y)$$

In the Direction of the Sweep Line

Obviously the intersection of $\mathrm{Vor}(P)$ and sweep line ℓ at the current time point is not known yet.

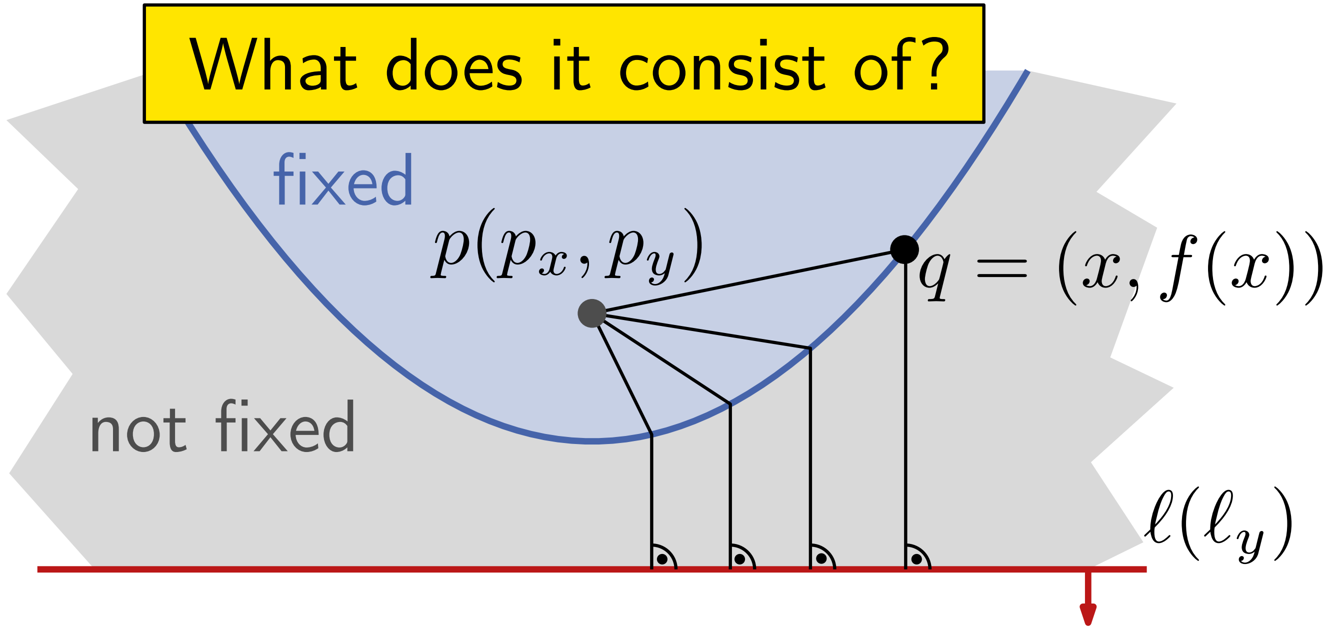
Instead, we store the part above ℓ that is already fixed!



In the Direction of the Sweep Line

Obviously the intersection of $\mathrm{Vor}(P)$ and sweep line ℓ at the current time point is not known yet.

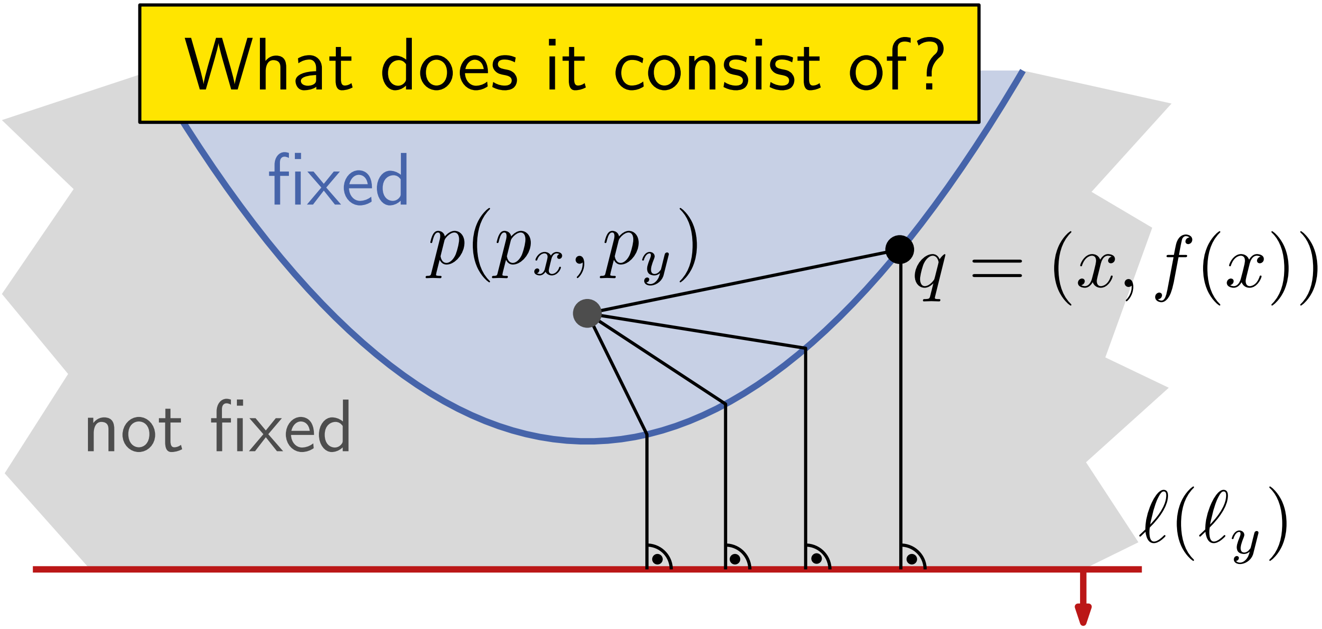
Instead, we store the part above ℓ that is already fixed!



In the Direction of the Sweep Line

Obviously the intersection of $\mathrm{Vor}(P)$ and sweep line ℓ at the current time point is not known yet.

Instead, we store the part above ℓ that is already fixed!



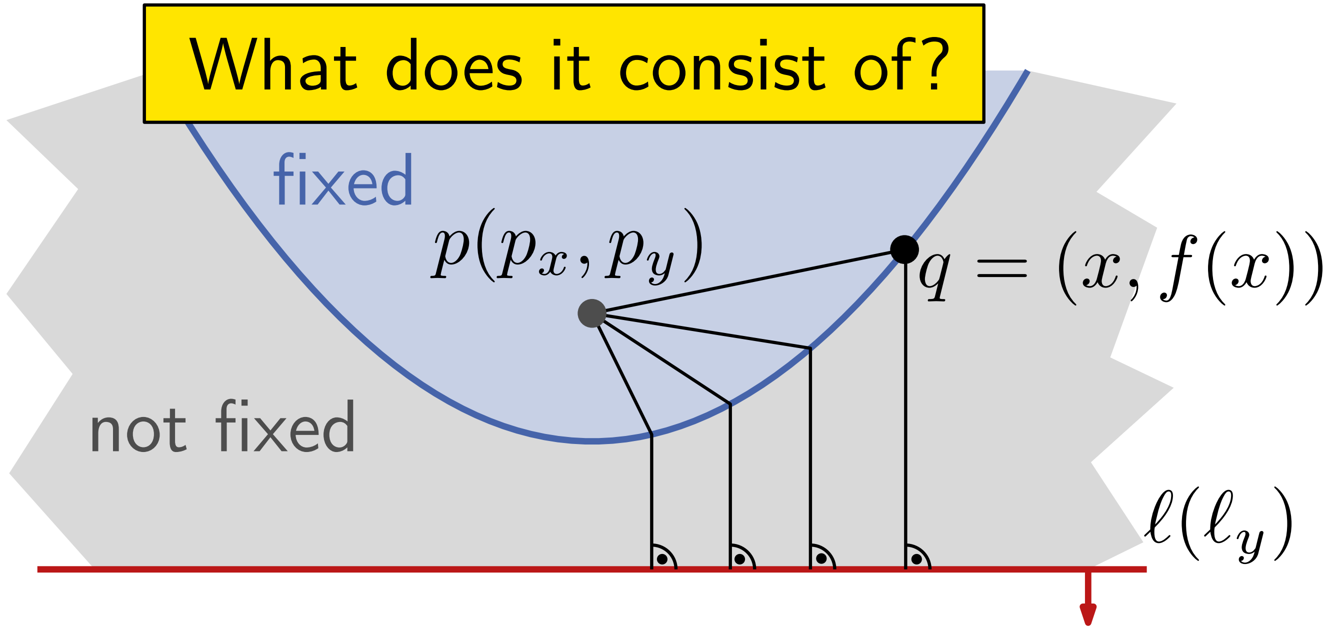
Enforcing the equality $|pq| = |q\ell|$ gives

$$f(x) = \frac{1}{2(p_y - \ell_y)}(x - p_x)^2 + \frac{p_y + \ell_y}{2}$$

In the Direction of the Sweep Line

Obviously the intersection of $\text{Vor}(P)$ and sweep line ℓ at the current time point is not known yet.

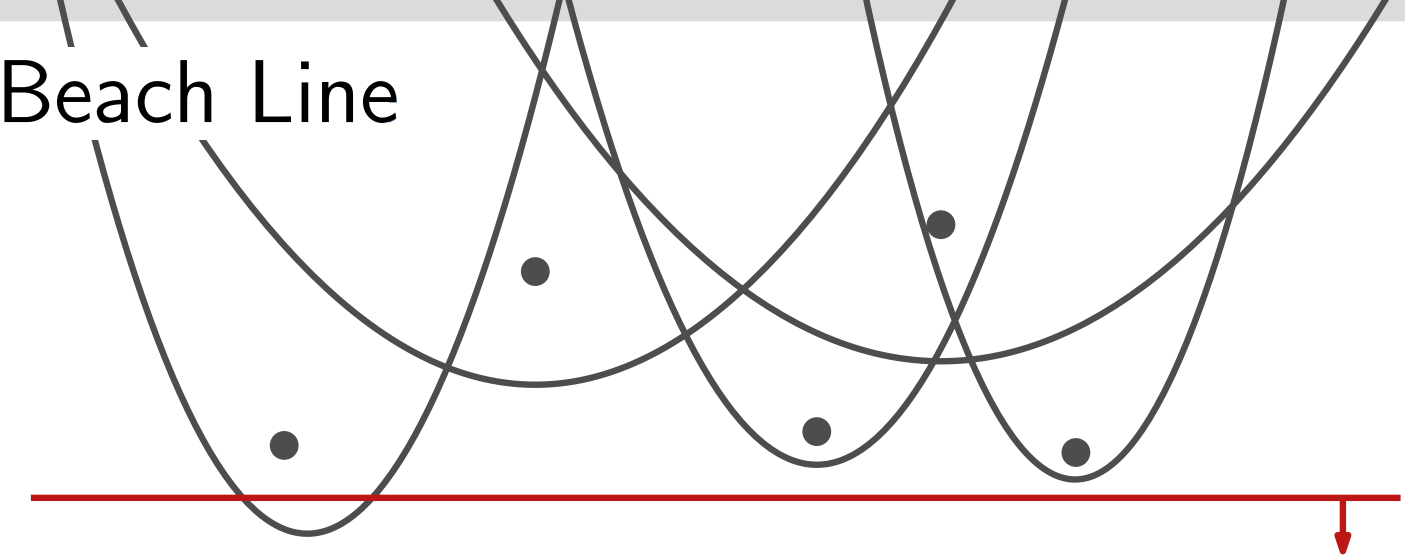
Instead, we store the part above ℓ that is already fixed!



Enforcing the equality $|pq| = |q\ell|$ gives

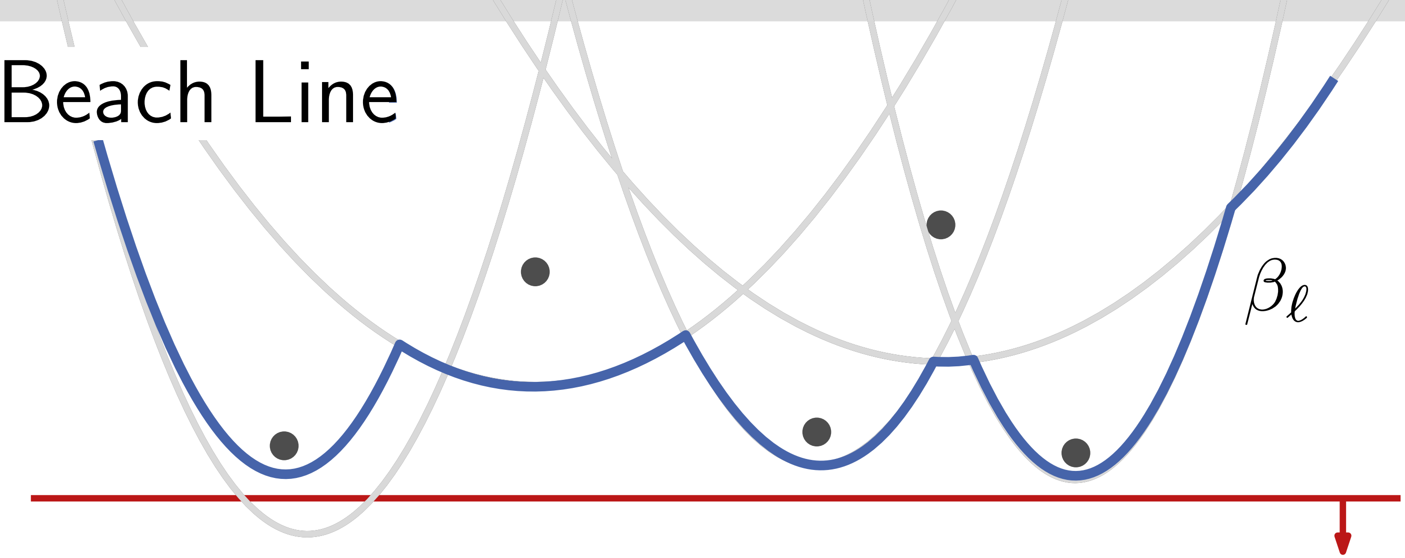
$$f_p^\ell(x) = f(x) = \frac{1}{2(p_y - \ell_y)}(x - p_x)^2 + \frac{p_y + \ell_y}{2}$$

The Beach Line



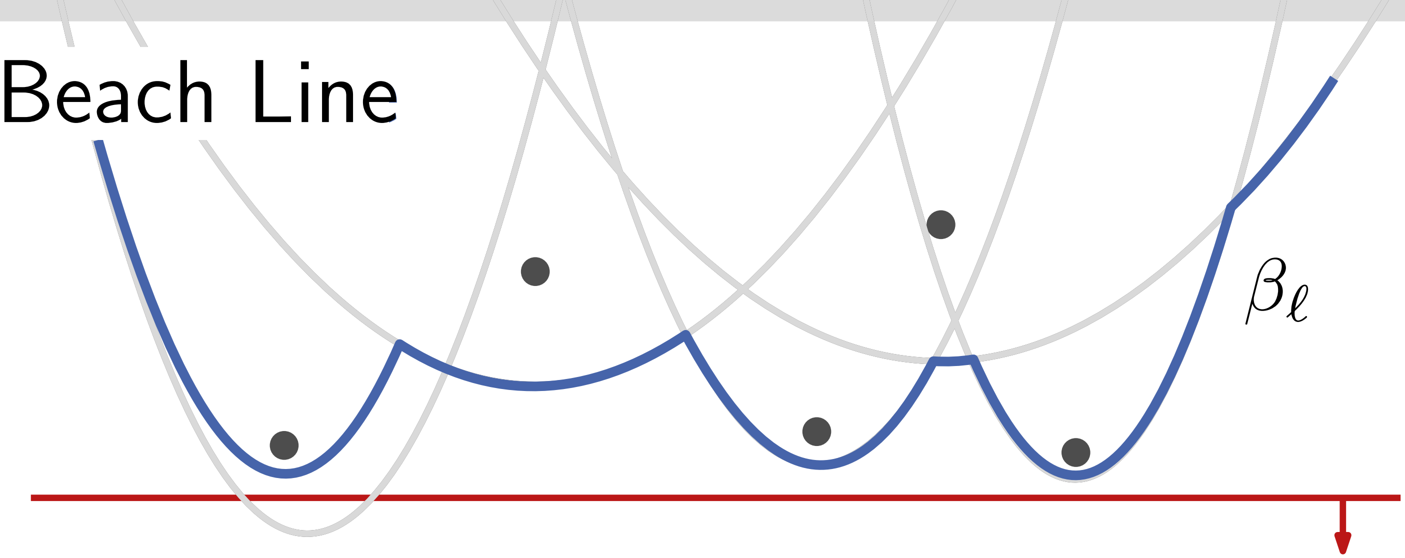
Definition: The **beach line** β_ℓ is the lower envelope of parabolas f_p^ℓ for the points already found.

The Beach Line



Definition: The **beach line** β_ℓ is the lower envelope of parabolas f_p^ℓ for the points already found.

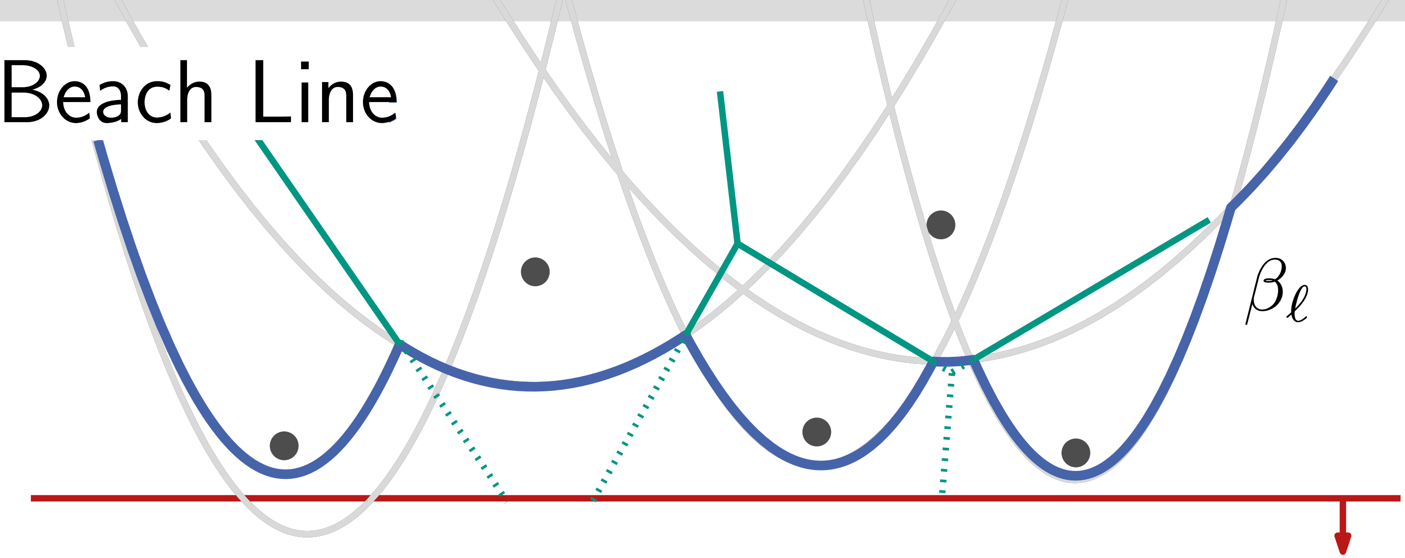
The Beach Line



Definition: The **beach line** β_ℓ is the lower envelope of parabolas f_p^ℓ for the points already found.

What does it have to do with $\mathrm{Vor}(P)$?

The Beach Line



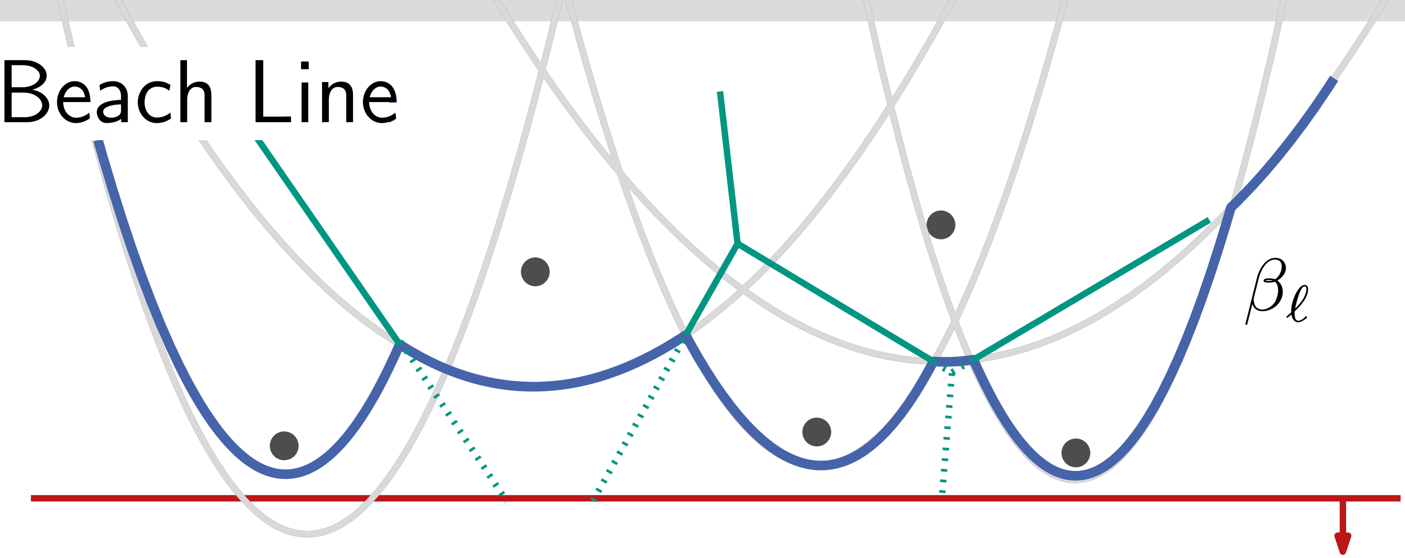
Definition: The **beach line** β_ℓ is the lower envelope of parabolas f_p^ℓ for the points already found.

What does it have to do with $\mathrm{Vor}(P)$?

Obs.:

- The beach line is x-monotone
- Intersection points in the beach line lie on Voronoi edges
- **As the sweep-line goes down**, intersection points run along $\mathrm{Vor}(P)$

The Beach Line



Definition: The **beach line** β_ℓ is the lower envelope of parabolas f_p^ℓ for the points already found.

What does it have to do with $\mathrm{Vor}(P)$?

Obs.:
- The beach line is x-monotone
- Intersection points in the beach line lie on Voronoi edges
- **As the sweep-line goes down**, intersection points run along $\mathrm{Vor}(P)$

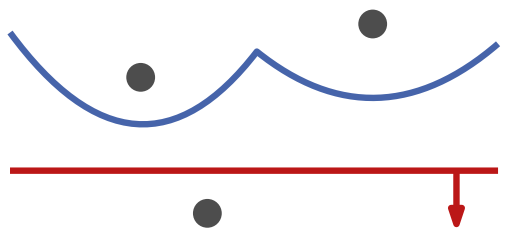
Goal: Store (implicit) contour β_ℓ instead of $\mathrm{Vor}(P) \cap \ell$

Before we proceed...

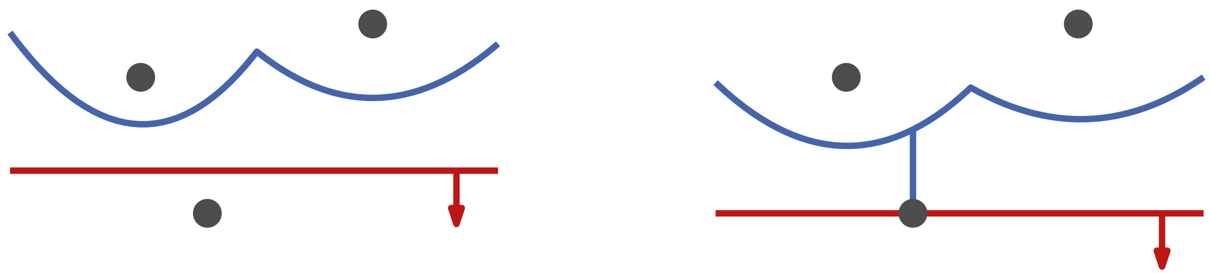
Demo

http://www.diku.dk/hjemmesider/studerende/duff/Fortune/

Point Events

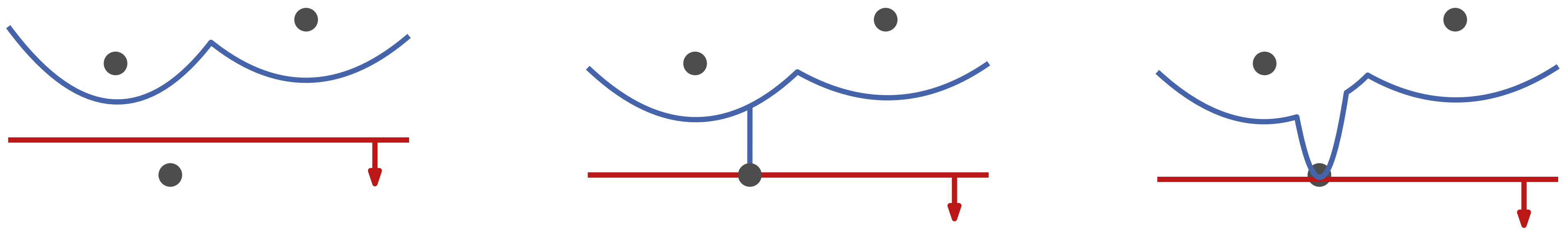


Point Events



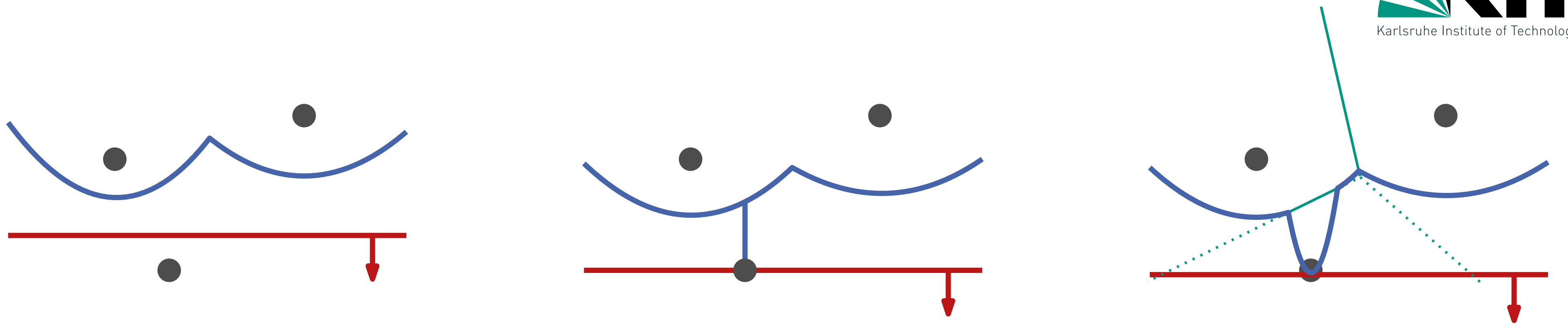
- If ℓ meets a point, then a new parabola is added to β_ℓ

Point Events



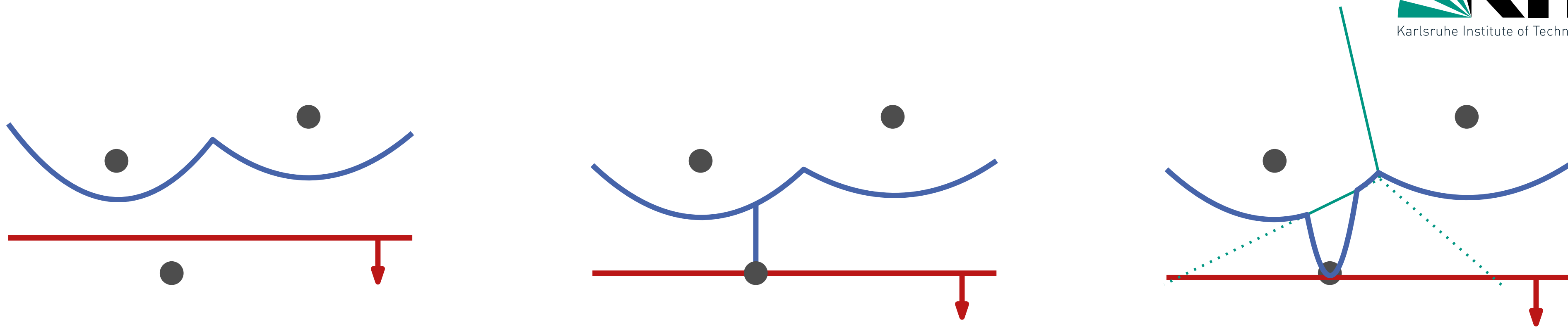
- If ℓ meets a point, then a new parabola is added to β_ℓ

Point Events



- If ℓ meets a point, then a new parabola is added to β_ℓ
- The two intersection points generate a new part of an edge.

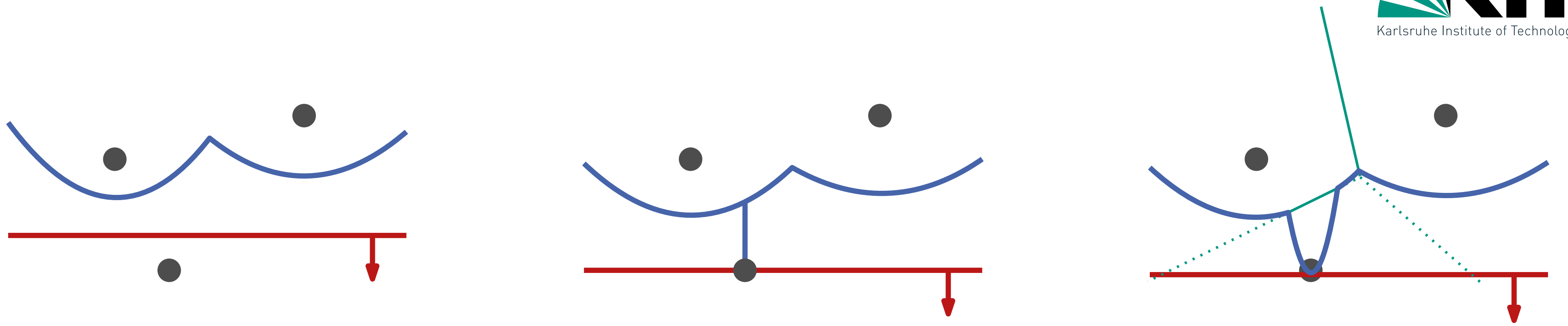
Point Events



- If ℓ meets a point, then a new parabola is added to β_ℓ
- The two intersection points generate a new part of an edge.

Lemma 1: New arcs on β_ℓ only come from point events.

Point Events

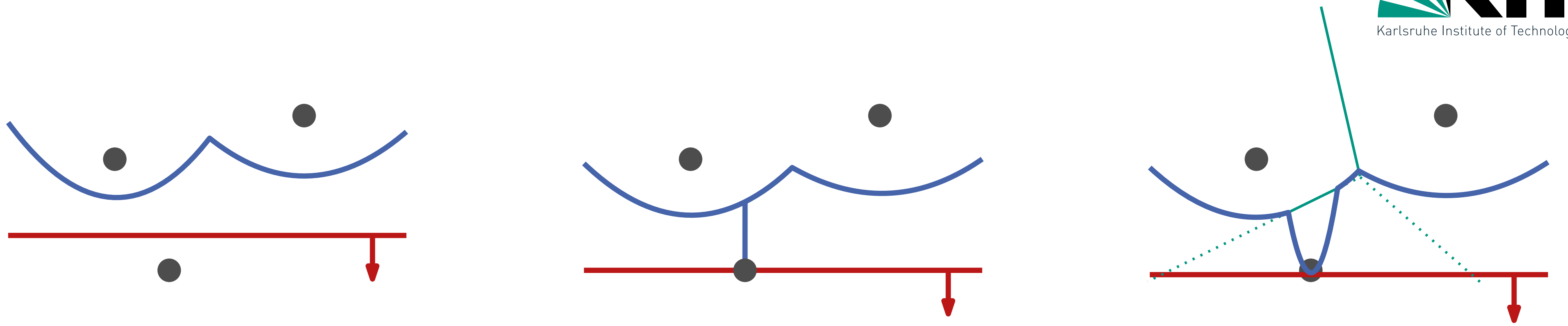


- If ℓ meets a point, then a new parabola is added to β_ℓ
- The two intersection points generate a new part of an edge.

Lemma 1: New arcs on β_ℓ only come from point events.

Corollary: β_ℓ is at most $2n - 1$ parabolic arcs

Point Events



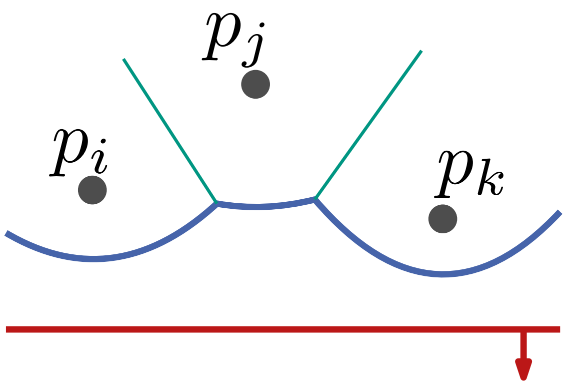
- If ℓ meets a point, then a new parabola is added to β_ℓ
- The two intersection points generate a new part of an edge.

Lemma 1: New arcs on β_ℓ only come from point events.

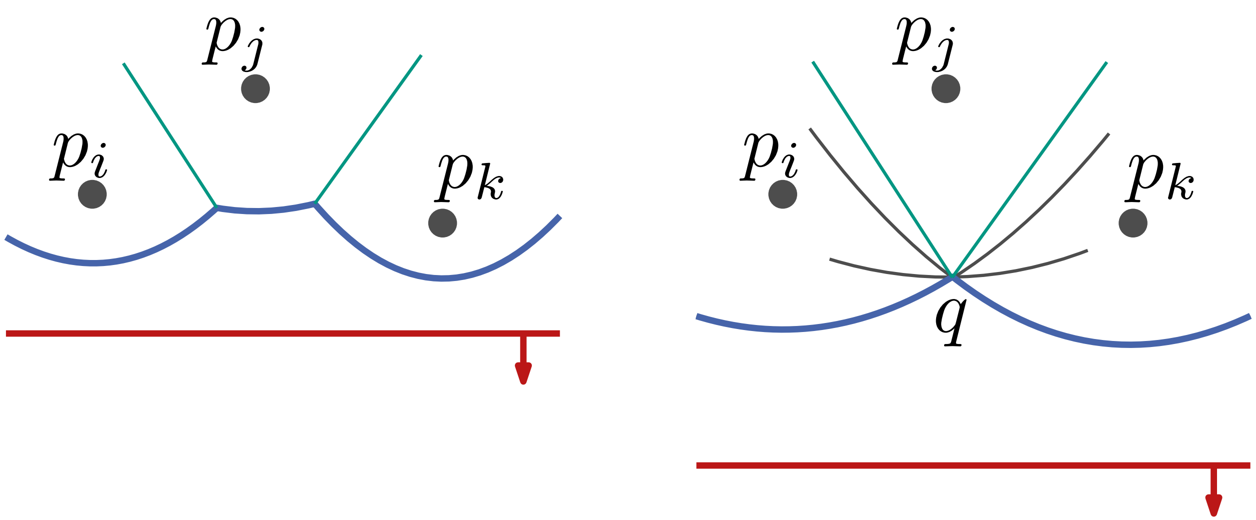
Corollary: β_ℓ is at most $2n - 1$ parabolic arcs

More about this in exercises...

Circle Events

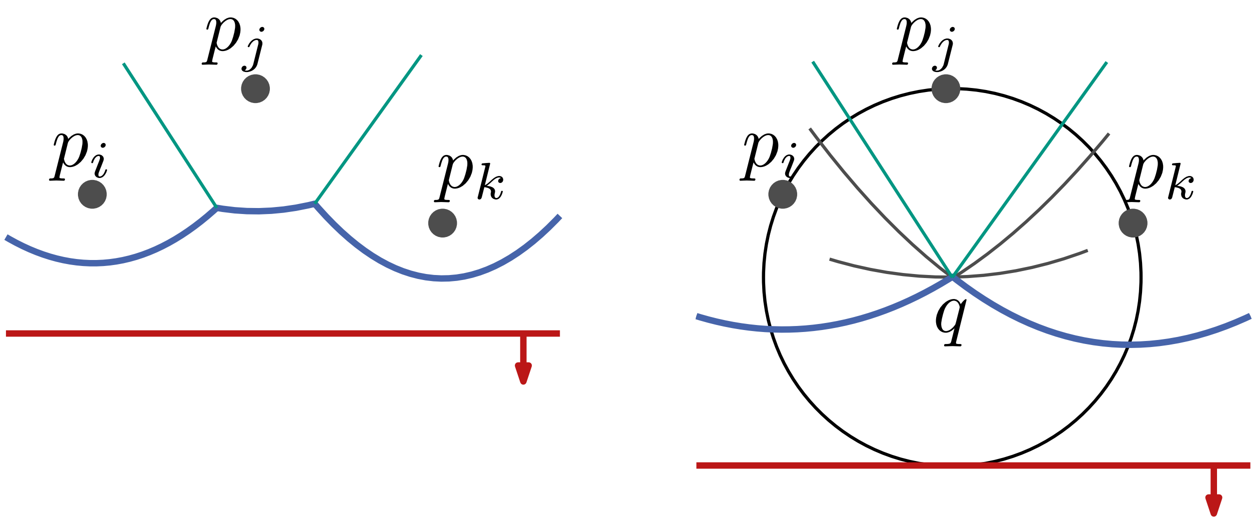


Circle Events



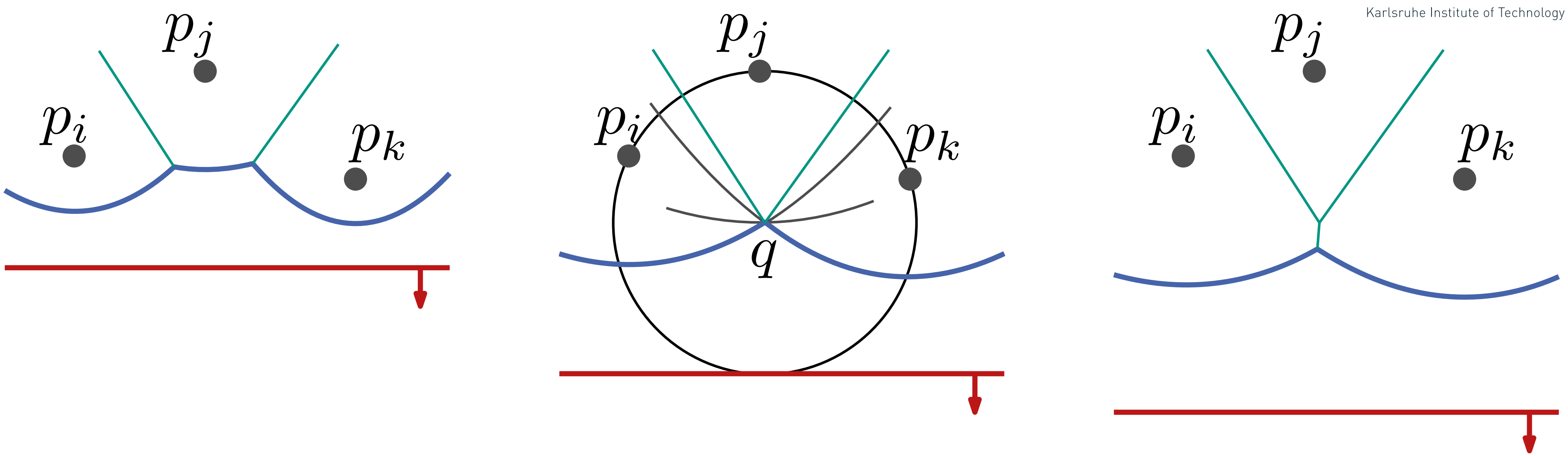
- A parabolic arc disappears from $f^{\ell}_{p_i}, f^{\ell}_{p_j}, f^{\ell}_{p_k}$ at a common point q

Circle Events



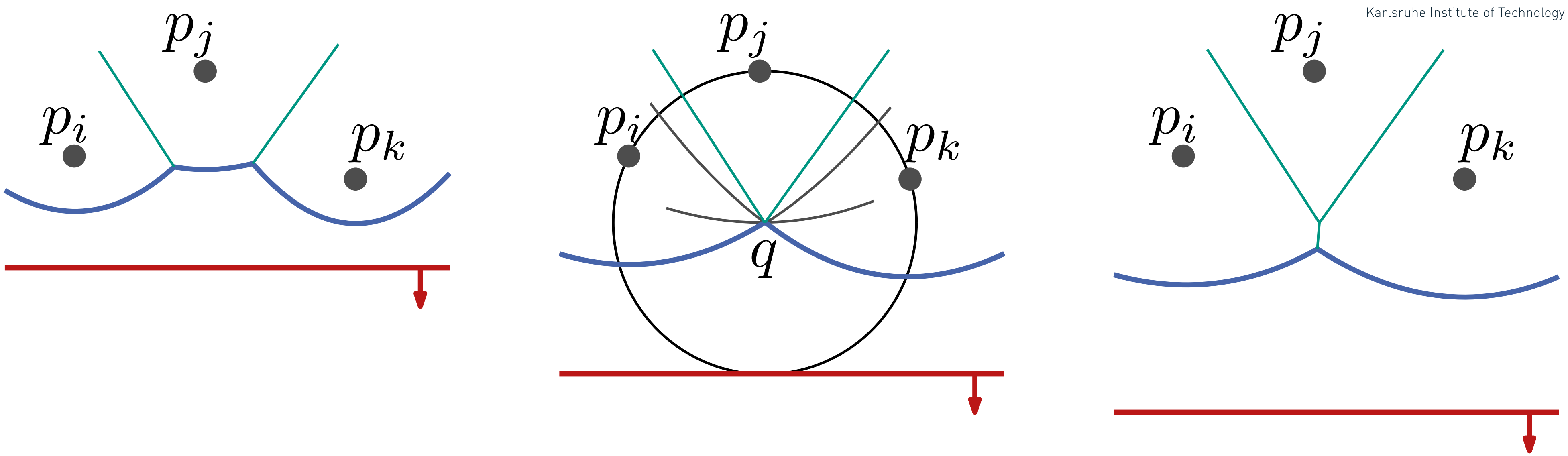
- A parabolic arc disappears from $f^{\ell}_{p_i}, f^{\ell}_{p_j}, f^{\ell}_{p_k}$ at a common point q
- The circle $C_P(q)$ that touches p_i, p_j, p_k and ℓ
  $\Rightarrow$ q is a Voronoi vertex

Circle Events



- A parabolic arc disappears from $f^\ell_{p_i}, f^\ell_{p_j}, f^\ell_{p_k}$ at a common point q
- The circle $C_P(q)$ that touches p_i, p_j, p_k and ℓ
  $\Rightarrow$ q is a Voronoi vertex

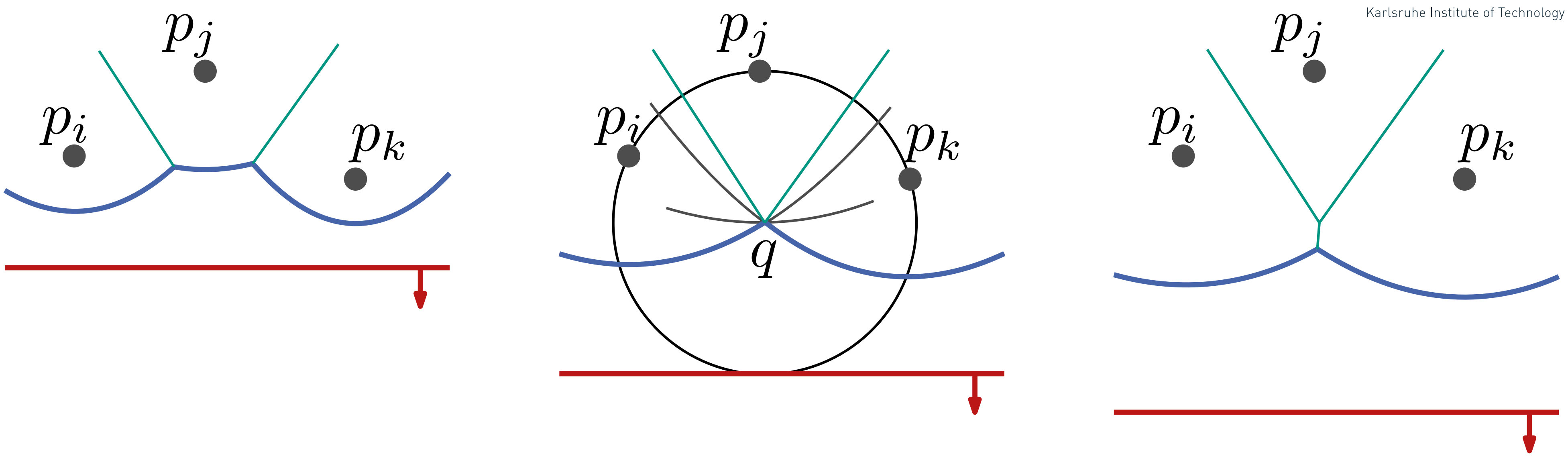
Circle Events



- A parabolic arc disappears from $f^{\ell}_{p_i}, f^{\ell}_{p_j}, f^{\ell}_{p_k}$ at a common point q
- The circle $C_P(q)$ that touches p_i, p_j, p_k and ℓ $\Rightarrow q$ is a Voronoi vertex

Def.: The lowest point of the circle defined by three points of consecutive arcs in β_ℓ defines a **circle event**.

Circle Events

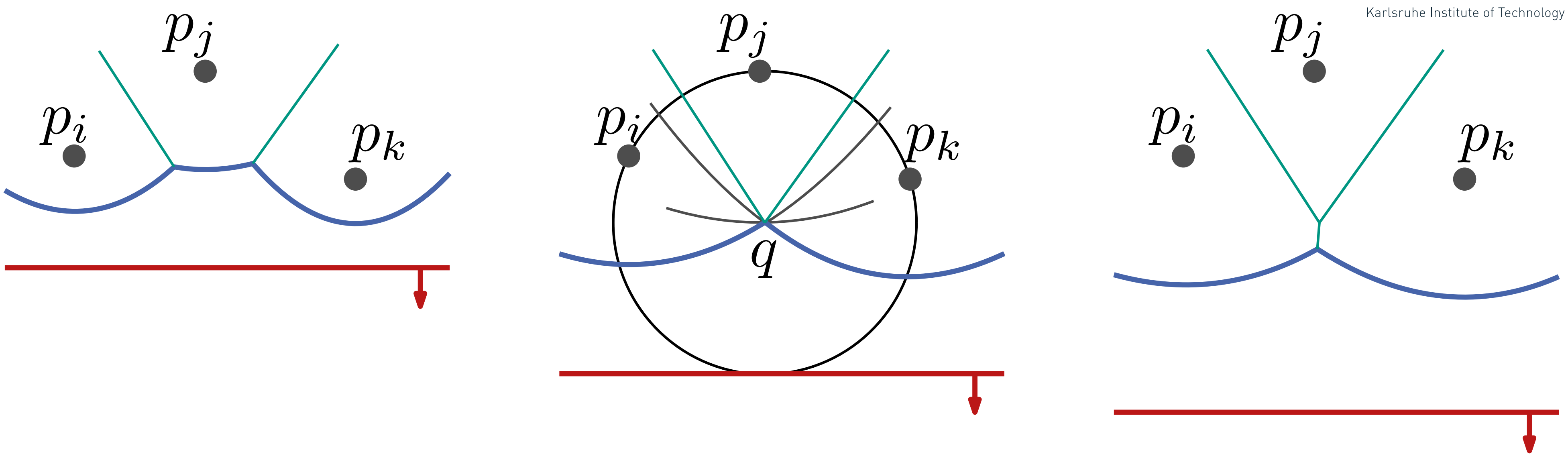


- A parabolic arc disappears from $f^\ell_{p_i}, f^\ell_{p_j}, f^\ell_{p_k}$ at a common point q
- The circle $C_P(q)$ that touches p_i, p_j, p_k and ℓ
  $\Rightarrow q$ is a Voronoi vertex

Def.: The lowest point of the circle defined by three points of consecutive arcs in β_ℓ defines a **circle event**.

Lemma 2: Arcs of β_ℓ only disappear through circle events.

Circle Events



- A parabolic arc disappears from $f^\ell_{p_i}, f^\ell_{p_j}, f^\ell_{p_k}$ at a common point q
- The circle $C_P(q)$ that touches p_i, p_j, p_k and ℓ
  $\Rightarrow$ q is a Voronoi vertex

Def.: The lowest point of the circle defined by three points of consecutive arcs in β_ℓ defines a **circle event**.

Lemma 2: Arcs of β_ℓ only disappear through circle events.

Lemma 3: For each Voronoi vertex there is a circle event.

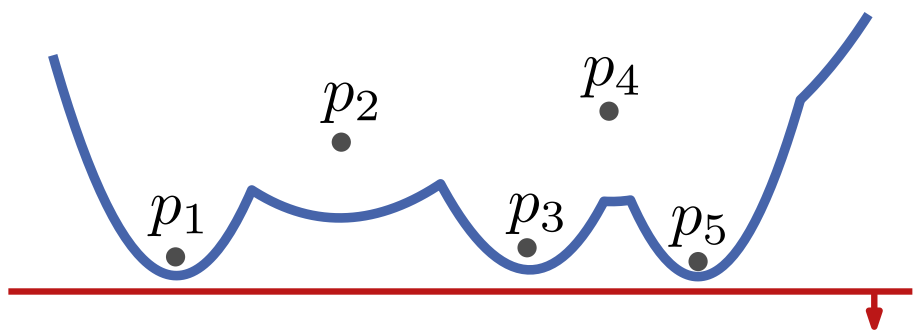
Data Structures

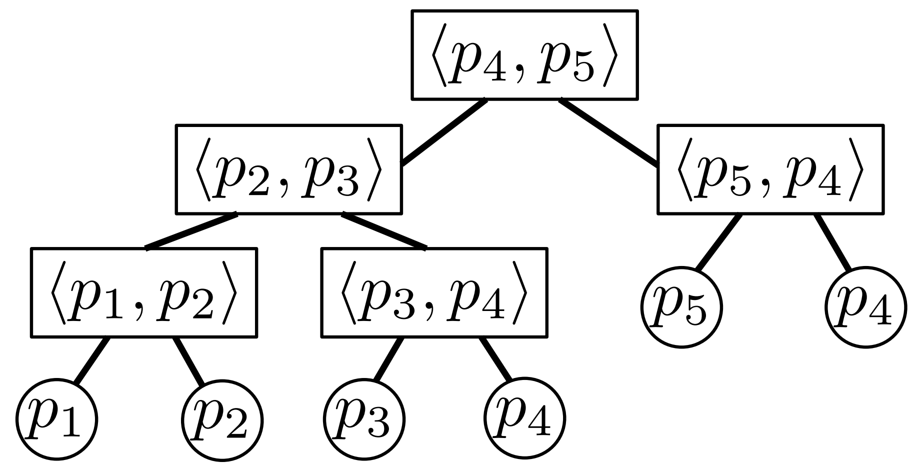
- Double-connected edge list (DCEL) $\mathcal{D}$ for $\mathrm{Vor}(P)$
  Warning: Include a bounding box to avoid half-lines

Data Structures

- Double-connected edge list (DCEL) $\mathcal{D}$ for $\mathrm{Vor}(P)$
  Warning: Include a bounding box to avoid half-lines

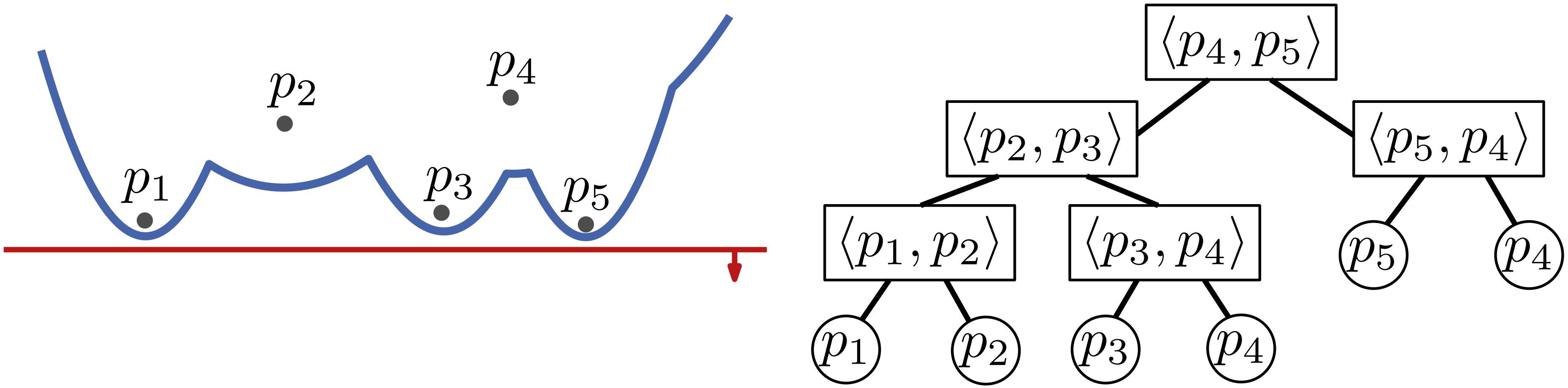
- Balanced binary search tree $\mathcal{T}$ for implicit beach line
  – Leaves represent parabolic arcs from left to right
  – Interior nodes $\langle p_i, p_j \rangle$ represent intersection points of f_{p_i} and f_{p_j}
  – Pointers from interior nodes to the corresponding edges in $\mathcal{D}$





Data Structures

- Double-connected edge list (DCEL) $\mathcal{D}$ for $\mathrm{Vor}(P)$
  Warning: Include a bounding box to avoid half-lines
- Balanced binary search tree $\mathcal{T}$ for implicit beach line
  - Leaves represent parabolic arcs from left to right
  - Interior nodes $\langle p_i, p_j \rangle$ represent intersection points of f_{p_i} and f_{p_j}
  - Pointers from interior nodes to the corresponding edges in $\mathcal{D}$



- Priority queue $\mathcal{Q}$ for the point and circle events
  - Pointer from circle event to corresponding leaf in $\mathcal{T}$ and vice versa

Fortune's Sweep Algorithm

Karlsruhe Institute of Technology

VoronoiDiagram($P \subset \mathbb{R}^2$)

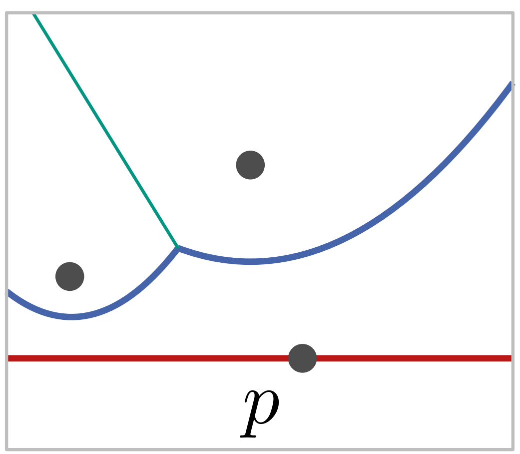
```
Q ← new PriorityQueue(P)    // Point events sorted by y
T ← new BalancedBinarySearchTree()    // sweep status (β)
D ← new DCEL()    // DS for Vor(P)
while not Q.empty() do
    p ← Q.ExtractMax()
    if p point event then
        HandlePointEvent(p)
    else
        α ← arcs of β to be removed
        HandleCircleEvent(α)
Handle interior remaining nodes of T (half-lines of Vor(P))
return D
```

Fortune's Sweep Algorithm

```
VoronoiDiagram(P ⊂ ℝ²)
  Q ← new PriorityQueue(P)    // Point events sorted by y
  T ← new BalancedBinarySearchTree()    // sweep status (β)
  D ← new DCEL()    // DS for Vor(P)
  while not Q.empty() do
      p ← Q.ExtractMax()
      if p point event then
          HandlePointEvent(p)
      else
          α ← arcs of β to be removed
          HandleCircleEvent(α)
  Handle interior remaining nodes of T (half-lines of Vor(P))
  return D
```

Handing Point Events

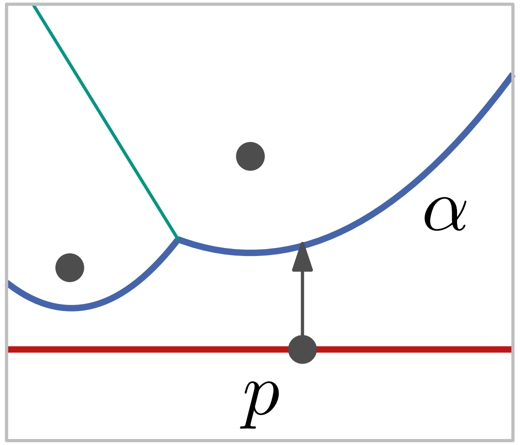
HandlePointEvent(point p)



Handing Point Events

HandlePointEvent(point p)

- Search in $\mathcal{T}$ for the arc α above p.
  If α has a pointer to a circle event in $\mathcal{Q}$, remove it from $\mathcal{Q}$.

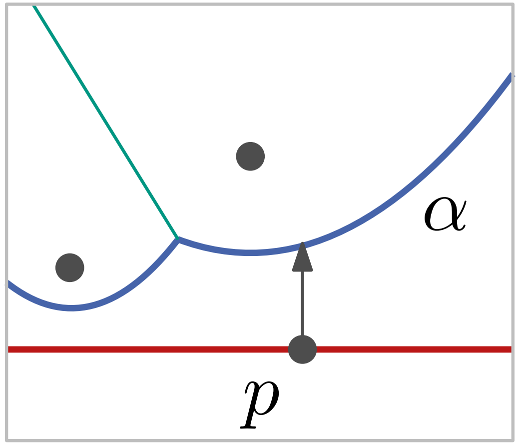


Handing Point Events

HandlePointEvent(point p)

- Search in $\mathcal{T}$ for the arc α above p.
  If α has a pointer to a circle event in $\mathcal{Q}$, remove it from $\mathcal{Q}$.

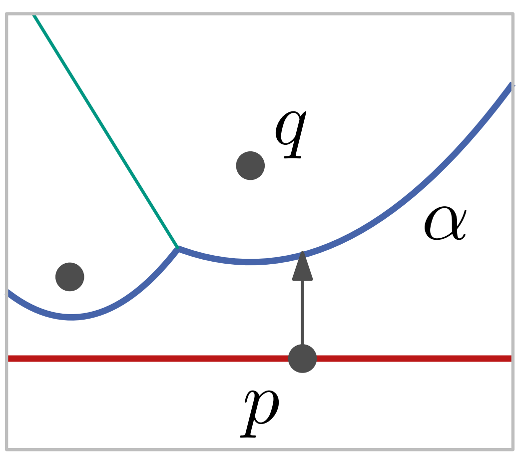
How?



Handing Point Events

HandlePointEvent(point p)

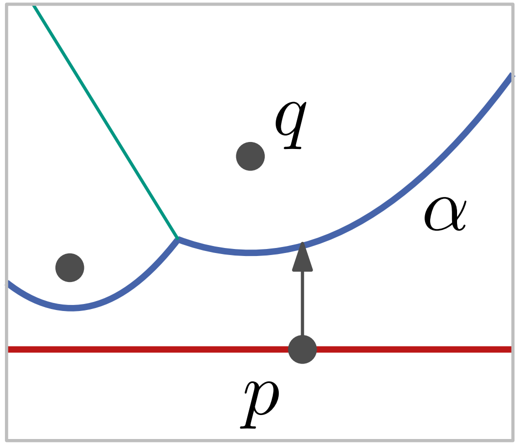
- Search in $\mathcal{T}$ for the arc α above p.
  If α has a pointer to a circle event in $\mathcal{Q}$, remove it from $\mathcal{Q}$.
- Split α into α_0 and α_2.
  Let α_1 be a new arc for p.



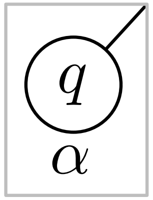
Handing Point Events

HandlePointEvent(point p)

- Search in $\mathcal{T}$ for the arc α above p.
  If α has a pointer to a circle event in $\mathcal{Q}$, remove it from $\mathcal{Q}$.
- Split α into α_0 and α_2.
  Let α_1 be a new arc for p.

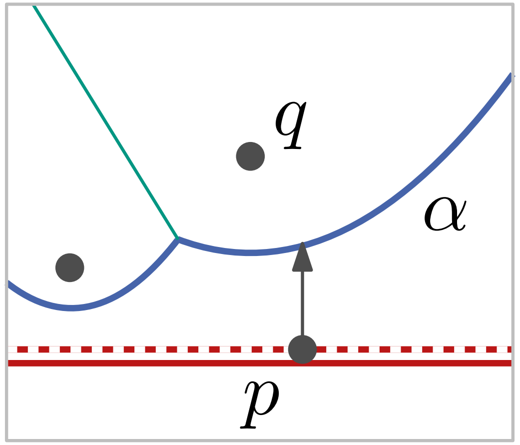


In $\mathcal{T}$:

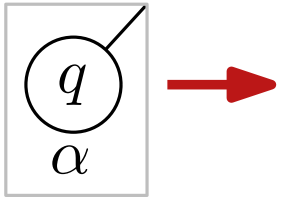


HandlePointEvent(point p)

- Search in $\mathcal{T}$ for the arc α above p.
  If α has a pointer to a circle event in $\mathcal{Q}$, remove it from $\mathcal{Q}$.
- Split α into α_0 and α_2.
  Let α_1 be a new arc for p.



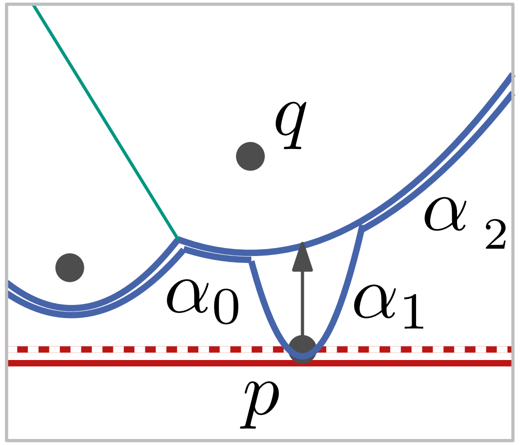
In $\mathcal{T}$:



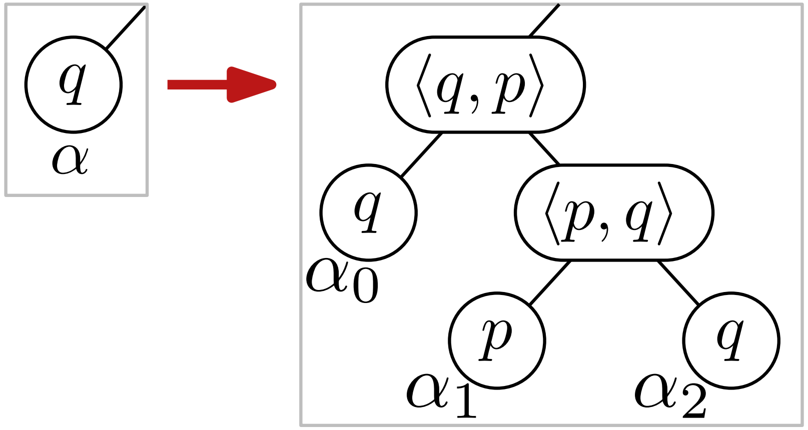
Handing Point Events

HandlePointEvent(point p)

- Search in $\mathcal{T}$ for the arc α above p.
  If α has a pointer to a circle event in $\mathcal{Q}$, remove it from $\mathcal{Q}$.
- Split α into α_0 and α_2.
  Let α_1 be a new arc for p.



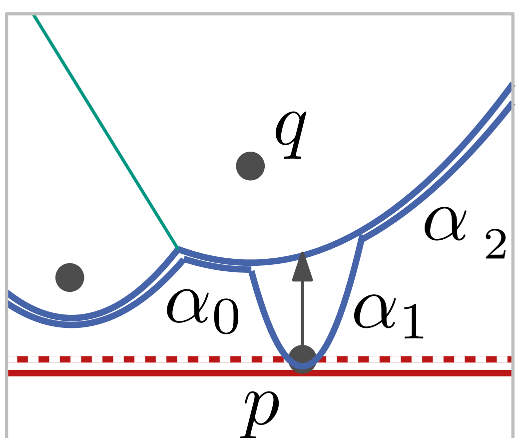
In $\mathcal{T}$:



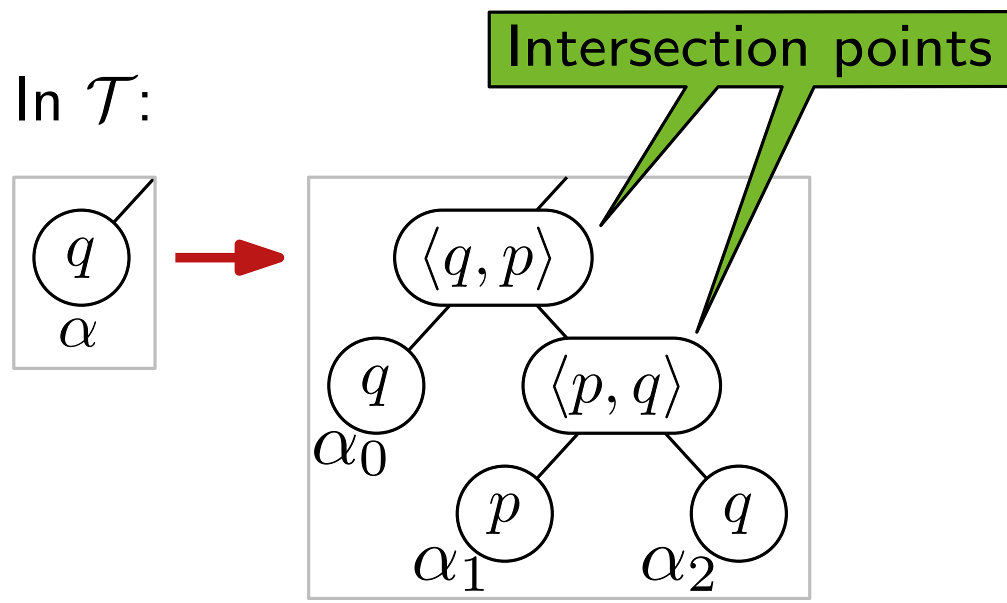
Handing Point Events

HandlePointEvent(point p)

- Search in $\mathcal{T}$ for the arc α above p.
  If α has a pointer to a circle event in $\mathcal{Q}$, remove it from $\mathcal{Q}$.
- Split α into α_0 and α_2.
  Let α_1 be a new arc for p.

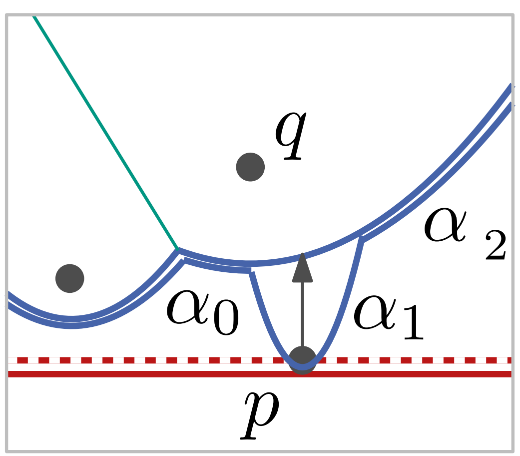


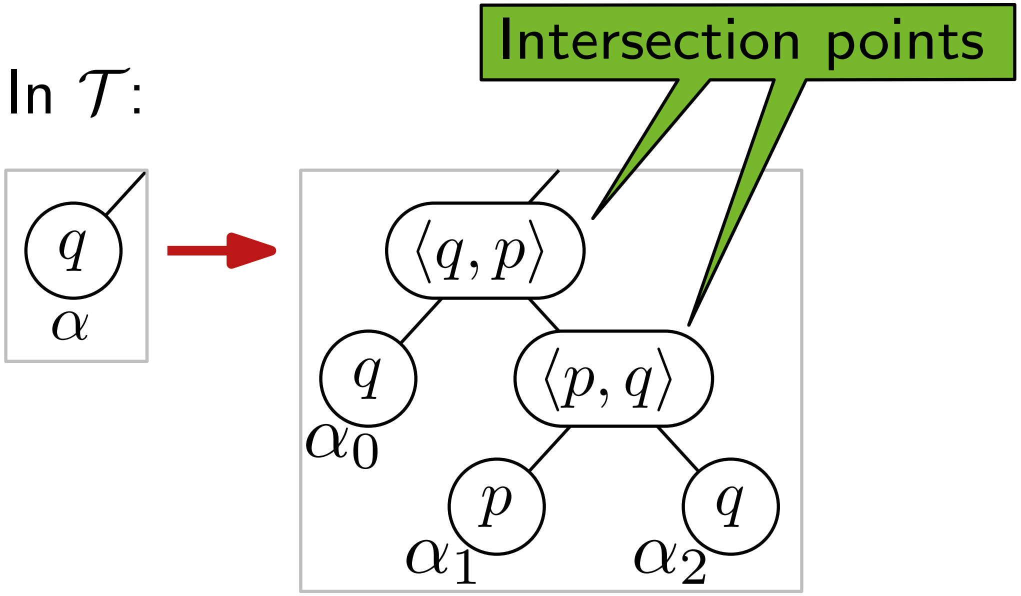
In $\mathcal{T}$:



HandlePointEvent(point p)

- Search in $\mathcal{T}$ for the arc α above p.
  If α has a pointer to a circle event in $\mathcal{Q}$, remove it from $\mathcal{Q}$.
- Split α into α_0 and α_2.
  Let α_1 be a new arc for p.
- Add edges $\langle q, p \rangle$ and $\langle p, q \rangle$ to $\mathcal{D}$.



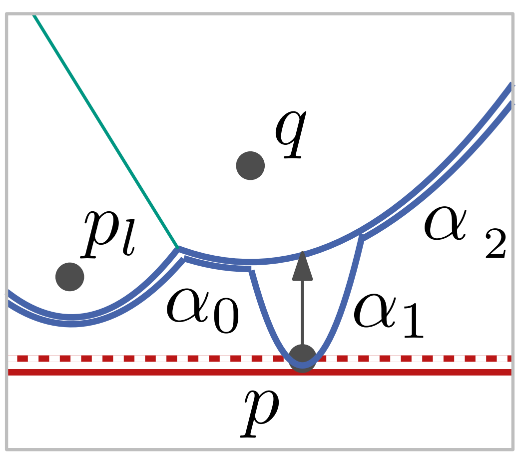


Handing Point Events

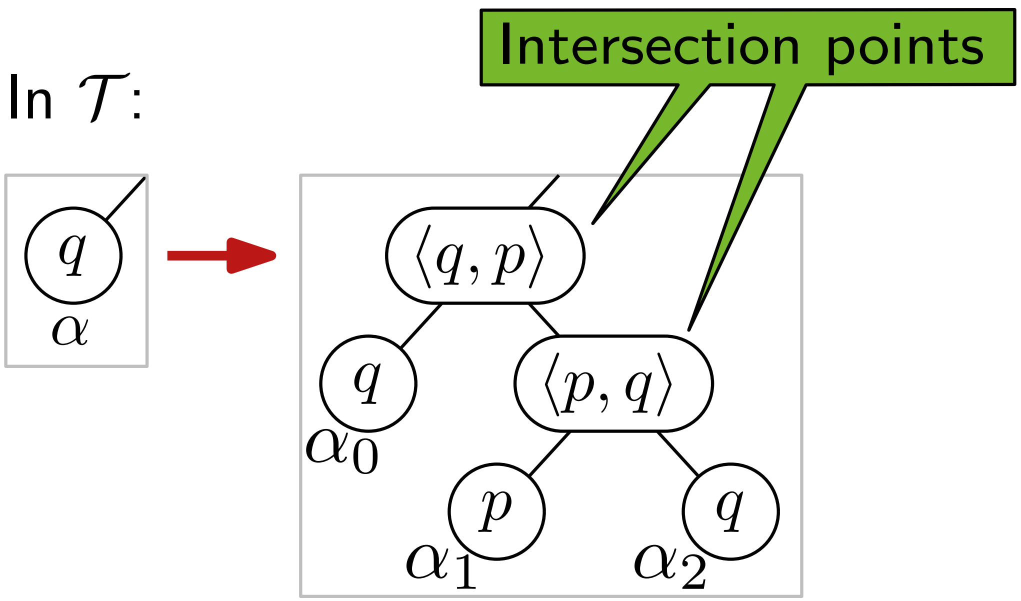
HandlePointEvent(point p)

- Search in $\mathcal{T}$ for the arc α above p.
  If α has a pointer to a circle event in $\mathcal{Q}$, remove it from $\mathcal{Q}$.
- Split α into α_0 and α_2.
  Let α_1 be a new arc for p.
- Add edges $\langle q, p \rangle$ and $\langle p, q \rangle$ to $\mathcal{D}$.
- Check $\langle p_l, q, p \rangle$ and $\langle p, q, p_r \rangle$ for circle events.

Neighbors of β



In $\mathcal{T}$:

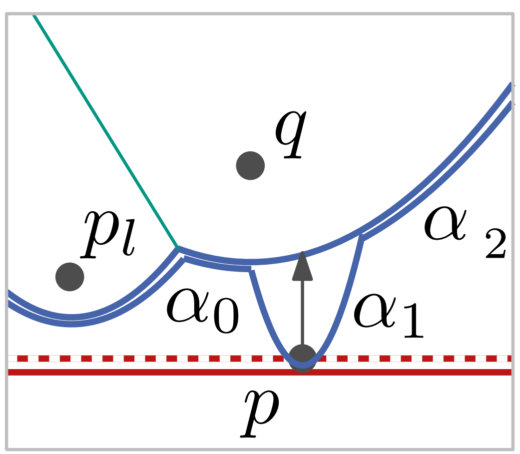


Handing Point Events

HandlePointEvent(point p)

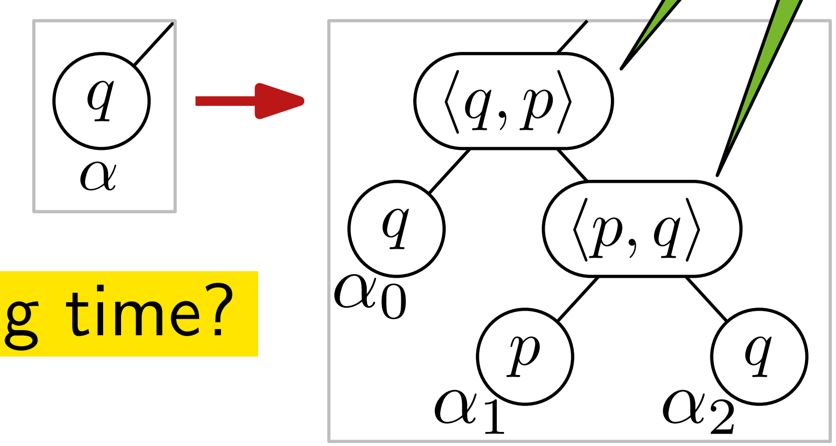
- Search in $\mathcal{T}$ for the arc α above p.
  If α has a pointer to a circle event in $\mathcal{Q}$, remove it from $\mathcal{Q}$.
- Split α into α_0 and α_2.
  Let α_1 be a new arc for p.
- Add edges $\langle q, p\rangle$ and $\langle p, q\rangle$ to $\mathcal{D}$.
- Check $\langle p_l, q, p\rangle$ and $\langle p, q, p_r\rangle$ for circle events.

Neighbors of β



In $\mathcal{T}$:

Intersection points



Running time?

Handing Point Events

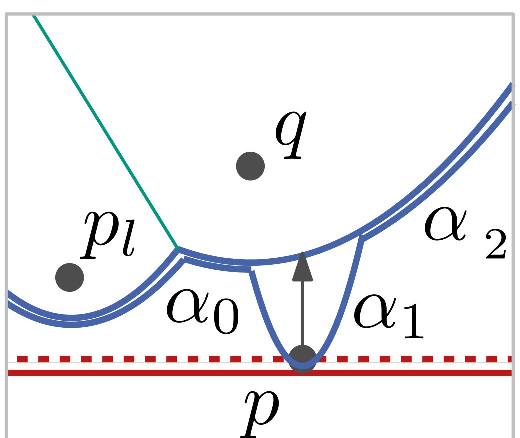
HandlePointEvent(point p)

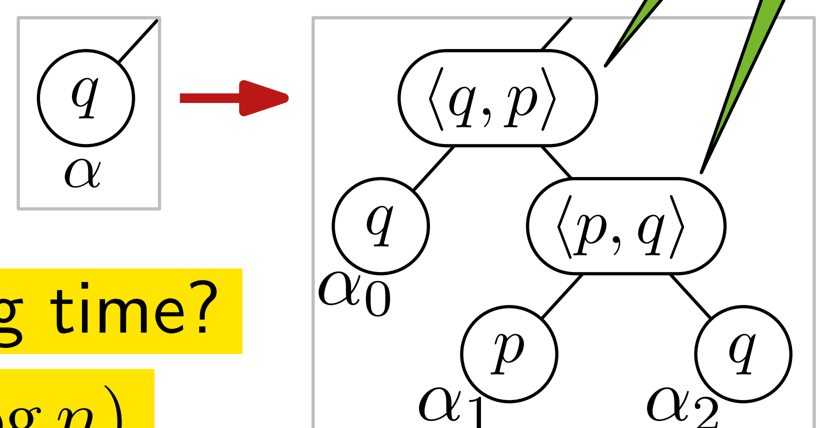
- Search in $\mathcal{T}$ for the arc α above p.
  If α has a pointer to a circle event in $\mathcal{Q}$, remove it from $\mathcal{Q}$.
- Split α into α_0 and α_2.
  Let α_1 be a new arc for p.
- Add edges $\langle q, p \rangle$ and $\langle p, q \rangle$ to $\mathcal{D}$.
- Check $\langle p_l, q, p \rangle$ and $\langle p, q, p_r \rangle$ for circle events.

Neighbors of β

In $\mathcal{T}$:

Intersection points



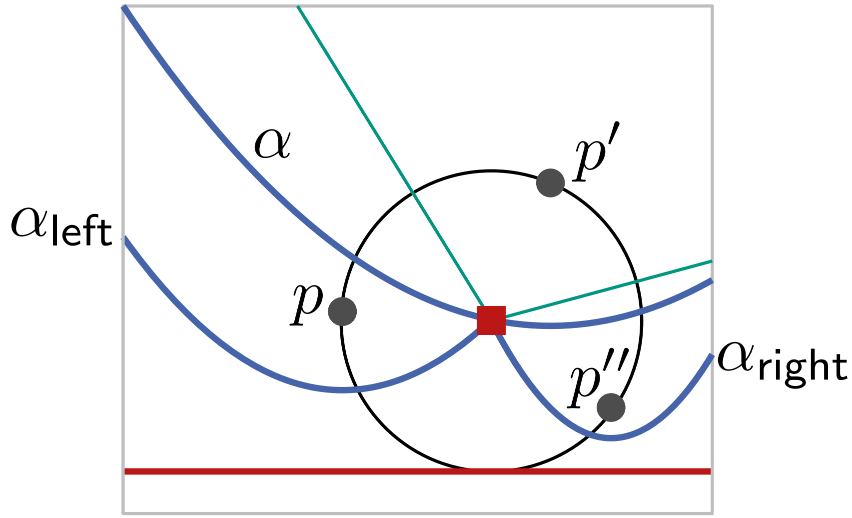


Running time?

$O(\log n)$

Handling Circle Events

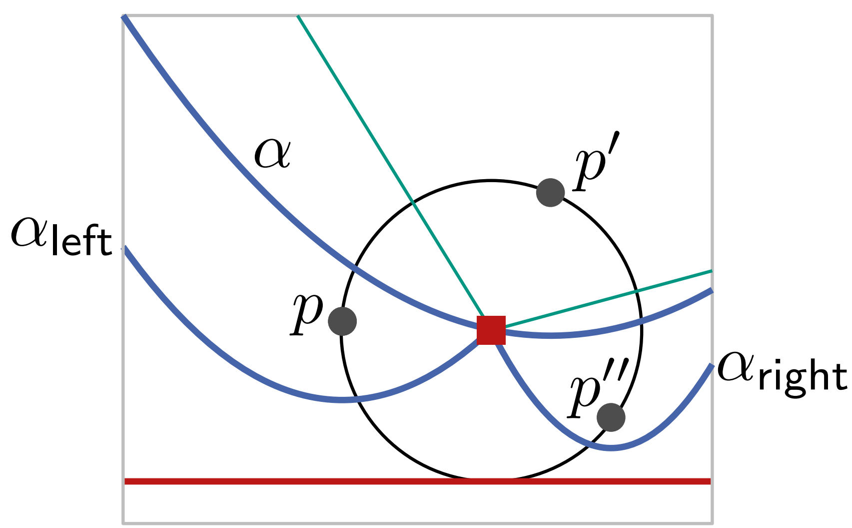
HandleCircleEvent(arc α)



Handling Circle Events

HandleCircleEvent(arc α)

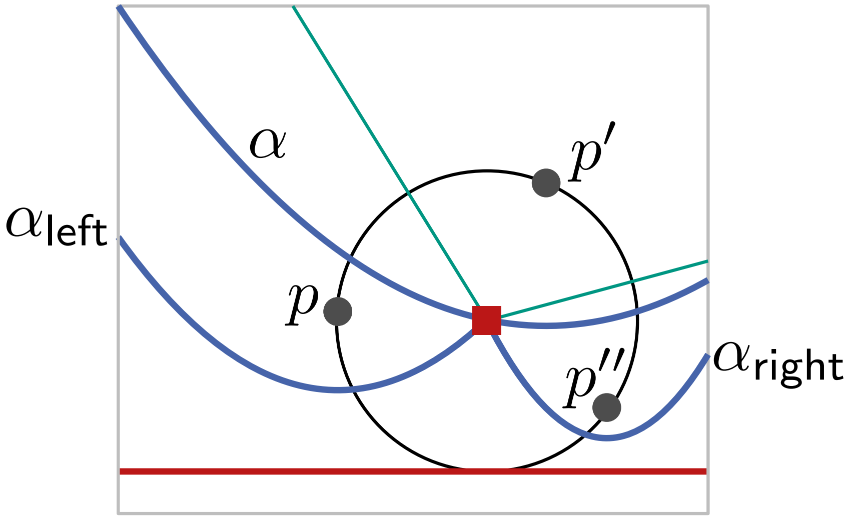
- $\mathcal{T}$.delete(α); Update intersection points in $\mathcal{T}$



Handling Circle Events

HandleCircleEvent(arc α)

- $\mathcal{T}$.delete(α); Update intersection points in $\mathcal{T}$
- Remove all circle events with middle arc α from $\mathcal{Q}$.

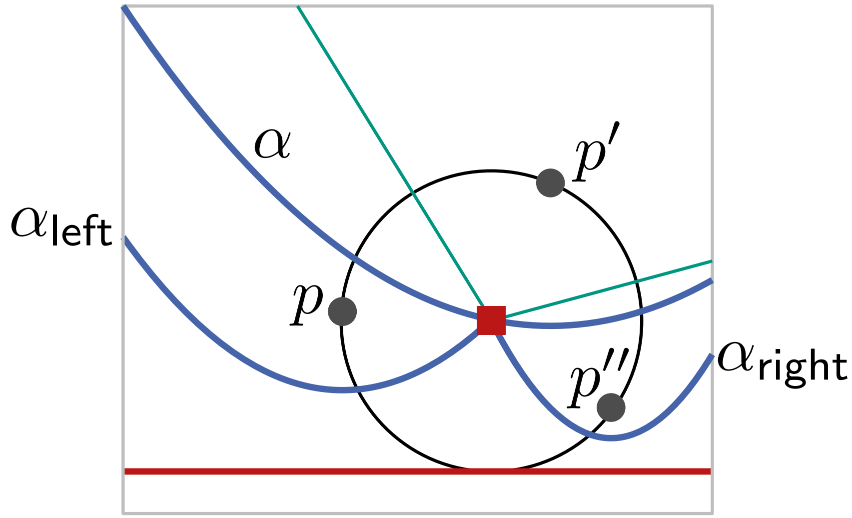


Handling Circle Events

HandleCircleEvent(arc α)

- $\mathcal{T}$.delete(α); Update intersection points in $\mathcal{T}$
- Remove all circle events with middle arc α from $\mathcal{Q}$.

false alarms

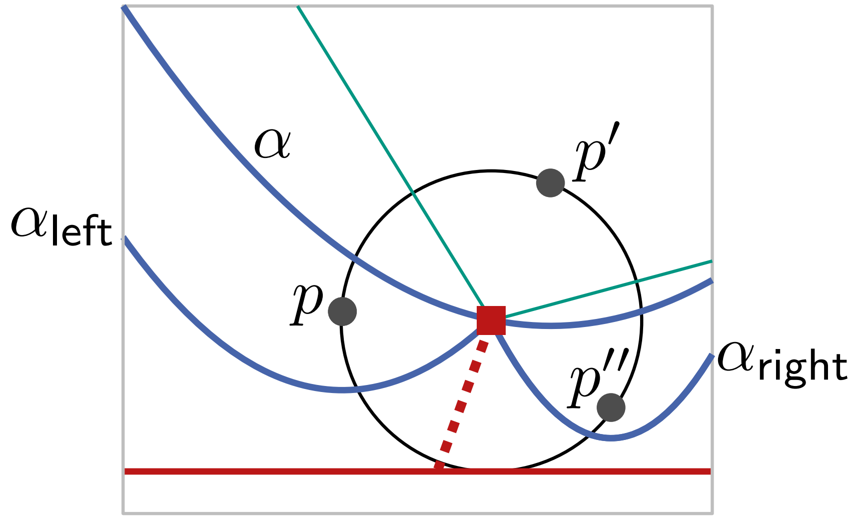


Handling Circle Events

HandleCircleEvent(arc α)

- $\mathcal{T}$.delete(α); Update intersection points in $\mathcal{T}$
- Remove all circle events with middle arc α from $\mathcal{Q}$.
- Add node $\mathcal{V}(\{p, p', p''\})$ and edges $\langle p, p'' \rangle$, $\langle p'', p \rangle$ to $\mathcal{D}$.

false alarms



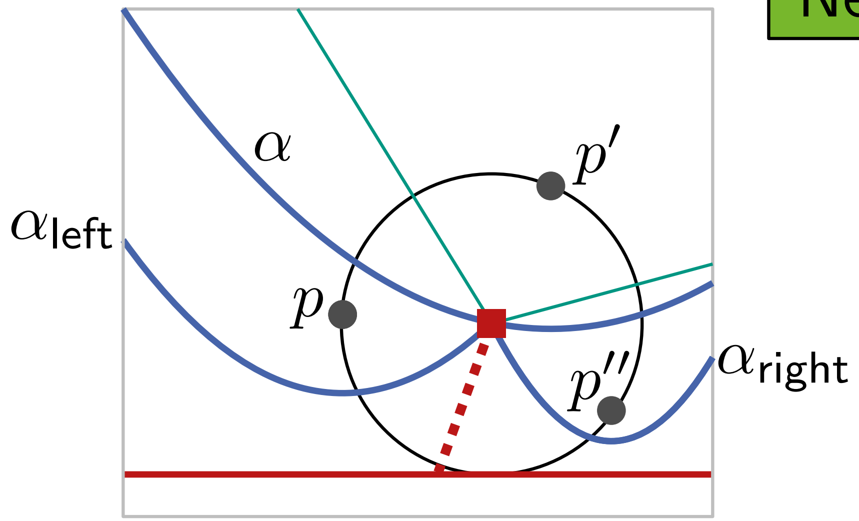
Handling Circle Events

HandleCircleEvent(arc α)

- $\mathcal{T}$.delete(α); Update intersection points in $\mathcal{T}$
- Remove all circle events with middle arc α from $\mathcal{Q}$.
- Add node $\mathcal{V}(\{p, p', p''\})$ and edges $\langle p, p''\rangle$, $\langle p'', p\rangle$ to $\mathcal{D}$.
- Add potential circle events $\langle p_l, p, p''\rangle$ and $\langle p, p'', p_r\rangle$ to $\mathcal{Q}$.

false alarms

Neighbors of β



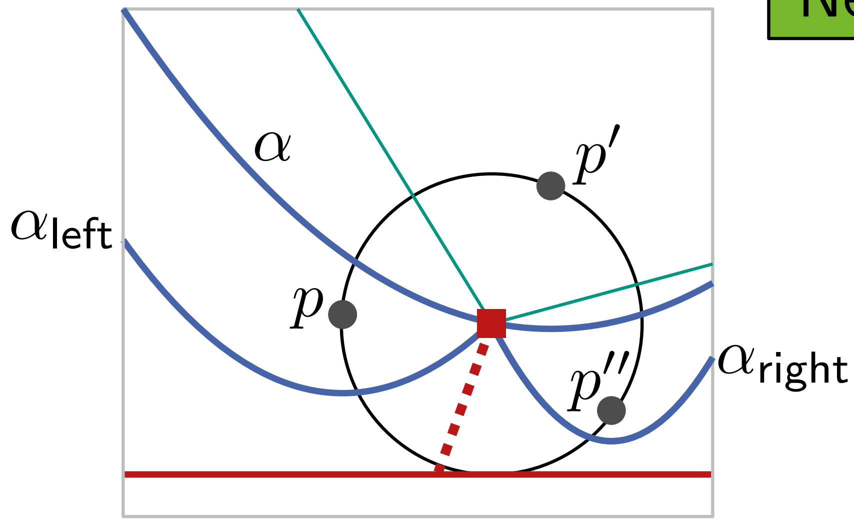
Handling Circle Events

HandleCircleEvent(arc α)

- $\mathcal{T}$.delete(α); Update intersection points in $\mathcal{T}$
- Remove all circle events with middle arc α from $\mathcal{Q}$.
- Add node $\mathcal{V}(\{p, p', p''\})$ and edges $\langle p, p'' \rangle$, $\langle p'', p \rangle$ to $\mathcal{D}$.
- Add potential circle events $\langle p_l, p, p'' \rangle$ and $\langle p, p'', p_r \rangle$ to $\mathcal{Q}$.

false alarms

Neighbors of β



Running time?

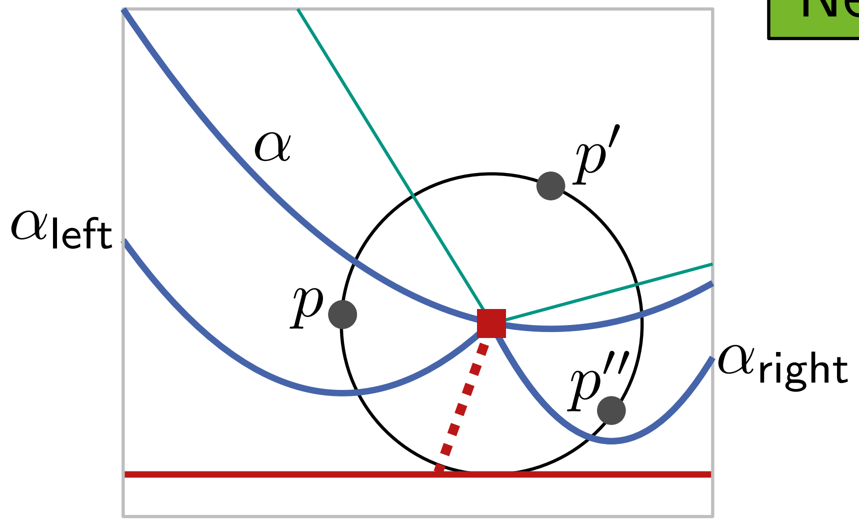
Handling Circle Events

HandleCircleEvent(arc α)

- $\mathcal{T}$.delete(α); Update intersection points in $\mathcal{T}$
- Remove all circle events with middle arc α from $\mathcal{Q}$.
- Add node $\mathcal{V}(\{p, p', p''\})$ and edges $\langle p, p'' \rangle$, $\langle p'', p \rangle$ to $\mathcal{D}$.
- Add potential circle events $\langle p_l, p, p'' \rangle$ and $\langle p, p'', p_r \rangle$ to $\mathcal{Q}$.

false alarms

Neighbors of β



Running time?

$O(\log n)$

Fortune's Sweep Algorithm

Theorem 4: For a set P of n points, Fortune's sweep algorithm computes the Voronoi Diagram $\mathrm{Vor}(P)$ in $O(n \log n)$ time and $O(n)$ space.

Fortune's Sweep Algorithm

Theorem 4: For a set P of n points, Fortune's sweep algorithm computes the Voronoi Diagram $\mathrm{Vor}(P)$ in $O(n \log n)$ time and $O(n)$ space.

Proof sketch:

- Each event requires $O(\log n)$ time
- n point events
- $\leq 2n - 5$ circle events (= #nodes of $\mathrm{Vor}(P)$)
- False alarms are deleted from $\mathcal{Q}$ before they are processed.

Fortune's Sweep Algorithm

Karlsruhe Institute of Technology

Theorem 4: For a set P of n points, Fortune's sweep algorithm computes the Voronoi Diagram $\text{Vor}(P)$ in $O(n \log n)$ time and $O(n)$ space.

Proof sketch:

- Each event requires $O(\log n)$ time
- n point events
- $\leq 2n - 5$ circle events ($=$ #nodes of $\text{Vor}(P)$)
- False alarms are deleted from $\mathcal{Q}$ before they are processed.

Discussion

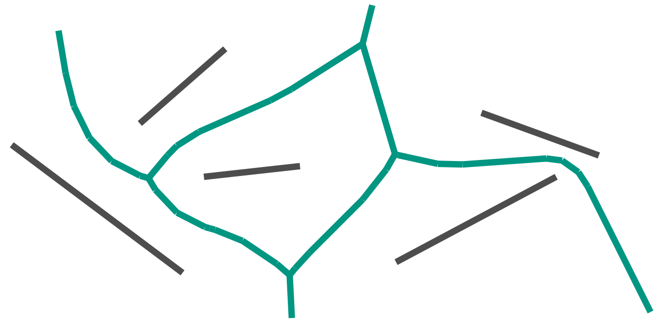
Are there other variants of Voronoi diagrams?

Discussion

Are there other variants of Voronoi diagrams?

Yes! For example, we can design an algorithm to compute the Voronoi diagram for line segments with the same running time and space.

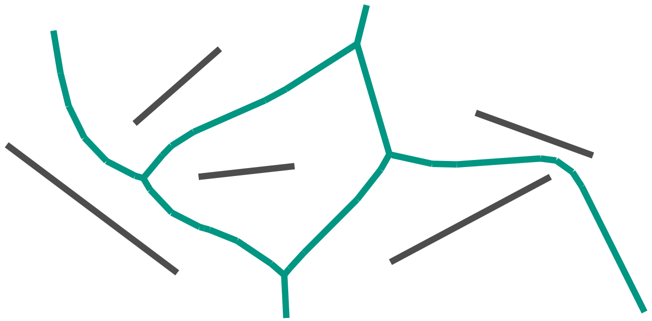
Other metrics like L_p or additive/multiplicative weighted Voronoi diagrams are possible.



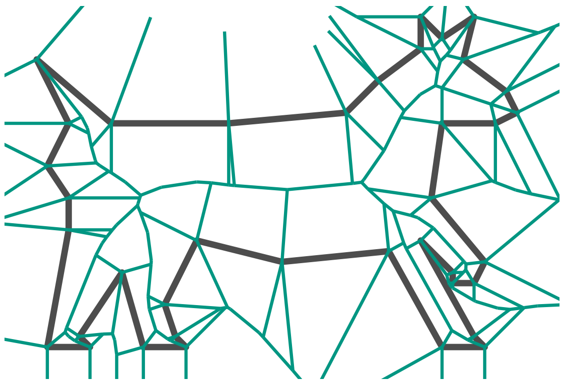
Discussion

Are there other variants of Voronoi diagrams?

Yes! For example, we can design an algorithm to compute the Voronoi diagram for line segments with the same running time and space.



Other metrics like L_p or additive/multiplicative weighted Voronoi diagrams are possible.



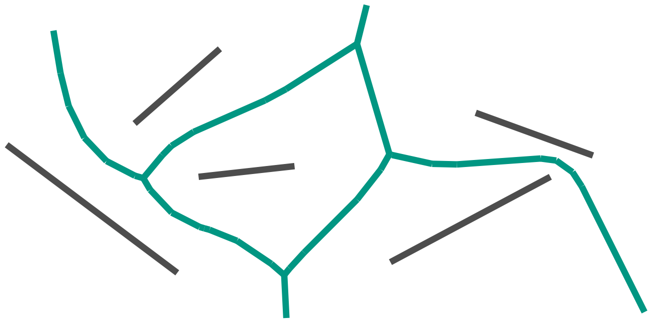
Voronoi diagrams for polygons define the so-called medial axis, (important in image processing).

Also farthest-point Voronoi diagrams are possible.

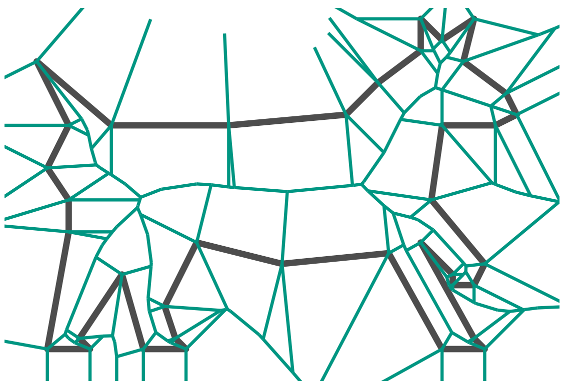
Discussion

Are there other variants of Voronoi diagrams?

Yes! For example, we can design an algorithm to compute the Voronoi diagram for line segments with the same running time and space.



Other metrics like L_p or additive/multiplicative weighted Voronoi diagrams are possible.



Voronoi diagrams for polygons define the so-called medial axis, (important in image processing).

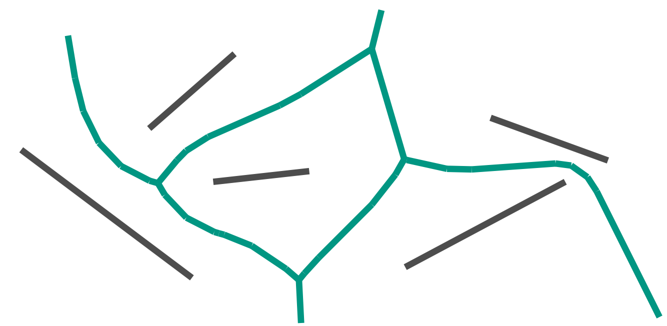
Also farthest-point Voronoi diagrams are possible.

What happens in higher dimensions?

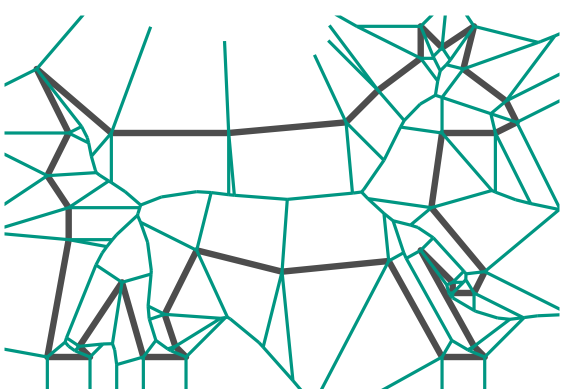
KIT Karlsruhe Institute of Technology

Are there other variants of Voronoi diagrams?

Yes! For example, we can design an algorithm to compute the Voronoi diagram for line segments with the same running time and space.



Other metrics like L_p or additive/multiplicative weighted Voronoi diagrams are possible.



Voronoi diagrams for polygons define the so-called medial axis, (important in image processing).

Also farthest-point Voronoi diagrams are possible.

What happens in higher dimensions?

The complexity of $\mathrm{Vor}(P)$ increases to $\Theta(n^{\lceil d/2 \rceil})$ and the running time to $O(n \log n + n^{\lceil d/2 \rceil})$.

From Voronoi to Art...

Geogebra