Computational Geometry

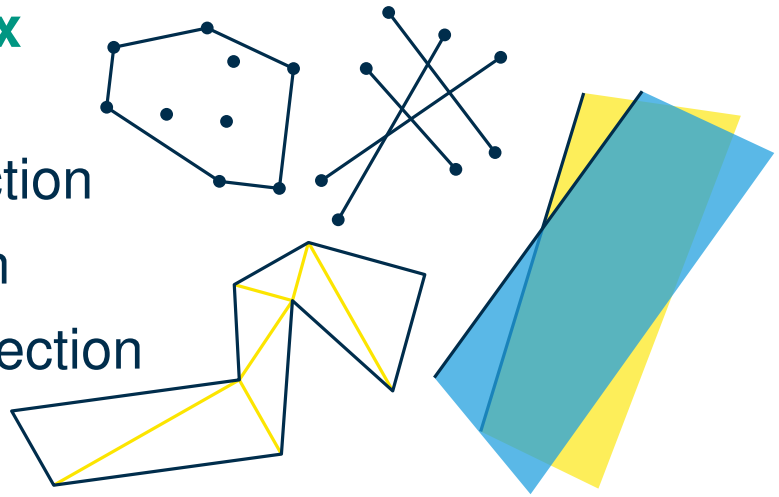
Voronoi Diagram: Fortunes Algorithm

Thomas Bläsius

Overview

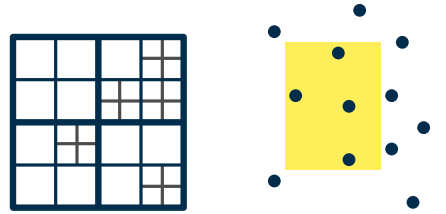
Basic Toolbox

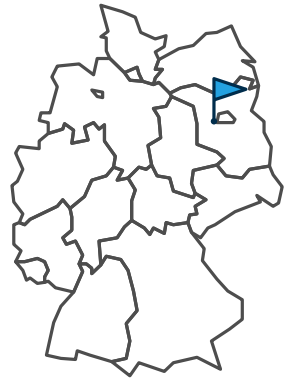
- convex hull
- line intersection
- triangulation
- plane intersection



Geometric Data Structures

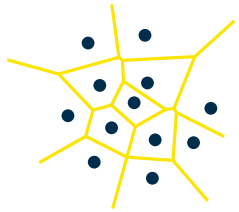
- orthogonal range searching
- space partitioning
- point location

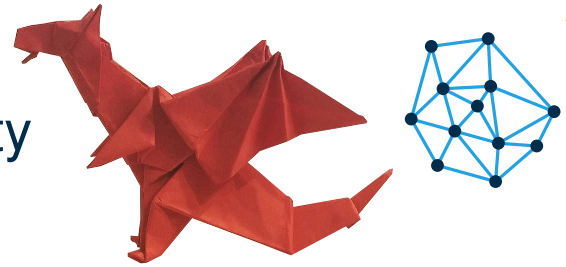




Advanced Toolbox

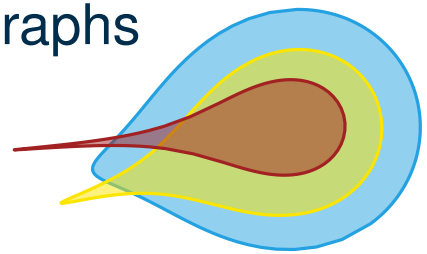
- Voronoi diagrams
- Delaunay triangulations
- origami
- complexity

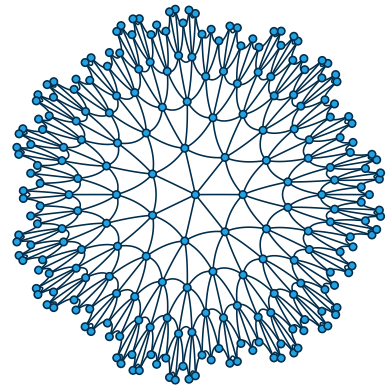




Related Topics

- What is geometry?
- hyperbolic geometry
- geometric graphs

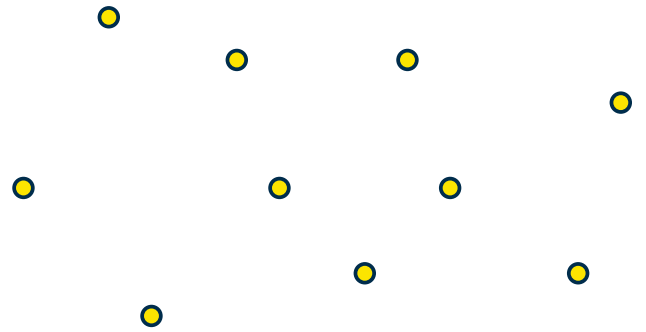




Who Uses Which Mailbox?

Situation

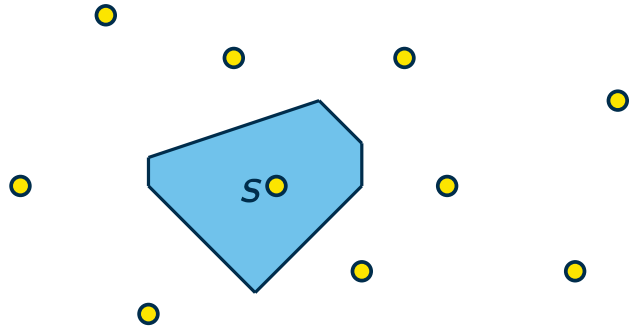
- given: set S of sites
- Which points are closest to which site?
  (assuming Euclidean distances)



Who Uses Which Mailbox?

Situation

- given: set S of sites
- Which points are closest to which site?
  (assuming Euclidean distances)



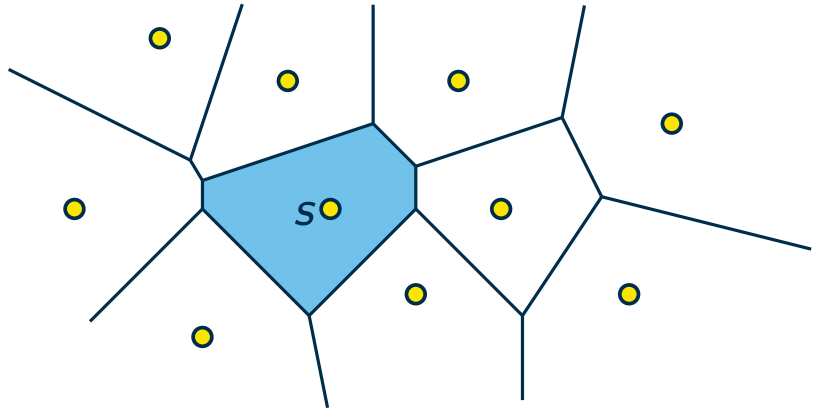
Voronoi-Diagramm – Some Definitions

- **Voronoi cell $\mathcal{V}(s)$** of $s \in S$: set of points closer to s than to any other site

Who Uses Which Mailbox?

Situation

- given: set S of sites
- Which points are closest to which site?
  (assuming Euclidean distances)



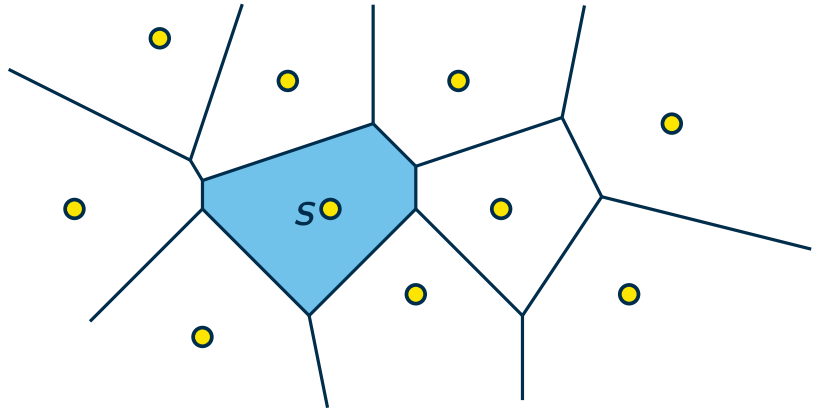
Voronoi-Diagramm – Some Definitions

- **Voronoi cell $\mathcal{V}(s)$** of $s \in S$: set of points closer to s than to any other site
- union of all cells is the **Voronoi diagram $\mathrm{Vor}(S)$**

Who Uses Which Mailbox?

Situation

- given: set S of sites
- Which points are closest to which site?
  (assuming Euclidean distances)



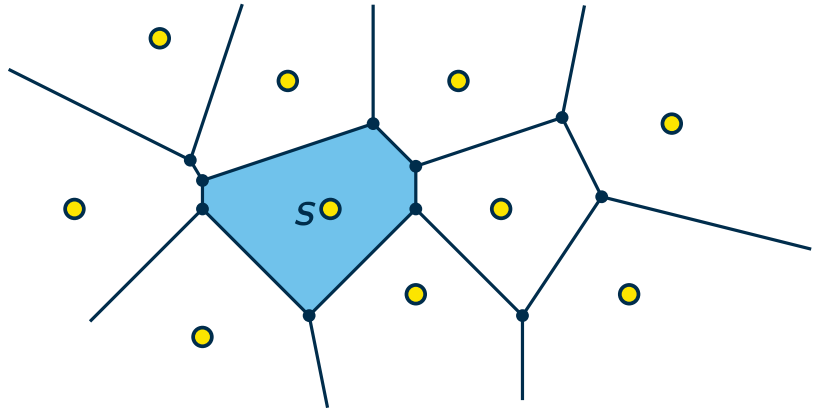
Voronoi-Diagramm – Some Definitions

- **Voronoi cell $\mathcal{V}(s)$** of $s \in S$: set of points closer to s than to any other site
- union of all cells is the **Voronoi diagram $\mathrm{Vor}(S)$**
- planar subdivision of the plane

Who Uses Which Mailbox?

Situation

- given: set S of sites
- Which points are closest to which site?
  (assuming Euclidean distances)



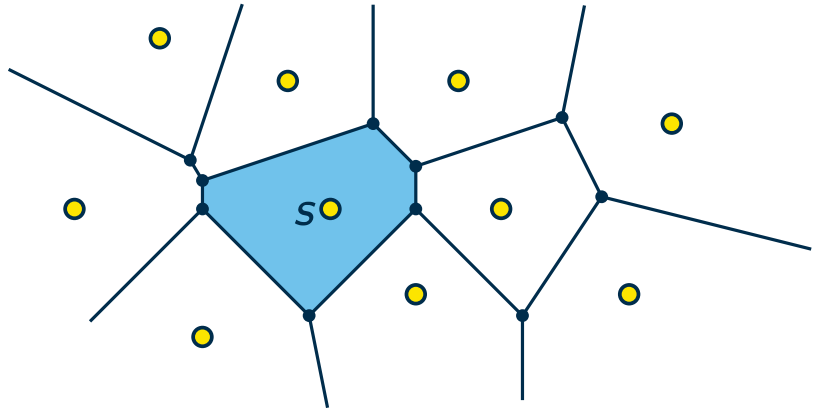
Voronoi-Diagramm – Some Definitions

- **Voronoi cell $\mathcal{V}(s)$** of $s \in S$: set of points closer to s than to any other site
- union of all cells is the **Voronoi diagram $\mathrm{Vor}(S)$**
- planar subdivision of the plane
- boundary between cells are called **edges**
- endpoints of these edges are **vertices**

Who Uses Which Mailbox?

Situation

- given: set S of sites
- Which points are closest to which site?
  (assuming Euclidean distances)

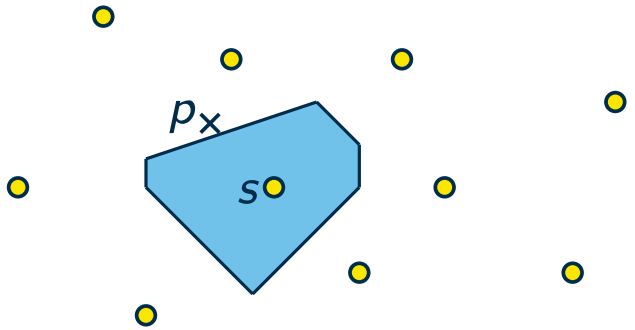


Voronoi-Diagramm – Some Definitions

- **Voronoi cell $\mathcal{V}(s)$** of $s \in S$: set of points closer to s than to any other site
- union of all cells is the **Voronoi diagram $\mathrm{Vor}(S)$**
- planar subdivision of the plane
- boundary between cells are called **edges**
- endpoints of these edges are **vertices**

Goal For Today: compute $\mathrm{Vor}(S)$

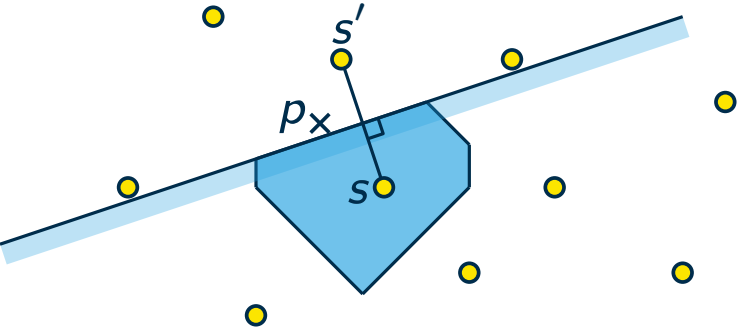
Basic Properties



Points Belonging To The Cell Of s

- Why does p not lie in the cell of s?

Basic Properties

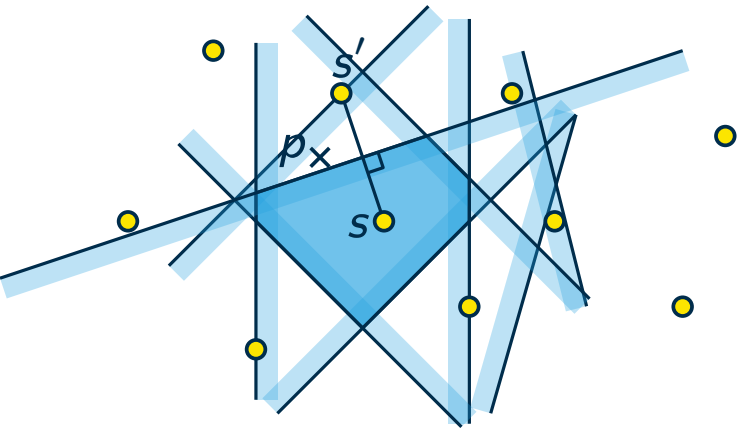


Points Belonging To The Cell Of s

- Why does p not lie in the cell of s?
- s' is closer $p \Leftrightarrow s \perp s'$ separates s from p

  perpendicular bisector of s and s'

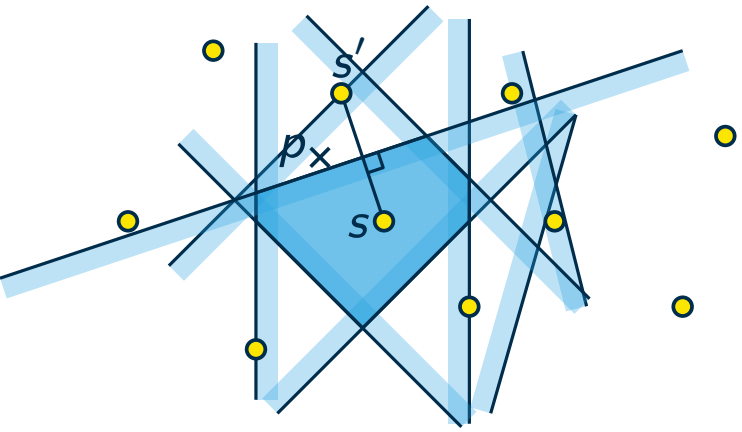
Basic Properties



Points Belonging To The Cell Of s

- Why does p not lie in the cell of s?
- s' is closer $p \Leftrightarrow s \perp s'$ separates s from p
- the cell of s is the intersection of half planes

Basic Properties

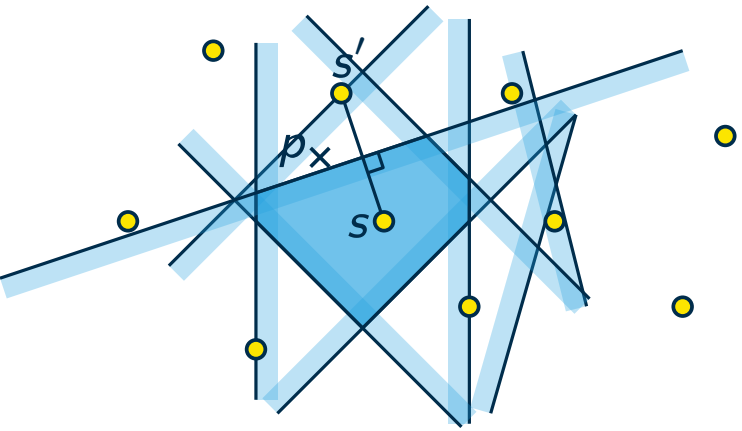


Points Belonging To The Cell Of s

- Why does p not lie in the cell of s?
- s' is closer $p \Leftrightarrow s \perp s'$ separates s from p
- the cell of s is the intersection of half planes
- this already yields a (slow) algorithm

How slow exactly?

Basic Properties



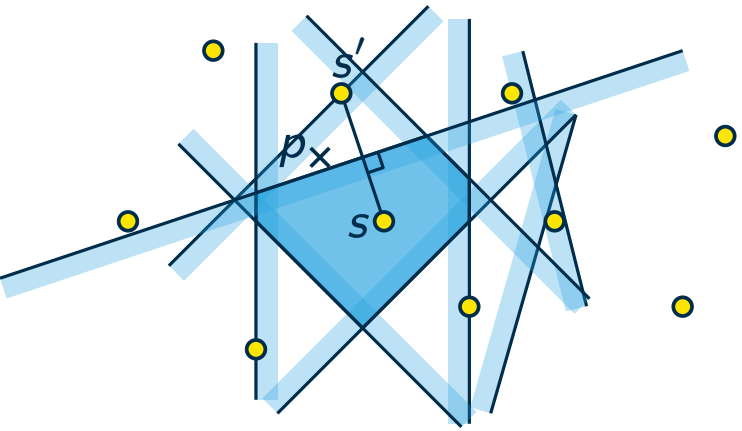
Connectedness

Points Belonging To The Cell Of s

- Why does p not lie in the cell of s?
- s' is closer $p \Leftrightarrow s \perp s'$ separates s from p
- the cell of s is the intersection of half planes
- this already yields a (slow) algorithm

How slow exactly?

Basic Properties



Points Belonging To The Cell Of s

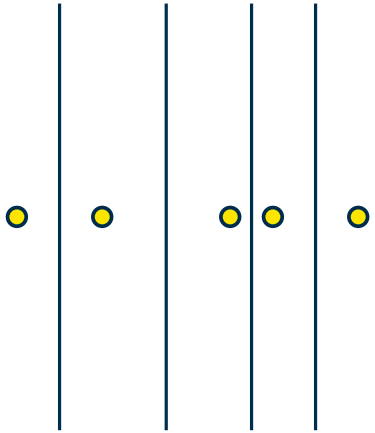
- Why does p not lie in the cell of s?
- s' is closer $p \Leftrightarrow s \perp s'$ separates s from p
- the cell of s is the intersection of half planes
- this already yields a (slow) algorithm

How slow exactly?

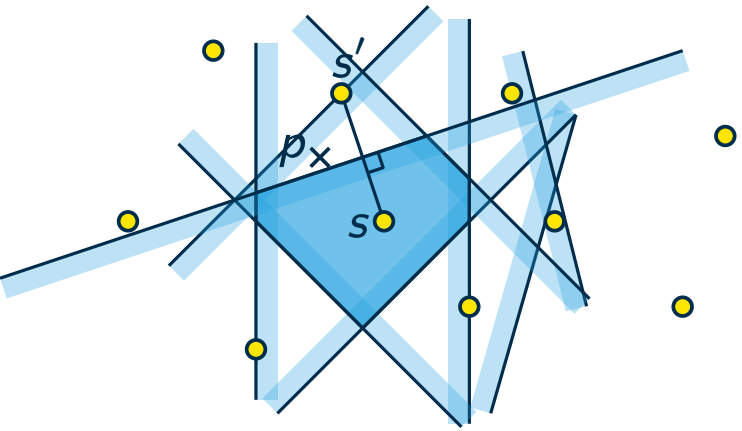
Connectedness

- not all sites collinear $\Rightarrow \mathrm{Vor}(S)$ is connected

(viewing $\mathrm{Vor}(S)$ as the union of edges, i.e., the geometric “graph”)



Basic Properties



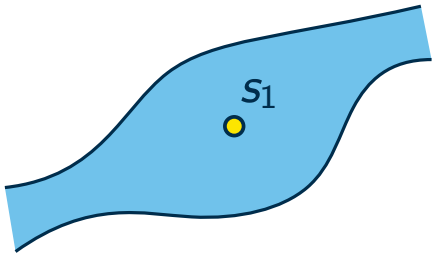
Points Belonging To The Cell Of s

- Why does p not lie in the cell of s?
- s' is closer $p \Leftrightarrow s \perp s'$ separates s from p
- the cell of s is the intersection of half planes
- this already yields a (slow) algorithm

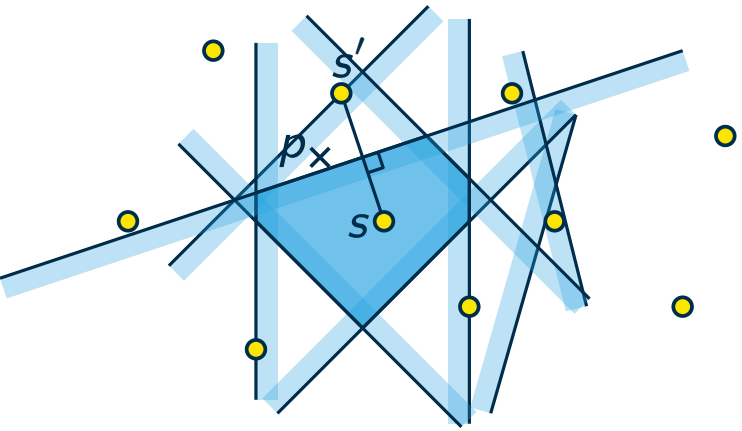
How slow exactly?

Connectedness

- not all sites collinear $\Rightarrow$ $\mathrm{Vor}(S)$ is connected (viewing $\mathrm{Vor}(S)$ as the union of edges, i.e., the geometric "graph")
- proof: assume it is disconnected
  - $\Rightarrow$ there is a site $s_1 \in S$, such that $\mathcal{V}(s_1)$ separates the plane



Basic Properties



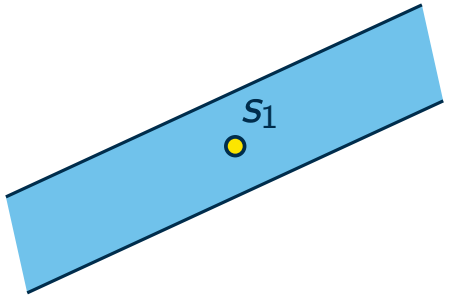
Points Belonging To The Cell Of s

- Why does p not lie in the cell of s?
- s' is closer $p \Leftrightarrow s \perp s'$ separates s from p
- the cell of s is the intersection of half planes
- this already yields a (slow) algorithm

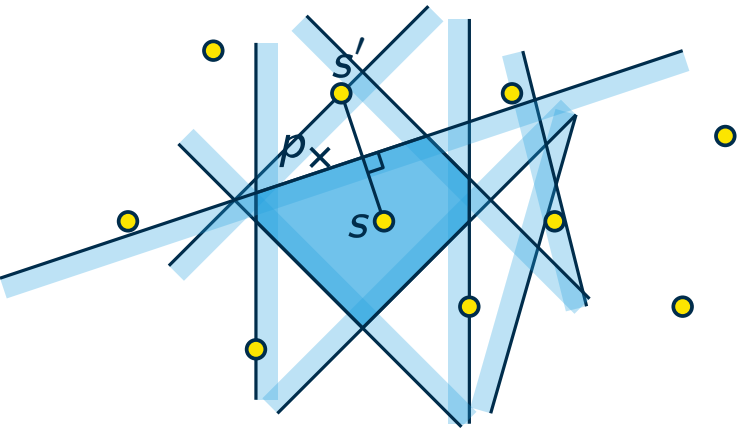
How slow exactly?

Connectedness

- not all sites collinear $\Rightarrow \mathrm{Vor}(S)$ is connected (viewing $\mathrm{Vor}(S)$ as the union of edges, i.e., the geometric "graph")
- proof: assume it is disconnected
  - $\Rightarrow$ there is a site $s_1 \in S$, such that $\mathcal{V}(s_1)$ separates the plane
  - $\mathcal{V}(s_1)$ convex $\Rightarrow$ must be bounded by two parallel lines



Basic Properties



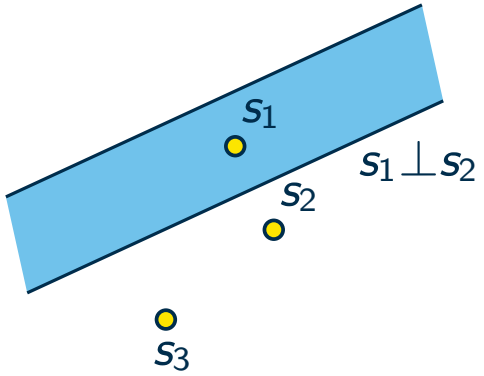
Points Belonging To The Cell Of s

- Why does p not lie in the cell of s?
- s' is closer $p \Leftrightarrow s \perp s'$ separates s from p
- the cell of s is the intersection of half planes
- this already yields a (slow) algorithm

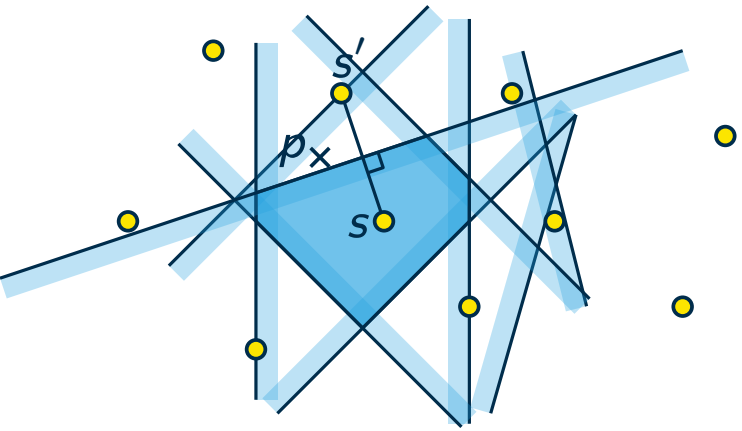
How slow exactly?

Connectedness

- not all sites collinear $\Rightarrow \mathrm{Vor}(S)$ is connected (viewing $\mathrm{Vor}(S)$ as the union of edges, i.e., the geometric "graph")
- proof: assume it is disconnected
  - $\Rightarrow$ there is a site $s_1 \in S$, such that $\mathcal{V}(s_1)$ separates the plane
  - $\mathcal{V}(s_1)$ convex $\Rightarrow$ must be bounded by two parallel lines
  - let $s_1 \perp s_2$ be one of these lines and let $s_3 \in S$ be non-collinear



Basic Properties



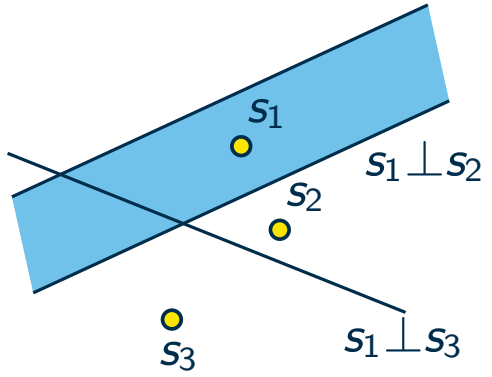
Points Belonging To The Cell Of s

- Why does p not lie in the cell of s?
- s' is closer $p \Leftrightarrow s \perp s'$ separates s from p
- the cell of s is the intersection of half planes
- this already yields a (slow) algorithm

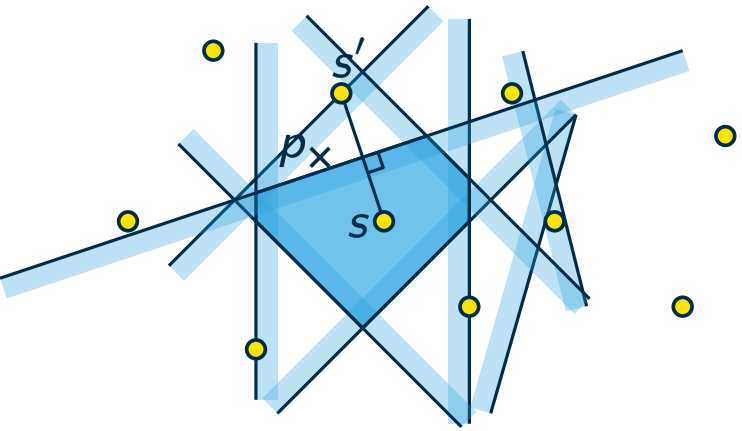
How slow exactly?

Connectedness

- not all sites collinear $\Rightarrow \mathrm{Vor}(S)$ is connected (viewing $\mathrm{Vor}(S)$ as the union of edges, i.e., the geometric "graph")
- proof: assume it is disconnected
  - $\Rightarrow$ there is a site $s_1 \in S$, such that $\mathcal{V}(s_1)$ separates the plane
  - $\mathcal{V}(s_1)$ convex $\Rightarrow$ must be bounded by two parallel lines
  - let $s_1 \perp s_2$ be one of these lines and let $s_3 \in S$ be non-collinear
  - $\Rightarrow s_1 \perp s_2$ intersects $s_1 \perp s_3$



Basic Properties



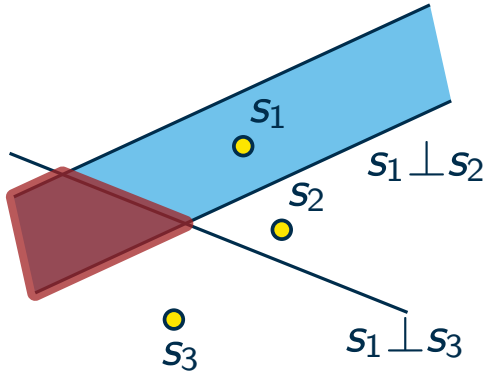
Points Belonging To The Cell Of s

- Why does p not lie in the cell of s?
- s' is closer $p \Leftrightarrow s \perp s'$ separates s from p
- the cell of s is the intersection of half planes
- this already yields a (slow) algorithm

How slow exactly?

Connectedness

- not all sites collinear $\Rightarrow$ $\mathrm{Vor}(S)$ is connected (viewing $\mathrm{Vor}(S)$ as the union of edges, i.e., the geometric "graph")
- proof: assume it is disconnected
  - $\Rightarrow$ there is a site $s_1 \in S$, such that $\mathcal{V}(s_1)$ separates the plane
  - $\mathcal{V}(s_1)$ convex $\Rightarrow$ must be bounded by two parallel lines
  - let $s_1 \perp s_2$ be one of these lines and let $s_3 \in S$ be non-collinear
  - $\Rightarrow$ $s_1 \perp s_2$ intersects $s_1 \perp s_3$ $\Rightarrow$ contradiction



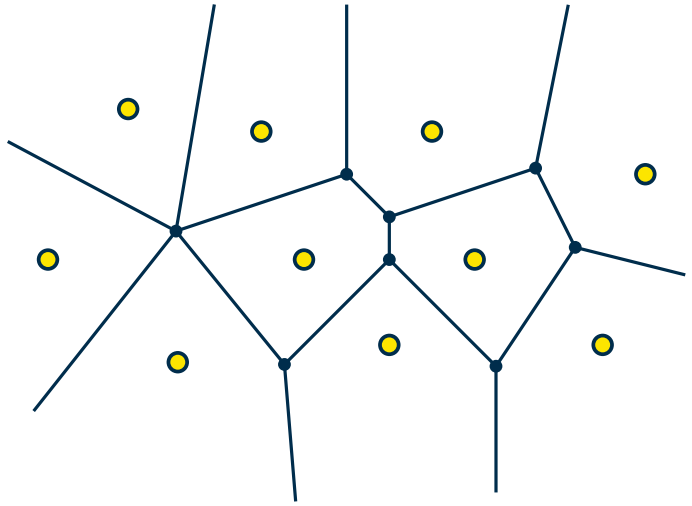
Basic Properties (2)

Theorem **(proof: exercise)**
The Voronoi diagram of $n \geq 3$ sites has at most $2n - 5$ vertices and at most $3n - 6$ edges.

Basic Properties (2)

Theorem **(proof: exercise)**
The Voronoi diagram of $n \geq 3$ sites has at most $2n - 5$ vertices and at most $3n - 6$ edges.

Which Points Are Vertices Of Vor(S)?

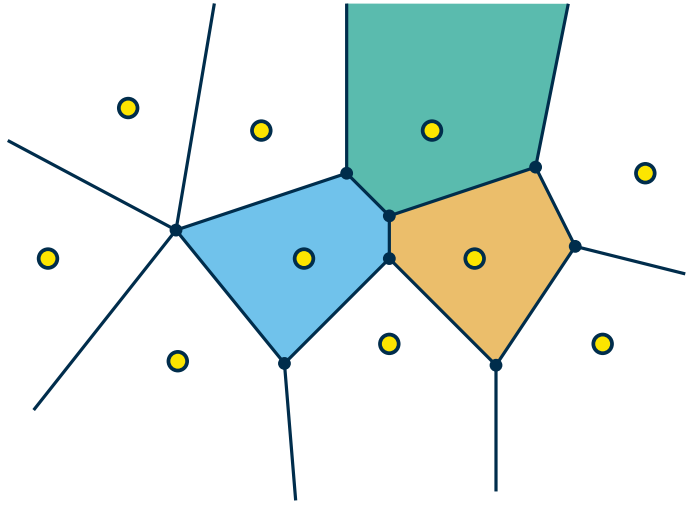


Basic Properties (2)

Theorem **(proof: exercise)**
The Voronoi diagram of $n \geq 3$ sites has at most $2n - 5$ vertices and at most $3n - 6$ edges.

Which Points Are Vertices Of $\mathrm{Vor}(S)$?

- at each a vertex v: ≥ 3 cells meet

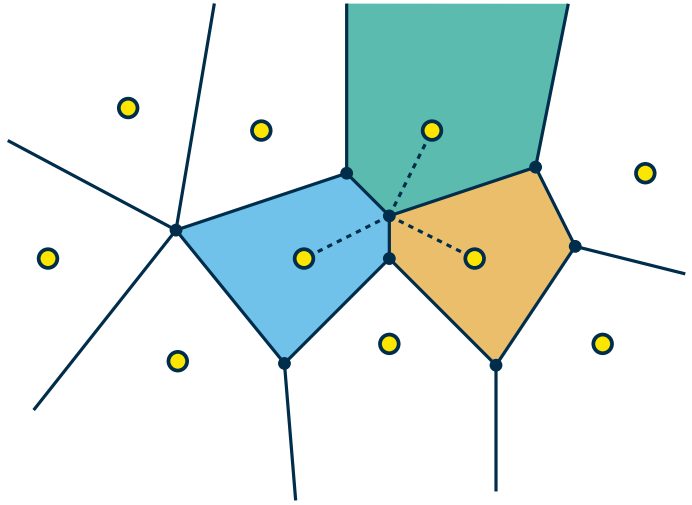


Basic Properties (2)

Theorem **(proof: exercise)**
The Voronoi diagram of $n \geq 3$ sites has at most $2n - 5$ vertices and at most $3n - 6$ edges.

Which Points Are Vertices Of Vor(S)?

- at each a vertex v: ≥ 3 cells meet
- v has equal distance to these sites

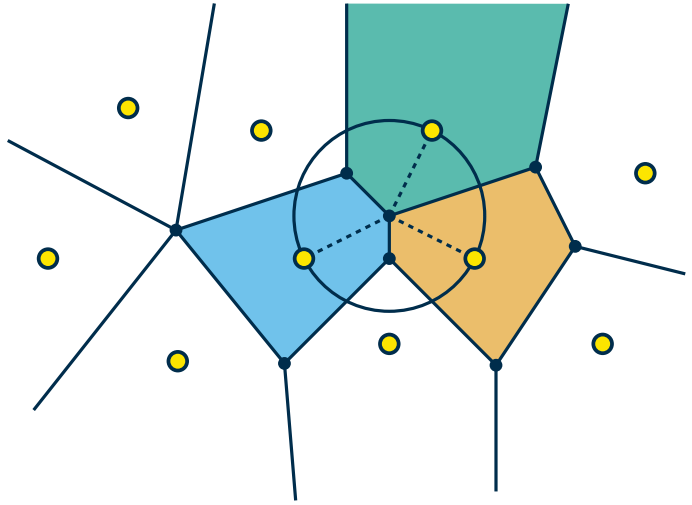


Basic Properties (2)

Theorem **(proof: exercise)**
The Voronoi diagram of $n \geq 3$ sites has at most $2n - 5$ vertices and at most $3n - 6$ edges.

Which Points Are Vertices Of Vor(S)?

- at each a vertex v: ≥ 3 cells meet
- v has equal distance to these sites $\Rightarrow$ circle with center v

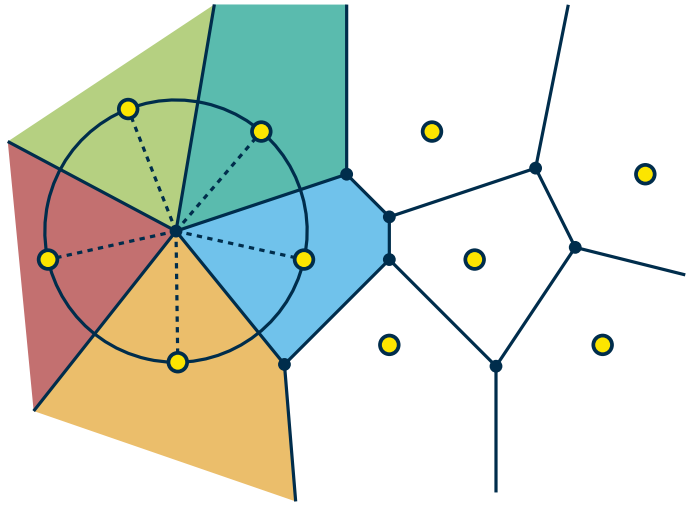


Basic Properties (2)

Theorem **(proof: exercise)**
The Voronoi diagram of $n \geq 3$ sites has at most $2n - 5$ vertices and at most $3n - 6$ edges.

Which Points Are Vertices Of Vor(S)?

- at each a vertex v: ≥ 3 cells meet
- v has equal distance to these sites $\Rightarrow$ circle with center v

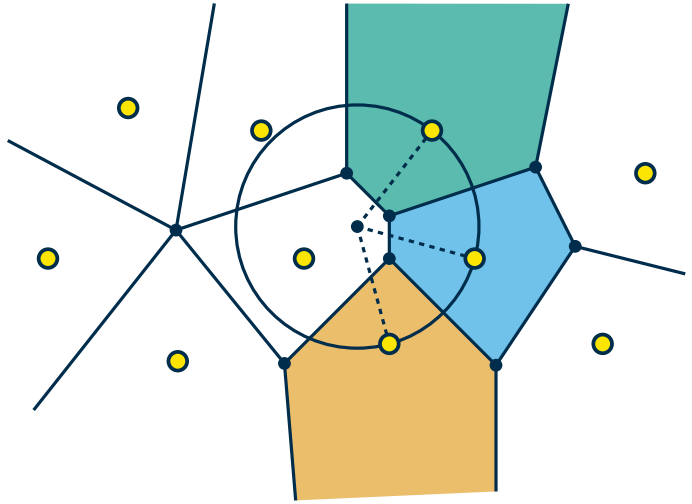


Basic Properties (2)

Theorem **(proof: exercise)**
The Voronoi diagram of $n \geq 3$ sites has at most $2n - 5$ vertices and at most $3n - 6$ edges.

Which Points Are Vertices Of Vor(S)?

- at each a vertex v: ≥ 3 cells meet
- v has equal distance to these sites $\Rightarrow$ circle with center v
- not every circle through ≥ 3 sites has vertex as center

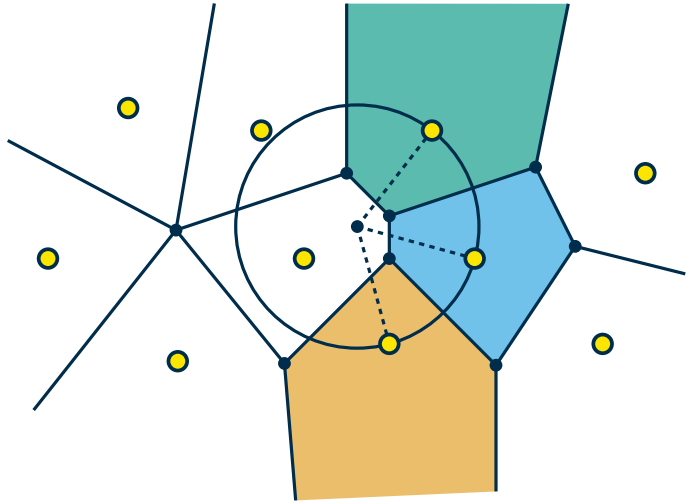


Basic Properties (2)

Theorem **(proof: exercise)**
The Voronoi diagram of $n \geq 3$ sites has at most $2n - 5$ vertices and at most $3n - 6$ edges.

Which Points Are Vertices Of Vor(S)?

- at each a vertex v: ≥ 3 cells meet
- v has equal distance to these sites $\Rightarrow$ circle with center v
- not every circle through ≥ 3 sites has vertex as center
- the circle cannot have any sites in its interior

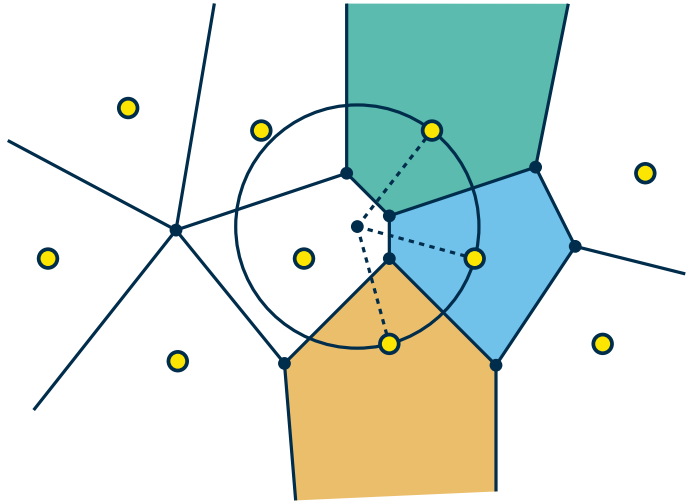


Basic Properties (2)

Theorem **(proof: exercise)**
The Voronoi diagram of $n \geq 3$ sites has at most $2n - 5$ vertices and at most $3n - 6$ edges.

Which Points Are Vertices Of Vor(S)?

- at each a vertex v: ≥ 3 cells meet
- v has equal distance to these sites $\Rightarrow$ circle with center v
- not every circle through ≥ 3 sites has vertex as center
- the circle cannot have any sites in its interior
- $C_S(p) =$ largest circle around p, with no site from S in its interior

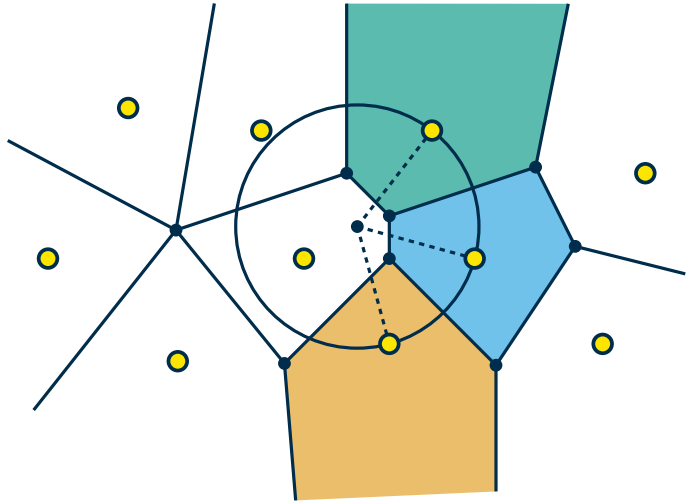


Basic Properties (2)

Theorem **(proof: exercise)**
The Voronoi diagram of $n \geq 3$ sites has at most $2n - 5$ vertices and at most $3n - 6$ edges.

Which Points Are Vertices Of Vor(S)?

- at each a vertex v: ≥ 3 cells meet
- v has equal distance to these sites $\Rightarrow$ circle with center v
- not every circle through ≥ 3 sites has vertex as center
- the circle cannot have any sites in its interior
- $C_S(p) =$ largest circle around p, with no site from S in its interior



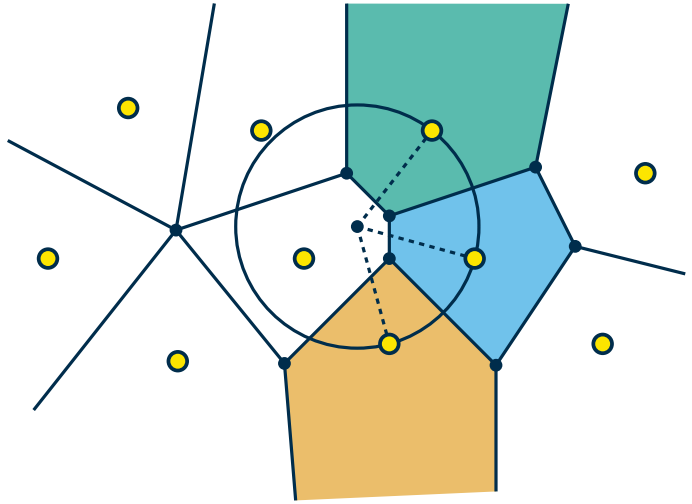
Theorem **(vertices of Vor(S))**
The point $p \in \mathbb{R}^2$ is a vertex of $\mathrm{Vor}(S)$ if and only if at least three sites lie on the circle $C_S(p)$.

Basic Properties (2)

Theorem **(proof: exercise)**
The Voronoi diagram of $n \geq 3$ sites has at most $2n - 5$ vertices and at most $3n - 6$ edges.

Which Points Are Vertices Of Vor(S)?

- at each a vertex v: ≥ 3 cells meet
- v has equal distance to these sites $\Rightarrow$ circle with center v
- not every circle through ≥ 3 sites has vertex as center
- the circle cannot have any sites in its interior
- $C_S(p) =$ largest circle around p, with no site from S in its interior



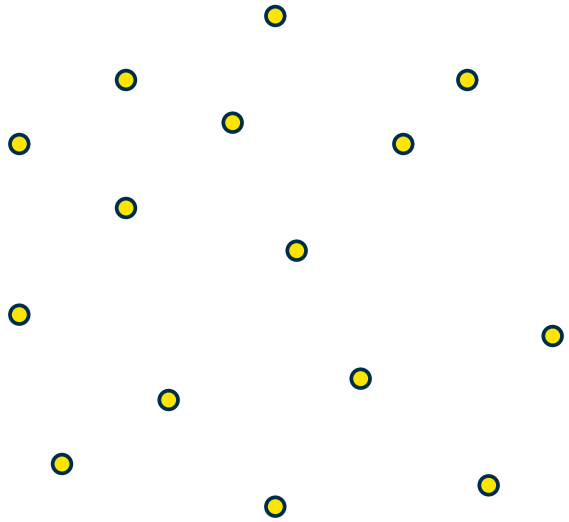
Theorem **(vertices of Vor(S))**
The point $p \in \mathbb{R}^2$ is a vertex of $\mathrm{Vor}(S)$ if and only if at least three sites lie on the circle $C_S(p)$.

Which similar statement holds for edges?

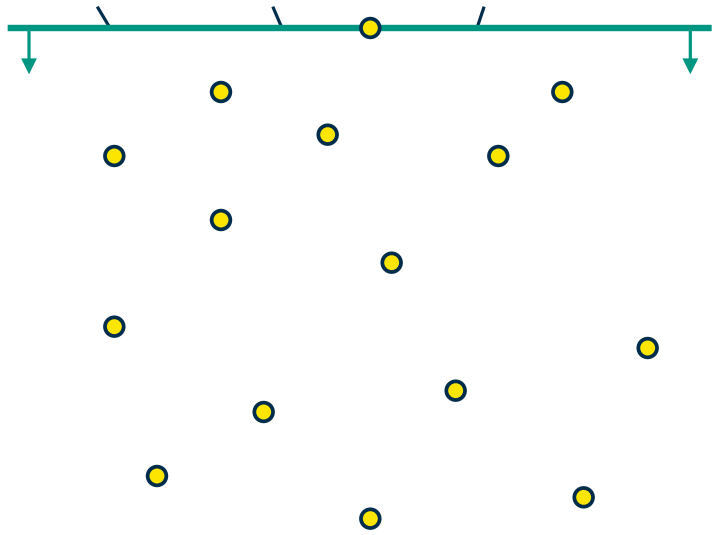
Sweep-Line Algo

General Approach: Sweep Line

- move sweep line from top to bottom
- at every point in time: know $\mathrm{Vor}(S)$ above the sweep line



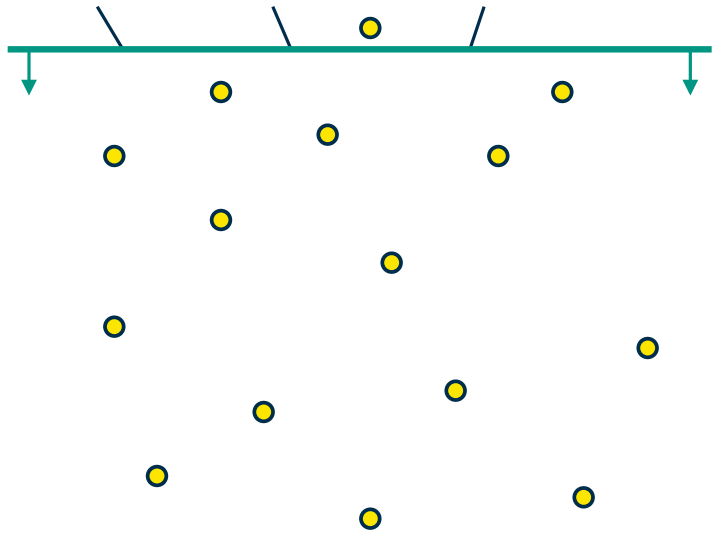
Sweep-Line Algo



General Approach: Sweep Line

- move sweep line from top to bottom
- at every point in time: know $\mathrm{Vor}(S)$ above the sweep line

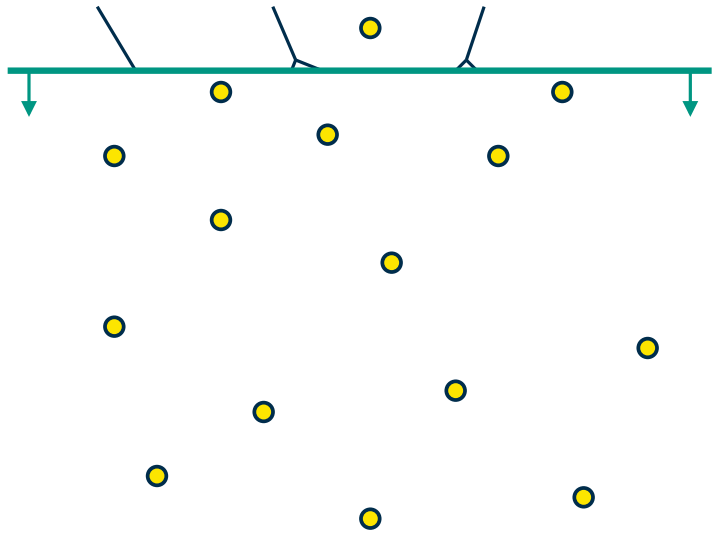
Sweep-Line Algo



General Approach: Sweep Line

- move sweep line from top to bottom
- at every point in time: know $\mathrm{Vor}(S)$ above the sweep line

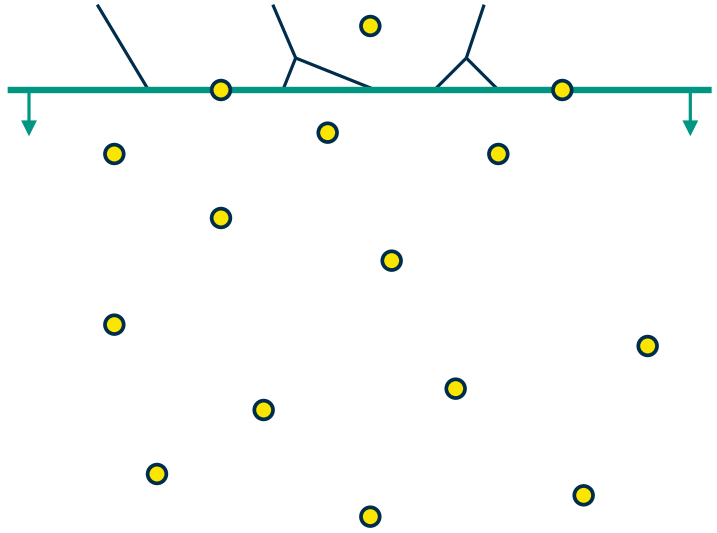
Sweep-Line Algo



General Approach: Sweep Line

- move sweep line from top to bottom
- at every point in time: know $\mathrm{Vor}(S)$ above the sweep line

Sweep-Line Algo



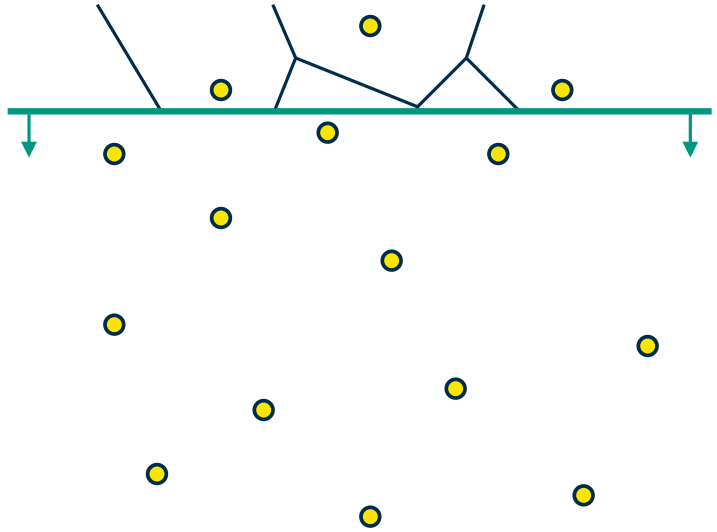
General Approach: Sweep Line

- move sweep line from top to bottom
- at every point in time: know $\mathrm{Vor}(S)$ above the sweep line

Sweep-Line Algo

General Approach: Sweep Line

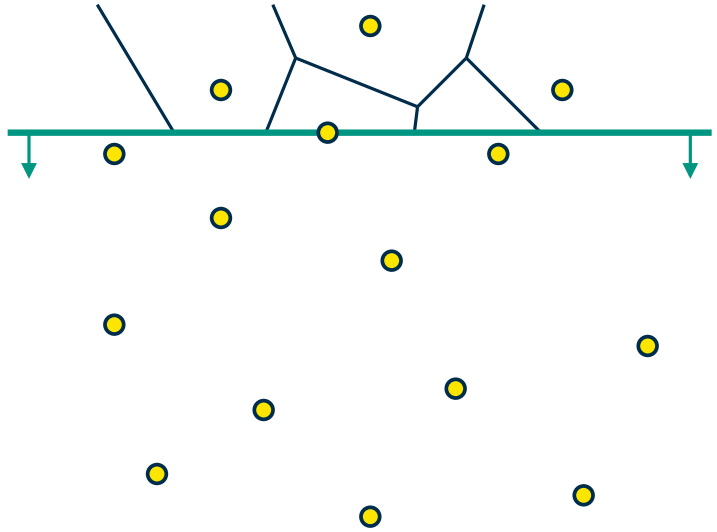
- move sweep line from top to bottom
- at every point in time: know $\mathrm{Vor}(S)$ above the sweep line



Sweep-Line Algo

General Approach: Sweep Line

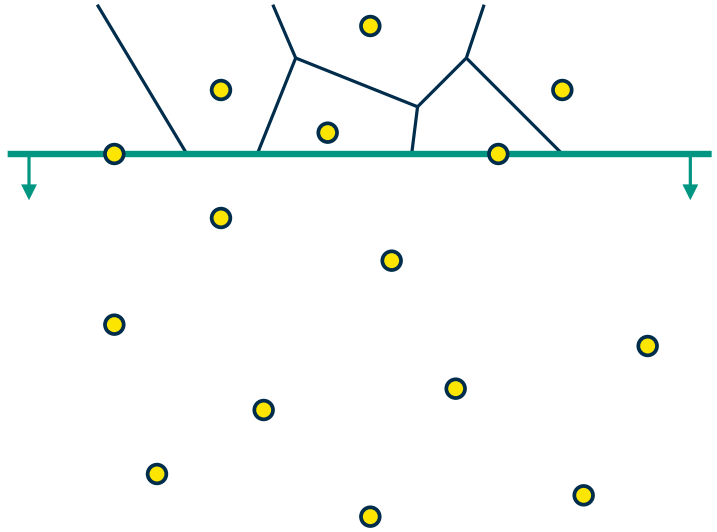
- move sweep line from top to bottom
- at every point in time: know $\mathrm{Vor}(S)$ above the sweep line



Sweep-Line Algo

General Approach: Sweep Line

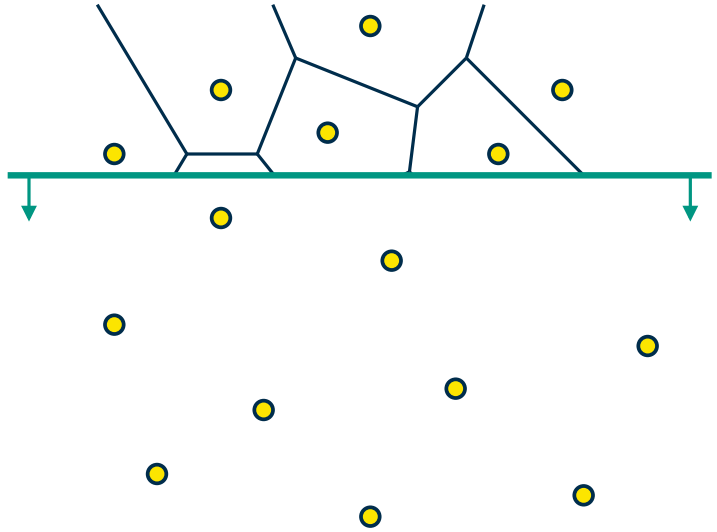
- move sweep line from top to bottom
- at every point in time: know $\mathrm{Vor}(S)$ above the sweep line



Sweep-Line Algo

General Approach: Sweep Line

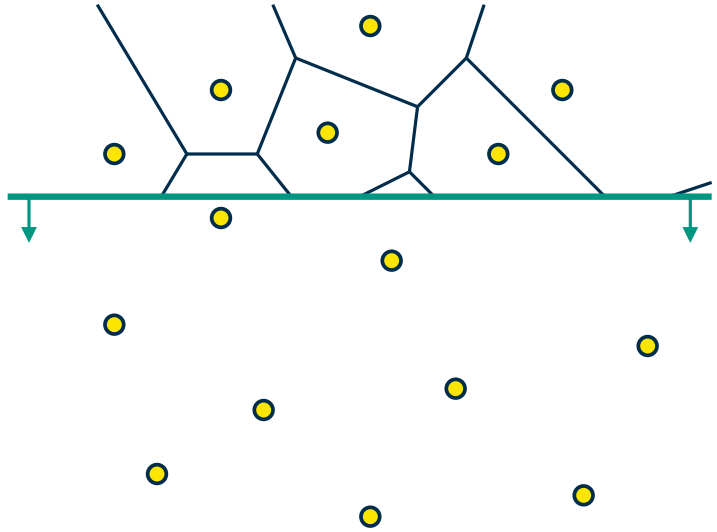
- move sweep line from top to bottom
- at every point in time: know $\mathrm{Vor}(S)$ above the sweep line



Sweep-Line Algo

General Approach: Sweep Line

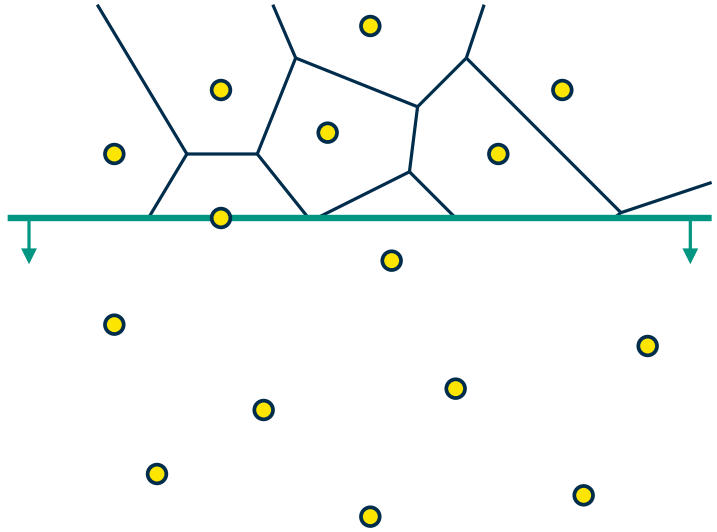
- move sweep line from top to bottom
- at every point in time: know $\mathrm{Vor}(S)$ above the sweep line



Sweep-Line Algo

General Approach: Sweep Line

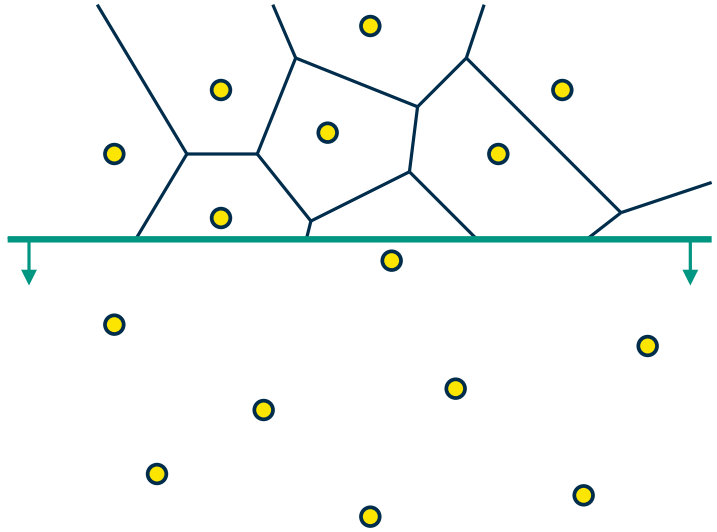
- move sweep line from top to bottom
- at every point in time: know $\mathrm{Vor}(S)$ above the sweep line



Sweep-Line Algo

General Approach: Sweep Line

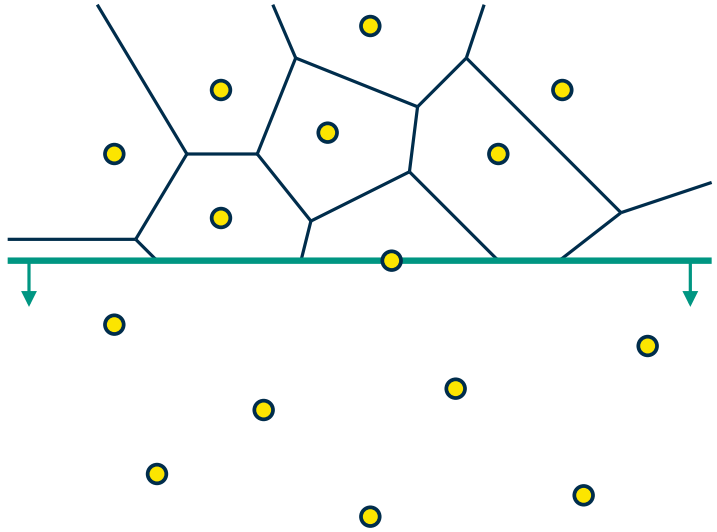
- move sweep line from top to bottom
- at every point in time: know $\mathrm{Vor}(S)$ above the sweep line



Sweep-Line Algo

General Approach: Sweep Line

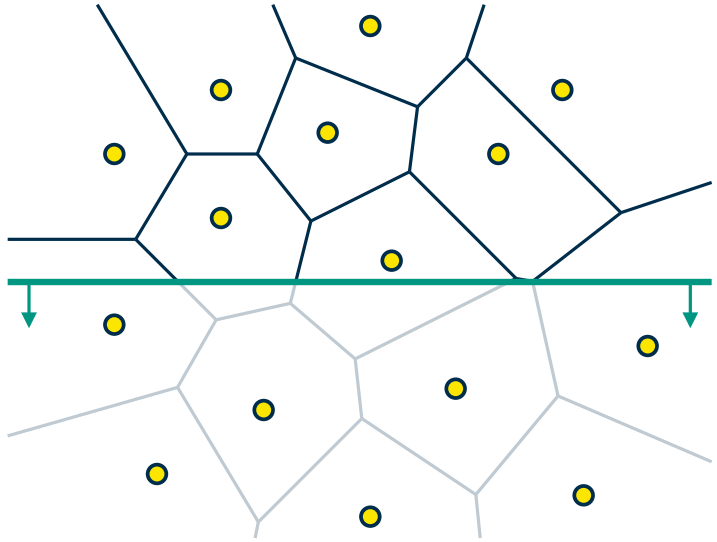
- move sweep line from top to bottom
- at every point in time: know $\mathrm{Vor}(S)$ above the sweep line



Sweep-Line Algo

General Approach: Sweep Line

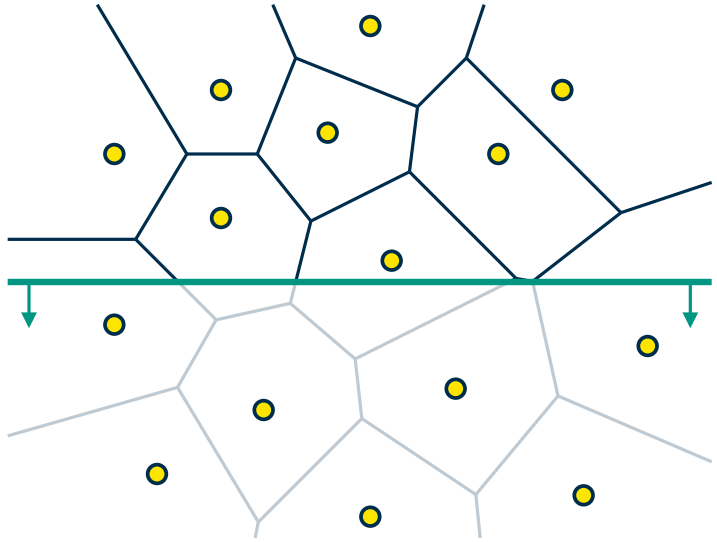
- move sweep line from top to bottom
- at every point in time: know $\mathrm{Vor}(S)$ above the sweep line



Sweep-Line Algo

General Approach: Sweep Line

- move sweep line from top to bottom
- at every point in time: know $\mathrm{Vor}(S)$ above the sweep line

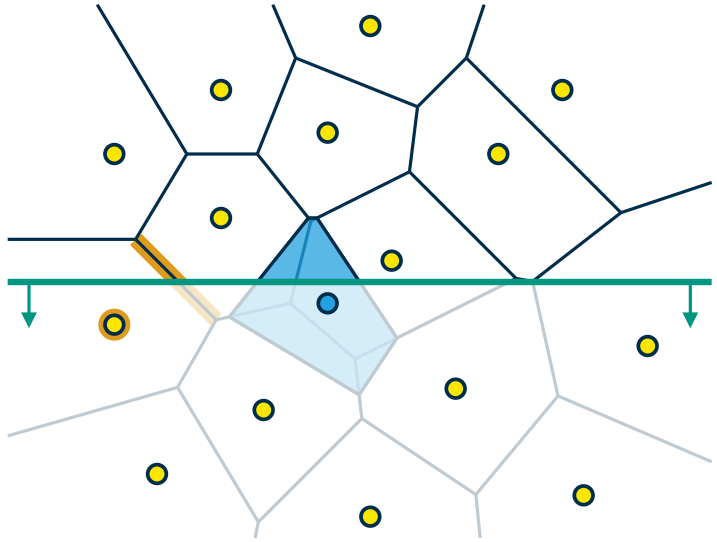


Problem

Sweep-Line Algo

General Approach: Sweep Line

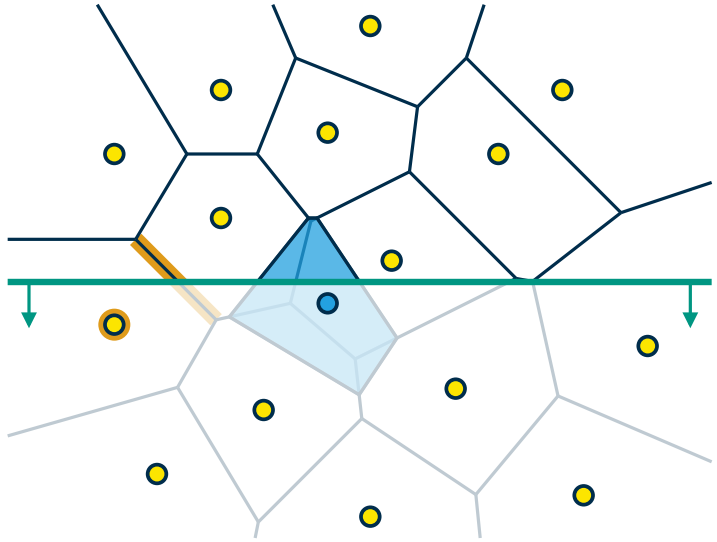
- move sweep line from top to bottom
- at every point in time: know $\mathrm{Vor}(S)$ above the sweep line



Problem

- parts above the sweep line depend on sites below the sweep line

Sweep-Line Algo



General Approach: Sweep Line

- move sweep line from top to bottom
- at every point in time: know $\mathrm{Vor}(S)$ above the sweep line

Problem

- parts above the sweep line depend on sites below the sweep line
- that (hopefully) happens only slightly above the sweep line!?!

Sweep-Line Algo

General Approach: Sweep Line

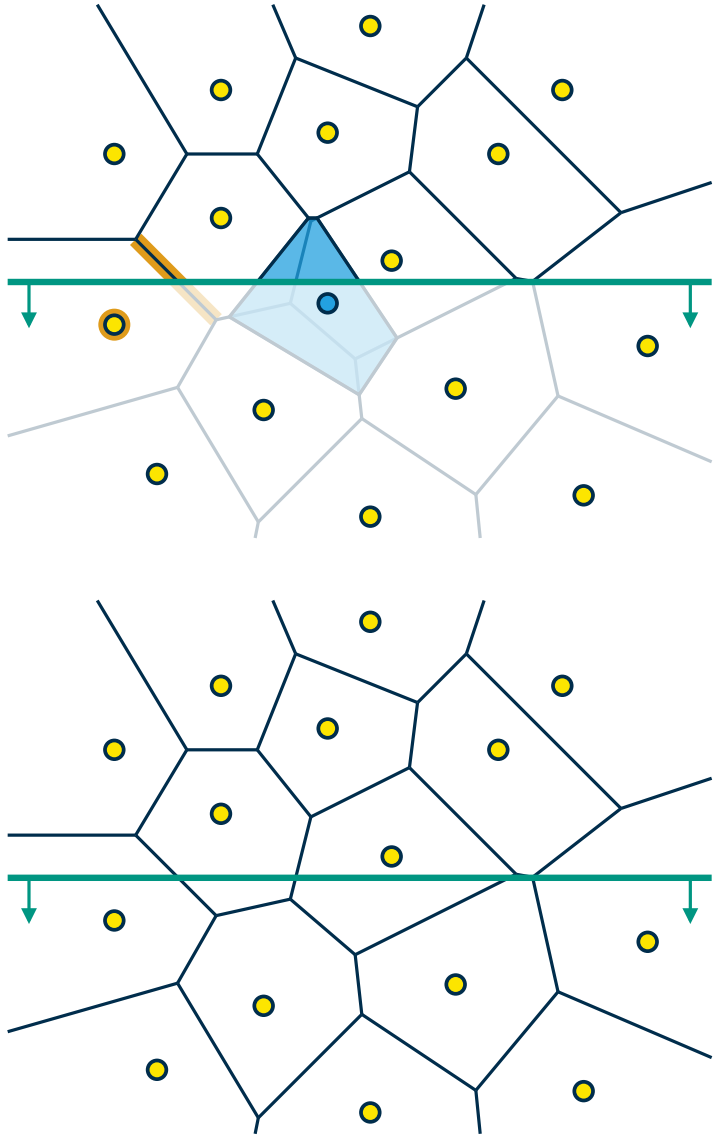
- move sweep line from top to bottom
- at every point in time: know $\mathrm{Vor}(S)$ above the sweep line

Problem

- parts above the sweep line depend on sites below the sweep line
- that (hopefully) happens only slightly above the sweep line!?!

Adjusted Sweep-Line Approach

- region of seen sites $\neq$ region in which $\mathrm{Vor}(S)$ is computed



Sweep-Line Algo

General Approach: Sweep Line

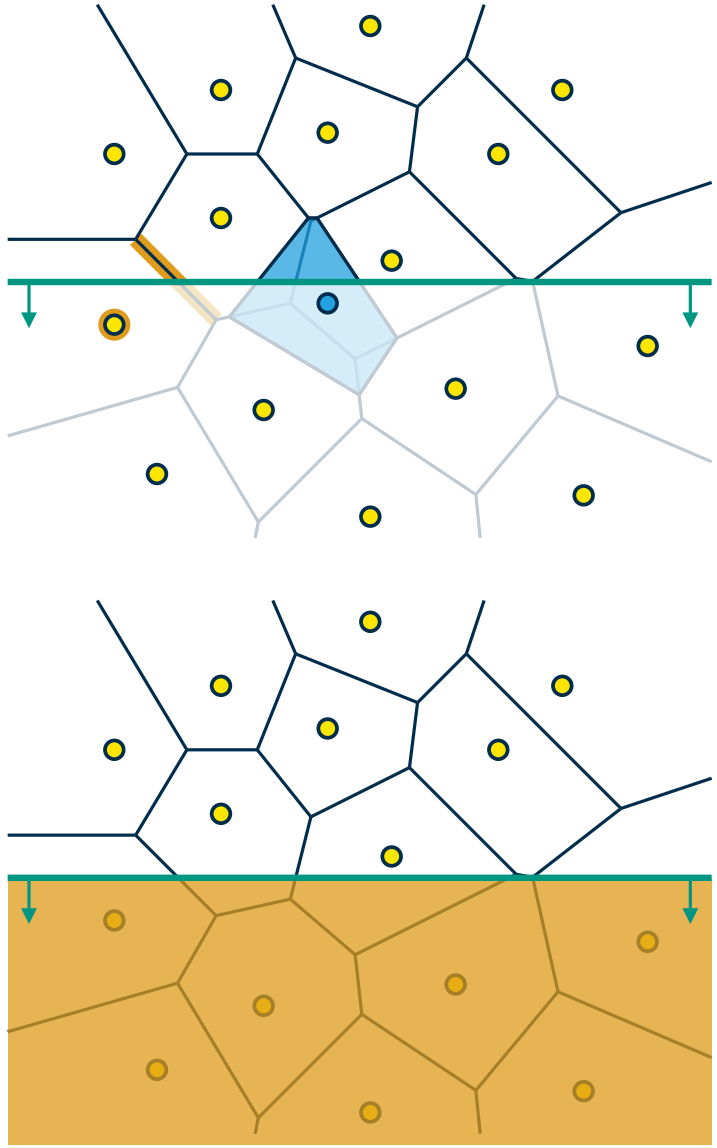
- move sweep line from top to bottom
- at every point in time: know $\mathrm{Vor}(S)$ above the sweep line

Problem

- parts above the sweep line depend on sites below the sweep line
- that (hopefully) happens only slightly above the sweep line!?!

Adjusted Sweep-Line Approach

- region of seen sites $\neq$ region in which $\mathrm{Vor}(S)$ is computed
- below the sweep line: unknown area



Sweep-Line Algo

General Approach: Sweep Line

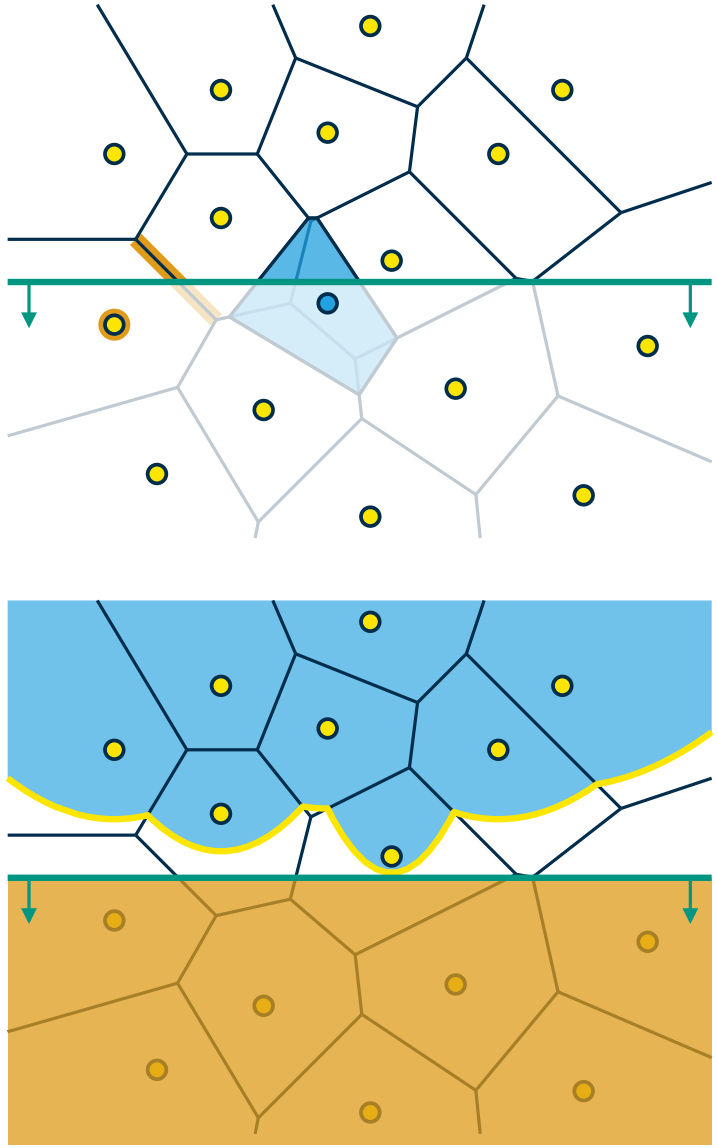
- move sweep line from top to bottom
- at every point in time: know $\mathrm{Vor}(S)$ above the sweep line

Problem

- parts above the sweep line depend on sites below the sweep line
- that (hopefully) happens only slightly above the sweep line!?!

Adjusted Sweep-Line Approach

- region of seen sites $\neq$ region in which $\mathrm{Vor}(S)$ is computed
- below the sweep line: unknown area
- some region above the sweep line: known area



Sweep-Line Algo

General Approach: Sweep Line

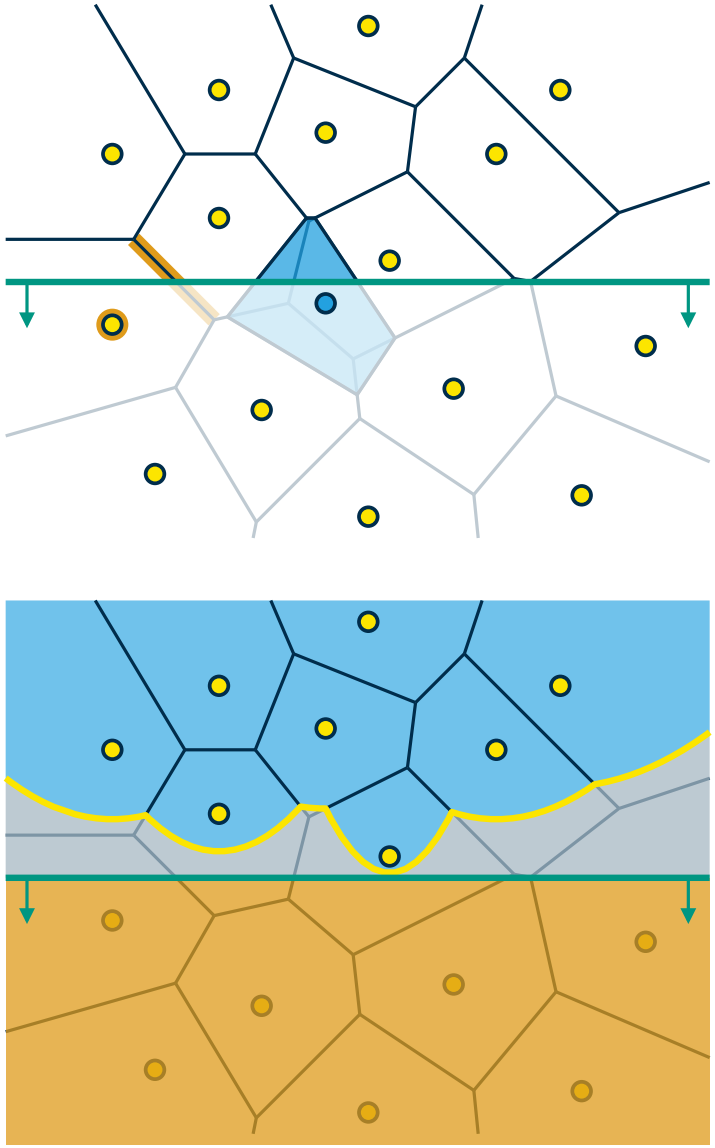
- move sweep line from top to bottom
- at every point in time: know $\mathrm{Vor}(S)$ above the sweep line

Problem

- parts above the sweep line depend on sites below the sweep line
- that (hopefully) happens only slightly above the sweep line!?!

Adjusted Sweep-Line Approach

- region of seen sites $\neq$ region in which $\mathrm{Vor}(S)$ is computed
- below the sweep line: unknown area
- some region above the sweep line: known area
- some gray area in between



Known Area

Where Does The Known Area End?

- consider a point $p \in \mathbb{R}^2$ above ℓ

Known Area

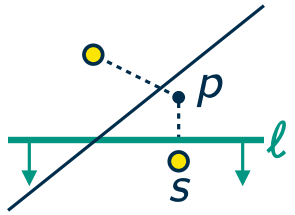
Where Does The Known Area End?

- consider a point $p \in \mathbb{R}^2$ above ℓ
- case 1: p is closer to ℓ than to any site above ℓ

Known Area

Where Does The Known Area End?

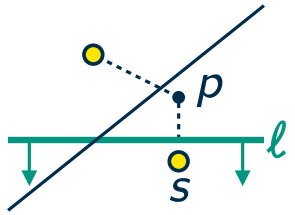
- consider a point $p \in \mathbb{R}^2$ above ℓ
- case 1: p is closer to ℓ than to any site above ℓ
  - there could be a site s below ℓ, such that $p \in \mathcal{V}(s)$



Known Area

Where Does The Known Area End?

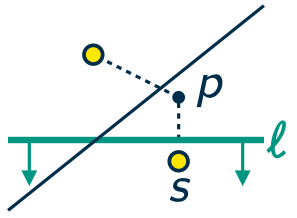
- consider a point $p \in \mathbb{R}^2$ above ℓ
- case 1: p is closer to ℓ than to any site above ℓ
  - there could be a site s below ℓ, such that $p \in \mathcal{V}(s)$
  - thus, p is part of the gray area

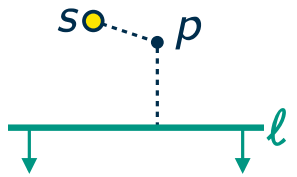


Known Area

Where Does The Known Area End?

- consider a point $p \in \mathbb{R}^2$ above ℓ
- case 1: p is closer to ℓ than to any site above ℓ
  - there could be a site s below ℓ, such that $p \in \mathcal{V}(s)$
  - thus, p is part of the gray area
- case 2: p is closer to a site s above ℓ as to ℓ

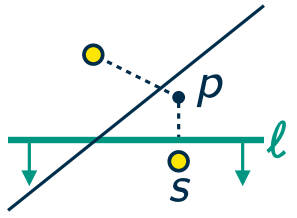


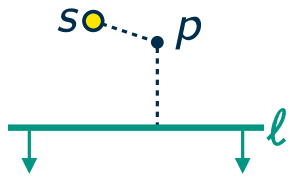


Known Area

Where Does The Known Area End?

- consider a point $p \in \mathbb{R}^2$ above ℓ
- case 1: p is closer to ℓ than to any site above ℓ
  - there could be a site s below ℓ, such that $p \in \mathcal{V}(s)$
  - thus, p is part of the gray area
- case 2: p is closer to a site s above ℓ as to ℓ
  - there is no site below ℓ that is closer to p
  - thus, we know the Voronoi cell of p

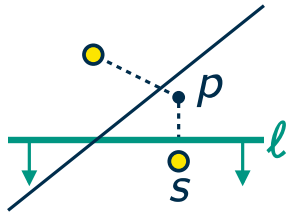


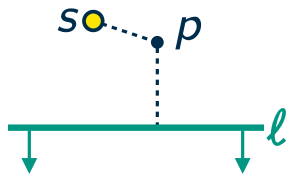


Known Area

Where Does The Known Area End?

- consider a point $p \in \mathbb{R}^2$ above ℓ
- case 1: p is closer to ℓ than to any site above ℓ
  - there could be a site s below ℓ, such that $p \in \mathcal{V}(s)$
  - thus, p is part of the gray area
- case 2: p is closer to a site s above ℓ as to ℓ
  - there is no site below ℓ that is closer to p
  - thus, we know the Voronoi cell of p
- known area: $\{p \in \mathbb{R}^2 \mid \exists s \in S \ \ \mathrm{dist}(p, s) \leq \mathrm{dist}(p, \ell)\}$

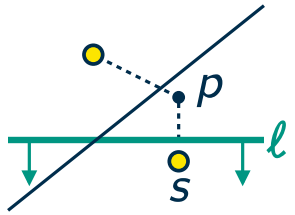


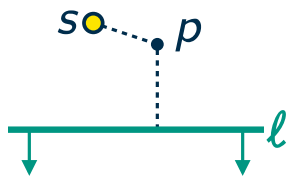


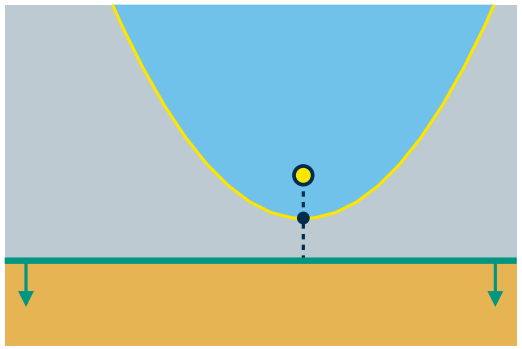
Known Area

Where Does The Known Area End?

- consider a point $p \in \mathbb{R}^2$ above ℓ
- case 1: p is closer to ℓ than to any site above ℓ
  - there could be a site s below ℓ, such that $p \in \mathcal{V}(s)$
  - thus, p is part of the gray area
- case 2: p is closer to a site s above ℓ as to ℓ
  - there is no site below ℓ that is closer to p
  - thus, we know the Voronoi cell of p
- known area: $\{p \in \mathbb{R}^2 \mid \exists s \in S \ \ \mathrm{dist}(p,s) \leq \mathrm{dist}(p,\ell)\}$



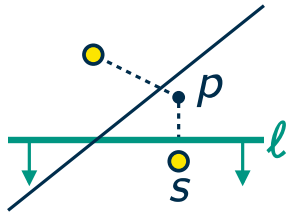


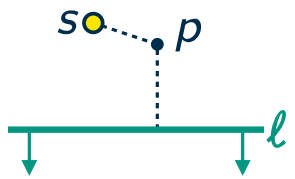


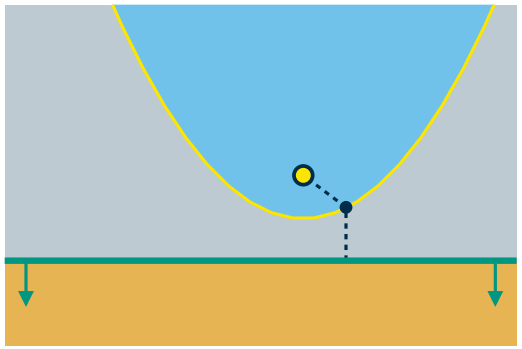
Known Area

Where Does The Known Area End?

- consider a point $p \in \mathbb{R}^2$ above ℓ
- case 1: p is closer to ℓ than to any site above ℓ
  - there could be a site s below ℓ, such that $p \in \mathcal{V}(s)$
  - thus, p is part of the gray area
- case 2: p is closer to a site s above ℓ as to ℓ
  - there is no site below ℓ that is closer to p
  - thus, we know the Voronoi cell of p
- known area: $\{p \in \mathbb{R}^2 \mid \exists s \in S \ \ \mathrm{dist}(p,s) \leq \mathrm{dist}(p,\ell)\}$



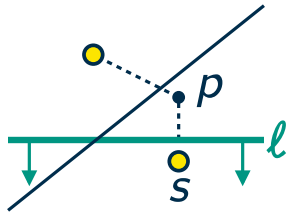


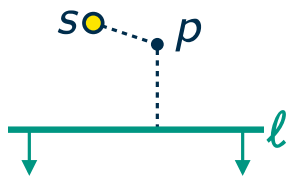


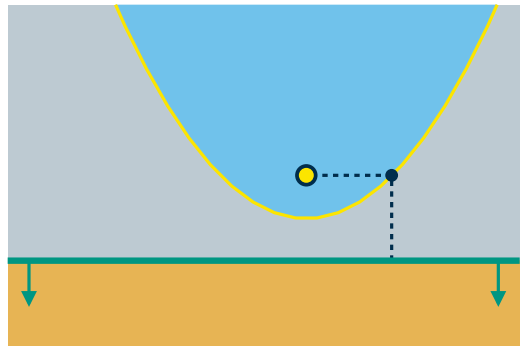
Known Area

Where Does The Known Area End?

- consider a point $p \in \mathbb{R}^2$ above ℓ
- case 1: p is closer to ℓ than to any site above ℓ
  - there could be a site s below ℓ, such that $p \in \mathcal{V}(s)$
  - thus, p is part of the gray area
- case 2: p is closer to a site s above ℓ as to ℓ
  - there is no site below ℓ that is closer to p
  - thus, we know the Voronoi cell of p
- known area: $\{p \in \mathbb{R}^2 \mid \exists s \in S \ \ \mathrm{dist}(p,s) \leq \mathrm{dist}(p,\ell)\}$



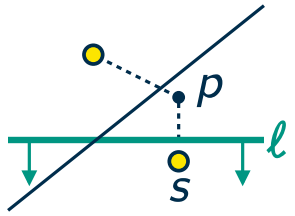


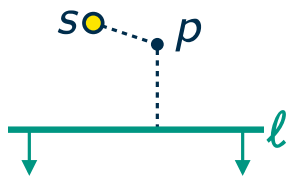


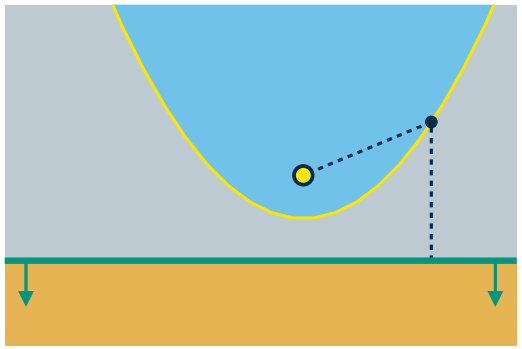
Known Area

Where Does The Known Area End?

- consider a point $p \in \mathbb{R}^2$ above ℓ
- case 1: p is closer to ℓ than to any site above ℓ
  - there could be a site s below ℓ, such that $p \in \mathcal{V}(s)$
  - thus, p is part of the gray area
- case 2: p is closer to a site s above ℓ as to ℓ
  - there is no site below ℓ that is closer to p
  - thus, we know the Voronoi cell of p
- known area: $\{p \in \mathbb{R}^2 \mid \exists s \in S \ \ \mathrm{dist}(p,s) \leq \mathrm{dist}(p,\ell)\}$



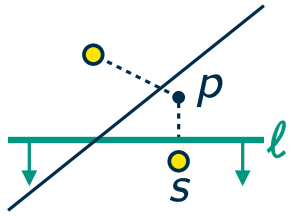


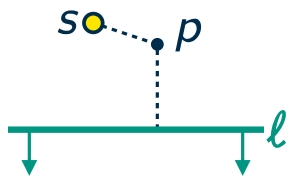


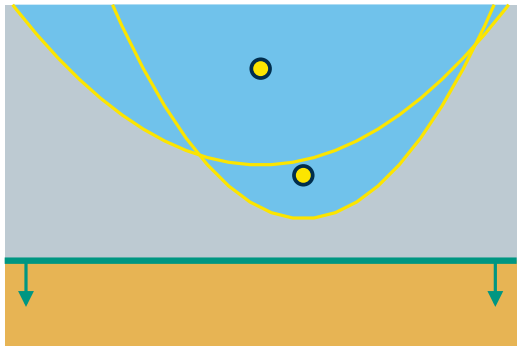
Known Area

Where Does The Known Area End?

- consider a point $p \in \mathbb{R}^2$ above ℓ
- case 1: p is closer to ℓ than to any site above ℓ
  - there could be a site s below ℓ, such that $p \in \mathcal{V}(s)$
  - thus, p is part of the gray area
- case 2: p is closer to a site s above ℓ as to ℓ
  - there is no site below ℓ that is closer to p
  - thus, we know the Voronoi cell of p
- known area: $\{p \in \mathbb{R}^2 \mid \exists s \in S \ \ \mathrm{dist}(p, s) \leq \mathrm{dist}(p, \ell)\}$



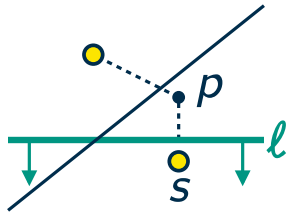


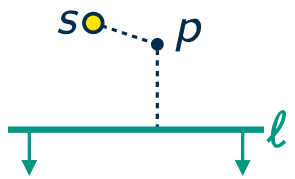


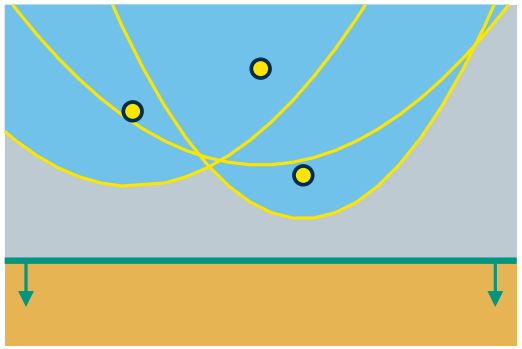
Known Area

Where Does The Known Area End?

- consider a point $p \in \mathbb{R}^2$ above ℓ
- case 1: p is closer to ℓ than to any site above ℓ
  - there could be a site s below ℓ, such that $p \in \mathcal{V}(s)$
  - thus, p is part of the gray area
- case 2: p is closer to a site s above ℓ as to ℓ
  - there is no site below ℓ that is closer to p
  - thus, we know the Voronoi cell of p
- known area: $\{p \in \mathbb{R}^2 \mid \exists s \in S \ \ \mathrm{dist}(p,s) \leq \mathrm{dist}(p,\ell)\}$



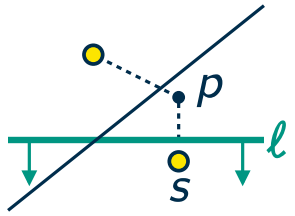


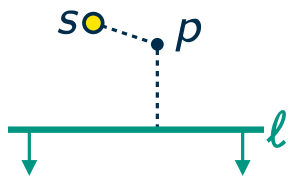


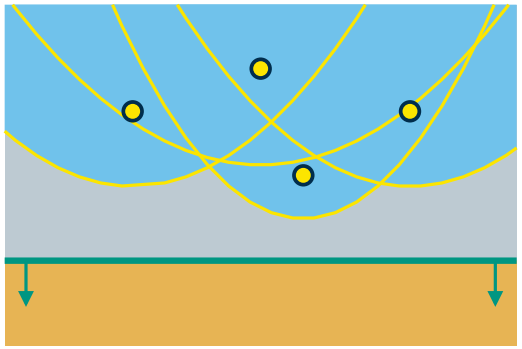
Known Area

Where Does The Known Area End?

- consider a point $p \in \mathbb{R}^2$ above ℓ
- case 1: p is closer to ℓ than to any site above ℓ
  - there could be a site s below ℓ, such that $p \in \mathcal{V}(s)$
  - thus, p is part of the gray area
- case 2: p is closer to a site s above ℓ as to ℓ
  - there is no site below ℓ that is closer to p
  - thus, we know the Voronoi cell of p
- known area: $\{p \in \mathbb{R}^2 \mid \exists s \in S \ \ \mathrm{dist}(p, s) \leq \mathrm{dist}(p, \ell)\}$



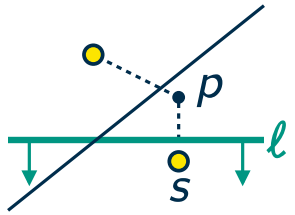


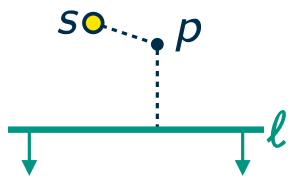


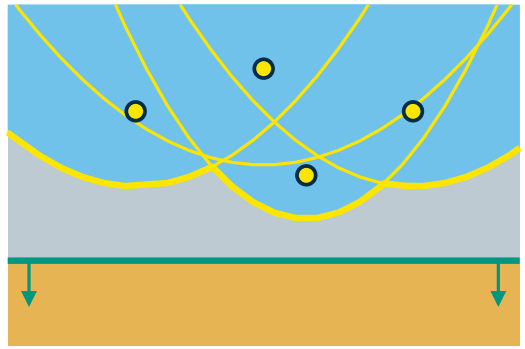
Known Area

Where Does The Known Area End?

- consider a point $p \in \mathbb{R}^2$ above ℓ
- case 1: p is closer to ℓ than to any site above ℓ
  - there could be a site s below ℓ, such that $p \in \mathcal{V}(s)$
  - thus, p is part of the gray area
- case 2: p is closer to a site s above ℓ as to ℓ
  - there is no site below ℓ that is closer to p
  - thus, we know the Voronoi cell of p
- known area: $\{p \in \mathbb{R}^2 \mid \exists s \in S \ \ \mathrm{dist}(p, s) \leq \mathrm{dist}(p, \ell)\}$
- the boundary between known area and gray area is called **beach line**



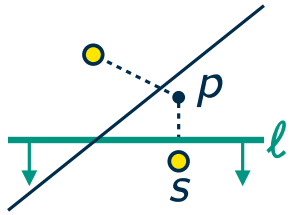


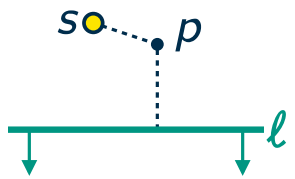


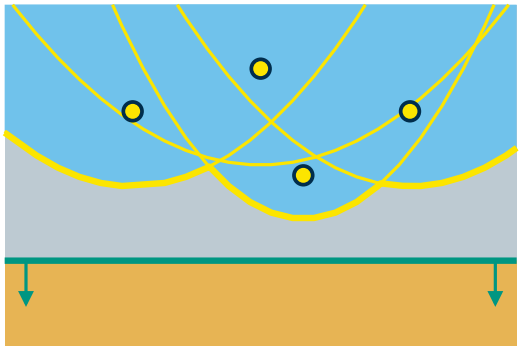
Known Area

Where Does The Known Area End?

- consider a point $p \in \mathbb{R}^2$ above ℓ
- case 1: p is closer to ℓ than to any site above ℓ
  - there could be a site s below ℓ, such that $p \in \mathcal{V}(s)$
  - thus, p is part of the gray area
- case 2: p is closer to a site s above ℓ as to ℓ
  - there is no site below ℓ that is closer to p
  - thus, we know the Voronoi cell of p
- known area: $\{p \in \mathbb{R}^2 \mid \exists s \in S \ \ \mathrm{dist}(p, s) \leq \mathrm{dist}(p, \ell)\}$
- the boundary between known area and gray area is called **beach line**
- each piece of the curve belongs to one site



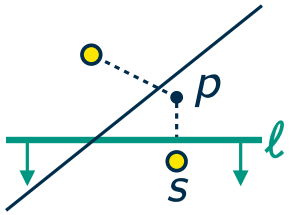


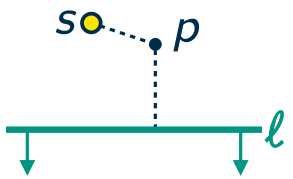


Known Area

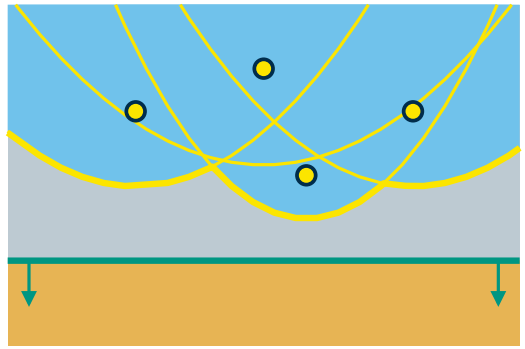
Where Does The Known Area End?

- consider a point $p \in \mathbb{R}^2$ above ℓ
- case 1: p is closer to ℓ than to any site above ℓ
  - there could be a site s below ℓ, such that $p \in \mathcal{V}(s)$
  - thus, p is part of the gray area
- case 2: p is closer to a site s above ℓ as to ℓ
  - there is no site below ℓ that is closer to p
  - thus, we know the Voronoi cell of p





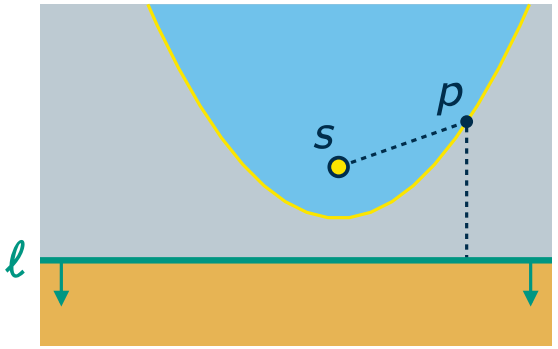
- known area: $\{p \in \mathbb{R}^2 \mid \exists s \in S \ \ \mathrm{dist}(p,s) \leq \mathrm{dist}(p,\ell)\}$
- the boundary between known area and gray area is called **beach line**
- each piece of the curve belongs to one site
- next slide: each piece is a parabola



Shape Of The Beach Line

Relevant Coordinates

- $s = (s_x, s_y)$
- $p = (p_x, p_y)$
- y-Koordinate von ℓ: ℓ_y



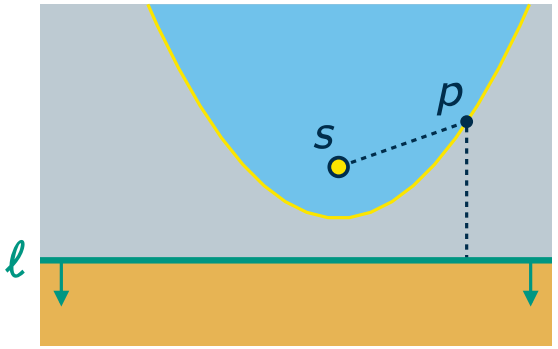
Shape Of The Beach Line

Distances

- $\text{dist}(s, p) = \sqrt{(p_x - s_x)^2 + (p_y - s_y)^2}$ and $\text{dist}(\ell, p) = p_y - \ell_y$

Relevant Coordinates

- $s = (s_x, s_y)$
- $p = (p_x, p_y)$
- y-Koordinate von ℓ: ℓ_y



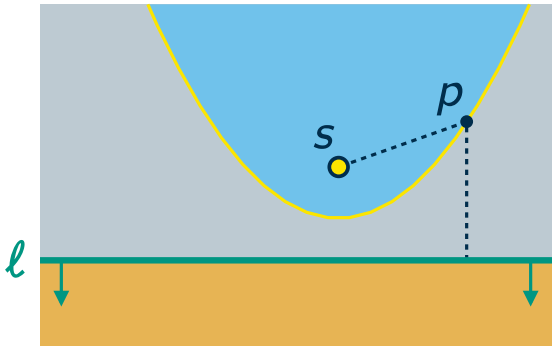
Shape Of The Beach Line

Distances

- $\text{dist}(s, p) = \sqrt{(p_x - s_x)^2 + (p_y - s_y)^2}$ and $\text{dist}(\ell, p) = p_y - \ell_y$
- p lies on the beach line $\Leftrightarrow \text{dist}(s, p) = \text{dist}(\ell, p)$

Relevant Coordinates

- $s = (s_x, s_y)$
- $p = (p_x, p_y)$
- y-Koordinate von ℓ: ℓ_y



Shape Of The Beach Line

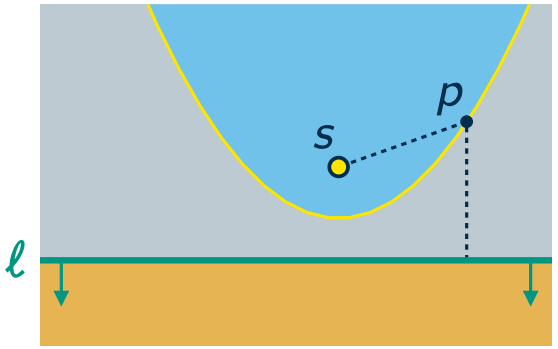
Distances

- $\text{dist}(s,p) = \sqrt{(p_x - s_x)^2 + (p_y - s_y)^2}$ and $\text{dist}(\ell,p) = p_y - \ell_y$
- p lies on the beach line $\Leftrightarrow \text{dist}(s,p) = \text{dist}(\ell,p)$

$$(p_x - s_x)^2 + (p_y - s_y)^2 = (p_y - \ell_y)^2$$

Relevant Coordinates

- $s = (s_x, s_y)$
- $p = (p_x, p_y)$
- y-Koordinate von ℓ: ℓ_y



Shape Of The Beach Line

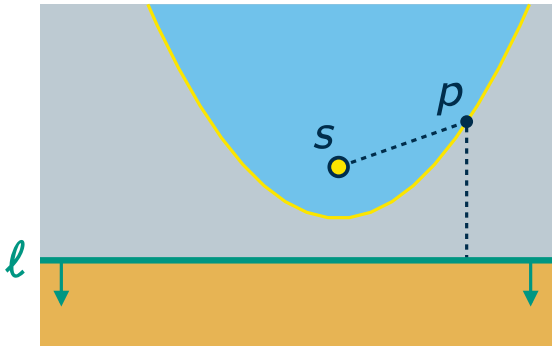
Distances

- $\text{dist}(s,p) = \sqrt{(p_x - s_x)^2 + (p_y - s_y)^2}$ and $\text{dist}(\ell,p) = p_y - \ell_y$
- p lies on the beach line $\Leftrightarrow \text{dist}(s,p) = \text{dist}(\ell,p)$

$$(p_x - s_x)^2 + (p_y - s_y)^2 = (p_y - \ell_y)^2$$
$$(p_x - s_x)^2 + p_y^2 - 2p_y s_y + s_y^2 = p_y^2 - 2p_y \ell_y + \ell_y^2$$

Relevant Coordinates

- $s = (s_x, s_y)$
- $p = (p_x, p_y)$
- y-Koordinate von ℓ: ℓ_y



Shape Of The Beach Line

Distances

- $\text{dist}(s,p) = \sqrt{(p_x - s_x)^2 + (p_y - s_y)^2}$ and $\text{dist}(\ell, p) = p_y - \ell_y$
- p lies on the beach line $\Leftrightarrow \text{dist}(s,p) = \text{dist}(\ell,p)$

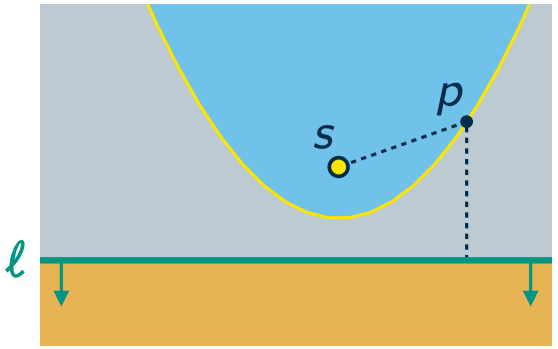
$$(p_x - s_x)^2 + (p_y - s_y)^2 = (p_y - \ell_y)^2$$

$$(p_x - s_x)^2 + p_y^2 - 2p_y s_y + s_y^2 = p_y^2 - 2p_y \ell_y + \ell_y^2$$

$$(p_x - s_x)^2 + s_y^2 - \ell_y^2 = 2p_y s_y - 2p_y \ell_y$$

Relevant Coordinates

- $s = (s_x, s_y)$
- $p = (p_x, p_y)$
- y-Koordinate von ℓ: ℓ_y



Shape Of The Beach Line

Distances

- $\text{dist}(s,p) = \sqrt{(p_x - s_x)^2 + (p_y - s_y)^2}$ and $\text{dist}(\ell, p) = p_y - \ell_y$
- p lies on the beach line $\Leftrightarrow \text{dist}(s,p) = \text{dist}(\ell, p)$

$$(p_x - s_x)^2 + (p_y - s_y)^2 = (p_y - \ell_y)^2$$

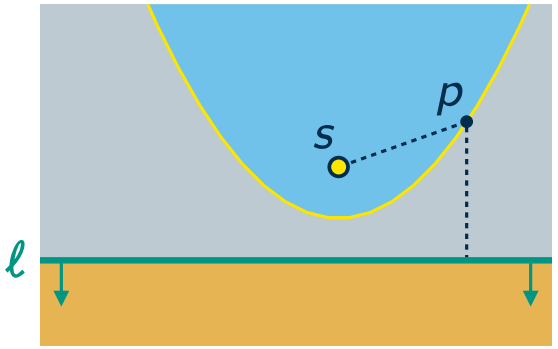
$$(p_x - s_x)^2 + p_y^2 - 2p_y s_y + s_y^2 = p_y^2 - 2p_y \ell_y + \ell_y^2$$

$$(p_x - s_x)^2 + s_y^2 - \ell_y^2 = 2p_y s_y - 2p_y \ell_y$$

$$\frac{(p_x - s_x)^2}{2s_y - 2\ell_y} + \frac{s_y^2 - \ell_y^2}{2s_y - 2\ell_y} = p_y$$

Relevant Coordinates

- $s = (s_x, s_y)$
- $p = (p_x, p_y)$
- y-Koordinate von ℓ: ℓ_y



Shape Of The Beach Line

Distances

- $\text{dist}(s,p) = \sqrt{(p_x - s_x)^2 + (p_y - s_y)^2}$ and $\text{dist}(\ell, p) = p_y - \ell_y$
- p lies on the beach line $\Leftrightarrow \text{dist}(s,p) = \text{dist}(\ell, p)$

$$(p_x - s_x)^2 + (p_y - s_y)^2 = (p_y - \ell_y)^2$$

$$(p_x - s_x)^2 + p_y^2 - 2p_y s_y + s_y^2 = p_y^2 - 2p_y \ell_y + \ell_y^2$$

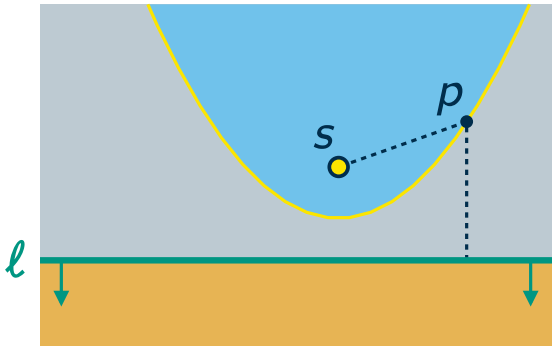
$$(p_x - s_x)^2 + s_y^2 - \ell_y^2 = 2p_y s_y - 2p_y \ell_y$$

$$\frac{(p_x - s_x)^2}{2s_y - 2\ell_y} + \frac{s_y^2 - \ell_y^2}{2s_y - 2\ell_y} = p_y$$

$$\frac{(p_x - s_x)^2}{2s_y - 2\ell_y} + \frac{s_y + \ell_y}{2} = p_y$$

Relevant Coordinates

- $s = (s_x, s_y)$
- $p = (p_x, p_y)$
- y-Koordinate von ℓ: ℓ_y



Shape Of The Beach Line

Distances

- $\text{dist}(s,p) = \sqrt{(p_x - s_x)^2 + (p_y - s_y)^2}$ and $\text{dist}(\ell, p) = p_y - \ell_y$
- p lies on the beach line $\Leftrightarrow \text{dist}(s,p) = \text{dist}(\ell, p)$

$$(p_x - s_x)^2 + (p_y - s_y)^2 = (p_y - \ell_y)^2$$

$$(p_x - s_x)^2 + p_y^2 - 2p_y s_y + s_y^2 = p_y^2 - 2p_y \ell_y + \ell_y^2$$

$$(p_x - s_x)^2 + s_y^2 - \ell_y^2 = 2p_y s_y - 2p_y \ell_y$$

$$\frac{(p_x - s_x)^2}{2s_y - 2\ell_y} + \frac{s_y^2 - \ell_y^2}{2s_y - 2\ell_y} = p_y$$

translation in x-direction

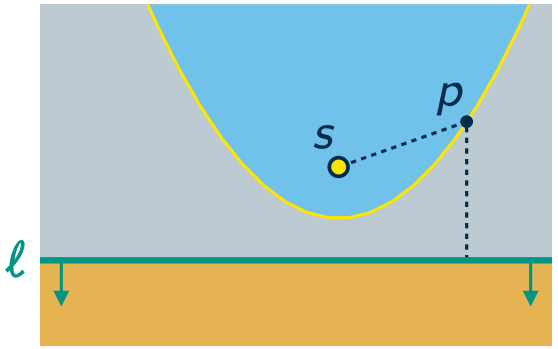
parabola

$$\frac{(p_x - s_x)^2}{2s_y - 2\ell_y} + \frac{s_y + \ell_y}{2} = p_y$$

translation in y-direction

Relevant Coordinates

- $s = (s_x, s_y)$
- $p = (p_x, p_y)$
- y-Koordinate von ℓ: ℓ_y



Shape Of The Beach Line

Distances

- $\text{dist}(s,p) = \sqrt{(p_x - s_x)^2 + (p_y - s_y)^2}$ and $\text{dist}(\ell,p) = p_y - \ell_y$
- p lies on the beach line $\Leftrightarrow \text{dist}(s,p) = \text{dist}(\ell,p)$

$$(p_x - s_x)^2 + (p_y - s_y)^2 = (p_y - \ell_y)^2$$

$$(p_x - s_x)^2 + p_y^2 - 2p_y s_y + s_y^2 = p_y^2 - 2p_y \ell_y + \ell_y^2$$

$$(p_x - s_x)^2 + s_y^2 - \ell_y^2 = 2p_y s_y - 2p_y \ell_y$$

$$\frac{(p_x - s_x)^2}{2s_y - 2\ell_y} + \frac{s_y^2 - \ell_y^2}{2s_y - 2\ell_y} = p_y$$

translation in x-direction

parabola

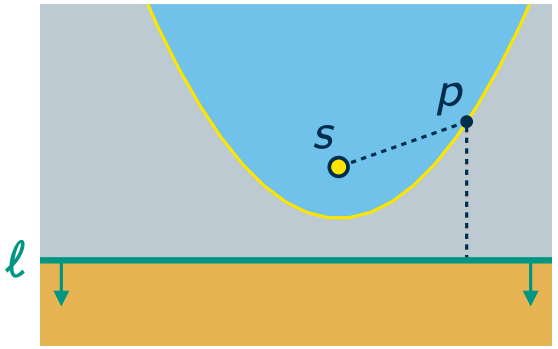
$$\frac{(p_x - s_x)^2}{2s_y - 2\ell_y} + \frac{s_y + \ell_y}{2} = p_y$$

translation in y-direction

- parabola with vertex at the center between s and ℓ
- larger $\text{dist}(s,\ell)$ yields a wider parabola

Relevant Coordinates

- $s = (s_x, s_y)$
- $p = (p_x, p_y)$
- y-Koordinate von ℓ: ℓ_y



When Does The Beach Line Change?

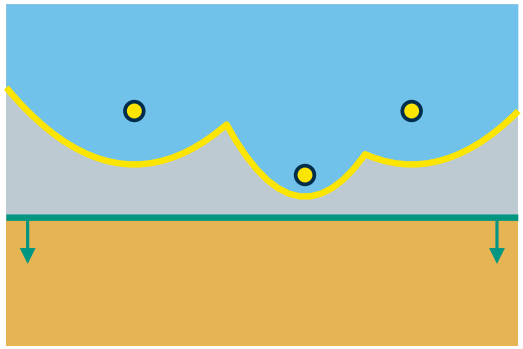
Sweep-Line Algo

- sweep-line status: state of the beach line
- events: points in time, when BL changes

When Does The Beach Line Change?

Sweep-Line Algo

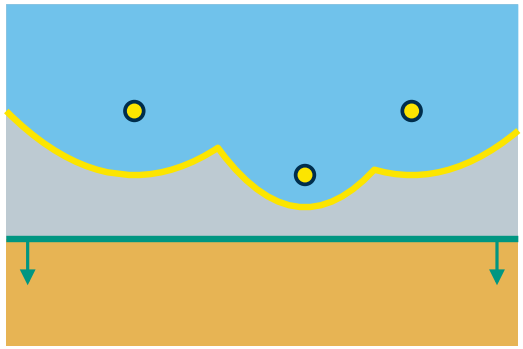
- sweep-line status: state of the beach line
- events: points in time, when BL changes
- What does “change” mean hear?



When Does The Beach Line Change?

Sweep-Line Algo

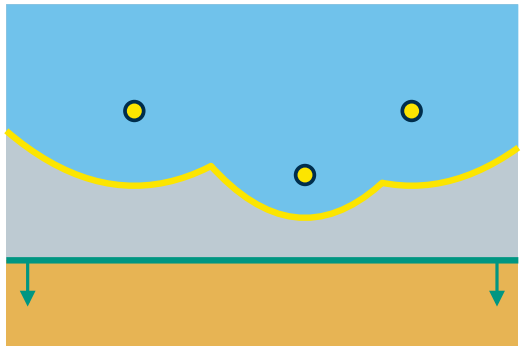
- sweep-line status: state of the beach line
- events: points in time, when BL changes
- What does “change” mean hear?



When Does The Beach Line Change?

Sweep-Line Algo

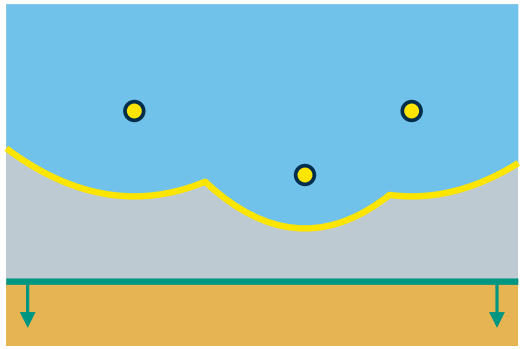
- sweep-line status: state of the beach line
- events: points in time, when BL changes
- What does “change” mean hear?



When Does The Beach Line Change?

Sweep-Line Algo

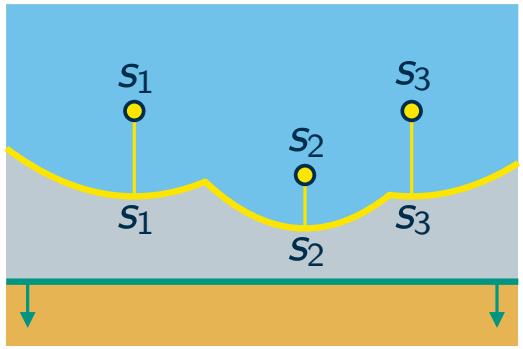
- sweep-line status: state of the beach line
- events: points in time, when BL changes
- What does “change” mean hear?
  - beach line changes all the time



When Does The Beach Line Change?

Sweep-Line Algo

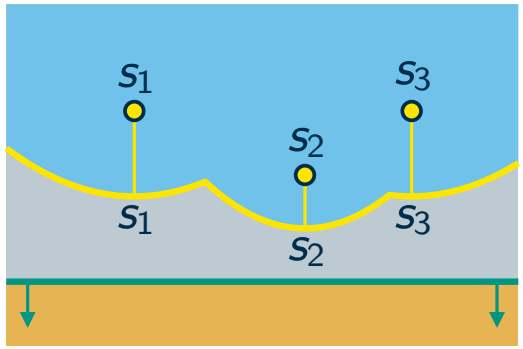
- sweep-line status: state of the beach line
- events: points in time, when BL changes
- What does “change” mean hear?
  - beach line changes all the time
  - sequence of corresponding sites rarely changes



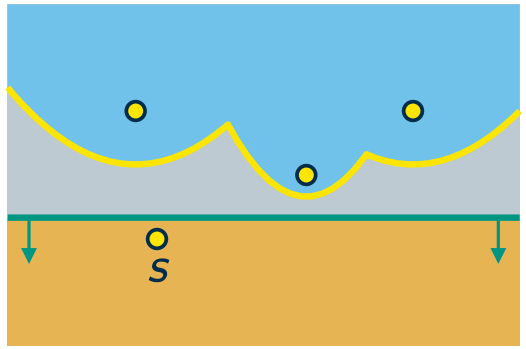
When Does The Beach Line Change?

Sweep-Line Algo

- sweep-line status: state of the beach line
- events: points in time, when BL changes
- What does “change” mean hear?
  - beach line changes all the time
  - sequence of corresponding sites rarely changes



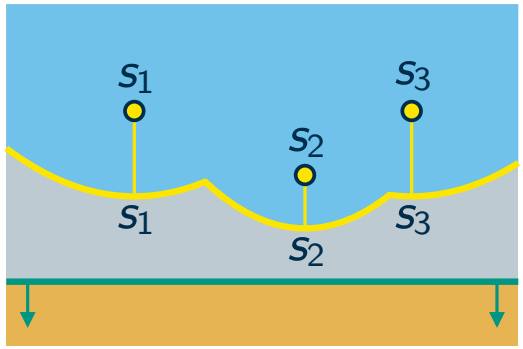
Event: New Site *s*

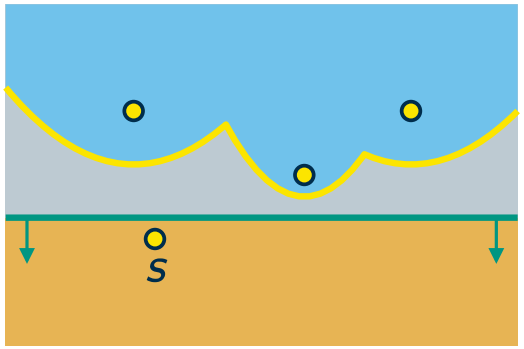


When Does The Beach Line Change?

Sweep-Line Algo

- sweep-line status: state of the beach line
- events: points in time, when BL changes
- What does “change” mean hear?
  - beach line changes all the time
  - sequence of corresponding sites rarely changes





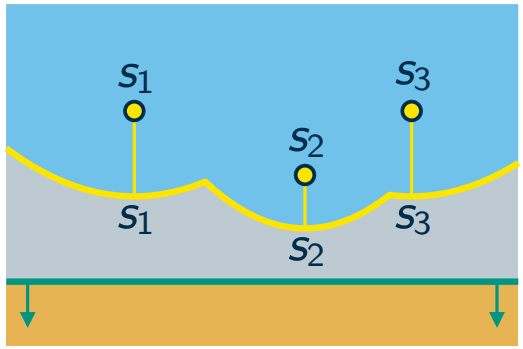
Event: New Site *s*

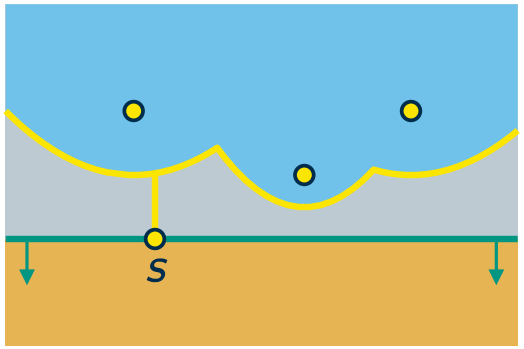
- new parabola is inserted into the BL

When Does The Beach Line Change?

Sweep-Line Algo

- sweep-line status: state of the beach line
- events: points in time, when BL changes
- What does “change” mean hear?
  - beach line changes all the time
  - sequence of corresponding sites rarely changes





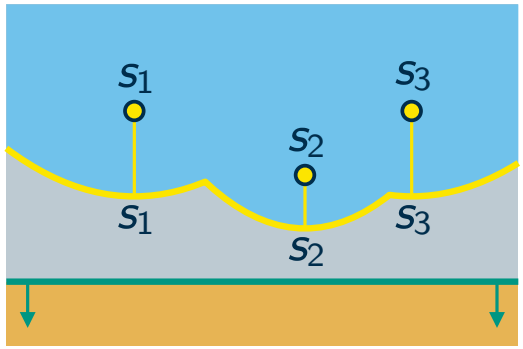
Event: New Site *s*

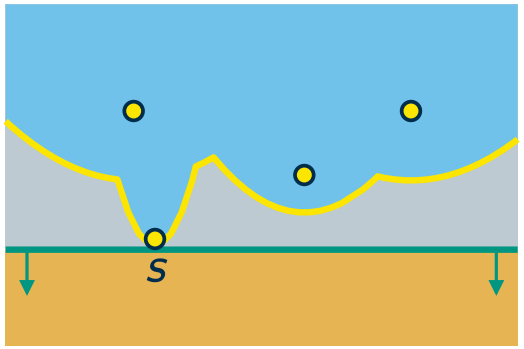
- new parabola is inserted into the BL

When Does The Beach Line Change?

Sweep-Line Algo

- sweep-line status: state of the beach line
- events: points in time, when BL changes
- What does “change” mean hear?
  - beach line changes all the time
  - sequence of corresponding sites rarely changes





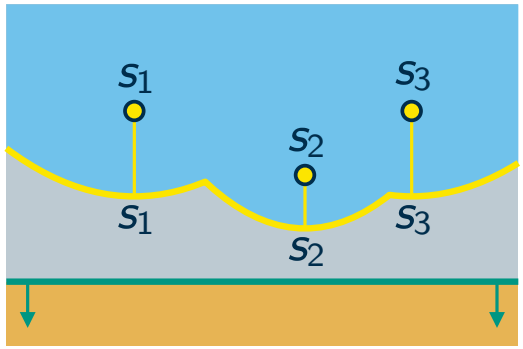
Event: New Site *s*

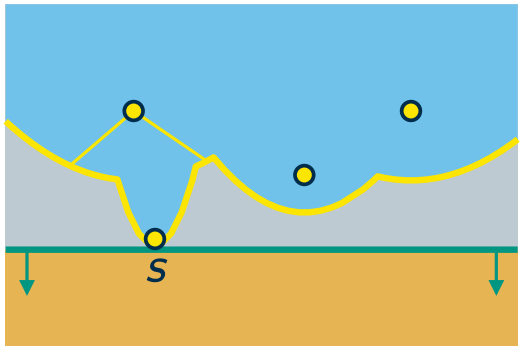
- new parabola is inserted into the BL

When Does The Beach Line Change?

Sweep-Line Algo

- sweep-line status: state of the beach line
- events: points in time, when BL changes
- What does “change” mean hear?
  - beach line changes all the time
  - sequence of corresponding sites rarely changes





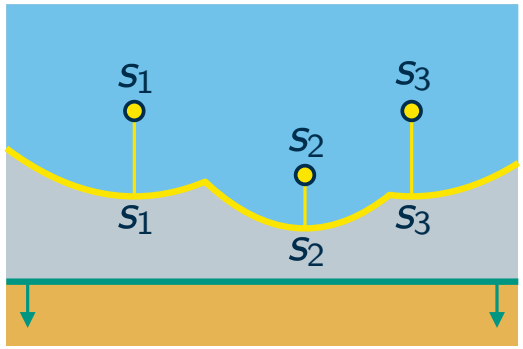
Event: New Site *s*

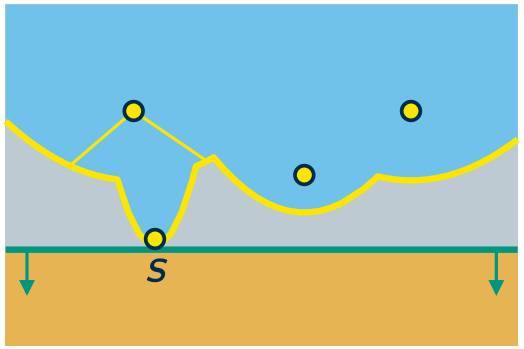
- new parabola is inserted into the BL
- splits parabola of current BL above *s*

When Does The Beach Line Change?

Sweep-Line Algo

- sweep-line status: state of the beach line
- events: points in time, when BL changes
- What does “change” mean hear?
  - beach line changes all the time
  - sequence of corresponding sites rarely changes





Event: New Site *s*

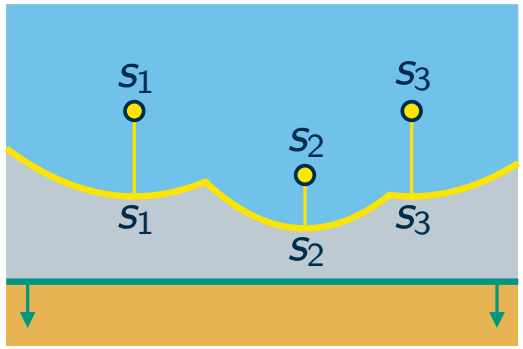
- new parabola is inserted into the BL
- splits parabola of current BL above *s*

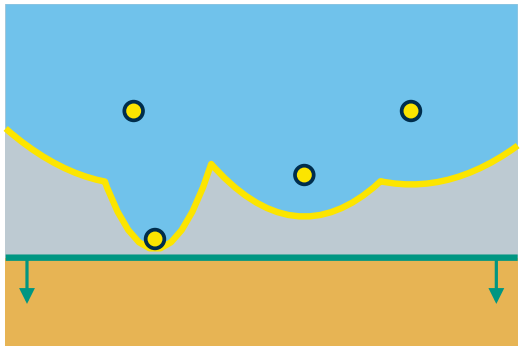
Event: Parabola Piece Disappears

When Does The Beach Line Change?

Sweep-Line Algo

- sweep-line status: state of the beach line
- events: points in time, when BL changes
- What does “change” mean hear?
  - beach line changes all the time
  - sequence of corresponding sites rarely changes





Event: New Site *s*

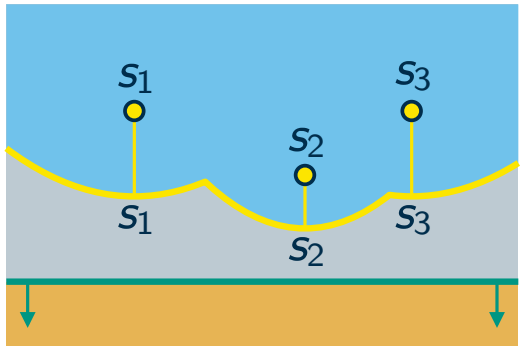
- new parabola is inserted into the BL
- splits parabola of current BL above *s*

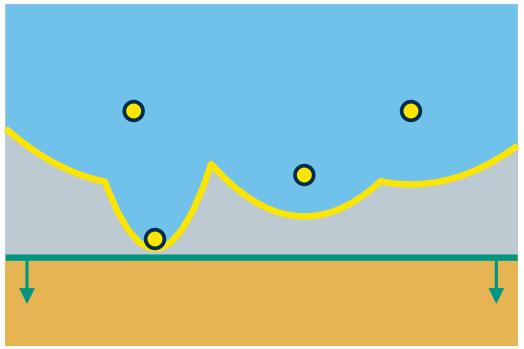
Event: Parabola Piece Disappears

When Does The Beach Line Change?

Sweep-Line Algo

- sweep-line status: state of the beach line
- events: points in time, when BL changes
- What does “change” mean hear?
  - beach line changes all the time
  - sequence of corresponding sites rarely changes





Event: New Site *s*

- new parabola is inserted into the BL
- splits parabola of current BL above *s*

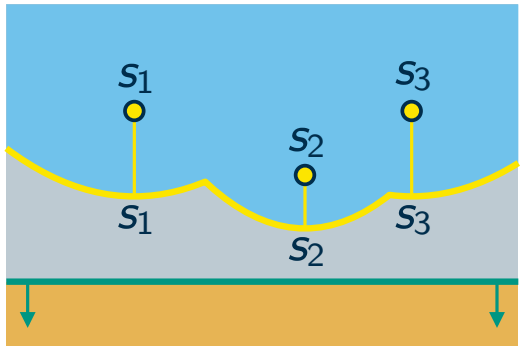
Event: Parabola Piece Disappears

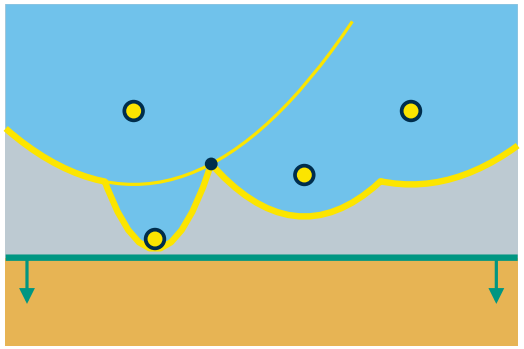
- How do we know when this happens?

When Does The Beach Line Change?

Sweep-Line Algo

- sweep-line status: state of the beach line
- events: points in time, when BL changes
- What does “change” mean hear?
  - beach line changes all the time
  - sequence of corresponding sites rarely changes





Event: New Site *s*

- new parabola is inserted into the BL
- splits parabola of current BL above *s*

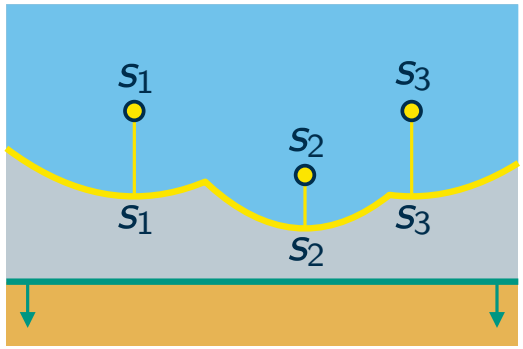
Event: Parabola Piece Disappears

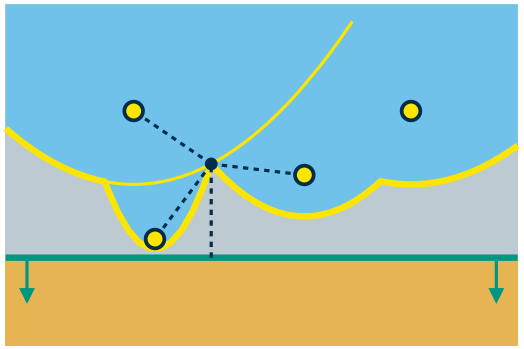
- How do we know when this happens?
- parabolas of three points intersect

When Does The Beach Line Change?

Sweep-Line Algo

- sweep-line status: state of the beach line
- events: points in time, when BL changes
- What does “change” mean hear?
  - beach line changes all the time
  - sequence of corresponding sites rarely changes





Event: New Site *s*

- new parabola is inserted into the BL
- splits parabola of current BL above *s*

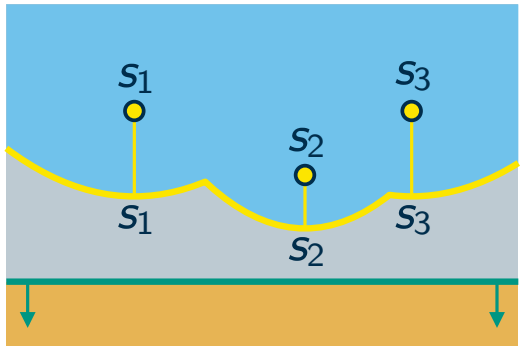
Event: Parabola Piece Disappears

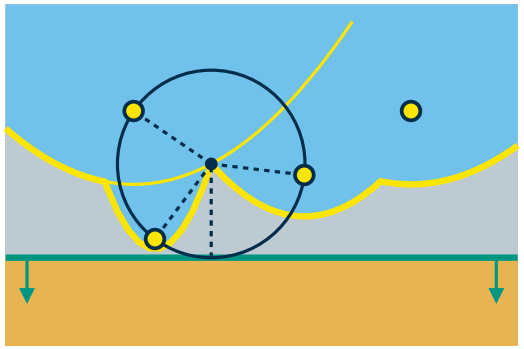
- How do we know when this happens?
- parabolas of three points intersect
- intersection has same distance to the three sites and to the sweep line

When Does The Beach Line Change?

Sweep-Line Algo

- sweep-line status: state of the beach line
- events: points in time, when BL changes
- What does “change” mean hear?
  - beach line changes all the time
  - sequence of corresponding sites rarely changes





Event: New Site *s*

- new parabola is inserted into the BL
- splits parabola of current BL above *s*

Event: Parabola Piece Disappears

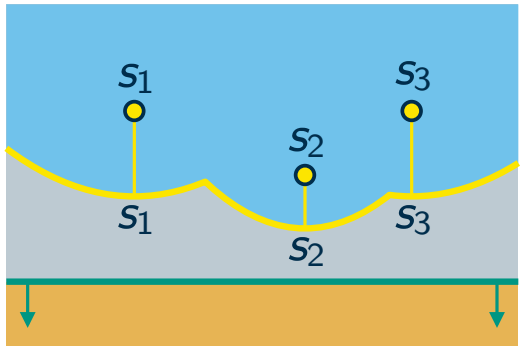
- How do we know when this happens?
- parabolas of three points intersect
- intersection has same distance to the three sites and to the sweep line
- no other site is closer

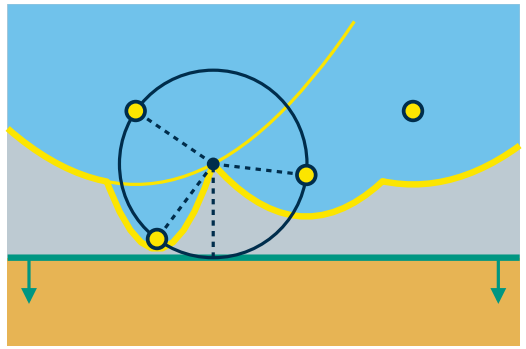
Why not?

When Does The Beach Line Change?

Sweep-Line Algo

- sweep-line status: state of the beach line
- events: points in time, when BL changes
- What does “change” mean hear?
  - beach line changes all the time
  - sequence of corresponding sites rarely changes





Event: New Site *s*

- new parabola is inserted into the BL
- splits parabola of current BL above *s*

Event: Parabola Piece Disappears

- How do we know when this happens?
- parabolas of three points intersect
- intersection has same distance to the three sites and to the sweep line
- no other site is closer
- we call this a circle event
- position defined by the sites corresponding to three consecutive parabola pieces

Why not?

What Do We Already Know?

What Do We Already Know?

Event Points: Beach Line Changes

- site event $\rightarrow$ insert new parabola piece
- circle event $\rightarrow$ remove parabola piece

(without proof: these are the only situations, in which the beach line changes)

What Do We Already Know?

Event Points: Beach Line Changes

- site event $\rightarrow$ insert new parabola piece
- circle event $\rightarrow$ remove parabola piece

(without proof: these are the only situations, in which the beach line changes)

Finding Circle Events

- every beach line change: new triples of consecutive parabola pieces $\rightarrow$ maybe new circle event

What Do We Already Know?

Event Points: Beach Line Changes

- site event $\rightarrow$ insert new parabola piece
- circle event $\rightarrow$ remove parabola piece

(without proof: these are the only situations, in which the beach line changes)

Finding Circle Events

- every beach line change: new triples of consecutive parabola pieces $\rightarrow$ maybe new circle event

Voronoi Diagram

- circle event $\rightarrow$ new vertex of the Voronoi diagram
- every vertex is found this way

Why?

What Do We Already Know?

Event Points: Beach Line Changes

- site event $\rightarrow$ insert new parabola piece
- circle event $\rightarrow$ remove parabola piece

(without proof: these are the only situations, in which the beach line changes)

Finding Circle Events

- every beach line change: new triples of consecutive parabola pieces $\rightarrow$ maybe new circle event

Voronoi Diagram

- circle event $\rightarrow$ new vertex of the Voronoi diagram
- every vertex is found this way

Why?

Open Questions

- How do we manage the beach line?

What Do We Already Know?

Event Points: Beach Line Changes

- site event $\rightarrow$ insert new parabola piece
- circle event $\rightarrow$ remove parabola piece

(without proof: these are the only situations, in which the beach line changes)

Finding Circle Events

- every beach line change: new triples of consecutive parabola pieces $\rightarrow$ maybe new circle event

Voronoi Diagram

- circle event $\rightarrow$ new vertex of the Voronoi diagram
- every vertex is found this way

Why?

Open Questions

- How do we manage the beach line?
- How do we manage the events?

What Do We Already Know?

Event Points: Beach Line Changes

- site event $\rightarrow$ insert new parabola piece
- circle event $\rightarrow$ remove parabola piece

(without proof: these are the only situations, in which the beach line changes)

Finding Circle Events

- every beach line change: new triples of consecutive parabola pieces $\rightarrow$ maybe new circle event

Voronoi Diagram

- circle event $\rightarrow$ new vertex of the Voronoi diagram
- every vertex is found this way

Why?

Open Questions

- How do we manage the beach line?
- How do we manage the events?
- How do we get the Voronoi diagram in the end?

What Do We Already Know?

Event Points: Beach Line Changes

- site event $\rightarrow$ insert new parabola piece
- circle event $\rightarrow$ remove parabola piece

(without proof: these are the only situations, in which the beach line changes)

Finding Circle Events

- every beach line change: new triples of consecutive parabola pieces $\rightarrow$ maybe new circle event

Voronoi Diagram

- circle event $\rightarrow$ new vertex of the Voronoi diagram
- every vertex is found this way

Why?

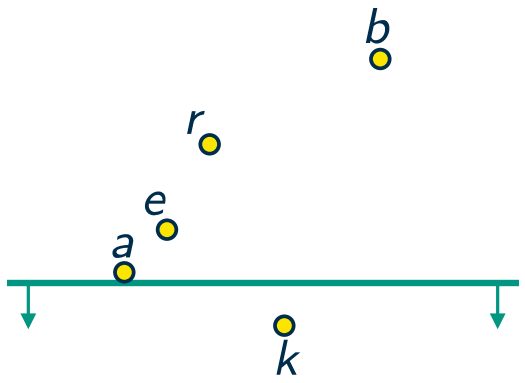
Open Questions

- How do we manage the beach line?
- How do we manage the events?
- How do we get the Voronoi diagram in the end?

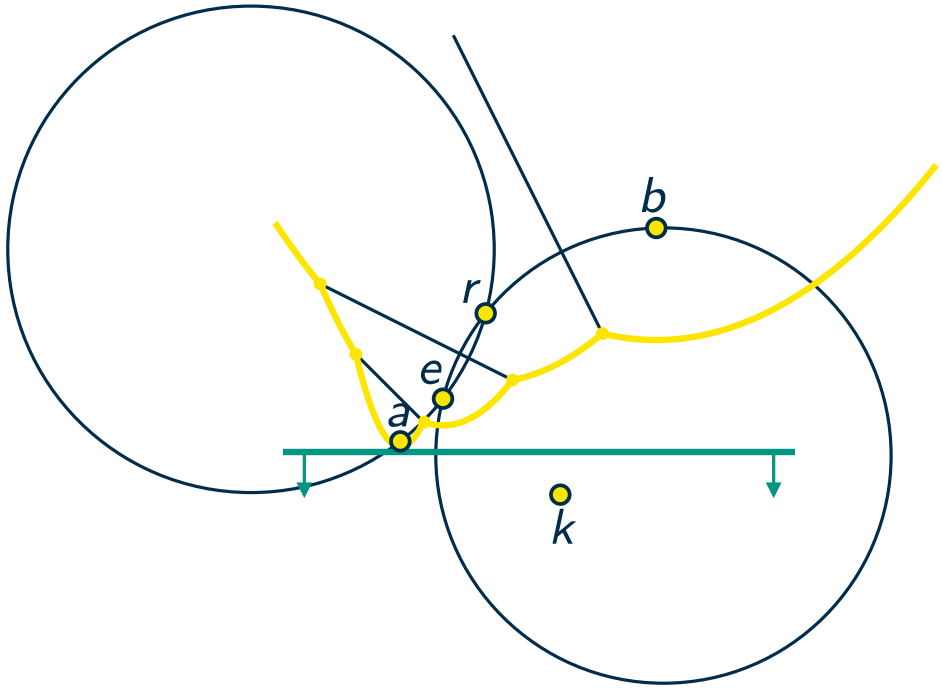
Special Cases

- sites with same y-coordinate
- vertices with degree > 3
- what else?

How Many Events Are In The Queue?

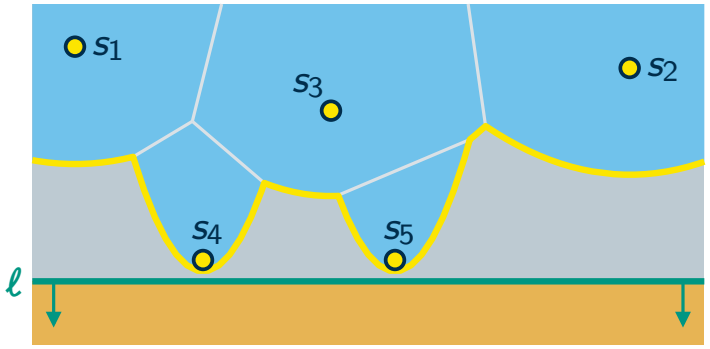


How Many Events Are In The Queue?



Answer: 3 (one site and two circle events)

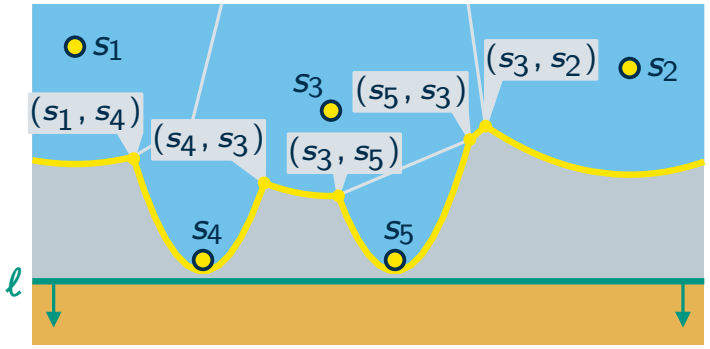
Managing The Beach Line



Observation

- parabolas can be split into multiple pieces
- knowing the site does not uniquely identify the parabola piece

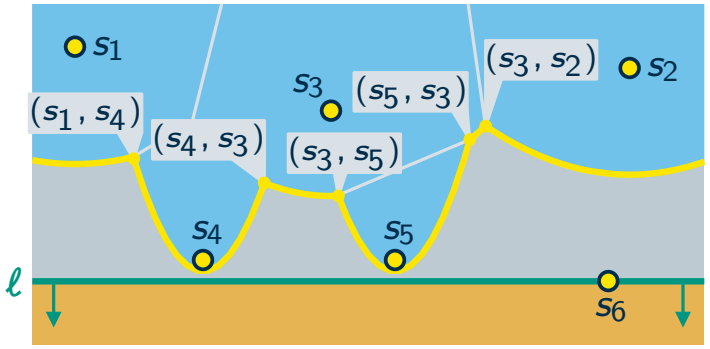
Managing The Beach Line



Observation

- parabolas can be split into multiple pieces
- knowing the site does not uniquely identify the parabola piece
- instead: store a representation of the intersection points that is independent of ℓ's exact position

Managing The Beach Line

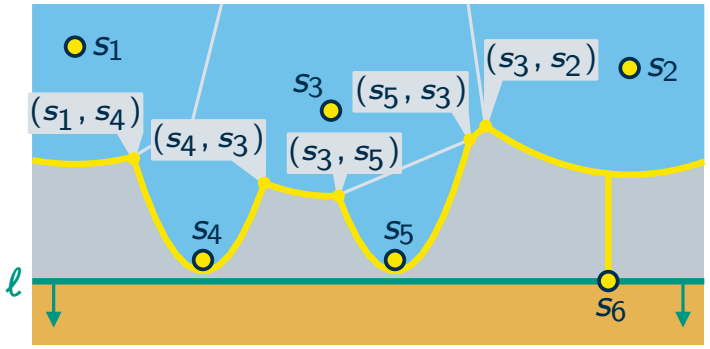


Observation

- parabolas can be split into multiple pieces
- knowing the site does not uniquely identify the parabola piece
- instead: store a representation of the intersection points that is independent of ℓ's exact position

Site Event

Managing The Beach Line



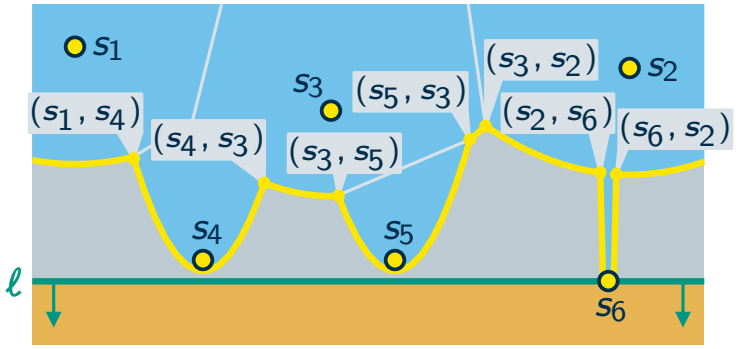
Observation

- parabolas can be split into multiple pieces
- knowing the site does not uniquely identify the parabola piece
- instead: store a representation of the intersection points that is independent of ℓ's exact position

Site Event

- find x-coordinate of the new site in the set of intersection points

Managing The Beach Line



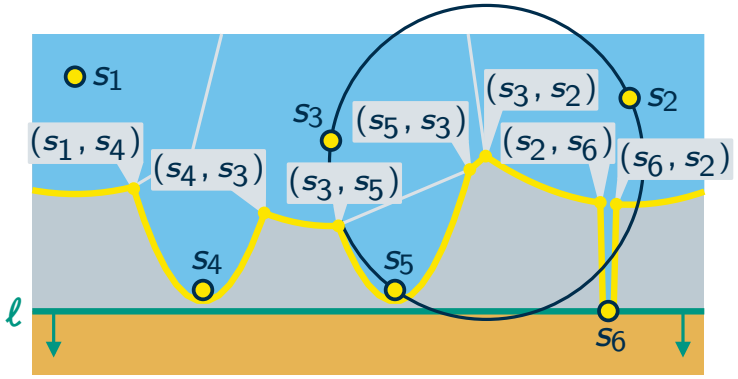
Observation

- parabolas can be split into multiple pieces
- knowing the site does not uniquely identify the parabola piece
- instead: store a representation of the intersection points that is independent of ℓ's exact position

Site Event

- find x-coordinate of the new site in the set of intersection points
- insert two new intersection points

Managing The Beach Line



Observation

- parabolas can be split into multiple pieces
- knowing the site does not uniquely identify the parabola piece
- instead: store a representation of the intersection points that is independent of ℓ's exact position

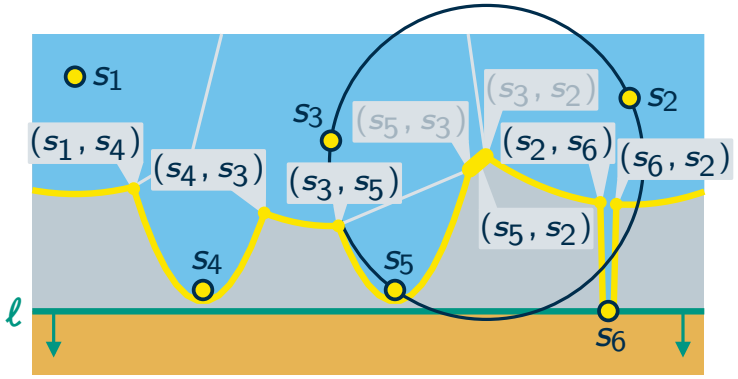
Site Event

- find x-coordinate of the new site in the set of intersection points
- insert two new intersection points

Circle Event

- belongs to two consecutive intersection points (here: (s_5, s_3) and (s_3, s_2))

Managing The Beach Line



Observation

- parabolas can be split into multiple pieces
- knowing the site does not uniquely identify the parabola piece
- instead: store a representation of the intersection points that is independent of ℓ's exact position

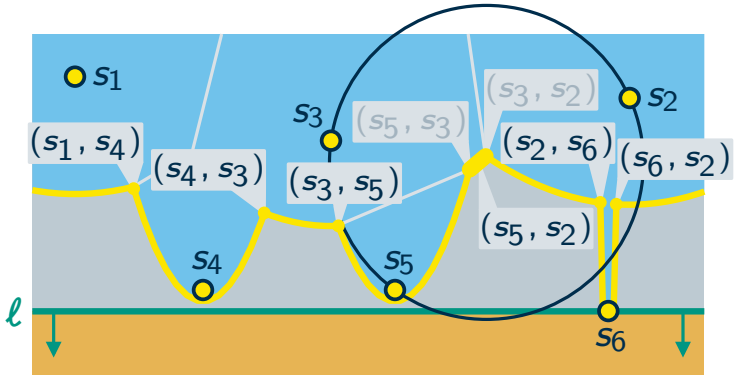
Site Event

- find x-coordinate of the new site in the set of intersection points
- insert two new intersection points

Circle Event

- belongs to two consecutive intersection points (here: (s_5, s_3) and (s_3, s_2))
- remove the two and insert the new one (here: (s_5, s_2))

Managing The Beach Line



Observation

- parabolas can be split into multiple pieces
- knowing the site does not uniquely identify the parabola piece
- instead: store a representation of the intersection points that is independent of ℓ's exact position

Site Event

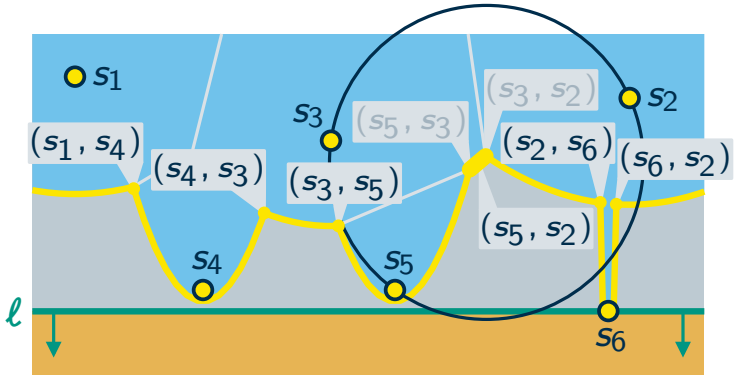
- find x-coordinate of the new site in the set of intersection points
- insert two new intersection points

Circle Event

- belongs to two consecutive intersection points (here: (s_5, s_3) and (s_3, s_2))
- remove the two and insert the new one (here: (s_5, s_2))

Data Structure: search (for x), insert, and delete

Managing The Beach Line



Observation

- parabolas can be split into multiple pieces
- knowing the site does not uniquely identify the parabola piece
- instead: store a representation of the intersection points that is independent of ℓ's exact position

Site Event

- find x-coordinate of the new site in the set of intersection points
- insert two new intersection points

Circle Event

- belongs to two consecutive intersection points (here: (s_5, s_3) and (s_3, s_2))
- remove the two and insert the new one (here: (s_5, s_2))

Data Structure: search (for x), insert, and delete $\rightarrow$ search tree ($O(\log n)$ per operation)

Managing The Events

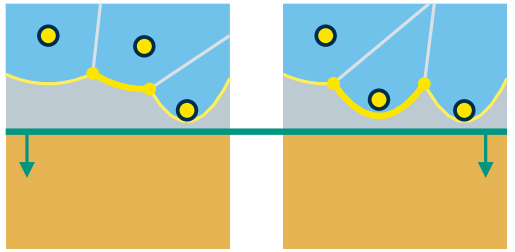
Observation

- we know all site events from the start
- circle events: defined by adjacent intersection points on the beach line

Managing The Events

Observation

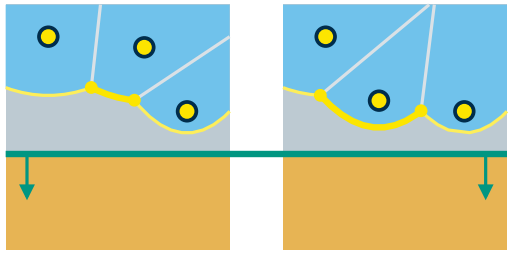
- we know all site events from the start
- circle events: defined by adjacent intersection points on the beach line
- not every pair of adjacent intersections yields a circle event



Managing The Events

Observation

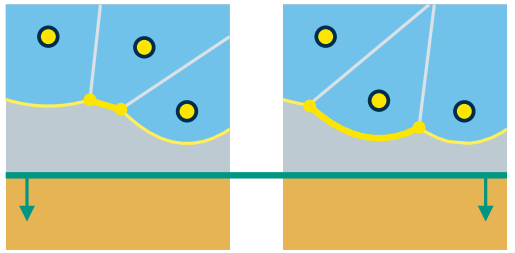
- we know all site events from the start
- circle events: defined by adjacent intersection points on the beach line
- not every pair of adjacent intersections yields a circle event



Managing The Events

Observation

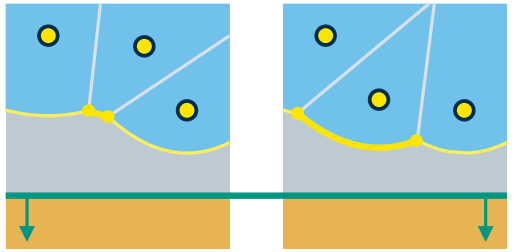
- we know all site events from the start
- circle events: defined by adjacent intersection points on the beach line
- not every pair of adjacent intersections yields a circle event



Managing The Events

Observation

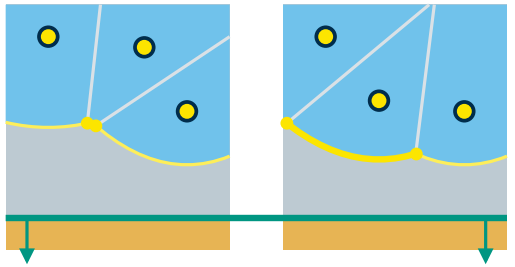
- we know all site events from the start
- circle events: defined by adjacent intersection points on the beach line
- not every pair of adjacent intersections yields a circle event



Managing The Events

Observation

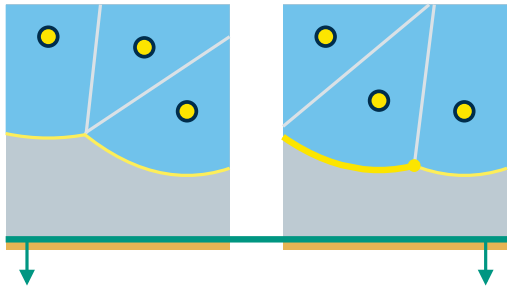
- we know all site events from the start
- circle events: defined by adjacent intersection points on the beach line
- not every pair of adjacent intersections yields a circle event



Managing The Events

Observation

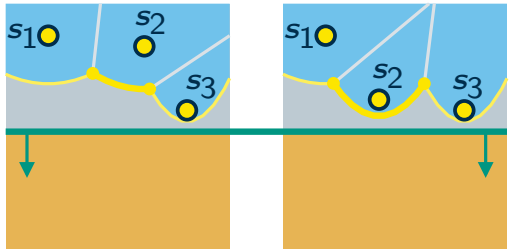
- we know all site events from the start
- circle events: defined by adjacent intersection points on the beach line
- not every pair of adjacent intersections yields a circle event



Managing The Events

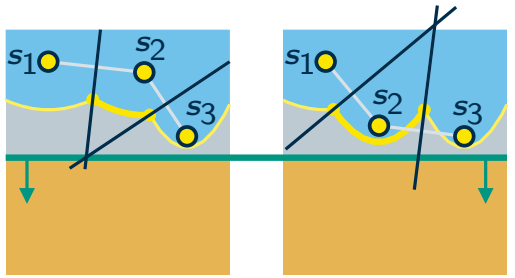
Observation

- we know all site events from the start
- circle events: defined by adjacent intersection points on the beach line
- not every pair of adjacent intersections yields a circle event
- whether we have a circle event can be determined based on the corresponding sites



Managing The Events

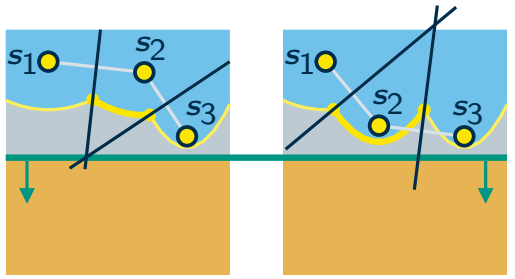
Observation



- we know all site events from the start
- circle events: defined by adjacent intersection points on the beach line
- not every pair of adjacent intersections yields a circle event
- whether we have a circle event can be determined based on the corresponding sites
- perpendicular bisectors converge $\rightarrow$ parabola piece shrinks $\rightarrow$ circle event
- perpendicular bisectors diverge $\rightarrow$ parabola piece grows $\rightarrow$ no circle event

Managing The Events

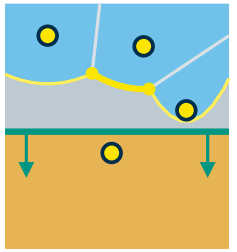
Observation



- we know all site events from the start
- circle events: defined by adjacent intersection points on the beach line
- not every pair of adjacent intersections yields a circle event
- whether we have a circle event can be determined based on the corresponding sites
- perpendicular bisectors converge $\rightarrow$ parabola piece shrinks $\rightarrow$ circle event
- perpendicular bisectors diverge $\rightarrow$ parabola piece grows $\rightarrow$ no circle event

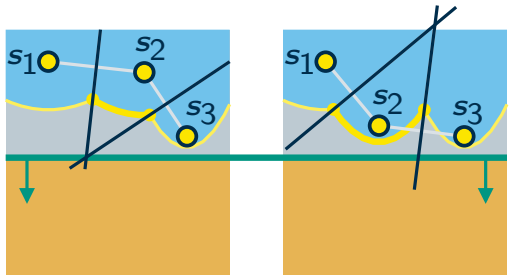
False Alarm

- some circle events to not actually happen



Managing The Events

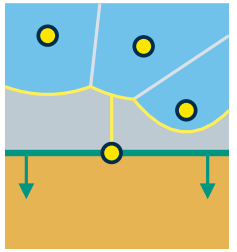
Observation



- we know all site events from the start
- circle events: defined by adjacent intersection points on the beach line
- not every pair of adjacent intersections yields a circle event
- whether we have a circle event can be determined based on the corresponding sites
- perpendicular bisectors converge $\rightarrow$ parabola piece shrinks $\rightarrow$ circle event
- perpendicular bisectors diverge $\rightarrow$ parabola piece grows $\rightarrow$ no circle event

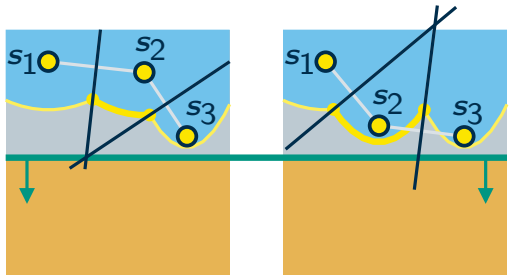
False Alarm

- some circle events to not actually happen



Managing The Events

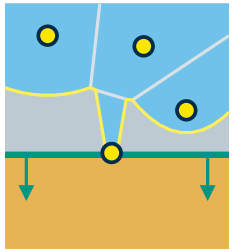
Observation



- we know all site events from the start
- circle events: defined by adjacent intersection points on the beach line
- not every pair of adjacent intersections yields a circle event
- whether we have a circle event can be determined based on the corresponding sites
- perpendicular bisectors converge $\rightarrow$ parabola piece shrinks $\rightarrow$ circle event
- perpendicular bisectors diverge $\rightarrow$ parabola piece grows $\rightarrow$ no circle event

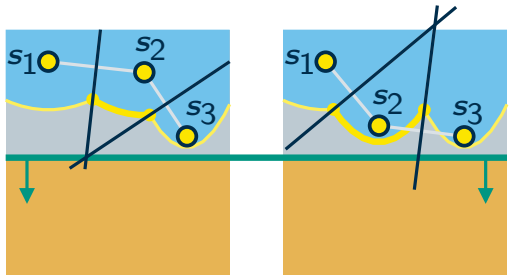
False Alarm

- some circle events to not actually happen



Managing The Events

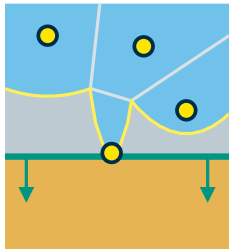
Observation



- we know all site events from the start
- circle events: defined by adjacent intersection points on the beach line
- not every pair of adjacent intersections yields a circle event
- whether we have a circle event can be determined based on the corresponding sites
- perpendicular bisectors converge $\rightarrow$ parabola piece shrinks $\rightarrow$ circle event
- perpendicular bisectors diverge $\rightarrow$ parabola piece grows $\rightarrow$ no circle event

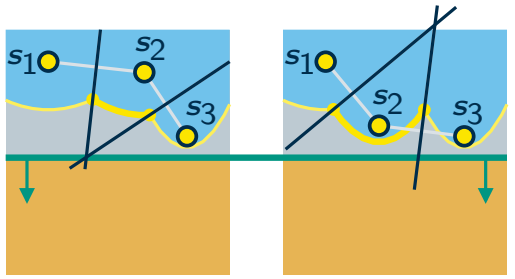
False Alarm

- some circle events to not actually happen



Managing The Events

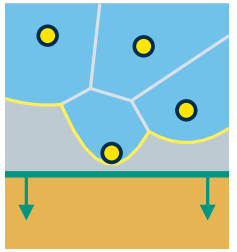
Observation



- we know all site events from the start
- circle events: defined by adjacent intersection points on the beach line
- not every pair of adjacent intersections yields a circle event
- whether we have a circle event can be determined based on the corresponding sites
- perpendicular bisectors converge $\rightarrow$ parabola piece shrinks $\rightarrow$ circle event
- perpendicular bisectors diverge $\rightarrow$ parabola piece grows $\rightarrow$ no circle event

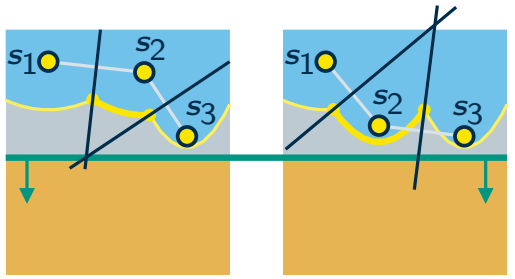
False Alarm

- some circle events to not actually happen



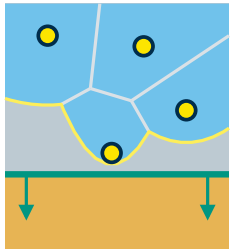
Managing The Events

Observation



- we know all site events from the start
- circle events: defined by adjacent intersection points on the beach line
- not every pair of adjacent intersections yields a circle event
- whether we have a circle event can be determined based on the corresponding sites
- perpendicular bisectors converge $\rightarrow$ parabola piece shrinks $\rightarrow$ circle event
- perpendicular bisectors diverge $\rightarrow$ parabola piece grows $\rightarrow$ no circle event

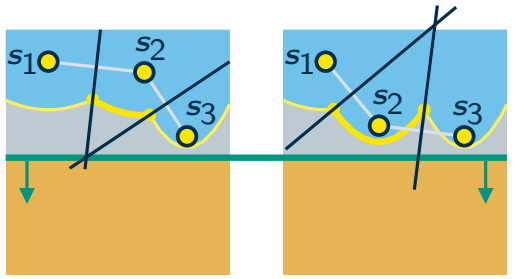
False Alarm



- some circle events to not actually happen
- at site event: remove circle event of split parabola (if it exists)

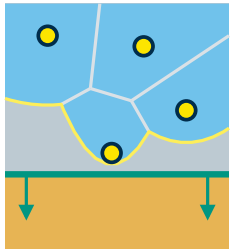
Managing The Events

Observation



- we know all site events from the start
- circle events: defined by adjacent intersection points on the beach line
- not every pair of adjacent intersections yields a circle event
- whether we have a circle event can be determined based on the corresponding sites
- perpendicular bisectors converge $\rightarrow$ parabola piece shrinks $\rightarrow$ circle event
- perpendicular bisectors diverge $\rightarrow$ parabola piece grows $\rightarrow$ no circle event

False Alarm

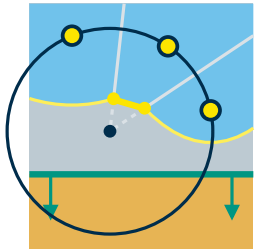


- some circle events to not actually happen
- at site event: remove circle event of split parabola (if it exists)
- no problem: use search tree for the event queue $\rightarrow O(\log n)$ per operation

Constructing The Voronoi Diagram

Already Seen

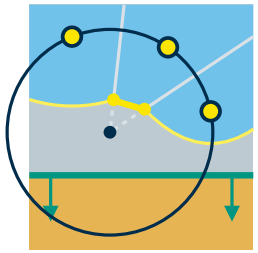
- point p is vertex $\Leftrightarrow$ p is the center for a circle event
- p lies on an edge $\Leftrightarrow$ p is at some point the intersection of parabola pieces



Constructing The Voronoi Diagram

Already Seen

- point p is vertex $\Leftrightarrow$ p is the center for a circle event
- p lies on an edge $\Leftrightarrow$ p is at some point the intersection of parabola pieces



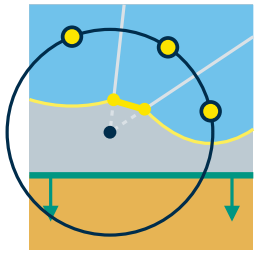
Plan

- compute doubly-connected edge list of $\mathrm{Vor}(S)$ during the sweep (with special handling for infinite edges)

Constructing The Voronoi Diagram

Already Seen

- point p is vertex $\Leftrightarrow$ p is the center for a circle event
- p lies on an edge $\Leftrightarrow$ p is at some point the intersection of parabola pieces



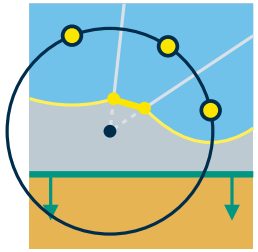
Plan

- compute doubly-connected edge list of $\mathrm{Vor}(S)$ during the sweep (with special handling for infinite edges)
- site event: new parabola $\rightarrow$ new intersections $\rightarrow$ new half edges

Constructing The Voronoi Diagram

Already Seen

- point p is vertex $\Leftrightarrow$ p is the center for a circle event
- p lies on an edge $\Leftrightarrow$ p is at some point the intersection of parabola pieces



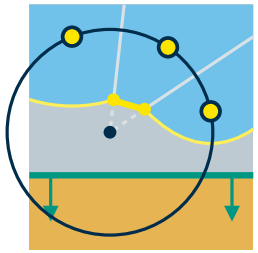
Plan

- compute doubly-connected edge list of $\mathrm{Vor}(S)$ during the sweep (with special handling for infinite edges)
- site event: new parabola $\rightarrow$ new intersections $\rightarrow$ new half edges
- circle event: new vertex $\rightarrow$ attach corresponding half edges to vertex
  new intersection $\rightarrow$ new half edge

Constructing The Voronoi Diagram

Already Seen

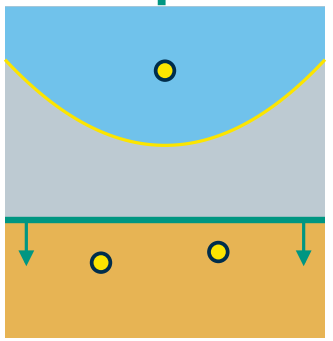
- point p is vertex $\Leftrightarrow$ p is the center for a circle event
- p lies on an edge $\Leftrightarrow$ p is at some point the intersection of parabola pieces



Plan

- compute doubly-connected edge list of $\mathrm{Vor}(S)$ during the sweep (with special handling for infinite edges)
- site event: new parabola $\rightarrow$ new intersections $\rightarrow$ new half edges
- circle event: new vertex $\rightarrow$ attach corresponding half edges to vertex
  new intersection $\rightarrow$ new half edge

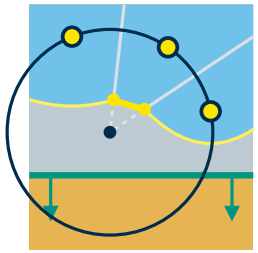
Example



Constructing The Voronoi Diagram

Already Seen

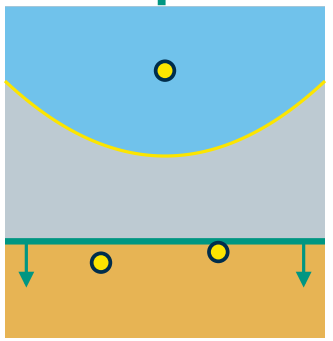
- point p is vertex $\Leftrightarrow$ p is the center for a circle event
- p lies on an edge $\Leftrightarrow$ p is at some point the intersection of parabola pieces



Plan

- compute doubly-connected edge list of $\mathrm{Vor}(S)$ during the sweep (with special handling for infinite edges)
- site event: new parabola $\rightarrow$ new intersections $\rightarrow$ new half edges
- circle event: new vertex $\rightarrow$ attach corresponding half edges to vertex
  new intersection $\rightarrow$ new half edge

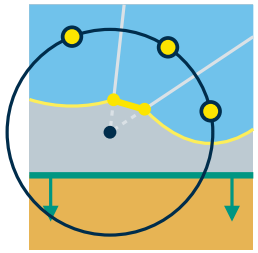
Example



Constructing The Voronoi Diagram

Already Seen

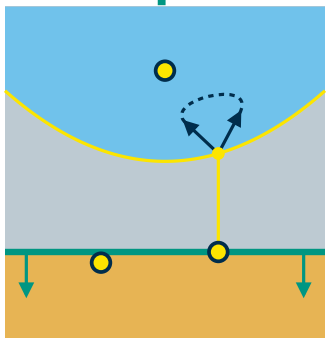
- point p is vertex $\Leftrightarrow$ p is the center for a circle event
- p lies on an edge $\Leftrightarrow$ p is at some point the intersection of parabola pieces



Plan

- compute doubly-connected edge list of $\mathrm{Vor}(S)$ during the sweep (with special handling for infinite edges)
- site event: new parabola $\rightarrow$ new intersections $\rightarrow$ new half edges
- circle event: new vertex $\rightarrow$ attach corresponding half edges to vertex
  new intersection $\rightarrow$ new half edge

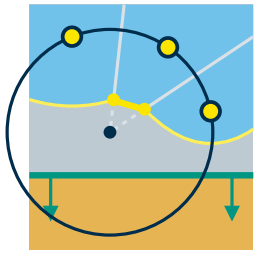
Example



Constructing The Voronoi Diagram

Already Seen

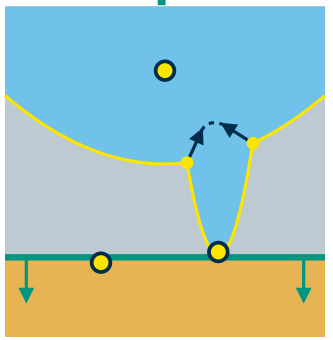
- point p is vertex $\Leftrightarrow$ p is the center for a circle event
- p lies on an edge $\Leftrightarrow$ p is at some point the intersection of parabola pieces



Plan

- compute doubly-connected edge list of $\mathrm{Vor}(S)$ during the sweep (with special handling for infinite edges)
- site event: new parabola $\rightarrow$ new intersections $\rightarrow$ new half edges
- circle event: new vertex $\rightarrow$ attach corresponding half edges to vertex
  new intersection $\rightarrow$ new half edge

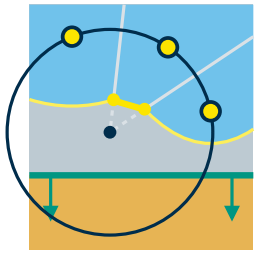
Example



Constructing The Voronoi Diagram

Already Seen

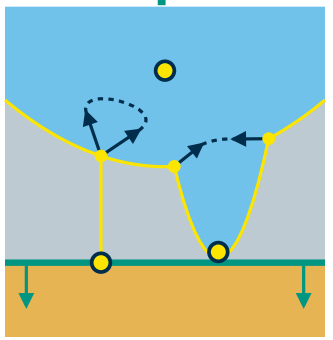
- point p is vertex $\Leftrightarrow$ p is the center for a circle event
- p lies on an edge $\Leftrightarrow$ p is at some point the intersection of parabola pieces



Plan

- compute doubly-connected edge list of $\mathrm{Vor}(S)$ during the sweep (with special handling for infinite edges)
- site event: new parabola $\rightarrow$ new intersections $\rightarrow$ new half edges
- circle event: new vertex $\rightarrow$ attach corresponding half edges to vertex
  new intersection $\rightarrow$ new half edge

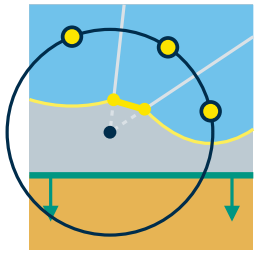
Example



Constructing The Voronoi Diagram

Already Seen

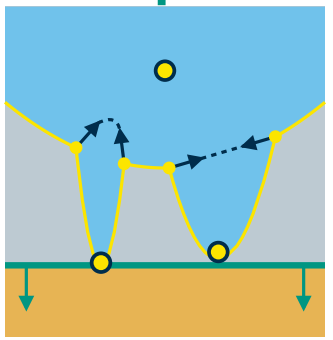
- point p is vertex $\Leftrightarrow$ p is the center for a circle event
- p lies on an edge $\Leftrightarrow$ p is at some point the intersection of parabola pieces



Plan

- compute doubly-connected edge list of $\mathrm{Vor}(S)$ during the sweep (with special handling for infinite edges)
- site event: new parabola $\rightarrow$ new intersections $\rightarrow$ new half edges
- circle event: new vertex $\rightarrow$ attach corresponding half edges to vertex
  new intersection $\rightarrow$ new half edge

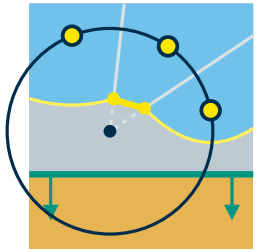
Example



Constructing The Voronoi Diagram

Already Seen

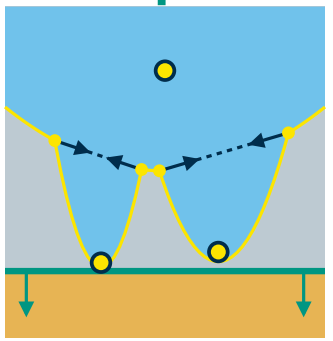
- point p is vertex $\Leftrightarrow$ p is the center for a circle event
- p lies on an edge $\Leftrightarrow$ p is at some point the intersection of parabola pieces



Plan

- compute doubly-connected edge list of $\mathrm{Vor}(S)$ during the sweep (with special handling for infinite edges)
- site event: new parabola $\rightarrow$ new intersections $\rightarrow$ new half edges
- circle event: new vertex $\rightarrow$ attach corresponding half edges to vertex
  new intersection $\rightarrow$ new half edge

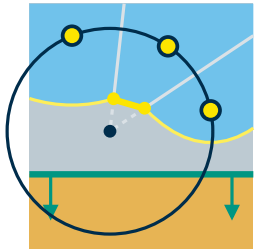
Example



Constructing The Voronoi Diagram

Already Seen

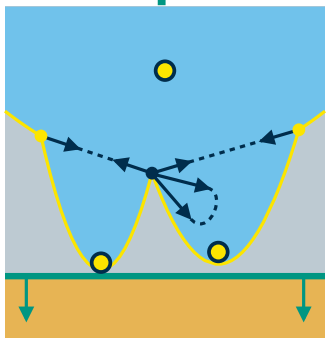
- point p is vertex $\Leftrightarrow$ p is the center for a circle event
- p lies on an edge $\Leftrightarrow$ p is at some point the intersection of parabola pieces



Plan

- compute doubly-connected edge list of $\mathrm{Vor}(S)$ during the sweep (with special handling for infinite edges)
- site event: new parabola $\rightarrow$ new intersections $\rightarrow$ new half edges
- circle event: new vertex $\rightarrow$ attach corresponding half edges to vertex
  new intersection $\rightarrow$ new half edge

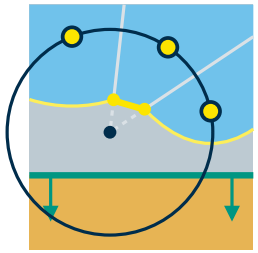
Example



Constructing The Voronoi Diagram

Already Seen

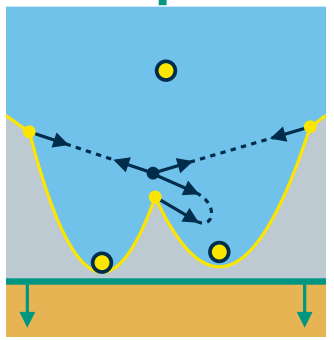
- point p is vertex $\Leftrightarrow$ p is the center for a circle event
- p lies on an edge $\Leftrightarrow$ p is at some point the intersection of parabola pieces



Plan

- compute doubly-connected edge list of $\mathrm{Vor}(S)$ during the sweep (with special handling for infinite edges)
- site event: new parabola $\rightarrow$ new intersections $\rightarrow$ new half edges
- circle event: new vertex $\rightarrow$ attach corresponding half edges to vertex
  new intersection $\rightarrow$ new half edge

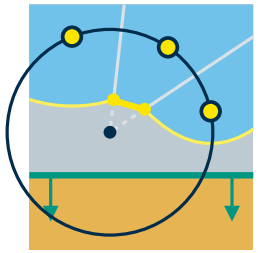
Example



Constructing The Voronoi Diagram

Already Seen

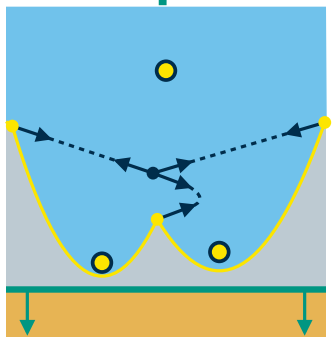
- point p is vertex $\Leftrightarrow$ p is the center for a circle event
- p lies on an edge $\Leftrightarrow$ p is at some point the intersection of parabola pieces



Plan

- compute doubly-connected edge list of $\mathrm{Vor}(S)$ during the sweep (with special handling for infinite edges)
- site event: new parabola $\rightarrow$ new intersections $\rightarrow$ new half edges
- circle event: new vertex $\rightarrow$ attach corresponding half edges to vertex
  new intersection $\rightarrow$ new half edge

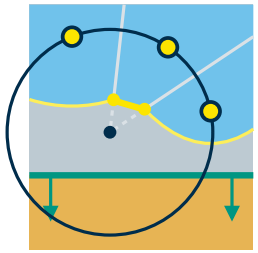
Example



Constructing The Voronoi Diagram

Already Seen

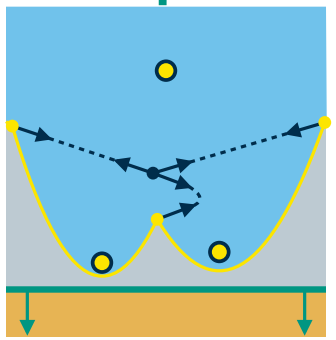
- point p is vertex $\Leftrightarrow$ p is the center for a circle event
- p lies on an edge $\Leftrightarrow$ p is at some point the intersection of parabola pieces



Plan

- compute doubly-connected edge list of $\mathrm{Vor}(S)$ during the sweep (with special handling for infinite edges)
- site event: new parabola $\rightarrow$ new intersections $\rightarrow$ new half edges
- circle event: new vertex $\rightarrow$ attach corresponding half edges to vertex
  new intersection $\rightarrow$ new half edge

Example

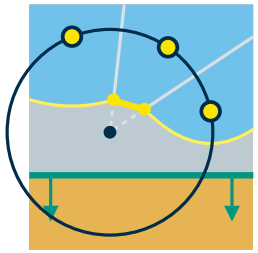


Infinite Edges

Constructing The Voronoi Diagram

Already Seen

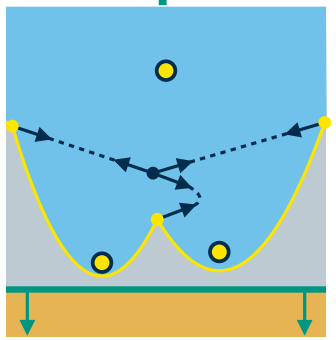
- point p is vertex $\Leftrightarrow$ p is the center for a circle event
- p lies on an edge $\Leftrightarrow$ p is at some point the intersection of parabola pieces



Plan

- compute doubly-connected edge list of $\mathrm{Vor}(S)$ during the sweep (with special handling for infinite edges)
- site event: new parabola $\rightarrow$ new intersections $\rightarrow$ new half edges
- circle event: new vertex $\rightarrow$ attach corresponding half edges to vertex
  new intersection $\rightarrow$ new half edge

Example

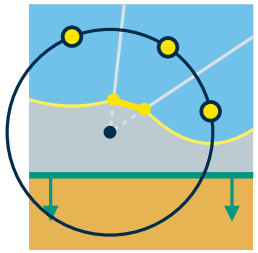


Infinite Edges

- some half edges do not end, as their parabola pieces never disappear

Constructing The Voronoi Diagram

Already Seen

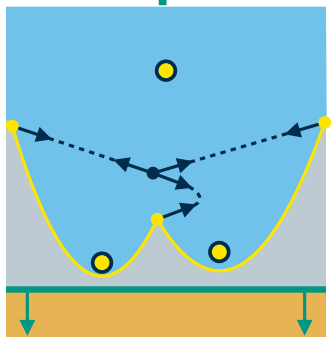


- point p is vertex $\Leftrightarrow$ p is the center for a circle event
- p lies on an edge $\Leftrightarrow$ p is at some point the intersection of parabola pieces

Plan

- compute doubly-connected edge list of $\mathrm{Vor}(S)$ during the sweep (with special handling for infinite edges)
- site event: new parabola $\rightarrow$ new intersections $\rightarrow$ new half edges
- circle event: new vertex $\rightarrow$ attach corresponding half edges to vertex
  new intersection $\rightarrow$ new half edge

Example

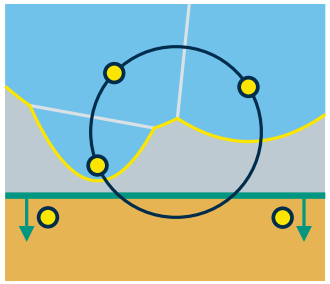


Infinite Edges

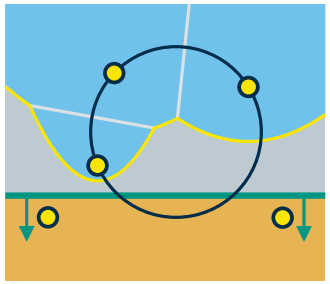
- some half edges do not end, as their parabola pieces never disappear
- empty queue $\rightarrow$ some final clean up remaining intersections in beach line
  (depending on how your doubly-connected edge list deals with infinite edges)

Special Cases

Events With Equal y-Coordinate



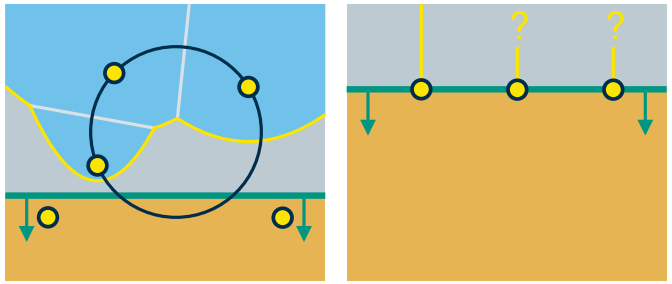
Special Cases



Events With Equal y-Coordinate

- handle in arbitrary order

Special Cases



Events With Equal *y*-Coordinate

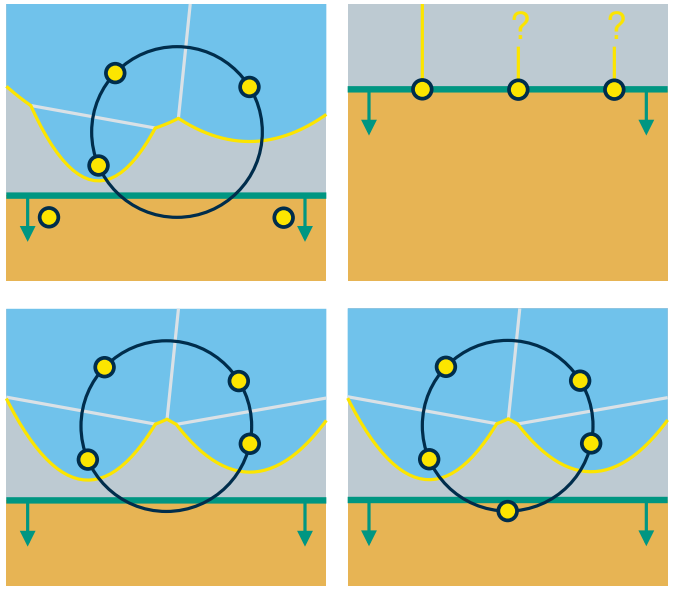
- handle in arbitrary order
- exception: initial site events (no parabola above exists)

Special Cases

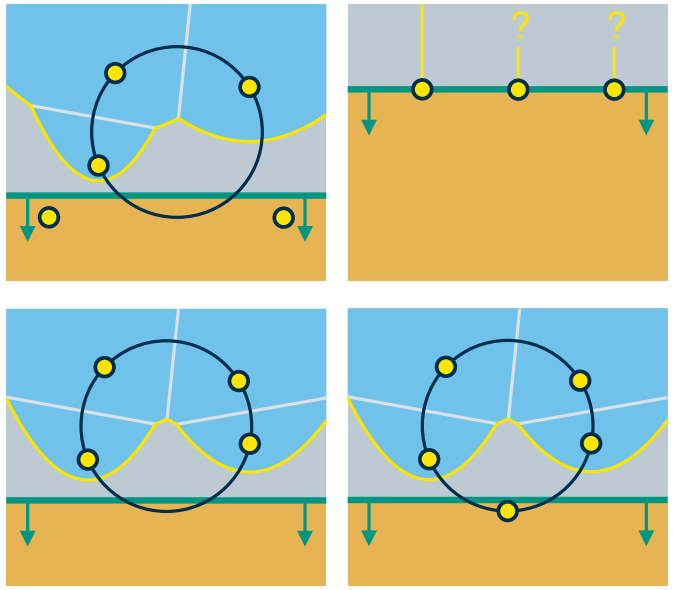
Events With Equal y-Coordinate

- handle in arbitrary order
- exception: initial site events (no parabola above exists)

Equal x- and y-Coordinate



Special Cases



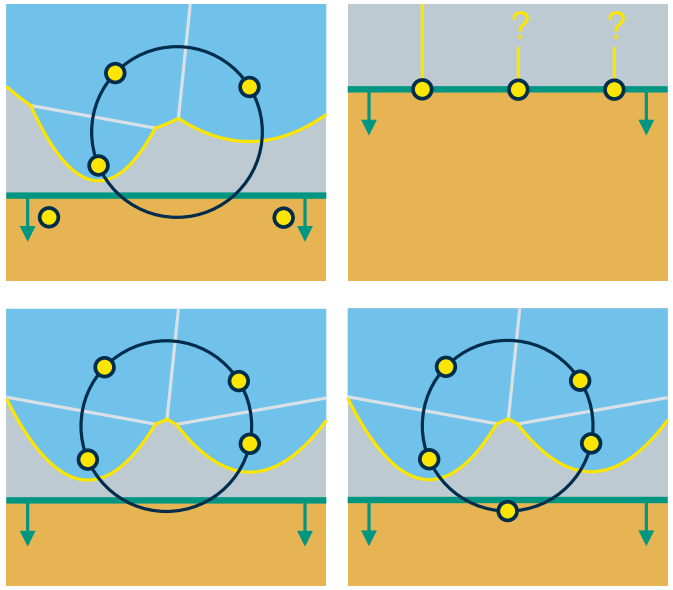
Events With Equal y-Coordinate

- handle in arbitrary order
- exception: initial site events (no parabola above exists)

Equal x- and y-Coordinate

- vertex in $\mathrm{Vor}(S)$ with degree > 3

Special Cases



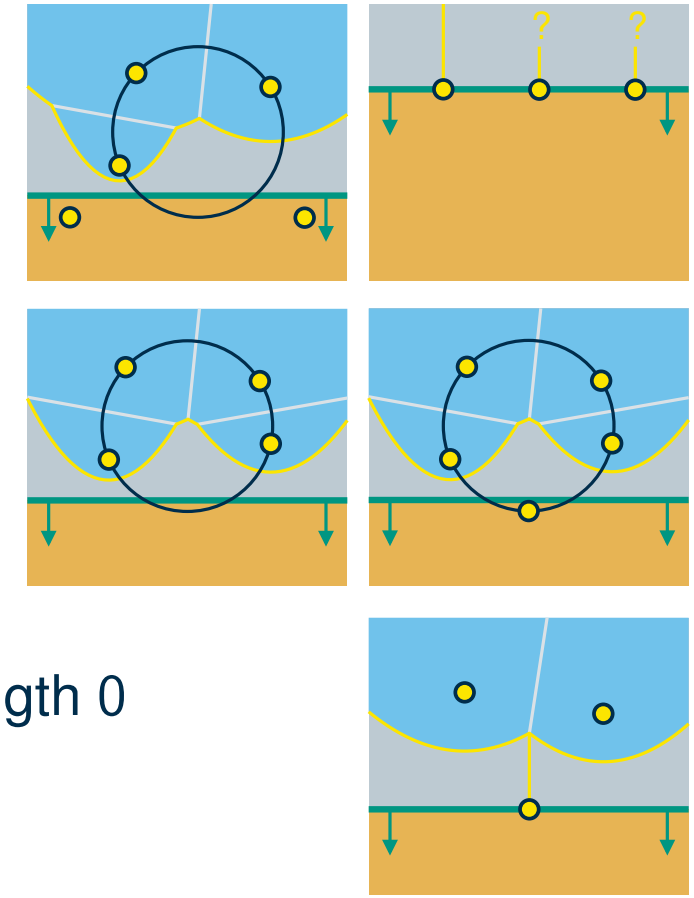
Events With Equal y-Coordinate

- handle in arbitrary order
- exception: initial site events (no parabola above exists)

Equal x- and y-Coordinate

- vertex in $\mathrm{Vor}(S)$ with degree > 3
- special handling $\rightarrow$ vertex with degree > 3
- no special handling $\rightarrow$ multiple degree-3 vertices with edges of length 0

Special Cases



Events With Equal y-Coordinate

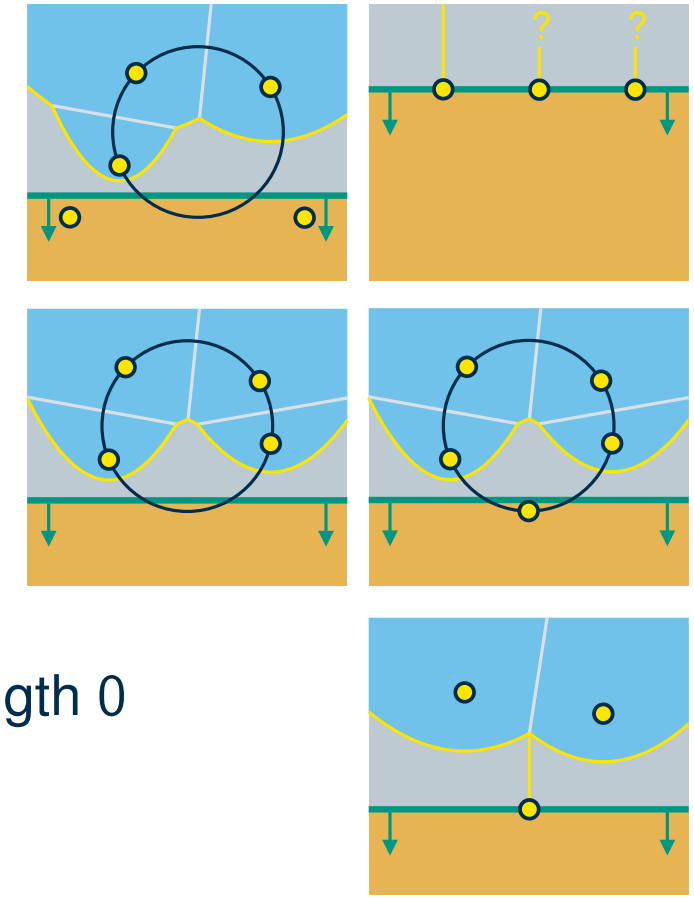
- handle in arbitrary order
- exception: initial site events (no parabola above exists)

Equal x- and y-Coordinate

- vertex in $\mathrm{Vor}(S)$ with degree > 3
- special handling $\rightarrow$ vertex with degree > 3
- no special handling $\rightarrow$ multiple degree-3 vertices with edges of length 0

New Parabola Starts At Intersection

Special Cases



Events With Equal y-Coordinate

- handle in arbitrary order
- exception: initial site events (no parabola above exists)

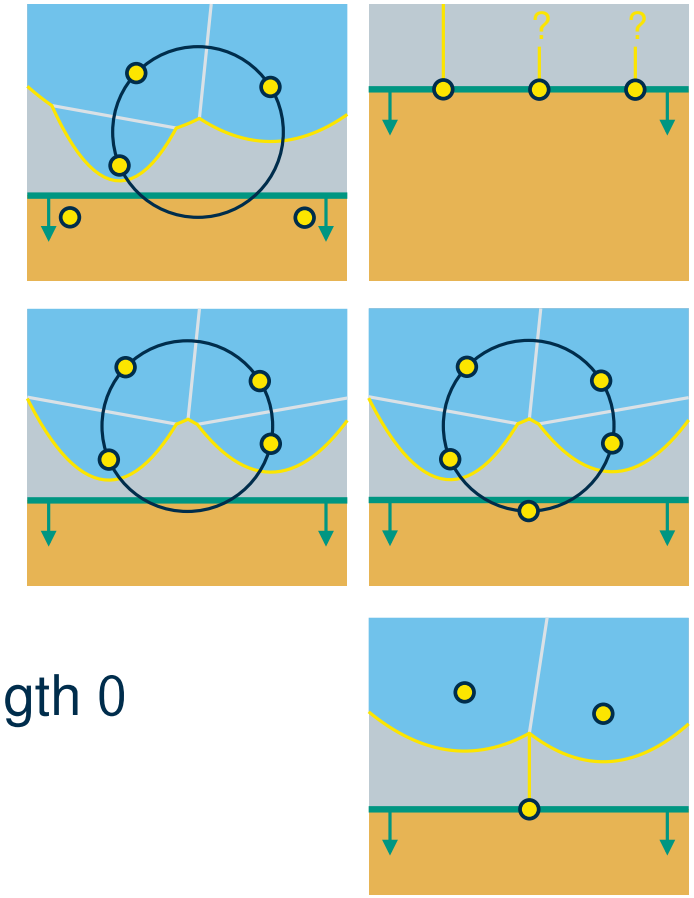
Equal x- and y-Coordinate

- vertex in $\mathrm{Vor}(S)$ with degree > 3
- special handling $\rightarrow$ vertex with degree > 3
- no special handling $\rightarrow$ multiple degree-3 vertices with edges of length 0

New Parabola Starts At Intersection

- a vertex should be created

Special Cases



Events With Equal y-Coordinate

- handle in arbitrary order
- exception: initial site events (no parabola above exists)

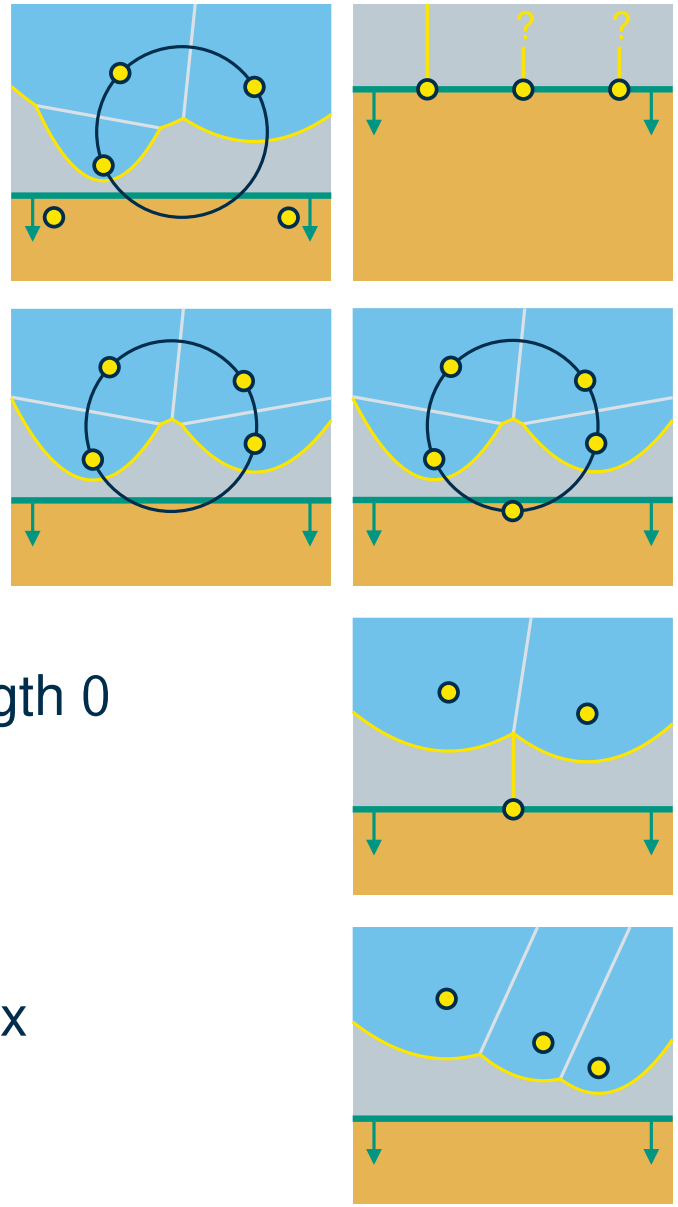
Equal x- and y-Coordinate

- vertex in $\mathrm{Vor}(S)$ with degree > 3
- special handling $\rightarrow$ vertex with degree > 3
- no special handling $\rightarrow$ multiple degree-3 vertices with edges of length 0

New Parabola Starts At Intersection

- a vertex should be created
- special handling: create vertex and half edges
- no special handling: split one parabola $\rightarrow$ circle event $\rightarrow$ new vertex

Special Cases



Events With Equal y-Coordinate

- handle in arbitrary order
- exception: initial site events (no parabola above exists)

Equal x- and y-Coordinate

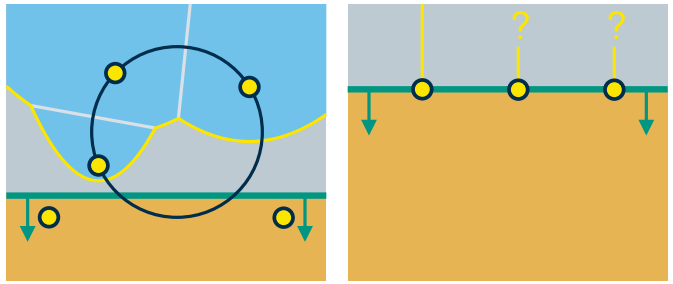
- vertex in $\mathrm{Vor}(S)$ with degree > 3
- special handling $\rightarrow$ vertex with degree > 3
- no special handling $\rightarrow$ multiple degree-3 vertices with edges of length 0

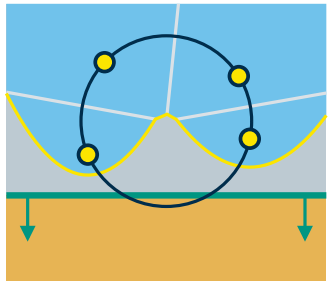
New Parabola Starts At Intersection

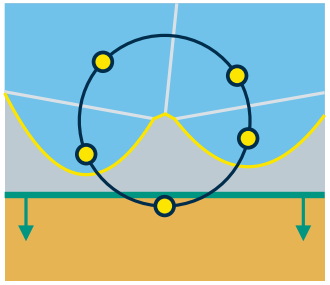
- a vertex should be created
- special handling: create vertex and half edges
- no special handling: split one parabola $\rightarrow$ circle event $\rightarrow$ new vertex

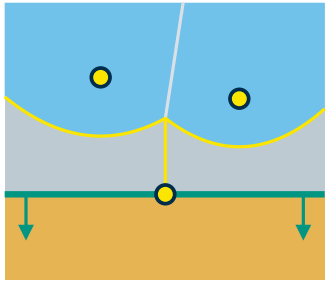
Collinear Sites With Consecutive Parabola Pieces

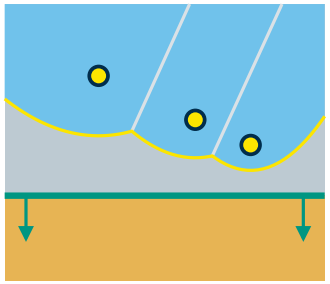
Special Cases











Events With Equal y-Coordinate

- handle in arbitrary order
- exception: initial site events (no parabola above exists)

Equal x- and y-Coordinate

- vertex in $\mathrm{Vor}(S)$ with degree > 3
- special handling $\rightarrow$ vertex with degree > 3
- no special handling $\rightarrow$ multiple degree-3 vertices with edges of length 0

New Parabola Starts At Intersection

- a vertex should be created
- special handling: create vertex and half edges
- no special handling: split one parabola $\rightarrow$ circle event $\rightarrow$ new vertex

Collinear Sites With Consecutive Parabola Pieces

- careful, when computing the circle

Overview

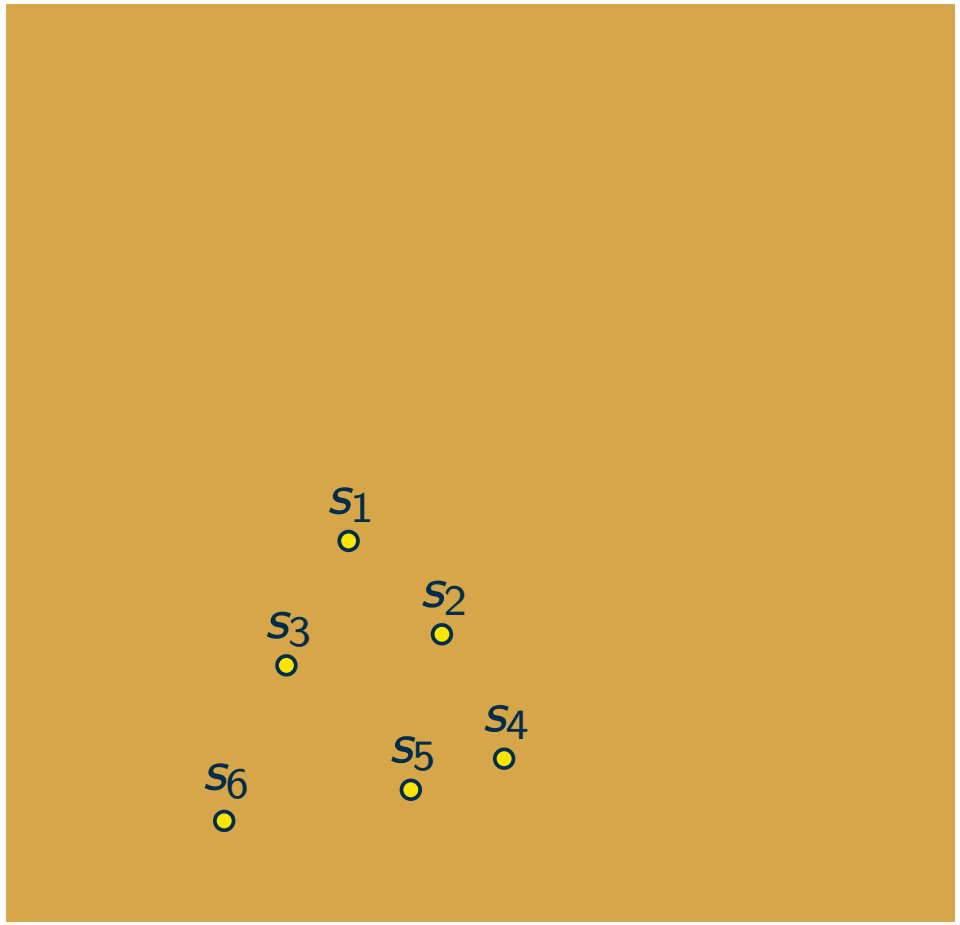
Site Event (Site s)

- find s_x in the beach line
- insert two new intersections
- handle false alarm for intersected parabola piece
- test new intersections for circle events
- new pair of half edges for the intersections

Circle Event (For Intersections $(s_i, s_j), (s_j, s_k)$)

- remove (s_i, s_j) and (s_j, s_k)
- insert (s_i, s_k)
- test (s_i, s_k) with BL-neighbors for circle events
- create new vertex v
- attach half edge of (s_i, s_j) and (s_j, s_k) to v
- new pair of half edges: one at v, one at (s_i, s_k)

Overview



beach line:

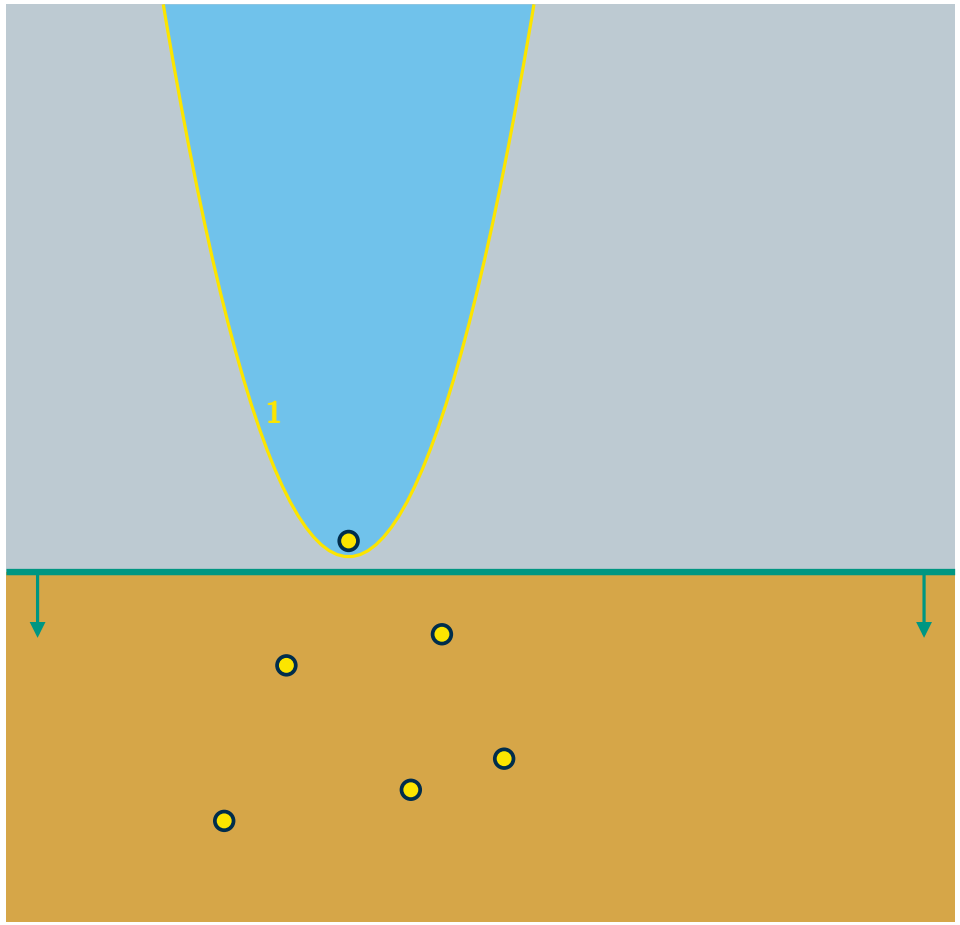
Site Event (Site s)

- find s_x in the beach line
- insert two new intersections
- handle false alarm for intersected parabola piece
- test new intersections for circle events
- new pair of half edges for the intersections

Circle Event (For Intersections $(s_i, s_j), (s_j, s_k)$)

- remove (s_i, s_j) and (s_j, s_k)
- insert (s_i, s_k)
- test (s_i, s_k) with BL-neighbors for circle events
- create new vertex v
- attach half edge of (s_i, s_j) and (s_j, s_k) to v
- new pair of half edges: one at v, one at (s_i, s_k)

Overview



beach line:

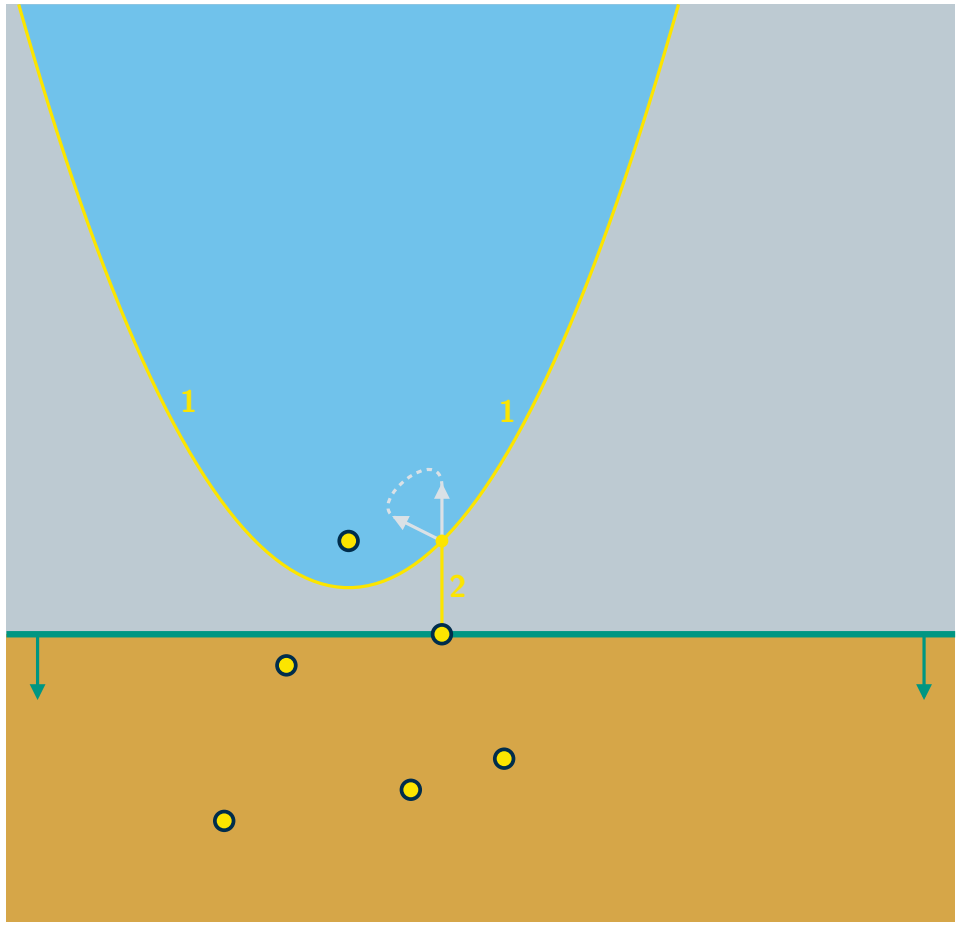
Site Event (Site s)

- find s_x in the beach line
- insert two new intersections
- handle false alarm for intersected parabola piece
- test new intersections for circle events
- new pair of half edges for the intersections

Circle Event (For Intersections $(s_i, s_j), (s_j, s_k)$)

- remove (s_i, s_j) and (s_j, s_k)
- insert (s_i, s_k)
- test (s_i, s_k) with BL-neighbors for circle events
- create new vertex v
- attach half edge of (s_i, s_j) and (s_j, s_k) to v
- new pair of half edges: one at v, one at (s_i, s_k)

Overview



beach line: (s_1, s_2) (s_2, s_1)

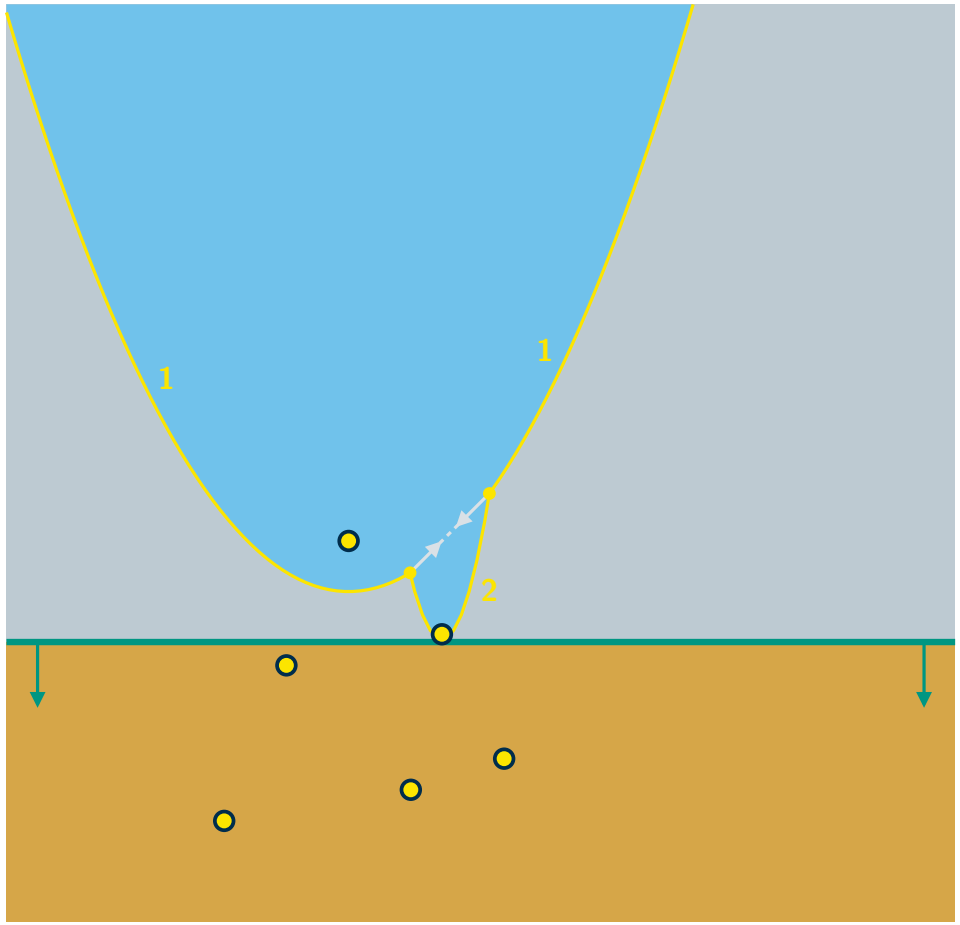
Site Event (Site s)

- find s_x in the beach line
- insert two new intersections
- handle false alarm for intersected parabola piece
- test new intersections for circle events
- new pair of half edges for the intersections

Circle Event (For Intersections (s_i, s_j), (s_j, s_k))

- remove (s_i, s_j) and (s_j, s_k)
- insert (s_i, s_k)
- test (s_i, s_k) with BL-neighbors for circle events
- create new vertex v
- attach half edge of (s_i, s_j) and (s_j, s_k) to v
- new pair of half edges: one at v, one at (s_i, s_k)

Overview



beach line: (s_1, s_2) (s_2, s_1)

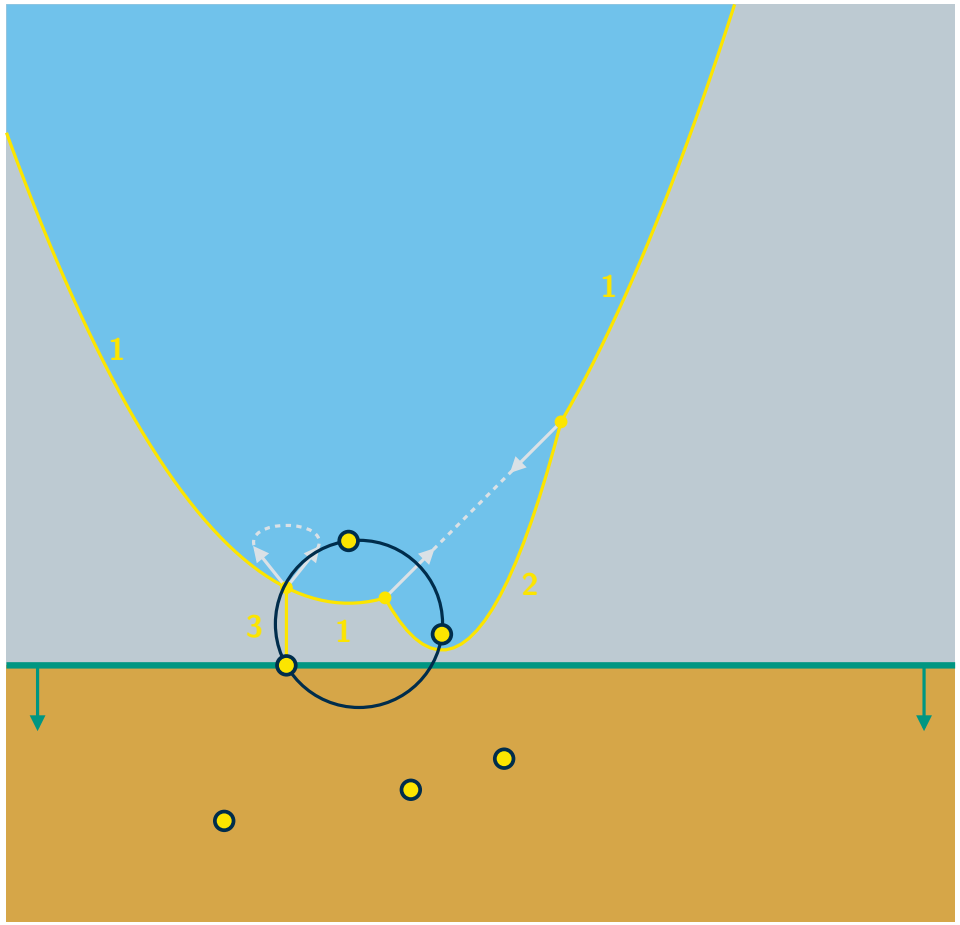
Site Event (Site s)

- find s_x in the beach line
- insert two new intersections
- handle false alarm for intersected parabola piece
- test new intersections for circle events
- new pair of half edges for the intersections

Circle Event (For Intersections $(s_i, s_j), (s_j, s_k)$)

- remove (s_i, s_j) and (s_j, s_k)
- insert (s_i, s_k)
- test (s_i, s_k) with BL-neighbors for circle events
- create new vertex v
- attach half edge of (s_i, s_j) and (s_j, s_k) to v
- new pair of half edges: one at v, one at (s_i, s_k)

Overview



beach line: (s_1, s_3) (s_3, s_1) (s_1, s_2) (s_2, s_1)

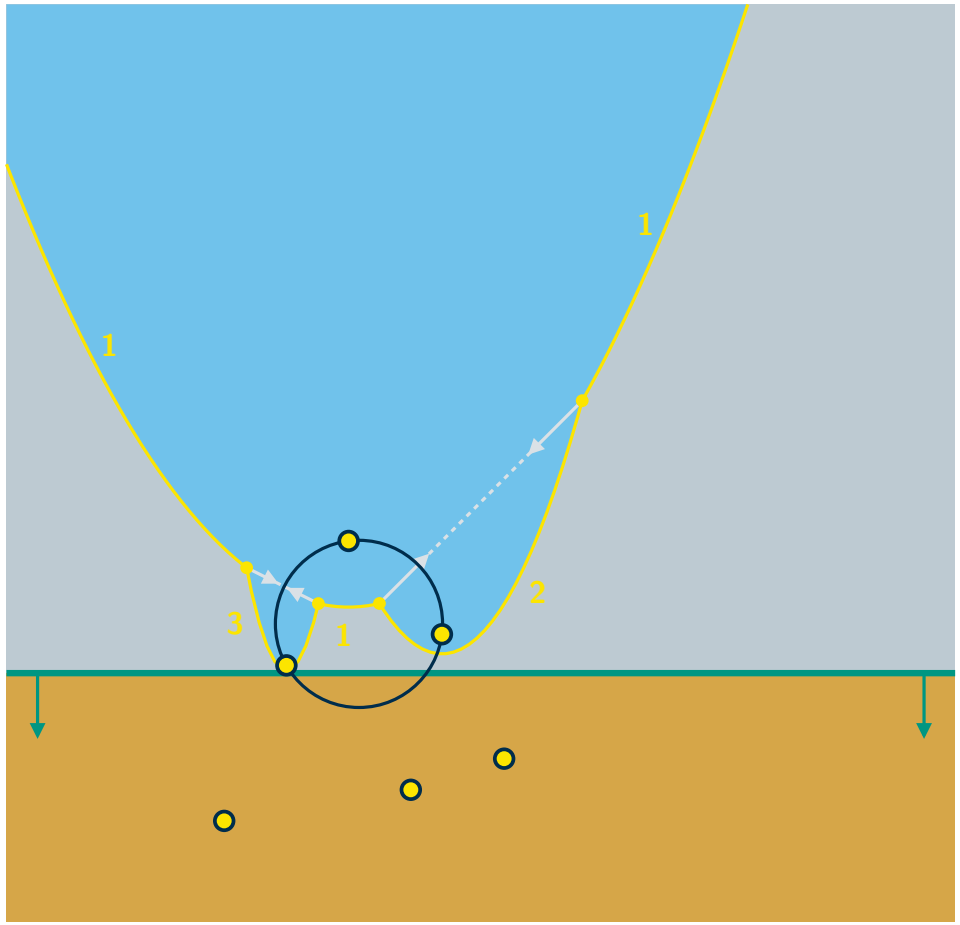
Site Event (Site s)

- find s_x in the beach line
- insert two new intersections
- handle false alarm for intersected parabola piece
- test new intersections for circle events
- new pair of half edges for the intersections

Circle Event (For Intersections $(s_i, s_j), (s_j, s_k)$)

- remove (s_i, s_j) and (s_j, s_k)
- insert (s_i, s_k)
- test (s_i, s_k) with BL-neighbors for circle events
- create new vertex v
- attach half edge of (s_i, s_j) and (s_j, s_k) to v
- new pair of half edges: one at v, one at (s_i, s_k)

Overview



beach line: (s_1, s_3) (s_3, s_1) (s_1, s_2) (s_2, s_1)

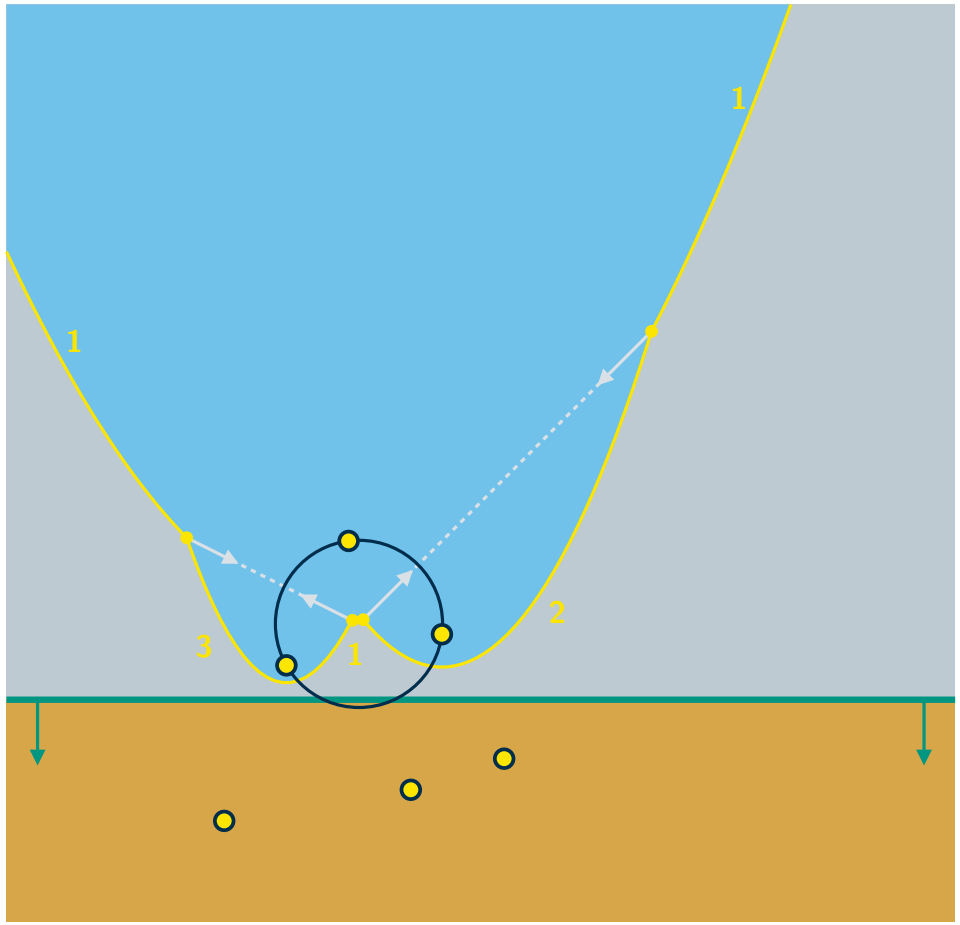
Site Event (Site s)

- find s_x in the beach line
- insert two new intersections
- handle false alarm for intersected parabola piece
- test new intersections for circle events
- new pair of half edges for the intersections

Circle Event (For Intersections $(s_i, s_j), (s_j, s_k)$)

- remove (s_i, s_j) and (s_j, s_k)
- insert (s_i, s_k)
- test (s_i, s_k) with BL-neighbors for circle events
- create new vertex v
- attach half edge of (s_i, s_j) and (s_j, s_k) to v
- new pair of half edges: one at v, one at (s_i, s_k)

Overview



beach line: (s_1, s_3) (s_3, s_1) (s_1, s_2) (s_2, s_1)

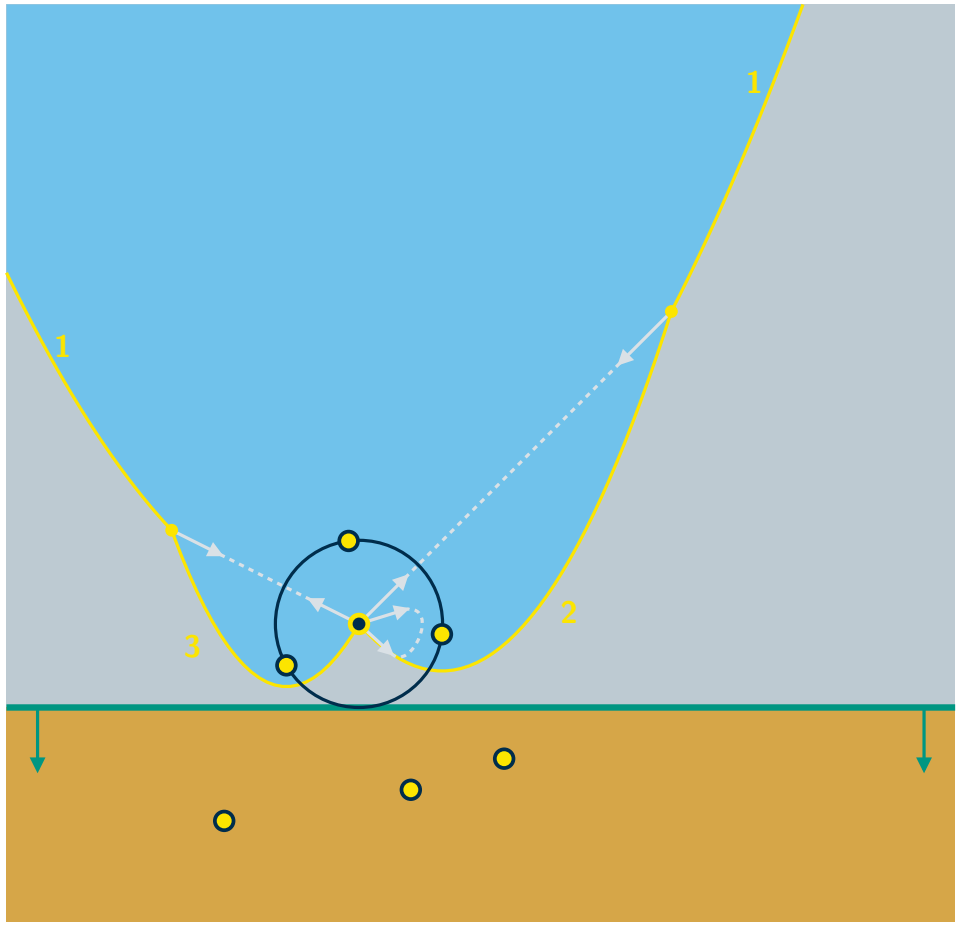
Site Event (Site s)

- find s_x in the beach line
- insert two new intersections
- handle false alarm for intersected parabola piece
- test new intersections for circle events
- new pair of half edges for the intersections

Circle Event (For Intersections $(s_i, s_j), (s_j, s_k)$)

- remove (s_i, s_j) and (s_j, s_k)
- insert (s_i, s_k)
- test (s_i, s_k) with BL-neighbors for circle events
- create new vertex v
- attach half edge of (s_i, s_j) and (s_j, s_k) to v
- new pair of half edges: one at v, one at (s_i, s_k)

Overview



beach line: (s_1, s_3) (s_3, s_2) (s_2, s_1)

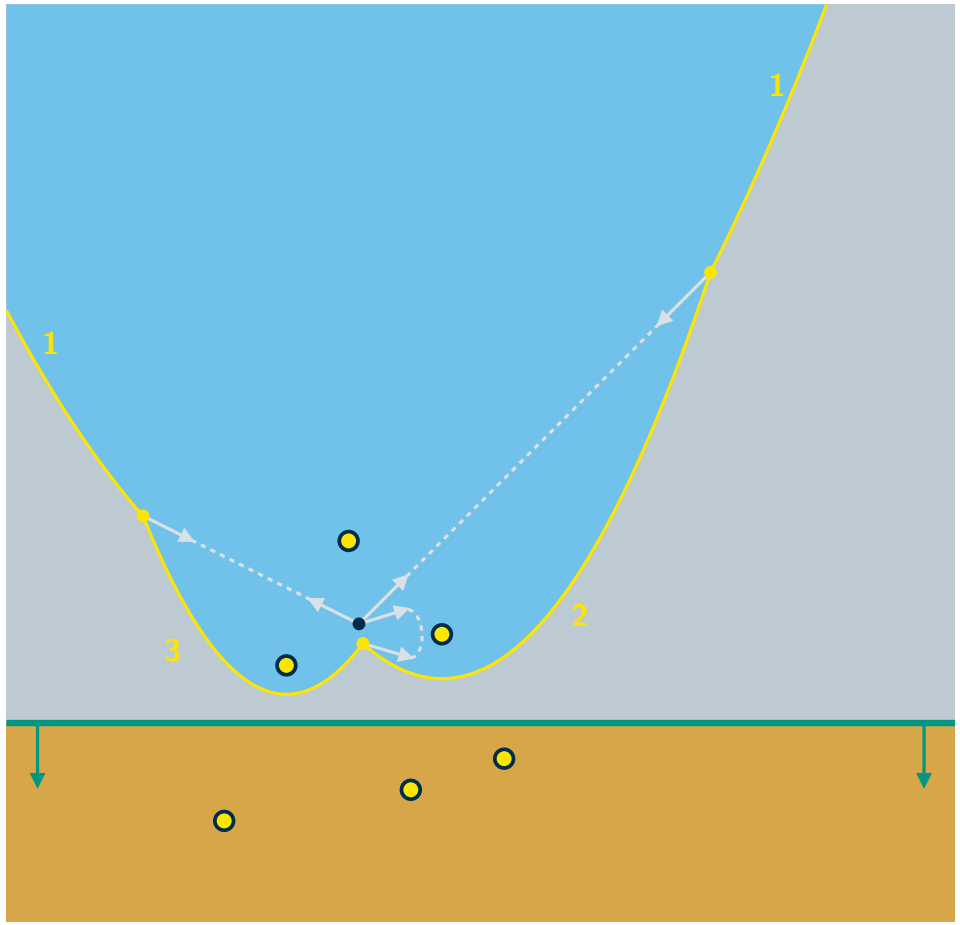
Site Event (Site s)

- find s_x in the beach line
- insert two new intersections
- handle false alarm for intersected parabola piece
- test new intersections for circle events
- new pair of half edges for the intersections

Circle Event (For Intersections (s_i, s_j), (s_j, s_k))

- remove (s_i, s_j) and (s_j, s_k)
- insert (s_i, s_k)
- test (s_i, s_k) with BL-neighbors for circle events
- create new vertex v
- attach half edge of (s_i, s_j) and (s_j, s_k) to v
- new pair of half edges: one at v, one at (s_i, s_k)

Overview



beach line: (s_1, s_3) (s_3, s_2) (s_2, s_1)

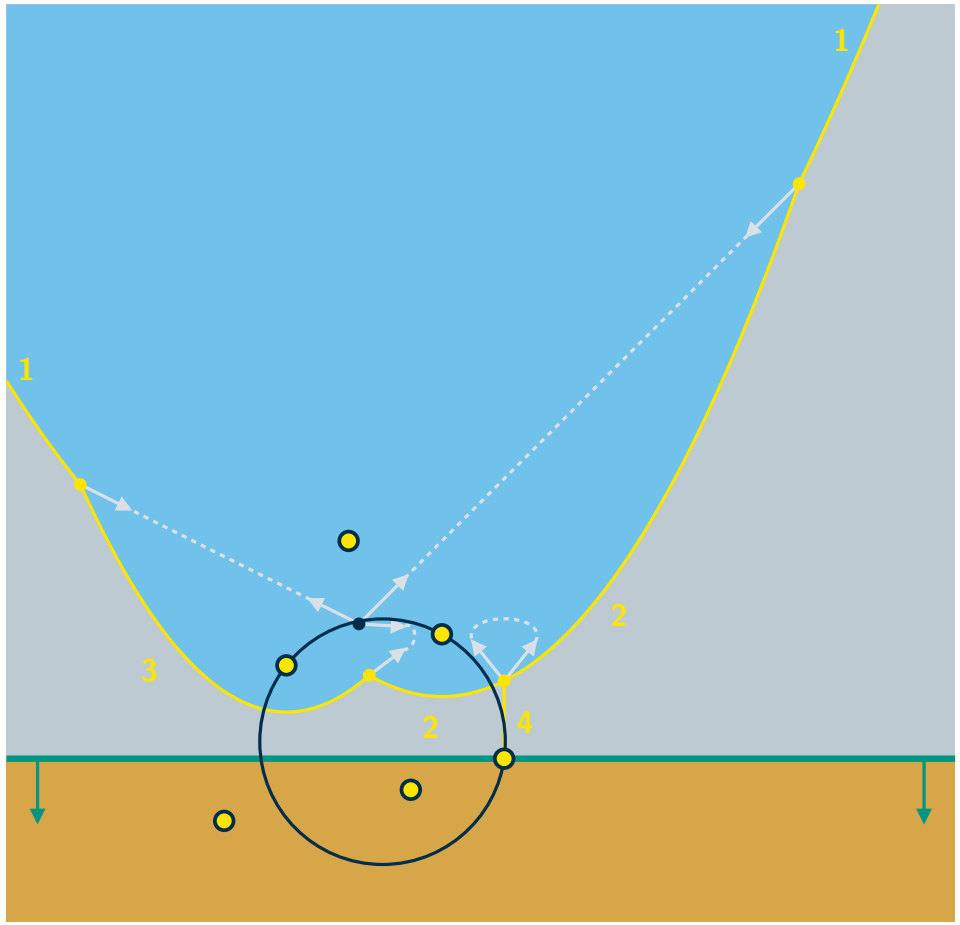
Site Event (Site s)

- find s_x in the beach line
- insert two new intersections
- handle false alarm for intersected parabola piece
- test new intersections for circle events
- new pair of half edges for the intersections

Circle Event (For Intersections $(s_i, s_j), (s_j, s_k)$)

- remove (s_i, s_j) and (s_j, s_k)
- insert (s_i, s_k)
- test (s_i, s_k) with BL-neighbors for circle events
- create new vertex v
- attach half edge of (s_i, s_j) and (s_j, s_k) to v
- new pair of half edges: one at v, one at (s_i, s_k)

Overview



beach line: (s_1, s_3) (s_3, s_2) (s_2, s_4) (s_4, s_2) (s_2, s_1)

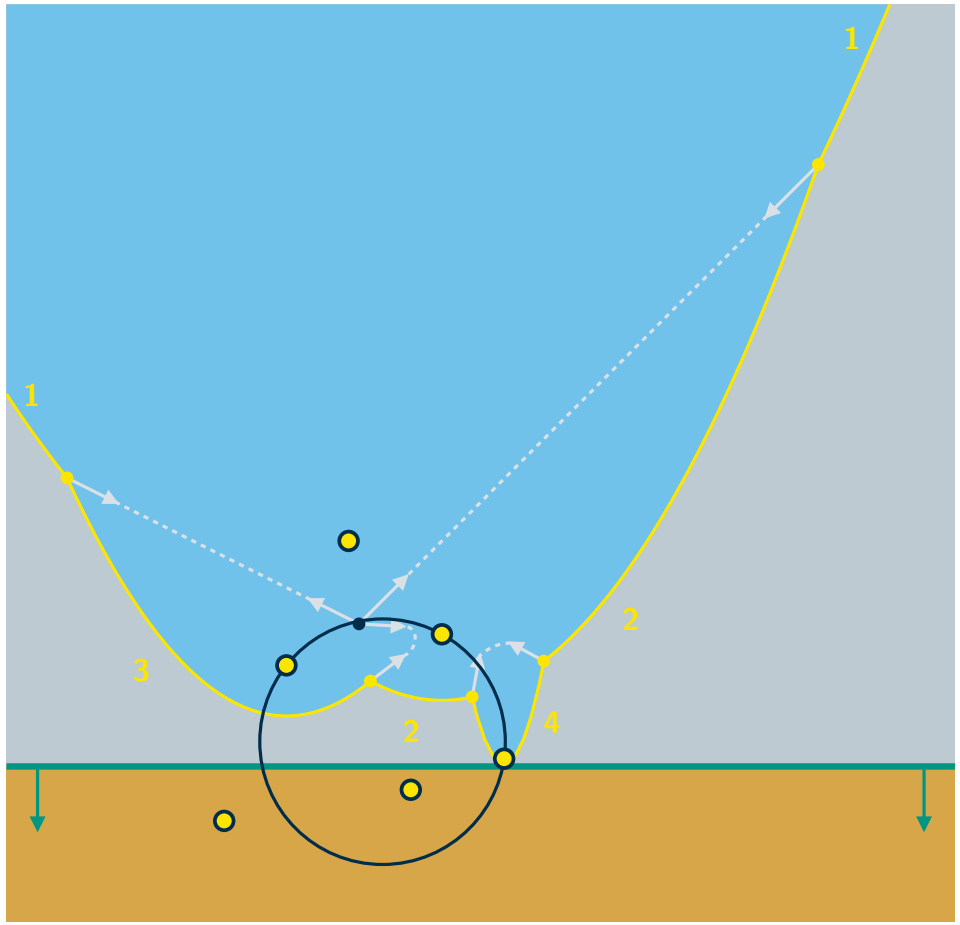
Site Event (Site s)

- find s_x in the beach line
- insert two new intersections
- handle false alarm for intersected parabola piece
- test new intersections for circle events
- new pair of half edges for the intersections

Circle Event (For Intersections (s_i, s_j), (s_j, s_k))

- remove (s_i, s_j) and (s_j, s_k)
- insert (s_i, s_k)
- test (s_i, s_k) with BL-neighbors for circle events
- create new vertex v
- attach half edge of (s_i, s_j) and (s_j, s_k) to v
- new pair of half edges: one at v, one at (s_i, s_k)

Overview



beach line: (s_1, s_3) (s_3, s_2) (s_2, s_4) (s_4, s_2) (s_2, s_1)

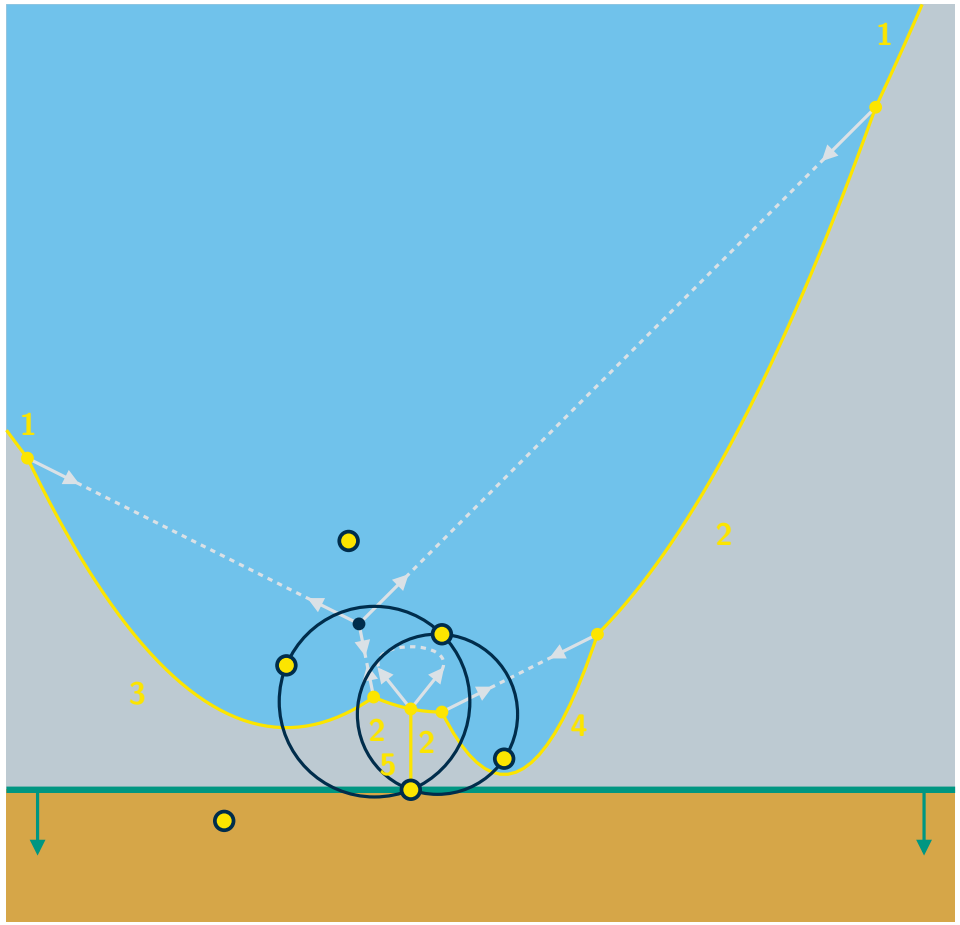
Site Event (Site s)

- find s_x in the beach line
- insert two new intersections
- handle false alarm for intersected parabola piece
- test new intersections for circle events
- new pair of half edges for the intersections

Circle Event (For Intersections $(s_i, s_j), (s_j, s_k)$)

- remove (s_i, s_j) and (s_j, s_k)
- insert (s_i, s_k)
- test (s_i, s_k) with BL-neighbors for circle events
- create new vertex v
- attach half edge of (s_i, s_j) and (s_j, s_k) to v
- new pair of half edges: one at v, one at (s_i, s_k)

Overview



Site Event (Site s)

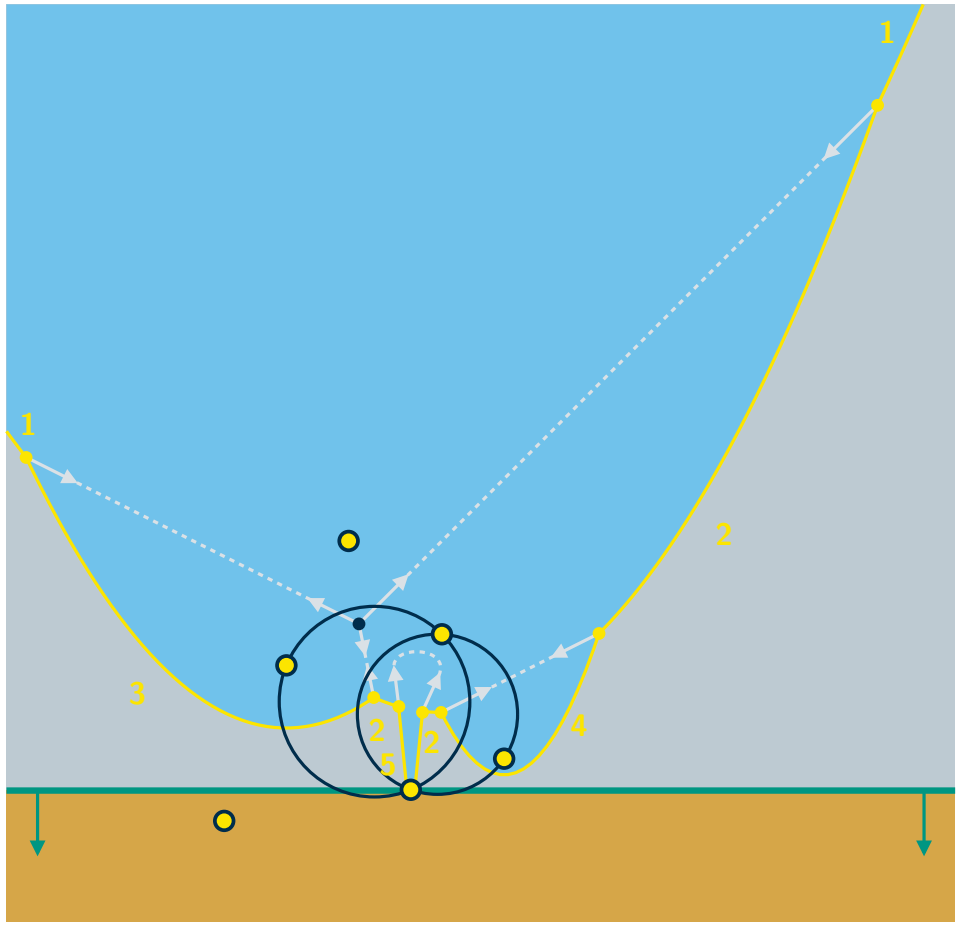
- find s_x in the beach line
- insert two new intersections
- handle false alarm for intersected parabola piece
- test new intersections for circle events
- new pair of half edges for the intersections

Circle Event (For Intersections $(s_i, s_j), (s_j, s_k)$)

- remove (s_i, s_j) and (s_j, s_k)
- insert (s_i, s_k)
- test (s_i, s_k) with BL-neighbors for circle events
- create new vertex v
- attach half edge of (s_i, s_j) and (s_j, s_k) to v
- new pair of half edges: one at v, one at (s_i, s_k)

beach line: (s_1, s_3) (s_3, s_2) (s_2, s_5) (s_5, s_2) (s_2, s_4) (s_4, s_2) (s_2, s_1)

Overview



Site Event (Site s)

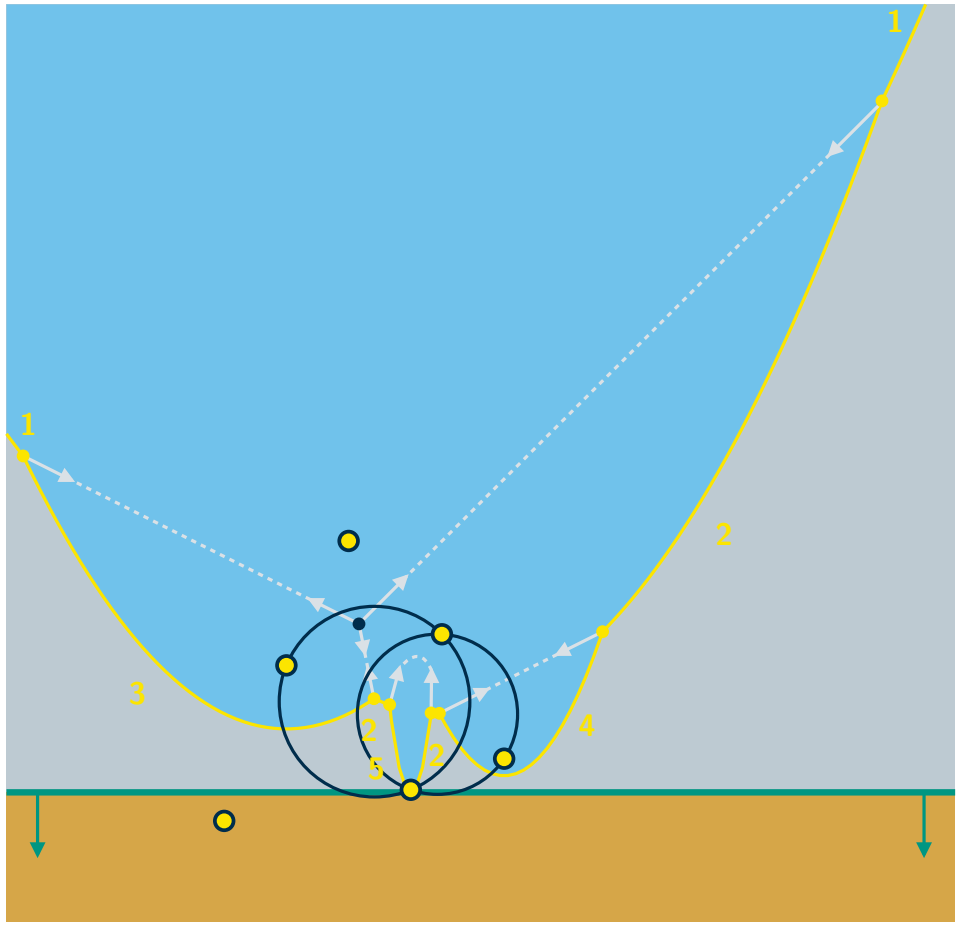
- find s_x in the beach line
- insert two new intersections
- handle false alarm for intersected parabola piece
- test new intersections for circle events
- new pair of half edges for the intersections

Circle Event (For Intersections $(s_i, s_j), (s_j, s_k)$)

- remove (s_i, s_j) and (s_j, s_k)
- insert (s_i, s_k)
- test (s_i, s_k) with BL-neighbors for circle events
- create new vertex v
- attach half edge of (s_i, s_j) and (s_j, s_k) to v
- new pair of half edges: one at v, one at (s_i, s_k)

beach line: (s_1, s_3) (s_3, s_2) (s_2, s_5) (s_5, s_2) (s_2, s_4) (s_4, s_2) (s_2, s_1)

Overview



Site Event (Site s)

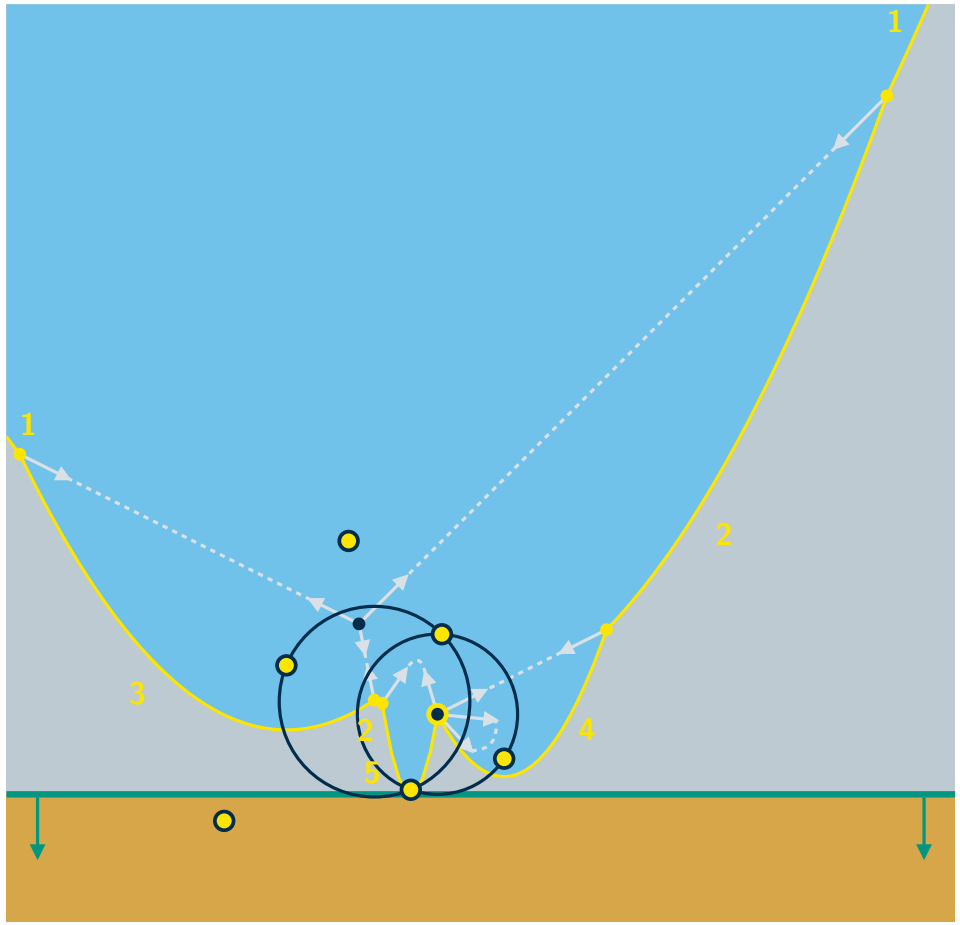
- find s_x in the beach line
- insert two new intersections
- handle false alarm for intersected parabola piece
- test new intersections for circle events
- new pair of half edges for the intersections

Circle Event (For Intersections $(s_i, s_j), (s_j, s_k)$)

- remove (s_i, s_j) and (s_j, s_k)
- insert (s_i, s_k)
- test (s_i, s_k) with BL-neighbors for circle events
- create new vertex v
- attach half edge of (s_i, s_j) and (s_j, s_k) to v
- new pair of half edges: one at v, one at (s_i, s_k)

beach line: (s_1, s_3) (s_3, s_2) (s_2, s_5) (s_5, s_2) (s_2, s_4) (s_4, s_2) (s_2, s_1)

Overview



Site Event (Site s)

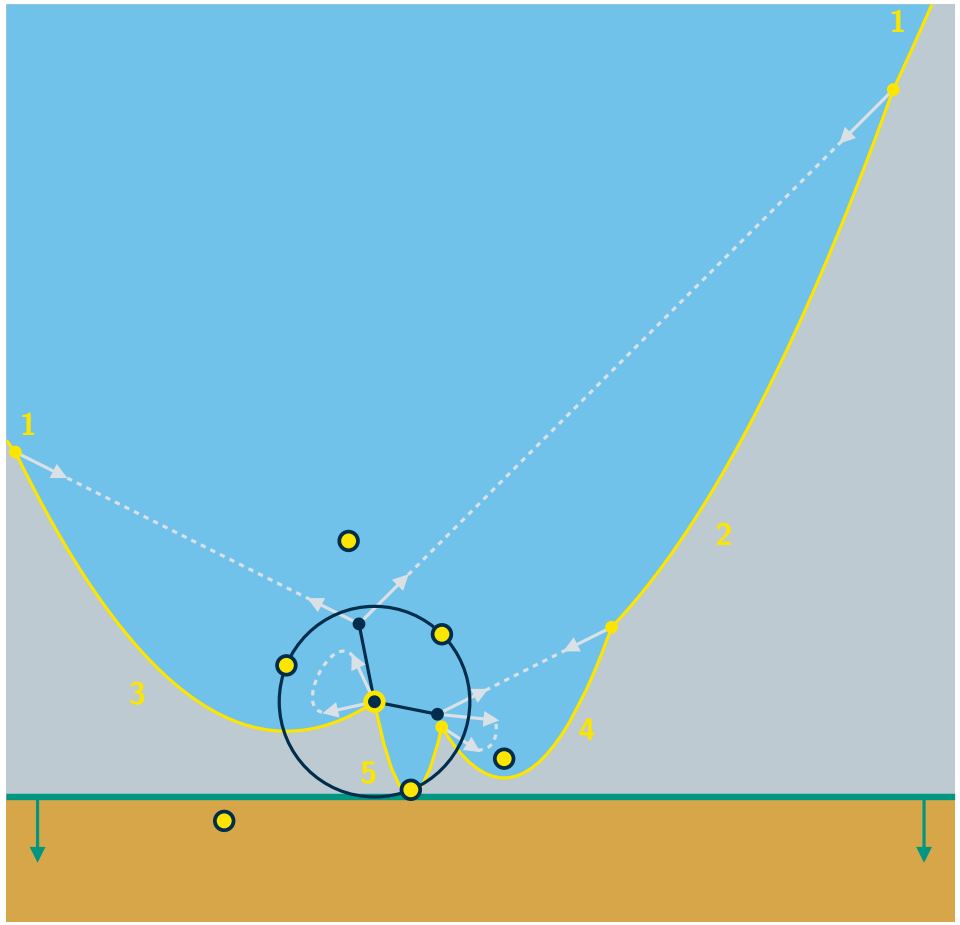
- find s_x in the beach line
- insert two new intersections
- handle false alarm for intersected parabola piece
- test new intersections for circle events
- new pair of half edges for the intersections

Circle Event (For Intersections $(s_i, s_j), (s_j, s_k)$)

- remove (s_i, s_j) and (s_j, s_k)
- insert (s_i, s_k)
- test (s_i, s_k) with BL-neighbors for circle events
- create new vertex v
- attach half edge of (s_i, s_j) and (s_j, s_k) to v
- new pair of half edges: one at v, one at (s_i, s_k)

beach line: (s_1, s_3) (s_3, s_2) (s_2, s_5) (s_5, s_4) (s_4, s_2) (s_2, s_1)

Overview



beach line: (s_1, s_3) (s_3, s_5) (s_5, s_4) (s_4, s_2) (s_2, s_1)

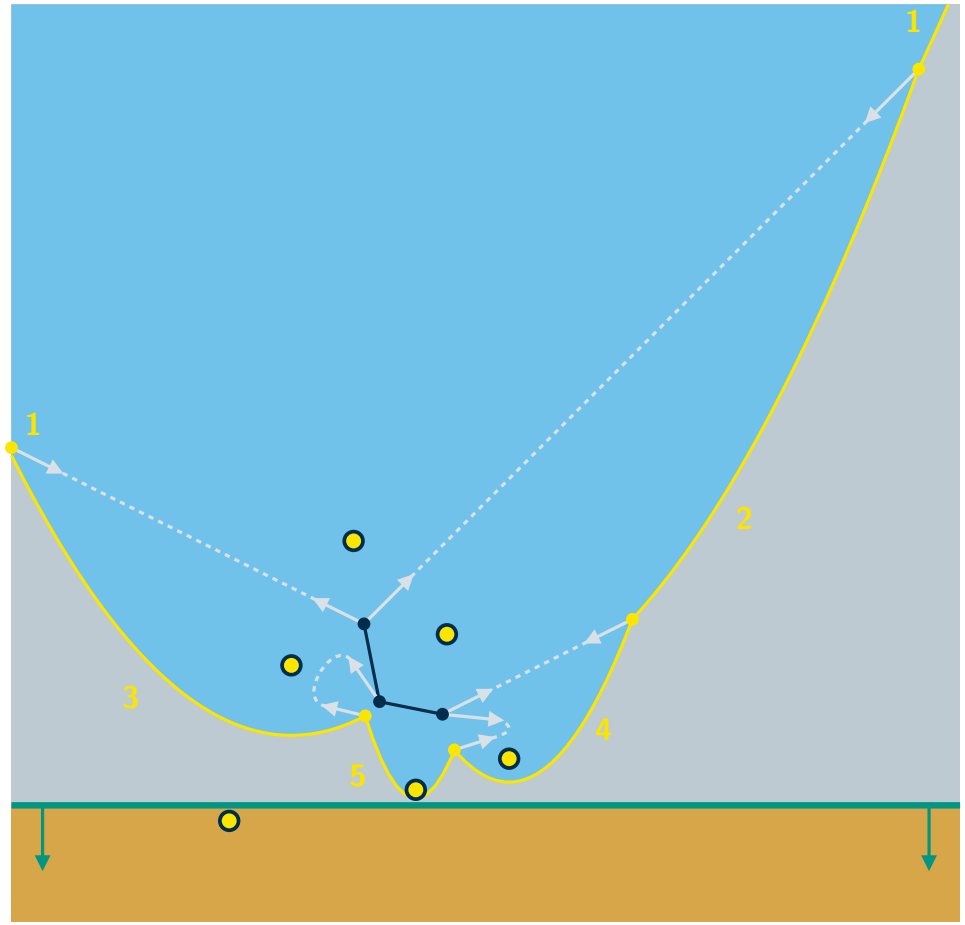
Site Event (Site s)

- find s_x in the beach line
- insert two new intersections
- handle false alarm for intersected parabola piece
- test new intersections for circle events
- new pair of half edges for the intersections

Circle Event (For Intersections $(s_i, s_j), (s_j, s_k)$)

- remove (s_i, s_j) and (s_j, s_k)
- insert (s_i, s_k)
- test (s_i, s_k) with BL-neighbors for circle events
- create new vertex v
- attach half edge of (s_i, s_j) and (s_j, s_k) to v
- new pair of half edges: one at v, one at (s_i, s_k)

Overview



beach line: (s_1, s_3) (s_3, s_5) (s_5, s_4) (s_4, s_2) (s_2, s_1)

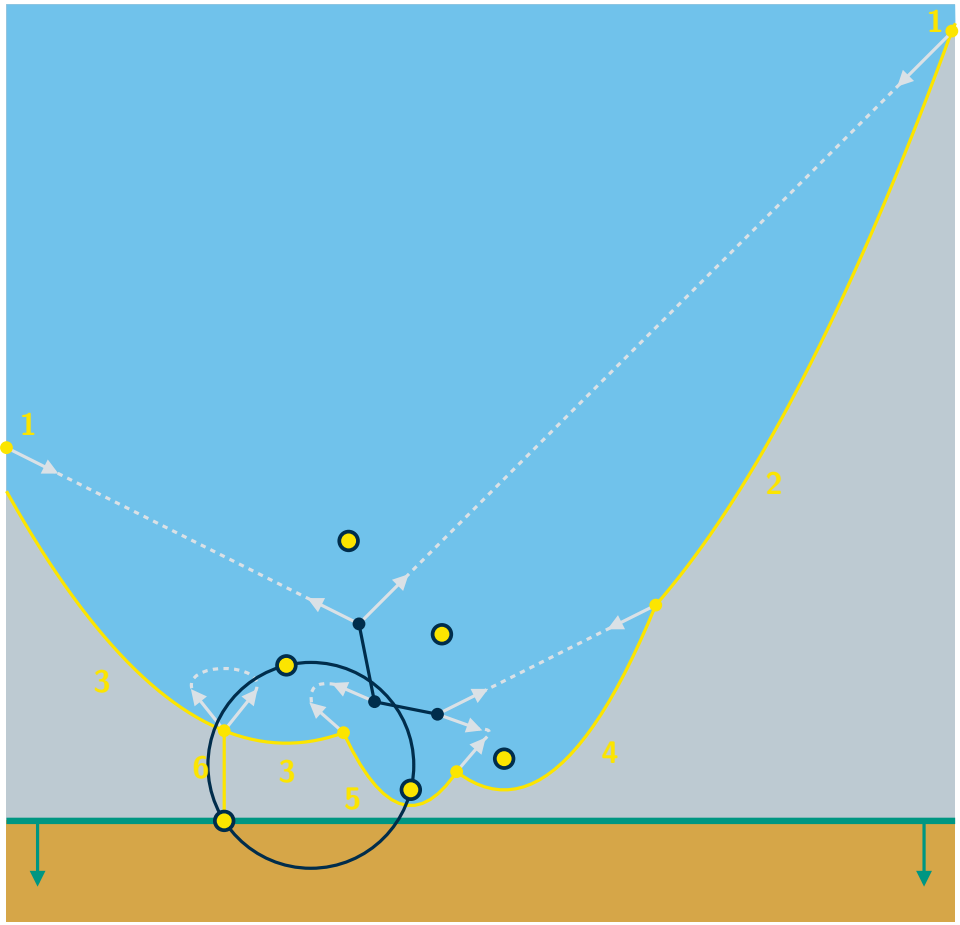
Site Event (Site s)

- find s_x in the beach line
- insert two new intersections
- handle false alarm for intersected parabola piece
- test new intersections for circle events
- new pair of half edges for the intersections

Circle Event (For Intersections $(s_i, s_j), (s_j, s_k)$)

- remove (s_i, s_j) and (s_j, s_k)
- insert (s_i, s_k)
- test (s_i, s_k) with BL-neighbors for circle events
- create new vertex v
- attach half edge of (s_i, s_j) and (s_j, s_k) to v
- new pair of half edges: one at v, one at (s_i, s_k)

Overview



Site Event (Site s)

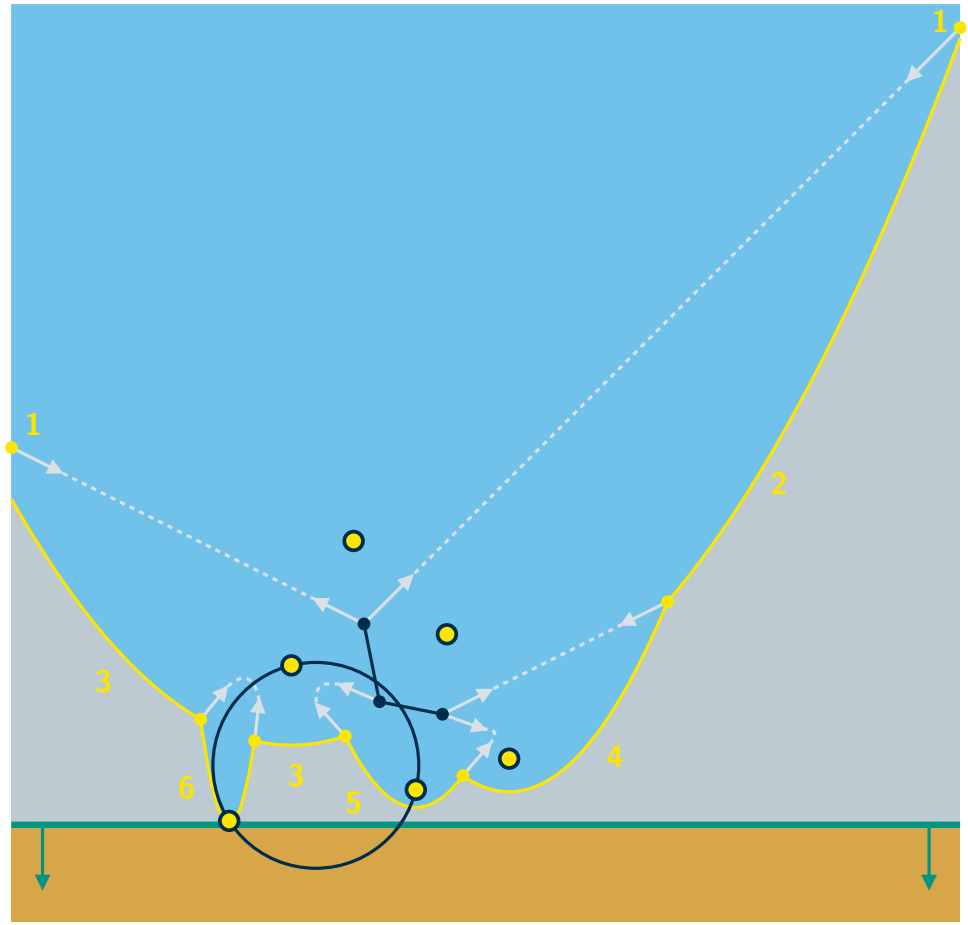
- find s_x in the beach line
- insert two new intersections
- handle false alarm for intersected parabola piece
- test new intersections for circle events
- new pair of half edges for the intersections

Circle Event (For Intersections $(s_i, s_j), (s_j, s_k)$)

- remove (s_i, s_j) and (s_j, s_k)
- insert (s_i, s_k)
- test (s_i, s_k) with BL-neighbors for circle events
- create new vertex v
- attach half edge of (s_i, s_j) and (s_j, s_k) to v
- new pair of half edges: one at v, one at (s_i, s_k)

beach line: (s_1, s_3) (s_3, s_6) (s_6, s_3) (s_3, s_5) (s_5, s_4) (s_4, s_2) (s_2, s_1)

Overview



Site Event (Site s)

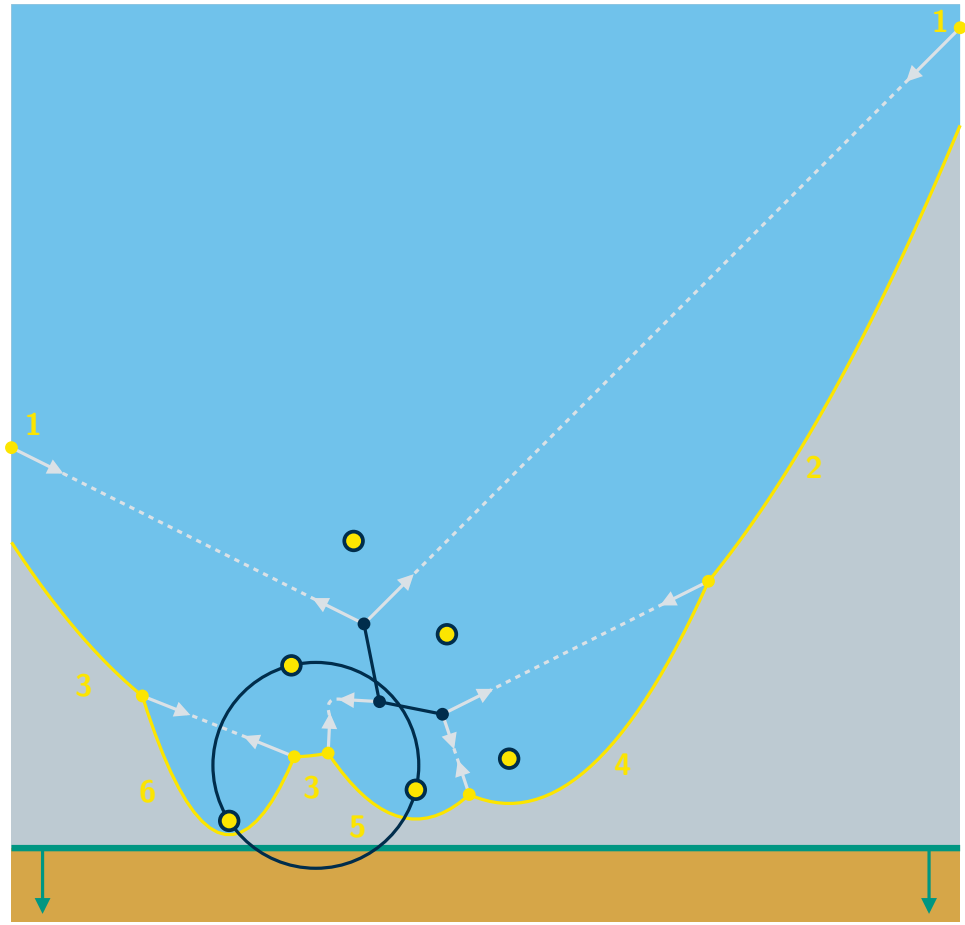
- find s_x in the beach line
- insert two new intersections
- handle false alarm for intersected parabola piece
- test new intersections for circle events
- new pair of half edges for the intersections

Circle Event (For Intersections $(s_i, s_j), (s_j, s_k)$)

- remove (s_i, s_j) and (s_j, s_k)
- insert (s_i, s_k)
- test (s_i, s_k) with BL-neighbors for circle events
- create new vertex v
- attach half edge of (s_i, s_j) and (s_j, s_k) to v
- new pair of half edges: one at v, one at (s_i, s_k)

beach line: (s_1, s_3) (s_3, s_6) (s_6, s_3) (s_3, s_5) (s_5, s_4) (s_4, s_2) (s_2, s_1)

Overview



Site Event (Site s)

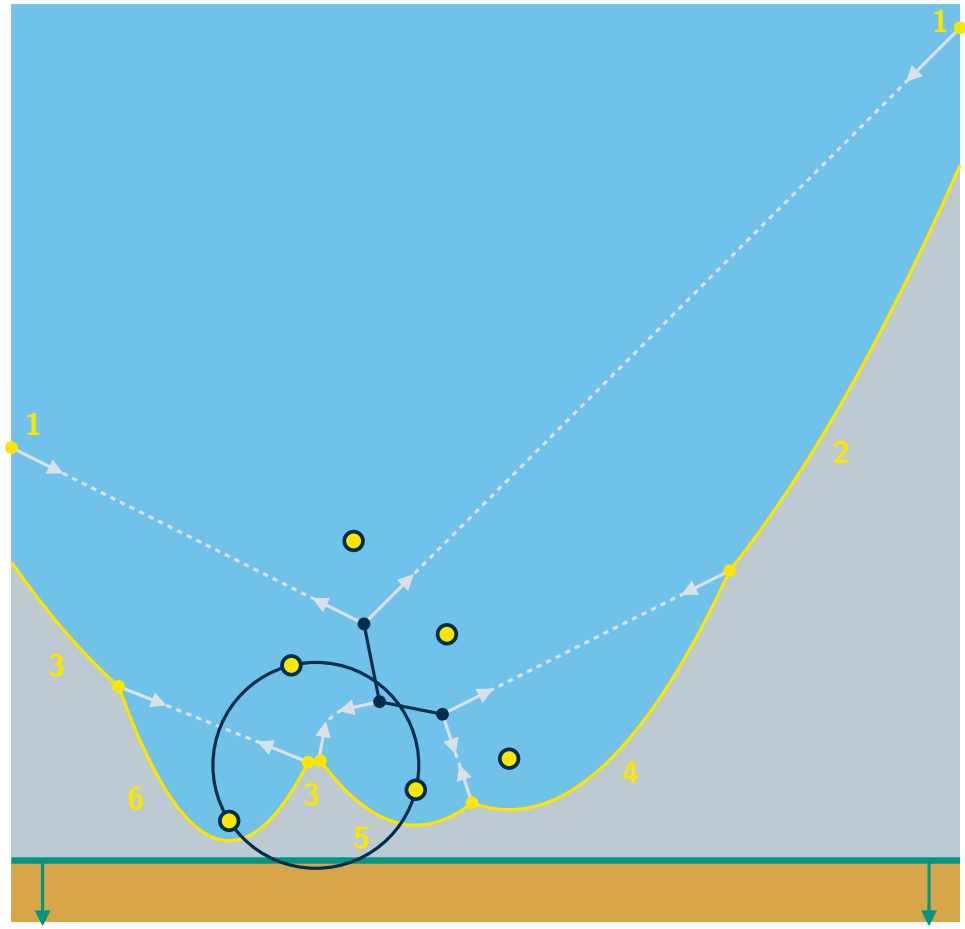
- find s_x in the beach line
- insert two new intersections
- handle false alarm for intersected parabola piece
- test new intersections for circle events
- new pair of half edges for the intersections

Circle Event (For Intersections $(s_i, s_j), (s_j, s_k)$)

- remove (s_i, s_j) and (s_j, s_k)
- insert (s_i, s_k)
- test (s_i, s_k) with BL-neighbors for circle events
- create new vertex v
- attach half edge of (s_i, s_j) and (s_j, s_k) to v
- new pair of half edges: one at v, one at (s_i, s_k)

beach line: (s_1, s_3) (s_3, s_6) (s_6, s_3) (s_3, s_5) (s_5, s_4) (s_4, s_2) (s_2, s_1)

Overview



Site Event (Site s)

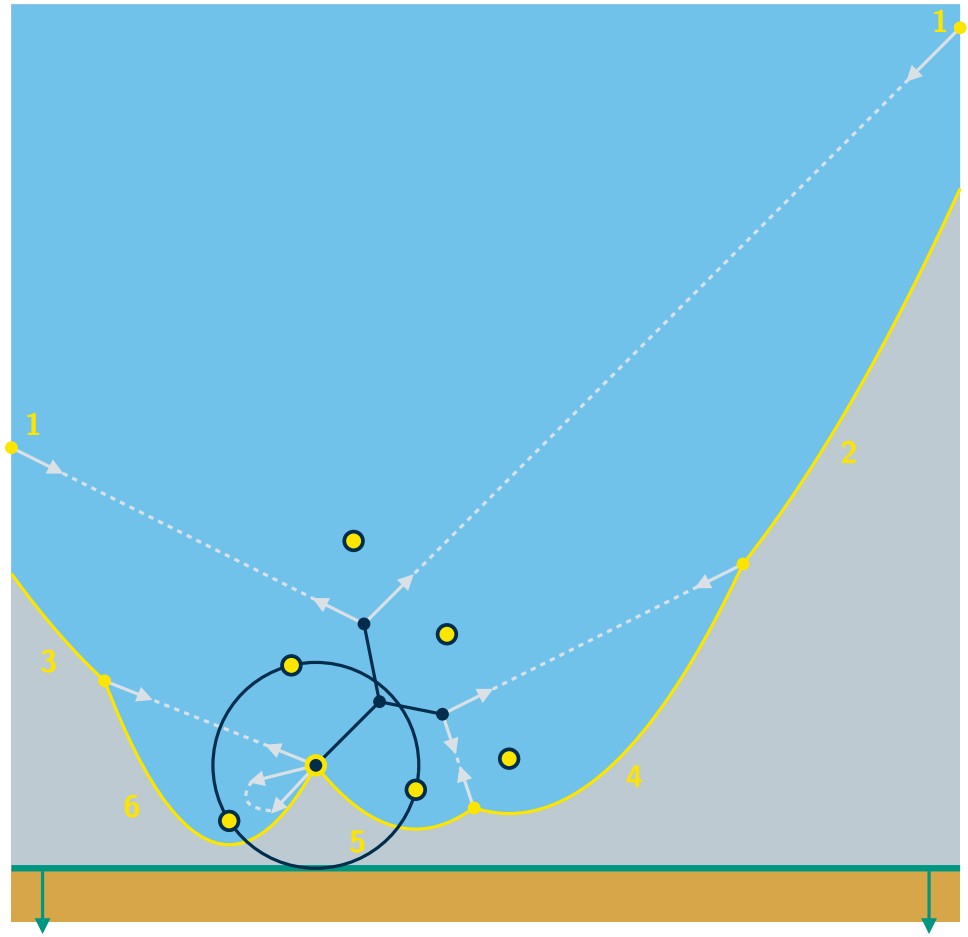
- find s_x in the beach line
- insert two new intersections
- handle false alarm for intersected parabola piece
- test new intersections for circle events
- new pair of half edges for the intersections

Circle Event (For Intersections $(s_i, s_j), (s_j, s_k)$)

- remove (s_i, s_j) and (s_j, s_k)
- insert (s_i, s_k)
- test (s_i, s_k) with BL-neighbors for circle events
- create new vertex v
- attach half edge of (s_i, s_j) and (s_j, s_k) to v
- new pair of half edges: one at v, one at (s_i, s_k)

beach line: (s_1, s_3) (s_3, s_6) (s_6, s_3) (s_3, s_5) (s_5, s_4) (s_4, s_2) (s_2, s_1)

Overview



beach line: (s_1, s_3) (s_3, s_6) (s_6, s_5) (s_5, s_4) (s_4, s_2) (s_2, s_1)

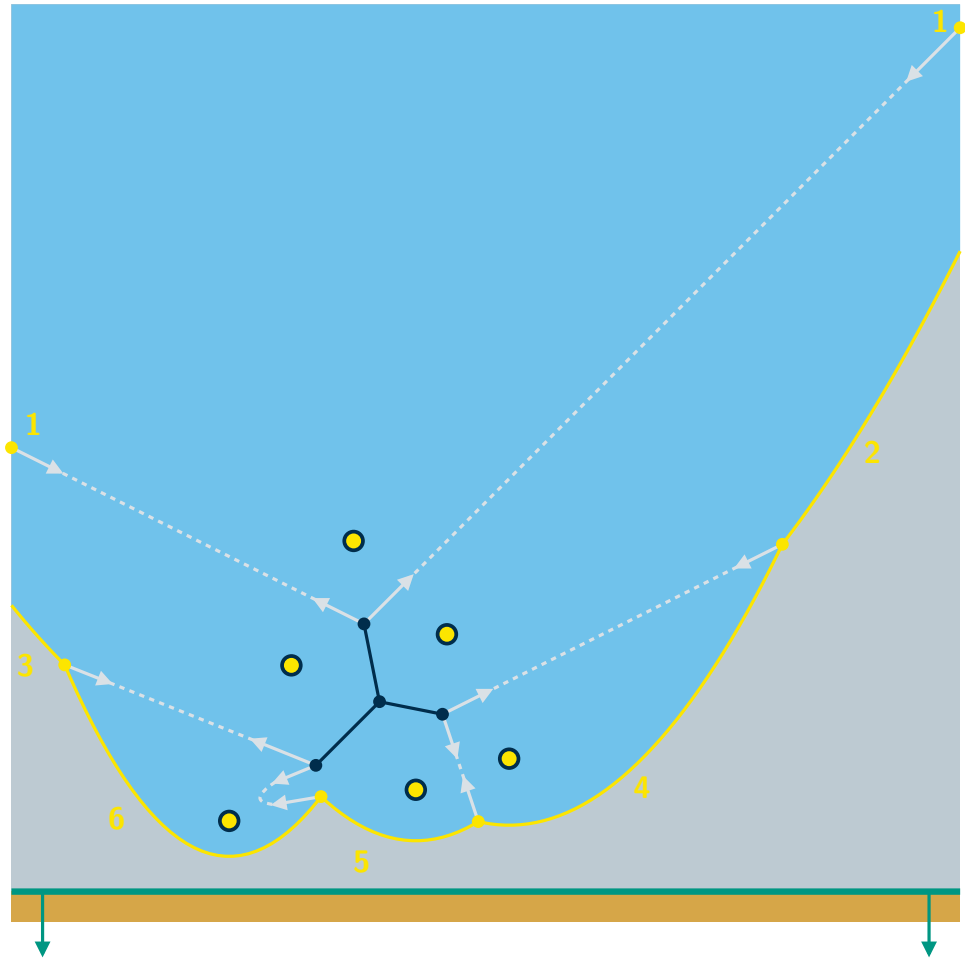
Site Event (Site s)

- find s_x in the beach line
- insert two new intersections
- handle false alarm for intersected parabola piece
- test new intersections for circle events
- new pair of half edges for the intersections

Circle Event (For Intersections $(s_i, s_j), (s_j, s_k)$)

- remove (s_i, s_j) and (s_j, s_k)
- insert (s_i, s_k)
- test (s_i, s_k) with BL-neighbors for circle events
- create new vertex v
- attach half edge of (s_i, s_j) and (s_j, s_k) to v
- new pair of half edges: one at v, one at (s_i, s_k)

Overview



beach line: (s_1, s_3) (s_3, s_6) (s_6, s_5) (s_5, s_4) (s_4, s_2) (s_2, s_1)

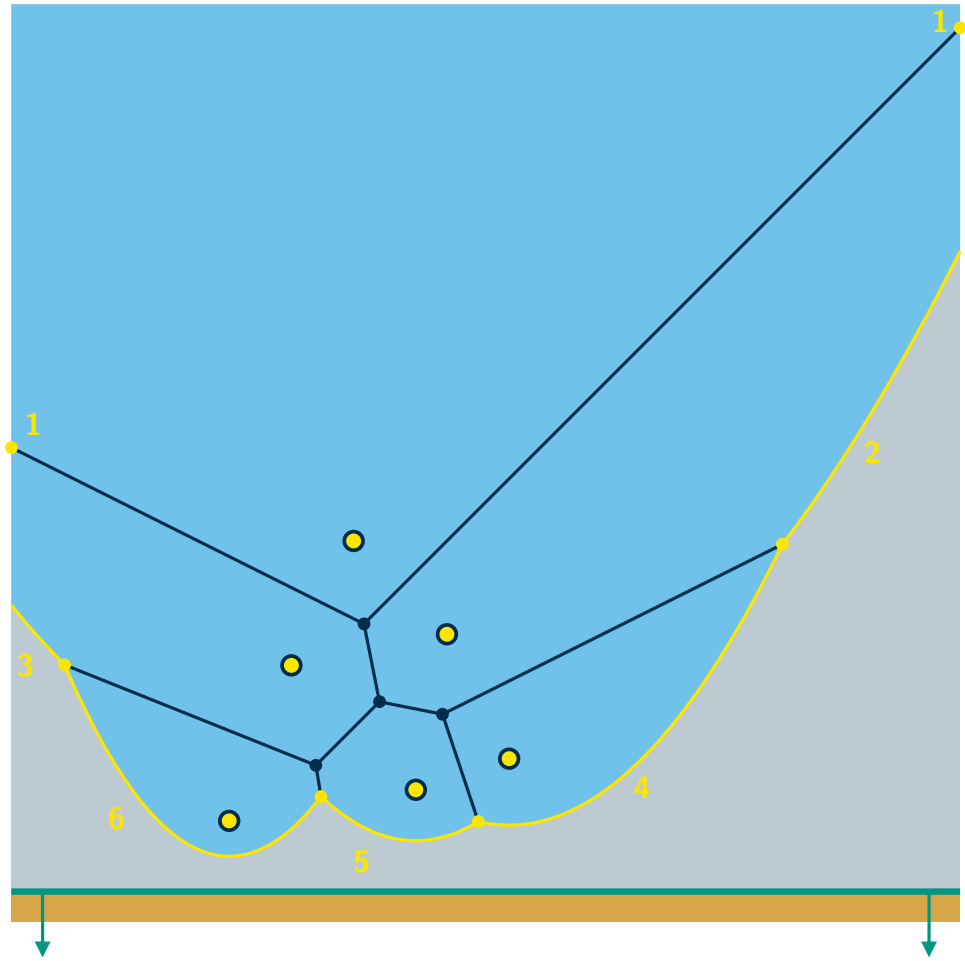
Site Event (Site s)

- find s_x in the beach line
- insert two new intersections
- handle false alarm for intersected parabola piece
- test new intersections for circle events
- new pair of half edges for the intersections

Circle Event (For Intersections $(s_i, s_j), (s_j, s_k)$)

- remove (s_i, s_j) and (s_j, s_k)
- insert (s_i, s_k)
- test (s_i, s_k) with BL-neighbors for circle events
- create new vertex v
- attach half edge of (s_i, s_j) and (s_j, s_k) to v
- new pair of half edges: one at v, one at (s_i, s_k)

Overview



beach line: (s_1, s_3) (s_3, s_6) (s_6, s_5) (s_5, s_4) (s_4, s_2) (s_2, s_1)

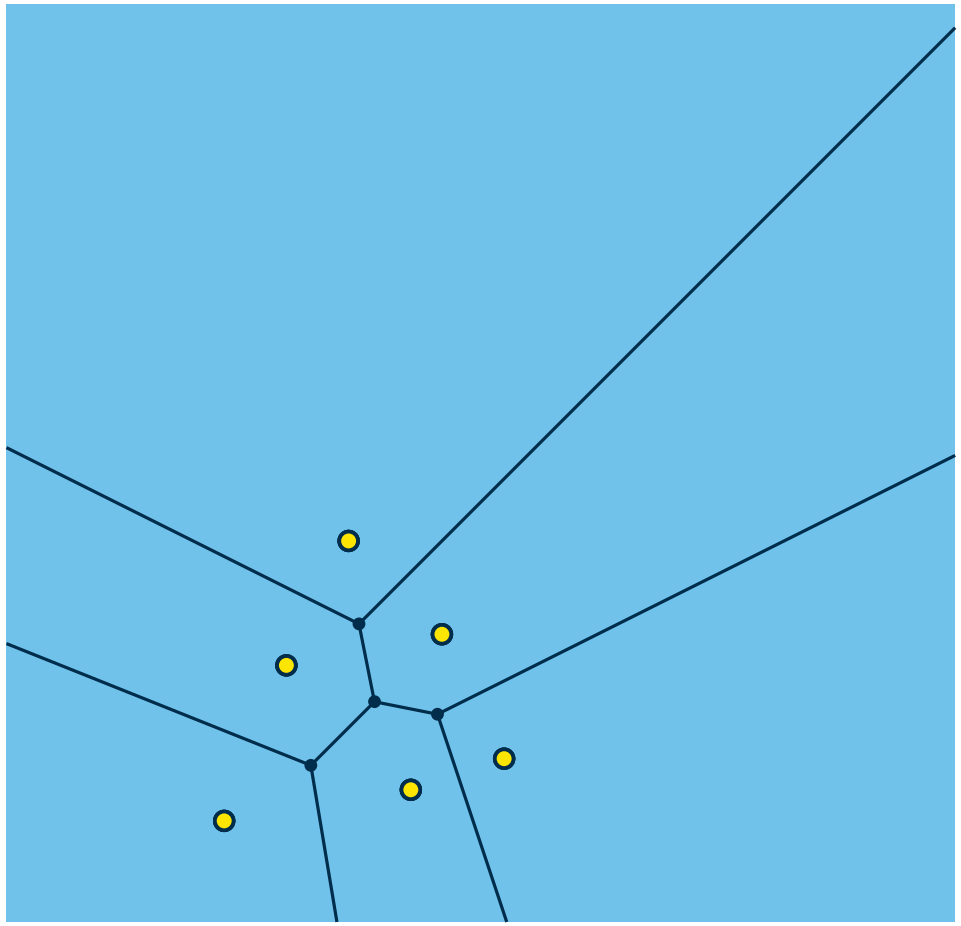
Site Event (Site s)

- find s_x in the beach line
- insert two new intersections
- handle false alarm for intersected parabola piece
- test new intersections for circle events
- new pair of half edges for the intersections

Circle Event (For Intersections $(s_i, s_j), (s_j, s_k)$)

- remove (s_i, s_j) and (s_j, s_k)
- insert (s_i, s_k)
- test (s_i, s_k) with BL-neighbors for circle events
- create new vertex v
- attach half edge of (s_i, s_j) and (s_j, s_k) to v
- new pair of half edges: one at v, one at (s_i, s_k)

Overview



beach line:

Site Event (Site s)

- find s_x in the beach line
- insert two new intersections
- handle false alarm for intersected parabola piece
- test new intersections for circle events
- new pair of half edges for the intersections

Circle Event (For Intersections $(s_i, s_j), (s_j, s_k)$)

- remove (s_i, s_j) and (s_j, s_k)
- insert (s_i, s_k)
- test (s_i, s_k) with BL-neighbors for circle events
- create new vertex v
- attach half edge of (s_i, s_j) and (s_j, s_k) to v
- new pair of half edges: one at v, one at (s_i, s_k)

Wrap-Up

Theorem
The Voronoi diagram of n sites can be computed in $O(n \log n)$ time using $O(n)$ space.

Wrap-Up

Theorem
The Voronoi diagram of n sites can be computed in $O(n \log n)$ time using $O(n)$ space.

What Else Is There?

- there are various variants
  - higher dimensions
  - different metrics
  - weighted sites
  - sites that are more complicated than just points (e.g., line segments)
  - Voronoi diagrams of higher order: What are the k closest sites?

Wrap-Up

Theorem
The Voronoi diagram of n sites can be computed in $O(n \log n)$ time using $O(n)$ space.

What Else Is There?

- there are various variants
  - higher dimensions
  - different metrics
  - weighted sites
  - sites that are more complicated than just points (e.g., line segments)
  - Voronoi diagrams of higher order: What are the k closest sites?
- visualization: `http://www.raymondhill.net/voronoi/rhill-voronoi.html`