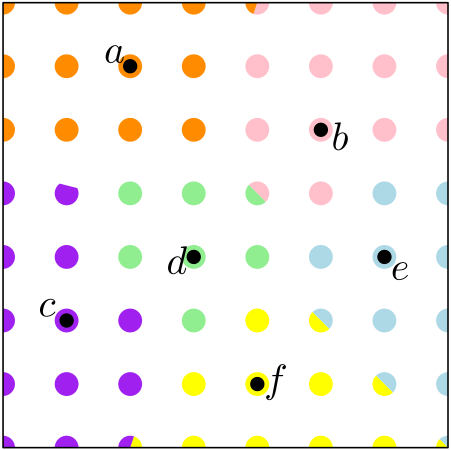
Computing Planar Voronoi Diagrams in Double Precision:
A Further Example of Degree-driven Algorithm Design

David L. Millman   Jack Snoeyink

University of North Carolina at Chapel Hill

June 16, 2010

The Problem

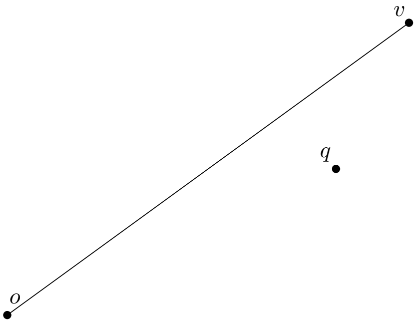


Given n sites on a pixel grid,
what is the closest site to each pixel?

How much precision is need to determine this?

Analyzing Precision[LPT99]

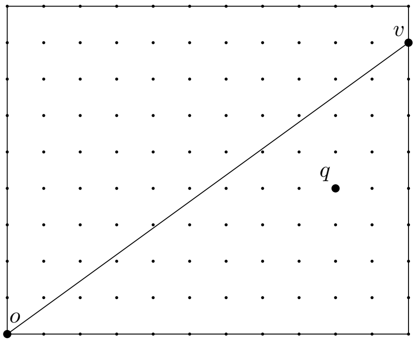
E.g., Precision of the orientation test:



`orientation`(o, v, q)

Analyzing Precision[LPT99]

E.g., Precision of the orientation test:

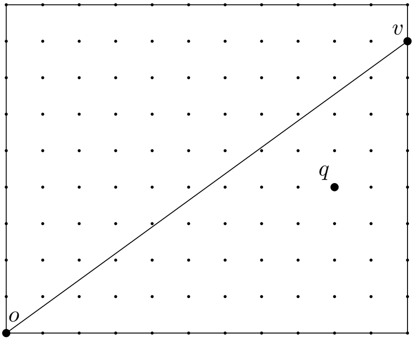


$\mathbb{U} = \{1, \ldots, U\}^2$
$o, v, q \in \mathbb{U}$

$\texttt{orientation}(o, v, q)$

Analyzing Precision[LPT99]

E.g., Precision of the orientation test:

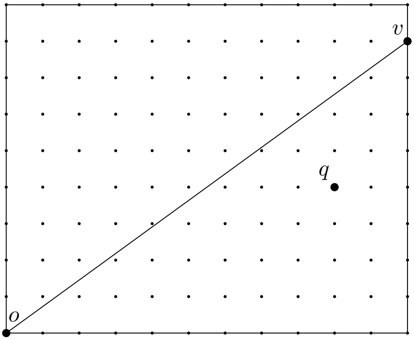


$\mathbb{U} = \{1, \ldots, U\}^2$
$o, v, q \in \mathbb{U}$
$o = (o_x, o_y)$
$v = (v_x, v_y)$
$q = (q_x, q_y)$

$$\texttt{orientation}(o, v, q) = \begin{vmatrix} 1 & o_x & o_y \\ 1 & v_x & v_y \\ 1 & q_x & q_y \end{vmatrix}$$

Analyzing Precision[LPT99]

E.g., Precision of the orientation test:

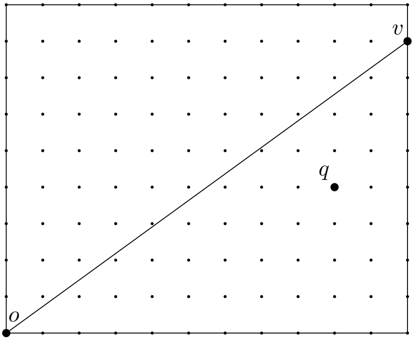


$\mathbb{U} = \{1, \ldots, U\}^2$
$o, v, q \in \mathbb{U}$
$o = (o_x, o_y)$
$v = (v_x, v_y)$
$q = (q_x, q_y)$

$$\begin{aligned}\texttt{orientation}(o, v, q) &= \begin{vmatrix} 1 & o_x & o_y \\ 1 & v_x & v_y \\ 1 & q_x & q_y \end{vmatrix} \\ &= v_x q_y - v_x o_y - o_x q_y + o_x o_y \\ &\quad - v_y q_x + v_y o_x + q_y q_x - q_y o_x\end{aligned}$$

Analyzing Precision[LPT99]

E.g., Precision of the orientation test:



$\mathbb{U} = \{1, \ldots, U\}^2$
$o, v, q \in \mathbb{U}$
$o = (o_x, o_y)$
$v = (v_x, v_y)$
$q = (q_x, q_y)$

$$\texttt{orientation}(o, v, q) = v_x q_y - v_x o_y - o_x q_y + o_x o_y - v_y q_x + v_y o_x + q_y q_x - q_y o_x$$

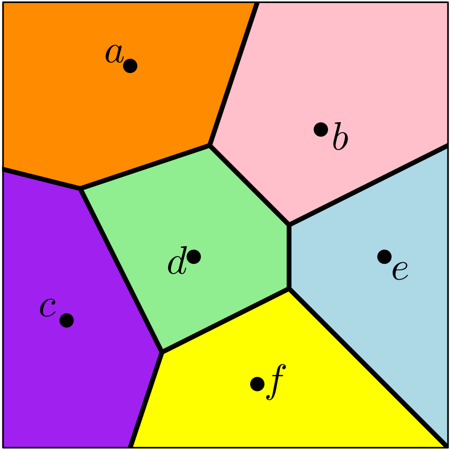
degree [2]

Other precision/robust approaches

Techniques for implementing geometric algorithms with finite precision computer arithmetic:

- Rely on machine precision (+epsilon)
- Exact Geometric Computation Y97
- Arithmetic Filters FW93,DP99
- Adaptive Predicates P92,S97
- Topological Consistency SI92
- Degree-driven algorithm design LPT99

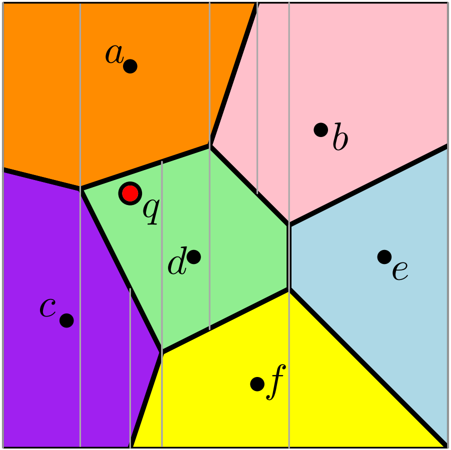
Precision of Voronoi Diagram/Trapeziod Graph



Voronoi diagram

- region
- edge
- vertex

Precision of Voronoi Diagram/Trapeziod Graph



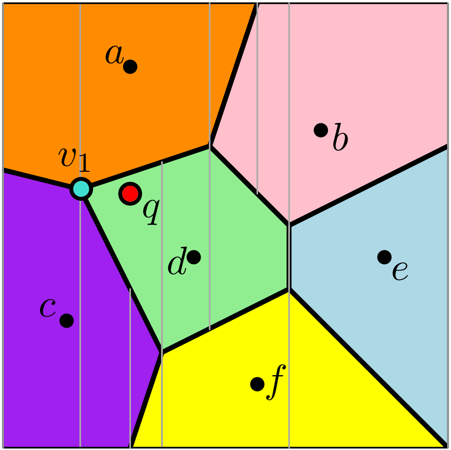
Voronoi diagram

- region
- edge
- vertex

Trapezoid graph for proximity queries

- x-`node()`
- y-`node()`

Precision of Voronoi Diagram/Trapeziod Graph



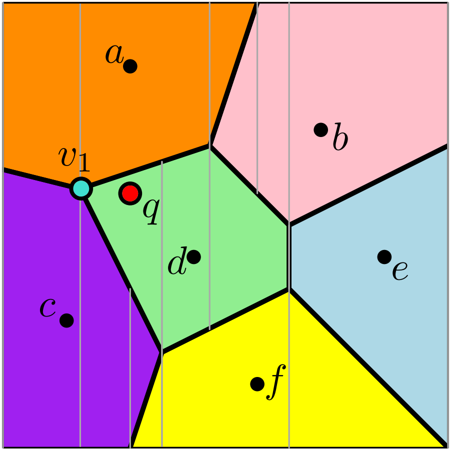
Voronoi diagram

- region
- edge
- vertex

Trapezoid graph for proximity queries

- *x*-`node()`
- *y*-`node()`

Precision of Voronoi Diagram/Trapeziod Graph



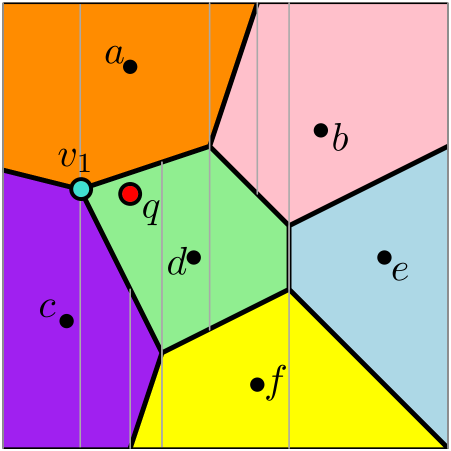
Voronoi diagram

- region
- edge
- vertex – rational: deg [3]/[2]

Trapezoid graph for proximity queries

- *x*-`node()`
- *y*-`node()`

Precision of Voronoi Diagram/Trapeziod Graph



Voronoi diagram

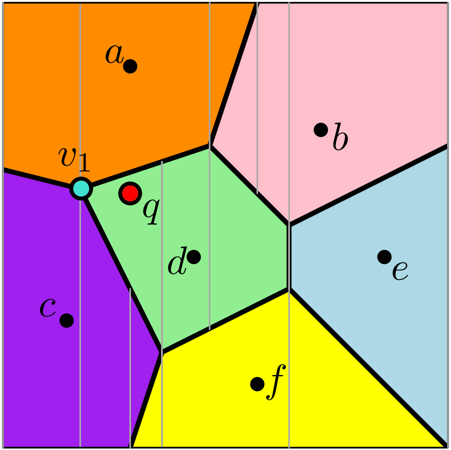
- region
- edge
- vertex – rational: deg [3]/[2]

Trapezoid graph for proximity queries

- x-node()
- y-node()

Precision of x-node test:
deg [3]/[2] $\gtrless$ deg [1]

Precision of Voronoi Diagram/Trapeziod Graph



Voronoi diagram

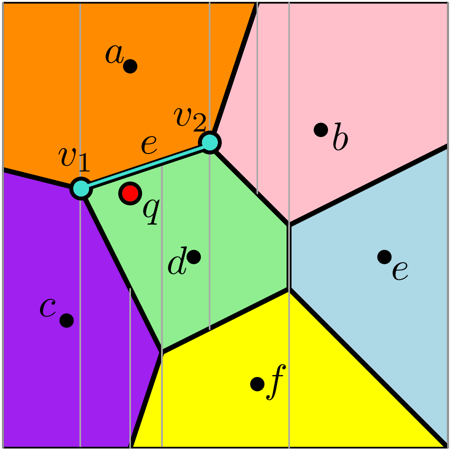
- region
- edge
- vertex – rational: deg [3]/[2]

Trapezoid graph for proximity queries

- x-node() – deg [3]
- y-node()

Precision of x-node test:
deg [3]/[2] $\gtrless$ deg [1]

Precision of Voronoi Diagram/Trapeziod Graph

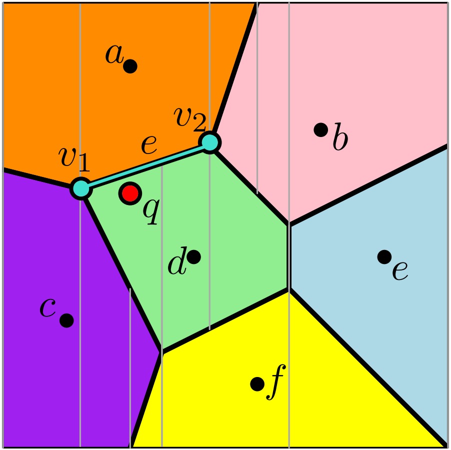


Voronoi diagram

- region
- edge
- vertex – rational: deg [3]/[2]

Trapezoid graph for proximity queries

- *x*-node() – deg [3]
- *y*-node()



Voronoi diagram

- region
- edge
- vertex – rational: deg [3]/[2]

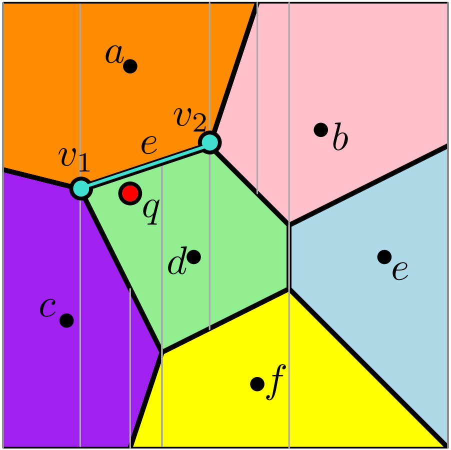
Trapezoid graph for proximity queries

- *x*-`node()` – deg [3]
- *y*-`node()`

Precision of *y*-node test:

$$\texttt{orientation()} = \begin{vmatrix} [2] & [3] & [3] \\ [2] & [3] & [3] \\ [0] & [1] & [1] \end{vmatrix}$$

Precision of Voronoi Diagram/Trapeziod Graph



Voronoi diagram

- region
- edge
- vertex – rational: deg [3]/[2]

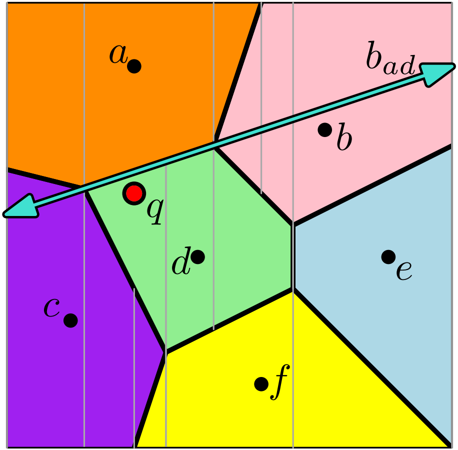
Trapezoid graph for proximity queries

- x-`node()` – deg [3]
- y-`node()` – deg [6]

Precision of y-node test:

$$\texttt{orientation()} = \begin{vmatrix} [2] & [3] & [3] \\ [2] & [3] & [3] \\ [0] & [1] & [1] \end{vmatrix}$$

Precision of Voronoi Diagram/Trapeziod Graph



Voronoi diagram

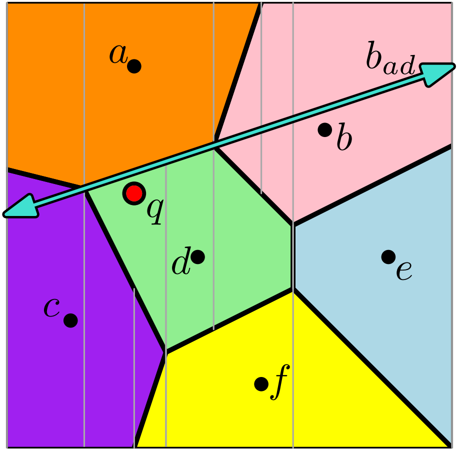
- region
- edge
- vertex – rational: deg [3]/[2]

Trapezoid graph for proximity queries

- x-`node()` – deg [3]
- y-`node()` – deg [6]

Precision of y-node test:

Precision of Voronoi Diagram/Trapeziod Graph



Voronoi diagram

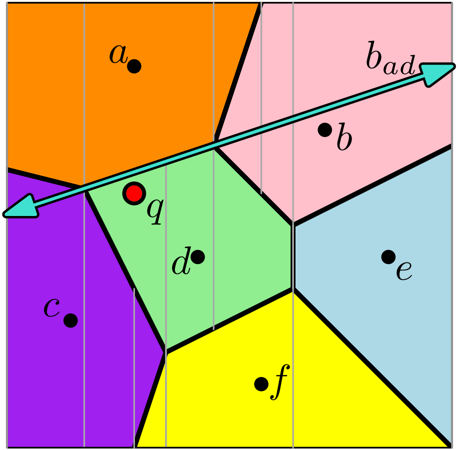
- region
- edge
- vertex – rational: deg [3]/[2]

Trapezoid graph for proximity queries

- x-`node()` – deg [3]
- y-`node()` – deg [6]

Precision of y-node test:
deg [2] $\gtrless$ deg [2]

Precision of Voronoi Diagram/Trapeziod Graph



Voronoi diagram

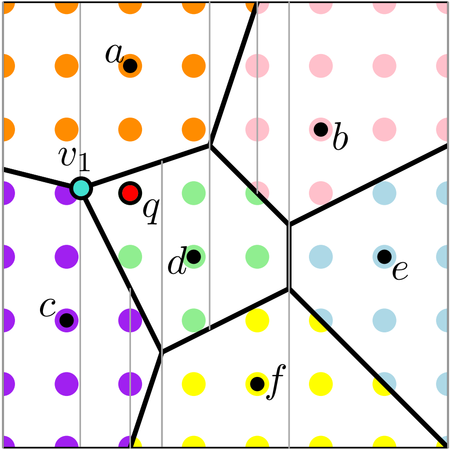
- region
- edge
- vertex – rational: deg [3]/[2]

Trapezoid graph for proximity queries

- x-node() – deg [3]
- y-node() – deg [2]

Precision of y-node test:
deg [2] $\gtrless$ deg [2]

Precision of Voronoi Diagram/Trapeziod Graph



Voronoi diagram

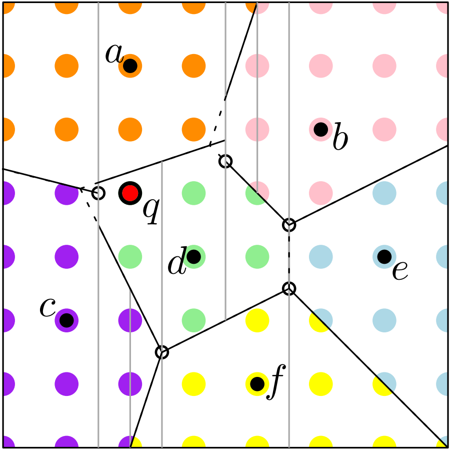
- region
- edge
- vertex – rational: deg [3]/[2]

Trapezoid graph for proximity queries

- x-node() – deg [3]
- y-node() – deg [2]

Precision of x-node test:

Precision of Voronoi Diagram/Trapeziod Graph



Voronoi diagram

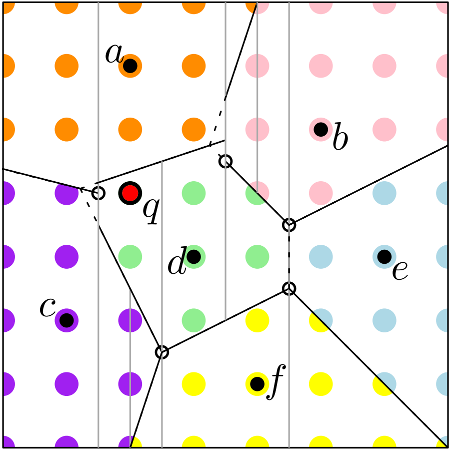
- region
- edge
- vertex – rational: deg [3]/[2]

Trapezoid graph for proximity queries

- x-node() – deg [3]
- y-node() – deg [2]

Precision of x-node test:
deg [1] $\gtrless$ deg [1]

Precision of Voronoi Diagram/Trapeziod Graph



Voronoi diagram

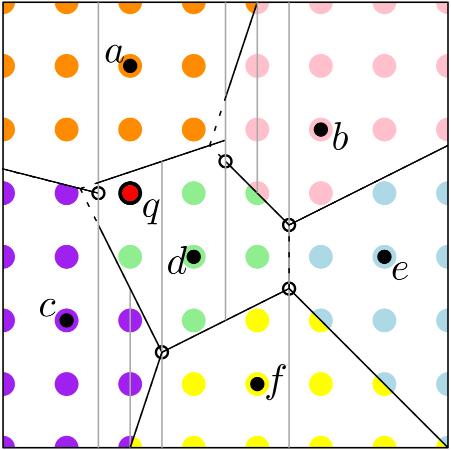
- region
- edge
- vertex – rational: deg [3]/[2]

Trapezoid graph for proximity queries

- *x*-`node()` – deg [1]
- *y*-`node()` – deg [2]

Precision of *x*-node test:
deg [1] $\gtrless$ deg [1]

Precision of Voronoi Diagram/Trapeziod Graph



Voronoi diagram

- region
- edge
- vertex – rational: deg [3]/[2]

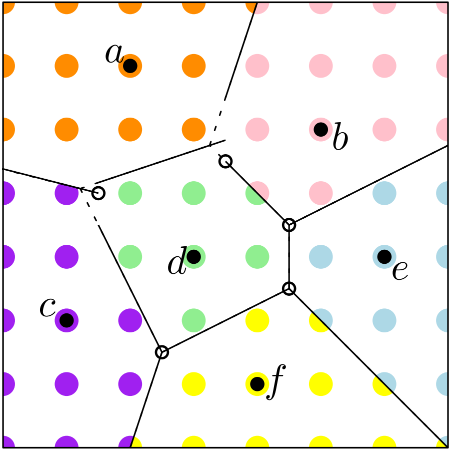
Trapezoid graph for proximity queries

- *x*-node() – deg [1]
- *y*-node() – deg [2]

This is a degree [2] trapezoid graph.

Implicit Voronoi diagram [LPT99]

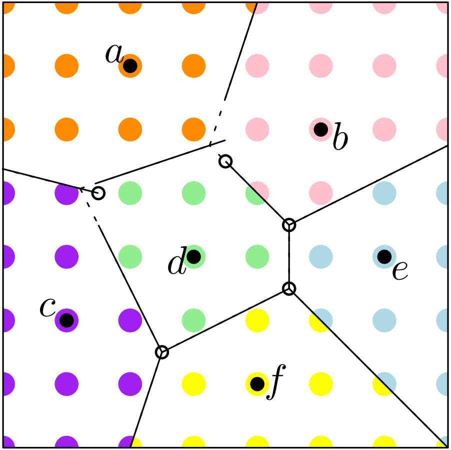
The implicit Voronoi diagram is a rounded Voronoi diagram.



- vertices – degree [1]
  Voronoi vertices
  snapped to half grid points.
- edges
  pointers to the two sites
  that define the bisector,
  which the edge is a subset of.

Implicit Voronoi diagram [LPT99]

The implicit Voronoi diagram is a rounded Voronoi diagram.

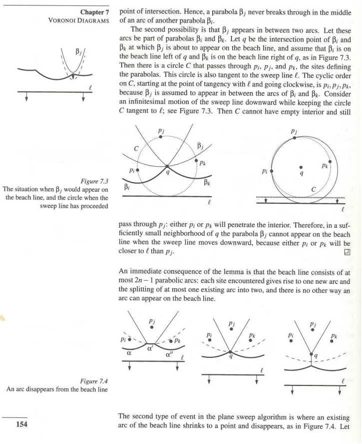


- vertices – degree [1]
  Voronoi vertices
  snapped to half grid points.
- edges
  pointers to the two sites
  that define the bisector,
  which the edge is a subset of.

How do we build
the Implicit Voronoi diagram
with low precision?

Precision of Constructing the Voronoi Diagram

Three well known ways to build the Voronoi diagram.



Chapter 7
VORONOI DIAGRAMS

point of intersection. Hence, a parabola β_j never breaks through in the middle of an arc of another parabola β_i.

The second possibility is that β_j appears in between two arcs. Let these arcs be part of parabolas β_i and β_k. Let q be the intersection point of β_i and β_k at which β_j is about to appear on the beach line, and assume that β_i is on the beach line left of q and β_k is on the beach line right of q, as in Figure 7.3. Then there is a circle C that passes through p_i, p_j, and p_k, the sites defining the parabolas. This circle is also tangent to the sweep line ℓ. The cyclic order on C, starting at the point of tangency with ℓ and going clockwise, is p_i, p_j, p_k, because β_j is assumed to appear in between the arcs of β_i and β_k. Consider an infinitesimal motion of the sweep line downward while keeping the circle C tangent to ℓ; see Figure 7.3. Then C cannot have empty interior and still pass through p_j: either p_i or p_k will penetrate the interior. Therefore, in a sufficiently small neighborhood of q the parabola β_j cannot appear on the beach line when the sweep line moves downward, because either p_i or p_k will be closer to ℓ than p_j. □

Figure 7.3
The situation when β_j would appear on the beach line, and the circle when the sweep line has proceeded

An immediate consequence of the lemma is that the beach line consists of at most $2n-1$ parabolic arcs: each site encountered gives rise to one new arc and the splitting of at most one existing arc into two, and there is no other way an arc can appear on the beach line.

Figure 7.4
An arc disappears from the beach line

The second type of event in the plane sweep algorithm is where an existing arc of the beach line shrinks to a point and disappears, as in Figure 7.4. Let

154

Sweepline[F87]
– degree [6]

Divide and Conquer[GS86]
– degree [4]

Tracing[SI92]
– degree [4]

Precision of Constructing the Voronoi Diagram

Three well known ways to build the Voronoi diagram.

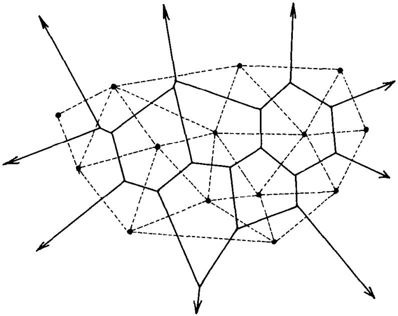


Fig. 15. The Voronoi diagram (solid) and the Delaunay diagram (dashed).

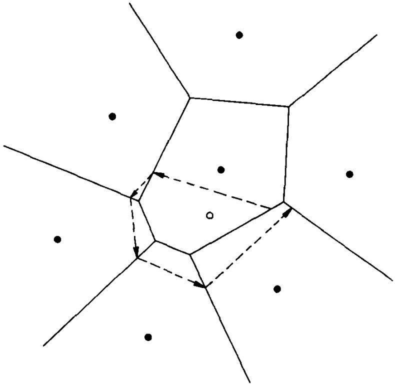
Sweepline[F87]
– degree [6]

Divide and Conquer[GS86]
– degree [4]

Tracing[SI92]
– degree [4]

Precision of Constructing the Voronoi Diagram

Three well known ways to build the Voronoi diagram.



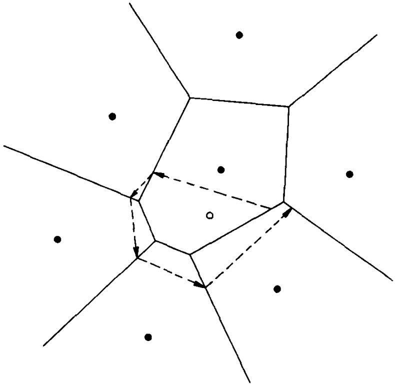
Sweepline[F87]
– degree [6]

Divide and Conquer[GS86]
– degree [4]

Tracing[SI92]
– degree [4]

Precision of Constructing the Voronoi Diagram

Three well known ways to build the Voronoi diagram.

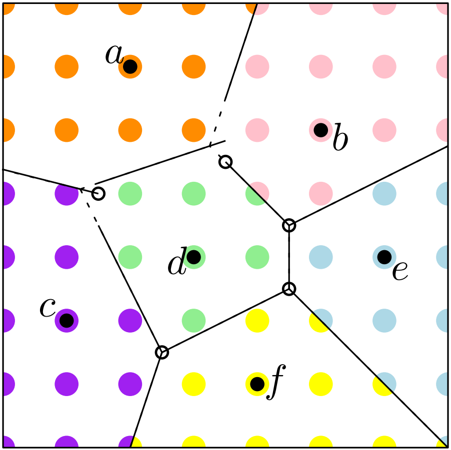


Sweepline[F87]
– degree [6]

Divide and Conquer[GS86]
– degree [4]

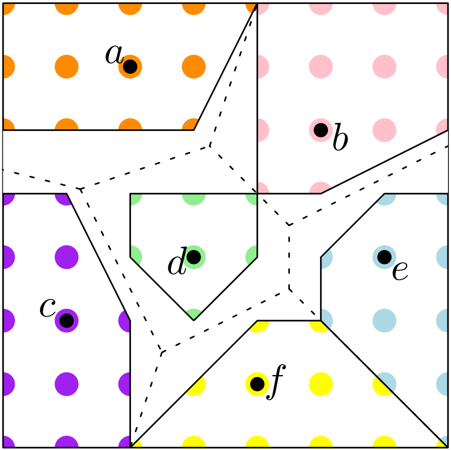
Tracing[SI92]
– degree [4]

How do we build a degree [2] trapezoid graph for proximity queries when we can't even construct a Voronoi vertex?



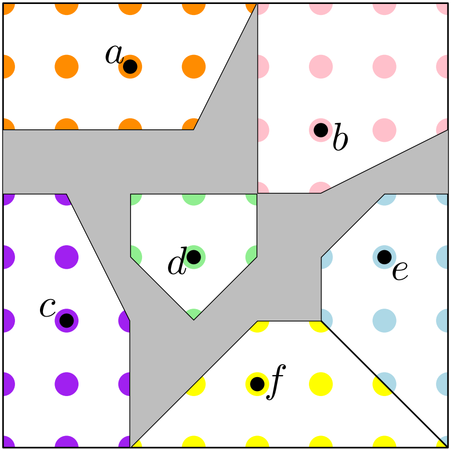
Implicit Voronoi diagram
is disconnected.

Voronoi Polygon Set



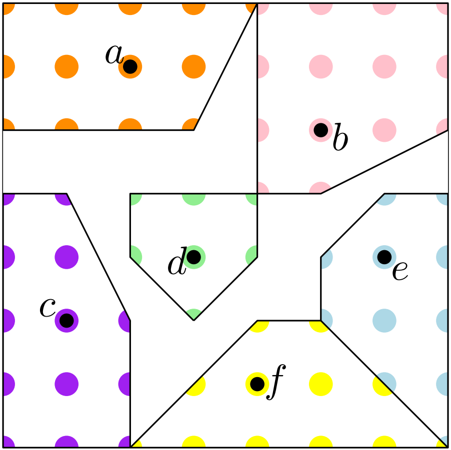
- *Voronoi polygon* is the convex hull of the grid points in a Voronoi cell.

Voronoi Polygon Set



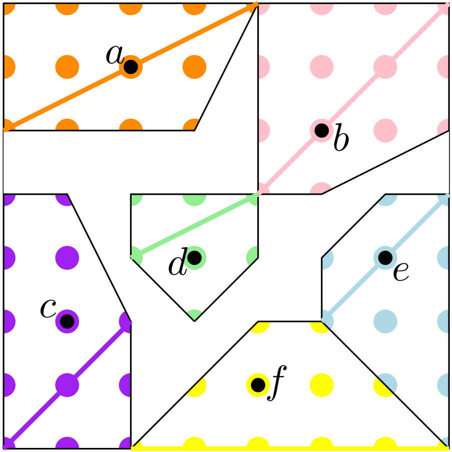
- *Voronoi polygon* is the convex hull of the grid points in a Voronoi cell.
- Gaps

Voronoi Polygon Set



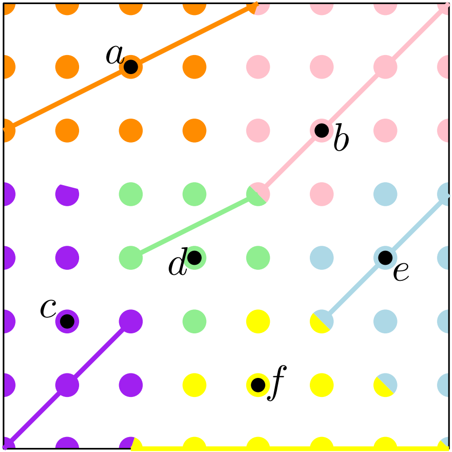
- *Voronoi polygon* is the convex hull of the grid points in a Voronoi cell.
- Gaps
- Total size $\Theta(n \log U)$.

Proxy Segments



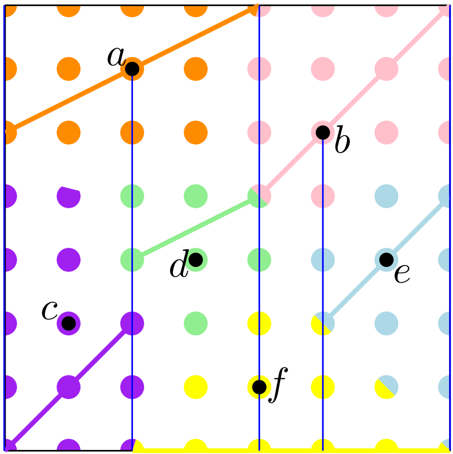
- *Proxy segment* -
  represent Voronoi polygons.

Proxy Segments



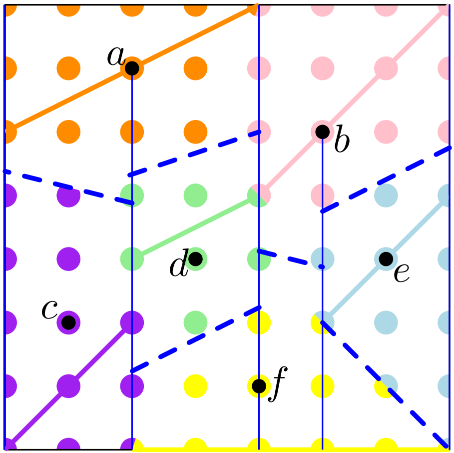
- *Proxy segment* -
  represent Voronoi polygons.

Proxy Segments



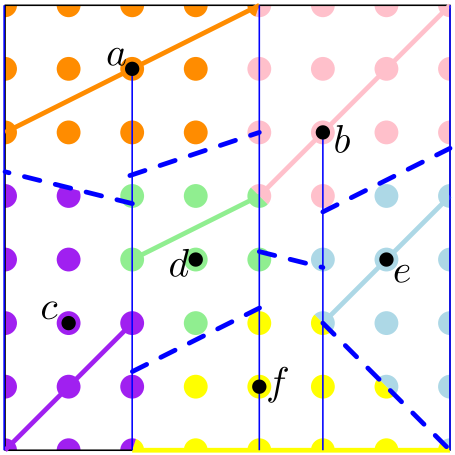
- *Proxy segment* -
  represent Voronoi polygons.
- *Proxy trapezoidation* -
  trapezoidation of the proxies.

Proxy Segments



- *Proxy segment* -
  represent Voronoi polygons.
- *Proxy trapezoidation* -
  trapezoidation of the proxies.
- *Voronoi Trapezoidation* -
  split the trapezoids
  of the Proxy trapezoidation
  with bisectors.

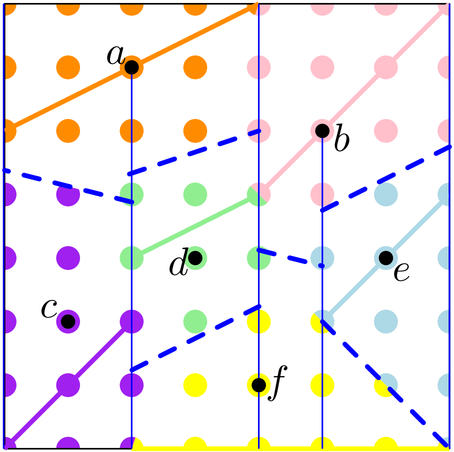
Proxy Segments



- *Proxy segment* -
  represent Voronoi polygons.
- *Proxy trapezoidation* -
  trapezoidation of the proxies.
- *Voronoi Trapezoidation* -
  split the trapezoids
  of the Proxy trapezoidation
  with bisectors.

Proxy Trapezoidation
is a degree [2] trapezoid graph.

Proxy Segments



- *Proxy segment* -
  represent Voronoi polygons.
- *Proxy trapezoidation* -
  trapezoidation of the proxies.
- *Voronoi Trapezoidation* -
  split the trapezoids
  of the Proxy trapezoidation
  with bisectors.

Proxy Trapezoidation
is a degree [2] trapezoid graph.

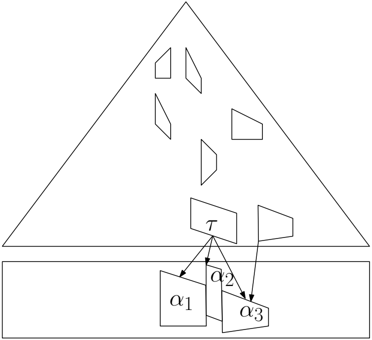
How do we build a Proxy trapezoidation with degree [2]?

Construction Sketch

Build the Proxy trapezoidation with a randomized incremental construction (RIC).

Each step creates and deletes trapezoids and introduces and modifies proxy segments.

Maintain a history of the trapezoids created and deleted in the RIC in a history DAG.

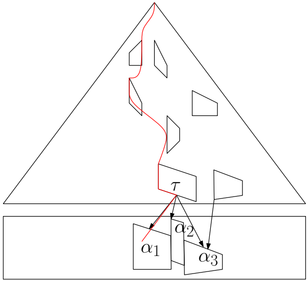


Suppose trap τ is deleted and replaced by traps $\alpha_1, \alpha_2, \alpha_3$.
The history DAG stores τ with pointers to α_i.

Construction Sketch

Insert site s_i:

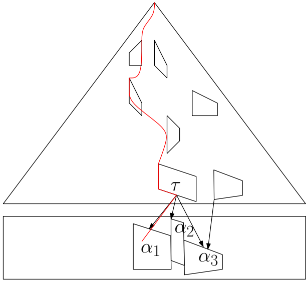
1. Identify proxy segment for s_i
2. Add new proxy to the Proxy Trapezoidation
3. Update old proxy segments
4. Update history.



Construction Sketch

Insert site s_i:

1. Identify proxy segment for s_i
2. Add new proxy to the Proxy Trapezoidation
3. Update old proxy segments
4. Update history.



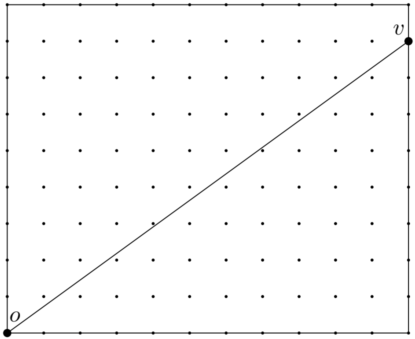
Analysis:
Use Mulmuley's general framework of stoppers and triggers (or definers and killers from the Dutch book) to show:

1. Expected size is $O(n)$
2. Expected time is $O(n \log n \log U)$

HullVertices[KS99]

Given a new site s_i and a trapezoid τ with s_{old} as closest neighbor.

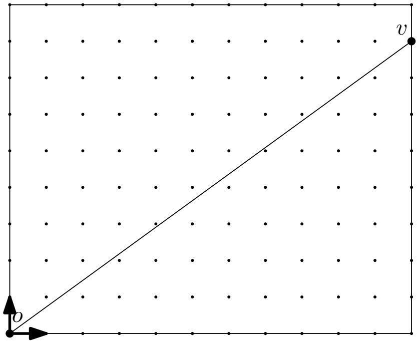
Compute the convex hull of the grid points
in the intersection of the Voronoi polygon for s_i and τ.



A geometric interpretation
of Euclid's GCD Algorithm
allows us, in degree [2],
to find increasingly better
rational approximations
to the slope of a line.

Given a new site s_i and a trapezoid τ with s_{old} as closest neighbor.

Compute the convex hull of the grid points
in the intersection of the Voronoi polygon for s_i and τ.

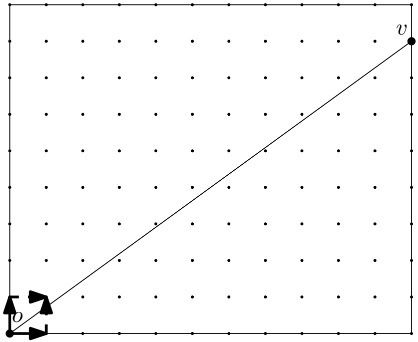


A geometric interpretation
of Euclid's GCD Algorithm
allows us, in degree [2],
to find increasingly better
rational approximations
to the slope of a line.

HullVertices[KS99]

Given a new site s_i and a trapezoid τ with s_{old} as closest neighbor.

Compute the convex hull of the grid points
in the intersection of the Voronoi polygon for s_i and τ.

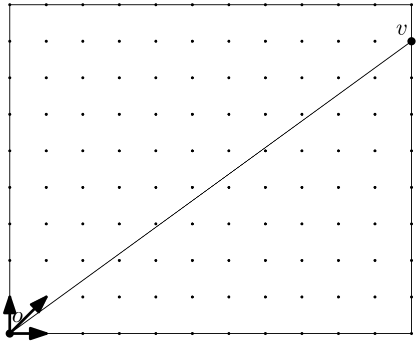


A geometric interpretation
of Euclid's GCD Algorithm
allows us, in degree [2],
to find increasingly better
rational approximations
to the slope of a line.

HullVertices[KS99]

Given a new site s_i and a trapezoid τ with s_{old} as closest neighbor.

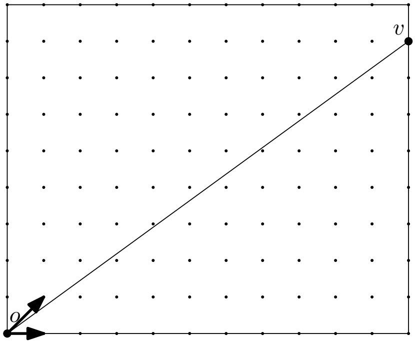
Compute the convex hull of the grid points
in the intersection of the Voronoi polygon for s_i and τ.



A geometric interpretation
of Euclid's GCD Algorithm
allows us, in degree [2],
to find increasingly better
rational approximations
to the slope of a line.

Given a new site s_i and a trapezoid τ with s_{old} as closest neighbor.

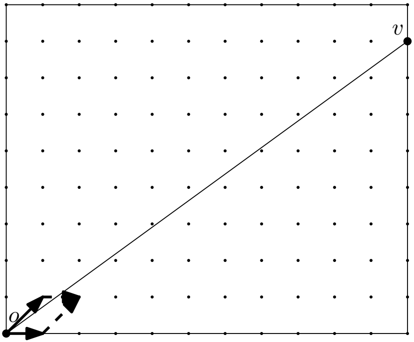
Compute the convex hull of the grid points
in the intersection of the Voronoi polygon for s_i and τ.



A geometric interpretation
of Euclid's GCD Algorithm
allows us, in degree [2],
to find increasingly better
rational approximations
to the slope of a line.

Given a new site s_i and a trapezoid τ with s_{old} as closest neighbor.

Compute the convex hull of the grid points
in the intersection of the Voronoi polygon for s_i and τ.

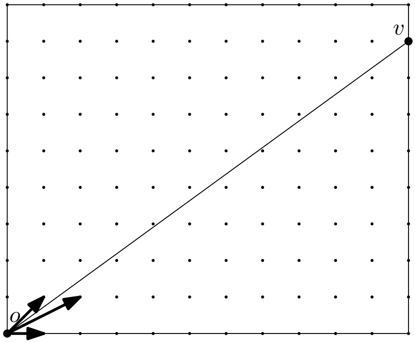


A geometric interpretation
of Euclid's GCD Algorithm
allows us, in degree [2],
to find increasingly better
rational approximations
to the slope of a line.

HullVertices[KS99]

Given a new site s_i and a trapezoid τ with s_{old} as closest neighbor.

Compute the convex hull of the grid points
in the intersection of the Voronoi polygon for s_i and τ.

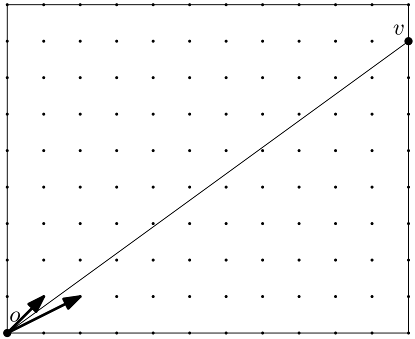


A geometric interpretation
of Euclid's GCD Algorithm
allows us, in degree [2],
to find increasingly better
rational approximations
to the slope of a line.

HullVertices[KS99]

Given a new site s_i and a trapezoid τ with s_{old} as closest neighbor.

Compute the convex hull of the grid points
in the intersection of the Voronoi polygon for s_i and τ.

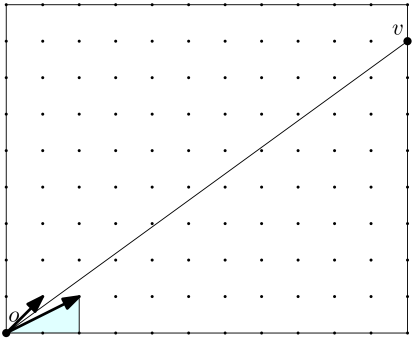


A geometric interpretation
of Euclid's GCD Algorithm
allows us, in degree [2],
to find increasingly better
rational approximations
to the slope of a line.

HullVertices[KS99]

Given a new site s_i and a trapezoid τ with s_{old} as closest neighbor.

Compute the convex hull of the grid points
in the intersection of the Voronoi polygon for s_i and τ.

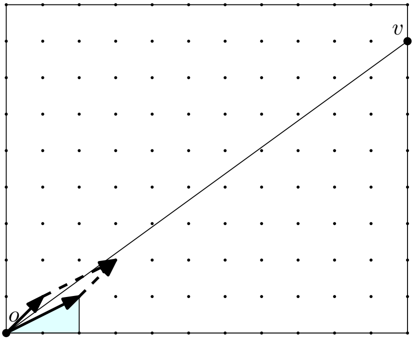


A geometric interpretation
of Euclid's GCD Algorithm
allows us, in degree [2],
to find increasingly better
rational approximations
to the slope of a line.

HullVertices[KS99]

Given a new site s_i and a trapezoid τ with s_{old} as closest neighbor.

Compute the convex hull of the grid points
in the intersection of the Voronoi polygon for s_i and τ.

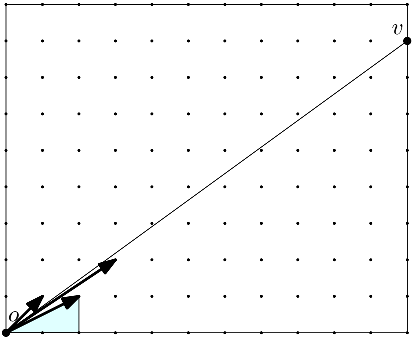


A geometric interpretation
of Euclid's GCD Algorithm
allows us, in degree [2],
to find increasingly better
rational approximations
to the slope of a line.

HullVertices[KS99]

Given a new site s_i and a trapezoid τ with s_{old} as closest neighbor.

Compute the convex hull of the grid points
in the intersection of the Voronoi polygon for s_i and τ.

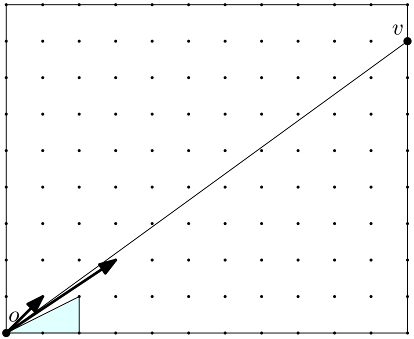


A geometric interpretation
of Euclid's GCD Algorithm
allows us, in degree [2],
to find increasingly better
rational approximations
to the slope of a line.

HullVertices[KS99]

Given a new site s_i and a trapezoid τ with s_{old} as closest neighbor.

Compute the convex hull of the grid points
in the intersection of the Voronoi polygon for s_i and τ.

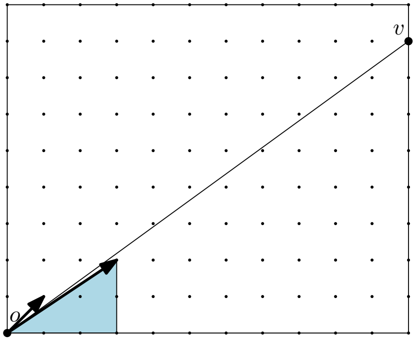


A geometric interpretation
of Euclid's GCD Algorithm
allows us, in degree [2],
to find increasingly better
rational approximations
to the slope of a line.

HullVertices[KS99]

Given a new site s_i and a trapezoid τ with s_{old} as closest neighbor.

Compute the convex hull of the grid points
in the intersection of the Voronoi polygon for s_i and τ.

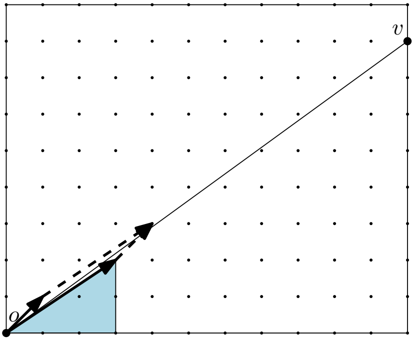


A geometric interpretation
of Euclid's GCD Algorithm
allows us, in degree [2],
to find increasingly better
rational approximations
to the slope of a line.

HullVertices[KS99]

Given a new site s_i and a trapezoid τ with s_{old} as closest neighbor.

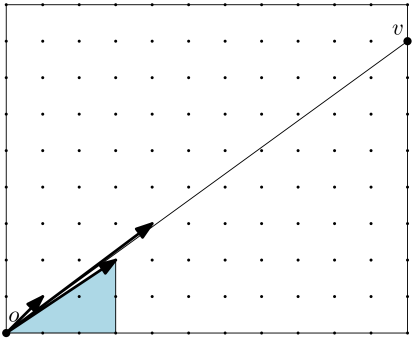
Compute the convex hull of the grid points
in the intersection of the Voronoi polygon for s_i and τ.



A geometric interpretation
of Euclid's GCD Algorithm
allows us, in degree [2],
to find increasingly better
rational approximations
to the slope of a line.

Given a new site s_i and a trapezoid τ with s_{old} as closest neighbor.

Compute the convex hull of the grid points
in the intersection of the Voronoi polygon for s_i and τ.

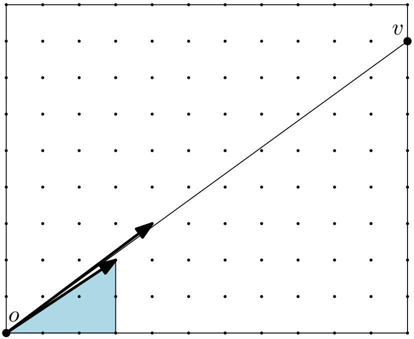


A geometric interpretation
of Euclid's GCD Algorithm
allows us, in degree [2],
to find increasingly better
rational approximations
to the slope of a line.

HullVertices[KS99]

Given a new site s_i and a trapezoid τ with s_{old} as closest neighbor.

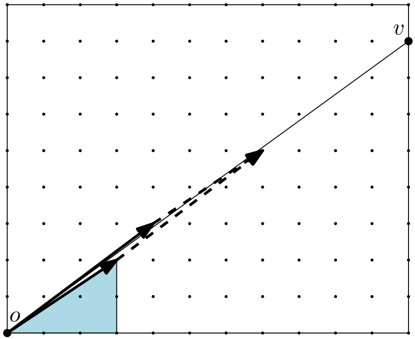
Compute the convex hull of the grid points
in the intersection of the Voronoi polygon for s_i and τ.



A geometric interpretation
of Euclid's GCD Algorithm
allows us, in degree [2],
to find increasingly better
rational approximations
to the slope of a line.

Given a new site s_i and a trapezoid τ with s_{old} as closest neighbor.

Compute the convex hull of the grid points
in the intersection of the Voronoi polygon for s_i and τ.

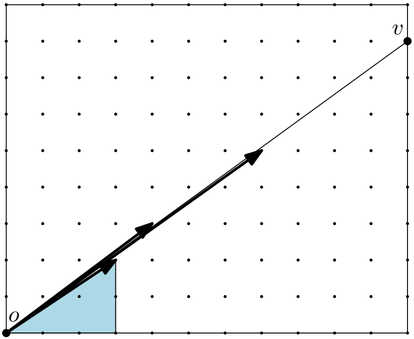


A geometric interpretation
of Euclid's GCD Algorithm
allows us, in degree [2],
to find increasingly better
rational approximations
to the slope of a line.

HullVertices[KS99]

Given a new site s_i and a trapezoid τ with s_{old} as closest neighbor.

Compute the convex hull of the grid points
in the intersection of the Voronoi polygon for s_i and τ.

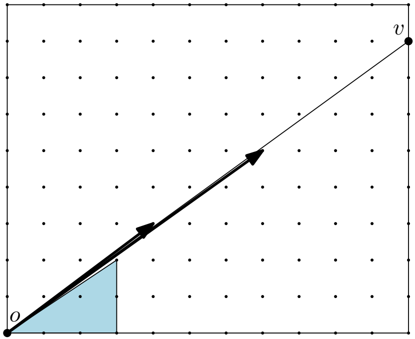


A geometric interpretation of Euclid's GCD Algorithm allows us, in degree [2], to find increasingly better rational approximations to the slope of a line.

HullVertices[KS99]

Given a new site s_i and a trapezoid τ with s_{old} as closest neighbor.

Compute the convex hull of the grid points
in the intersection of the Voronoi polygon for s_i and τ.

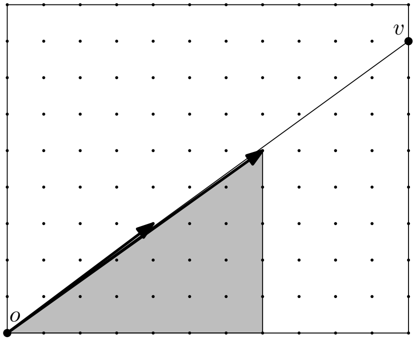


A geometric interpretation
of Euclid's GCD Algorithm
allows us, in degree [2],
to find increasingly better
rational approximations
to the slope of a line.

HullVertices[KS99]

Given a new site s_i and a trapezoid τ with s_{old} as closest neighbor.

Compute the convex hull of the grid points
in the intersection of the Voronoi polygon for s_i and τ.

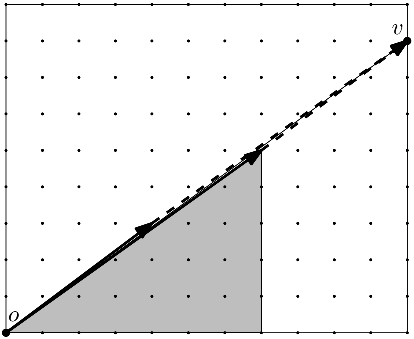


A geometric interpretation of Euclid's GCD Algorithm allows us, in degree [2], to find increasingly better rational approximations to the slope of a line.

HullVertices[KS99]

Given a new site s_i and a trapezoid τ with s_{old} as closest neighbor.

Compute the convex hull of the grid points
in the intersection of the Voronoi polygon for s_i and τ.

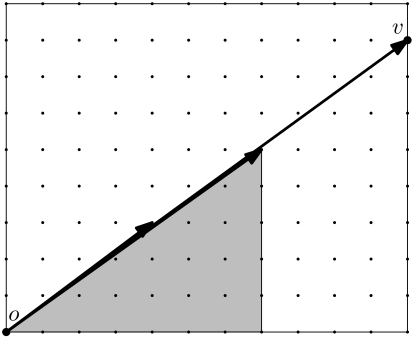


A geometric interpretation
of Euclid's GCD Algorithm
allows us, in degree [2],
to find increasingly better
rational approximations
to the slope of a line.

HullVertices[KS99]

Given a new site s_i and a trapezoid τ with s_{old} as closest neighbor.

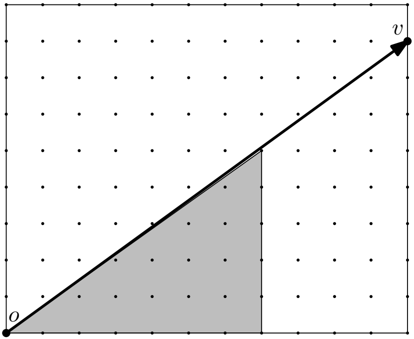
Compute the convex hull of the grid points
in the intersection of the Voronoi polygon for s_i and τ.



A geometric interpretation of Euclid's GCD Algorithm allows us, in degree [2], to find increasingly better rational approximations to the slope of a line.

Given a new site s_i and a trapezoid τ with s_{old} as closest neighbor.

Compute the convex hull of the grid points
in the intersection of the Voronoi polygon for s_i and τ.

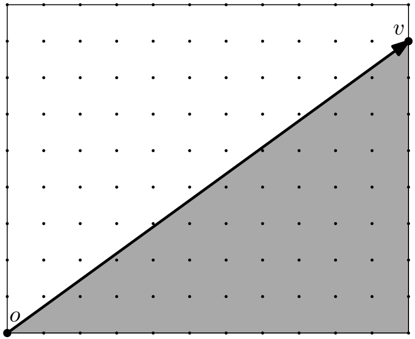


A geometric interpretation
of Euclid's GCD Algorithm
allows us, in degree [2],
to find increasingly better
rational approximations
to the slope of a line.

HullVertices[KS99]

Given a new site s_i and a trapezoid τ with s_{old} as closest neighbor.

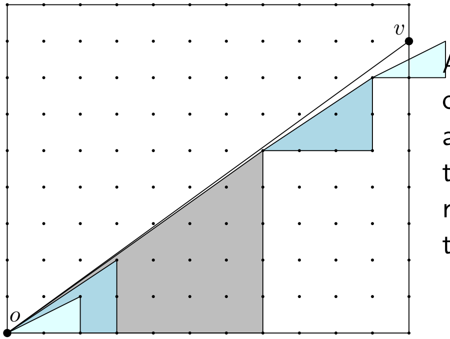
Compute the convex hull of the grid points
in the intersection of the Voronoi polygon for s_i and τ.



A geometric interpretation
of Euclid's GCD Algorithm
allows us, in degree [2],
to find increasingly better
rational approximations
to the slope of a line.

Given a new site s_i and a trapezoid τ with s_{old} as closest neighbor.

Compute the convex hull of the grid points
in the intersection of the Voronoi polygon for s_i and τ.



A geometric interpretation of Euclid's GCD Algorithm allows us, in degree [2], to find increasingly better rational approximations to the slope of a line.

Result

Given

sites $S = \{s_1, \ldots, s_n\}$ and query points on a $U \times U$.

Proxy Trapezoidation construction

- Time: $O(n \log n \log U)$ expected
- Space: $O(n)$ expected
- Precision: degree [2]

Queries on Proxy Trapezoidation

- Time: $O(\log n)$
- Precision: degree [2]

Conclusions

Building a point location data structure:

- Degree [4], $O(n \log n)$ time.
- Degree [3], $O(n(\log n + \log U))$ expected time.
- Degree [2], $O(n(\log n \log U))$ expected time.

Open problems

- Inherent loss of efficiency with restricted predicates?
- Limited precision proximity queries in higher dimension?
- What other problems have limited precision solutions?
  - Triangulations?
  - Voronoi generalizations?
  - Ray tracing?
  - Approximation algorithms?