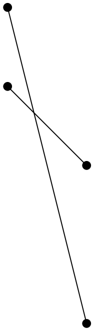
Thesis Proposal: Degree-driven Geometric Algorithm Design

David L. Millman

University of North Carolina at Chapel Hill

April 25, 2011

A Motivational Problem



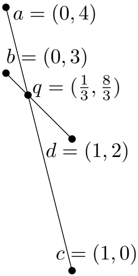
IsSegInter: Given two segments, defined by their 2D endpoints, with no three endpoints collinear, do the segments intersect?

How much precision is needed to determine this?

Thesis Statement: Degree-driven analysis supports the development of new, robust geometric algorithms, as I have demonstrated for computing Post-office query search structures, Nearest Neighbor Transforms, and Triangulations.

Input Representation

Input: Geometric configuration specified by numerical coords.



E.g. IsSegInter problem:

Numerical Coords: $(0, 4, 0, 3, 1, 0, 1, 2)$

Geometric interpretations:

$a = (a_x, a_y) = (0, 4)$,
$b = (b_x, b_y) = (0, 3)$,
$c = (c_x, c_y) = (1, 0)$,
$d = (d_x, d_y) = (1, 2)$,
$\overline{ac} = (a, c)$, and
$\overline{bd} = (b, d)$.

Illustration of an Algorithm that solves IsSegInter

`IsSegInterByConstruction`(a, c, b, d): Determine if $\overline{ac}$ and $\overline{bd}$ intersect; if so return INTERSECT, if not return NOINTERSECT

Require: no three points are collinear
if $\overleftrightarrow{ac} \parallel \overleftrightarrow{bd}$ **then**
    return NOINTERSECT
end if
Point $q = \overleftrightarrow{ac} \cap \overleftrightarrow{bd}$
Real $t_1 = (q_x - a_x)/(c_x - a_x)$
Real $t_2 = (q_x - b_x)/(d_x - b_x)$
if $t_1, t_2 \in [0, 1]$ **then**
    return INTERSECT
else
    return NOINTERSECT
end if

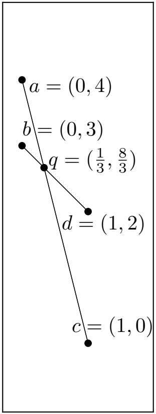
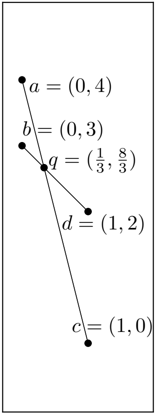


Illustration of an Algorithm that solves IsSegInter

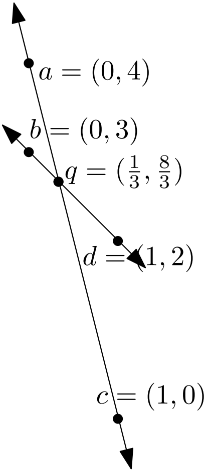
`IsSegInterByConstruction`(a, c, b, d): Determine if $\overline{ac}$ and $\overline{bd}$ intersect; if so return INTERSECT, if not return NOINTERSECT

Require: no three points are collinear

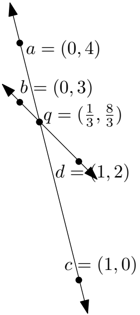
if $\overleftrightarrow{ac} \parallel \overleftrightarrow{bd}$ **then**
    return NOINTERSECT
end if
Point $q = \overleftrightarrow{ac} \cap \overleftrightarrow{bd}$
Real $t_1 = (q_x - a_x)/(c_x - a_x)$
Real $t_2 = (q_x - b_x)/(d_x - b_x)$
if $t_1, t_2 \in [0, 1]$ **then**
    return INTERSECT
else
    return NOINTERSECT
end if



Line 4: Point $q = \overleftrightarrow{ac} \cap \overleftrightarrow{bd}$



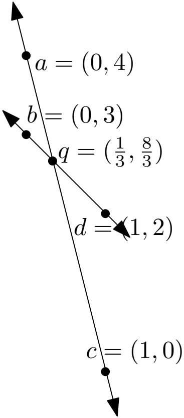
The `Intersect`(a, c, b, d) construction:



Input: single-precision coordinates of a, c, b and d defining non-parallel lines $\overleftrightarrow{ac}$ and $\overleftrightarrow{bd}$.
Construct: the intersection q of $\overleftrightarrow{ac}$ and $\overleftrightarrow{bd}$.

$$q_x = \frac{\begin{vmatrix} a_x c_y - c_x a_y & a_x - c_x \\ b_x d_y - d_x b_y & b_x - d_x \end{vmatrix}}{\begin{vmatrix} a_x - c_x & a_y - c_y \\ b_x - d_x & b_y - d_y \end{vmatrix}}, q_y = \frac{\begin{vmatrix} a_x c_y - c_x a_y & a_y - c_y \\ b_x d_y - d_x b_y & b_y - d_y \end{vmatrix}}{\begin{vmatrix} a_x - c_x & a_y - c_y \\ b_x - d_x & b_y - d_y \end{vmatrix}}.$$

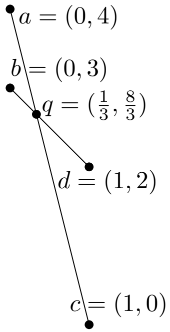
The `Intersect`(a, c, b, d) *construction*:



Input: single-precision coordinates of a, c, b and d defining non-parallel lines $\overleftrightarrow{ac}$ and $\overleftrightarrow{bd}$.
Construct: the intersection q of $\overleftrightarrow{ac}$ and $\overleftrightarrow{bd}$.

$$q_x = .\overline{3}$$
$$q_y = 2.\overline{6}.$$

The `Intersect`(a, c, b, d) construction:

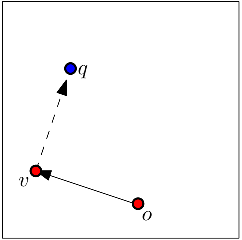


Input: single-precision coordinates of a, c, b and d defining non-parallel lines $\overleftrightarrow{ac}$ and $\overleftrightarrow{bd}$.
Construct: the intersection q of $\overleftrightarrow{ac}$ and $\overleftrightarrow{bd}$.
In Python with *numpy.float32* type:

$$
\begin{aligned}
\texttt{fl}(q_x) &\approx 0.33333334 \\
\texttt{fl}(q_y) &\approx 2.6666667 \\
\texttt{fl}(q) &\notin \texttt{fl}(\overline{ac}) \\
\texttt{fl}(q) &\notin \texttt{fl}(\overline{bd})
\end{aligned}
$$

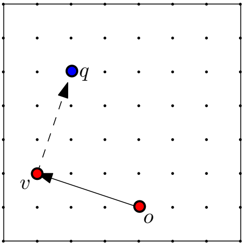
Predicates and Operations; Analyzing Precision [LPT99]

Precision used by the `Orientation` operation:



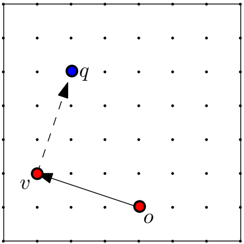
Input: single-precision coordinates of *o*, *v* and *q*.
Return: whether the straight line path from *o* to *v* to *q* forms a right turn, left turn or follows a straight line.

Precision used by the `Orientation` operation:



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

Precision used by the `Orientation` operation:

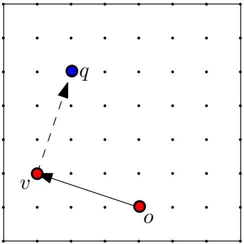


$\mathbb{U} = \{1, \ldots, U\}^2$
$o, v, q \in \mathbb{U}$
$o = (o_x, o_y)$
$v = (v_x, v_y)$
$q = (q_x, q_y)$

A *predicate* is a test of the sign of a multivariate polynomial with variables from the input coordinates.

$$\begin{aligned} P(o, v, q) &= \operatorname{sign}\left(\begin{vmatrix} v_x - o_x & v_y - o_y \\ q_x - o_x & q_y - o_y \end{vmatrix}\right) \\ &= \operatorname{sign}(v_x q_y - v_x o_y - o_x q_y + o_x o_y - v_y q_x + v_y o_x + q_y q_x - q_y o_x) \end{aligned}$$

Precision used by the `Orientation` operation:

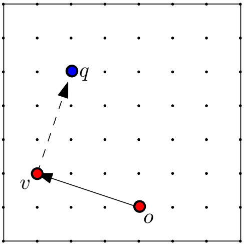


$$\mathbb{U} = \{1, \ldots, U\}^2$$
$$o, v, q \in \mathbb{U}$$
$$o = (o_x, o_y)$$
$$v = (v_x, v_y)$$
$$q = (q_x, q_y)$$

A *predicate* is a test of the sign of a multivariate polynomial with variables from the input coordinates.

$$\begin{aligned} P(o, v, q) &= \operatorname{sign}\left(\begin{vmatrix} v_x - o_x & v_y - o_y \\ q_x - o_x & q_y - o_y \end{vmatrix}\right) \\ &= \operatorname{sign}(v_x q_y - v_x o_y - o_x q_y + o_x o_y - v_y q_x + v_y o_x + q_y q_x - q_y o_x) \\ &= \operatorname{sign}(2) \end{aligned}$$

Precision used by the `Orientation` operation:



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

$P(o, v, q)$ is degree 2

A *predicate* is a test of the sign of a multivariate polynomial with variables from the input coordinates.

$$\begin{aligned} P(o, v, q) &= \operatorname{sign}\left(\begin{vmatrix} v_x - o_x & v_y - o_y \\ q_x - o_x & q_y - o_y \end{vmatrix}\right) \\ &= \operatorname{sign}(v_x q_y - v_x o_y - o_x q_y + o_x o_y - v_y q_x + v_y o_x + q_y q_x - q_y o_x) \\ &= \operatorname{sign}(2) \end{aligned}$$

Predicates and Operations; Analyzing Precision [LPT99]

How the degree of a predicate relates to precision.

Consider multivariate polynomial $Q(x)$ of degree k and a monomials.
The coordinates of x are b-bit integers.
Each monomial is in $\{-2^{bk}, \ldots, 2^{bk}\}$ (ignoring mult by a constant).
The value of $Q(x)$ is in $\{-a2^{bk}, \ldots, a2^{bk}\}$.
$\implies$ Values of $Q(x)$ are represented with $kb + \log(a) + O(1)$ bits.

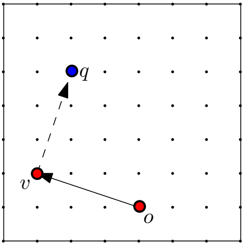
Predicates and Operations; Analyzing Precision [LPT99]

How the degree of a predicate relates to precision.

Consider multivariate polynomial $Q(x)$ of degree k and a monomials.
The coordinates of x are b-bit integers.
Each monomial is in $\{-2^{bk}, \ldots, 2^{bk}\}$ (ignoring mult by a constant).
The value of $Q(x)$ is in $\{-a2^{bk}, \ldots, a2^{bk}\}$.
$\implies$ Values of $Q(x)$ are represented with $kb + \log(a) + O(1)$ bits.

Note that kb bits is enough to evaluate the sign.

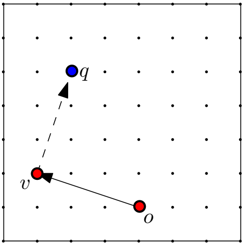
Precision used by the `Orientation` operation:



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

$P(o, v, q)$ is degree 2

Precision used by the `Orientation` operation:



$\mathbb{U} = \{1, \dots, U\}^2$

$o, v, q \in \mathbb{U}$

$o = (o_x, o_y)$

$v = (v_x, v_y)$

$q = (q_x, q_y)$

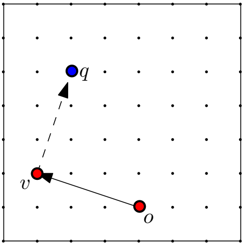
$P(o, v, q)$ is degree 2

Operation:

`Orientation`(o, v, q):

1: Sign $eval = P(o, v, q)$
2: **if** $eval > 0$ **then**
3: **return** LEFT
4: **else if** $eval < 0$ **then**
5: **return** RIGHT
6: **else**
7: **return** STRAIGHT
8: **end if**

Precision used by the `Orientation` operation:



$\mathbb{U} = \{1, \ldots, U\}^2$

$o, v, q \in \mathbb{U}$

$o = (o_x, o_y)$

$v = (v_x, v_y)$

$q = (q_x, q_y)$

$P(o, v, q)$ is degree 2

`Orientation` is degree 2

Operation:

`Orientation`(o, v, q):

Sign $eval = P(o, v, q)$
if $eval > 0$ **then**
    return LEFT
else if $eval < 0$ **then**
    return RIGHT
else
    return STRAIGHT
end if

Illustration of an Alg. that solves IsSegInter w/o construction

`IsSegInterByOrientation`(a, c, b, d): Determine if $\overline{ac}$ and $\overline{bd}$ intersect; if so return INTERSECT, if not return NOINTERSECT

Require: no three points are collinear

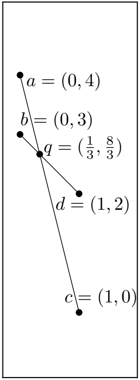
if `Orientation`$(a, c, b) \neq$ `Orientation`(a, c, d) and `Orientation`$(b, d, a) \neq$ `Orientation`(b, d, c) **then**

  return INTERSECT

else

  return NOINTERSECT

end if



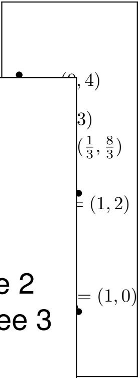
`IsSegInterByOrientation`(a, c, b, d): Determine if $\overline{ac}$ and $\overline{bd}$ intersect; if so return INTERSECT, if not return NOINTERSECT

Require: no three points are collinear

if `Orientation`$(a, c, b) \neq$ `Orientation`(a, c, d) and `Orientation`$(b, d, a) \neq$ `Orientation`(b, d, c) **then**

  return INTERSECT

else



In summary:

P predicate is degree 2

`Orientation` Operation is degree 2

`Intersect` construction is degree 3/2

`IsSegInterByOrientation` algorithm is degree 2

`IsSegInterByConstruction` algorithm is degree 3

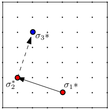
Other precision approaches

Approaches for implementing geometric algorithms with finite precision computer arithmetic:

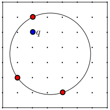
- Rely on machine precision ($+\epsilon$) [NAT90,LTH86,KMP*08]
- Exact Geometric Computation [Y97,C92,ABO*97,BEP*97]
- Arithmetic Filters [FW93,FW96,BBP01,DP98,DP99]
- Adaptive Predicates [P92,S97,BF09]
- Topological Consistency [S99,S01,SI90,SI92,SII*00]
- Degree-driven algorithm design
  [LPT99,BP00,BS00,C00,MS01,MS09,CMS09,MS10]

Goal: Descriptions, precision analysis and book-quality code for all predicates, operations and constructions, discussed in the thesis. This chapter conclude with results on lower bounds on degree and irreducibility.

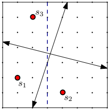
Simple Examples:



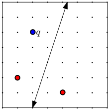
`Orientation` degree 2



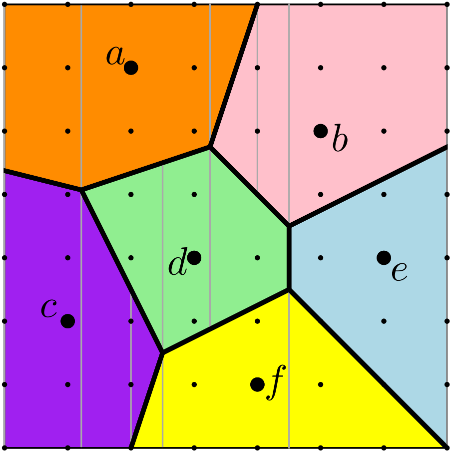
`InCircle` degree 4



`OrderOnLine` degree 3



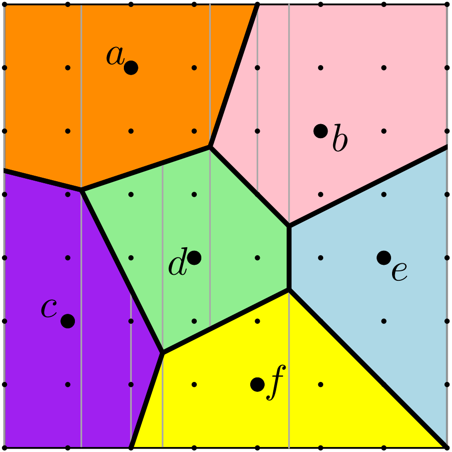
`SideOfBisector` degree 2



Goal: Compute a PO Query search structure with degree 2.
I propose to provide:

- degree 2 algorithm
- analysis and implementation
- book-quality code
- experimental results

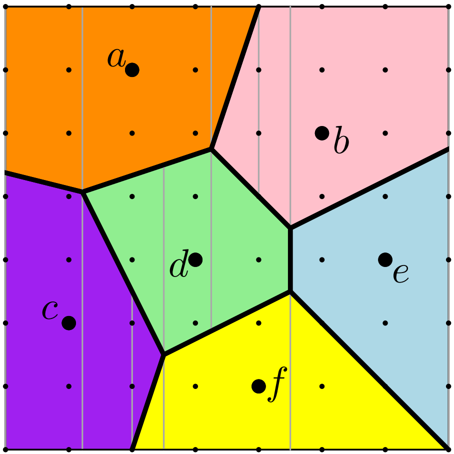
Post-office Query structure



Given
A grid of size U and sites $S = \{s_1, \ldots, s_n\} \subset \mathbb{U}$

Compute
A data structure capable of returning the closet $s_i \in S$ to a query point $q \in \mathbb{U}$ in $O(\log n)$ time

Precision of Voronoi Diagram/Trapeziod Graph



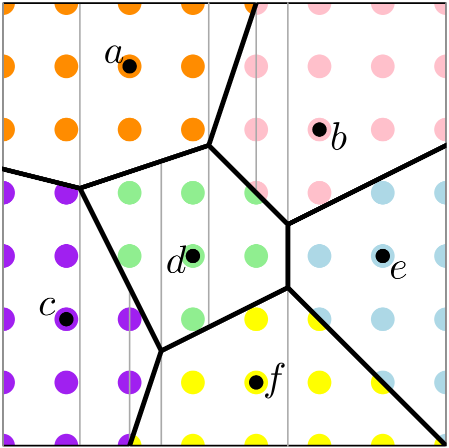
Voronoi diagram

- region
- edge
- vertex – rational degree 3/2

Trapezoid graph for proximity queries

- *x*-`node`() – degree 3
- *y*-`node`() – degree 6

Precision of Voronoi Diagram/Trapeziod Graph

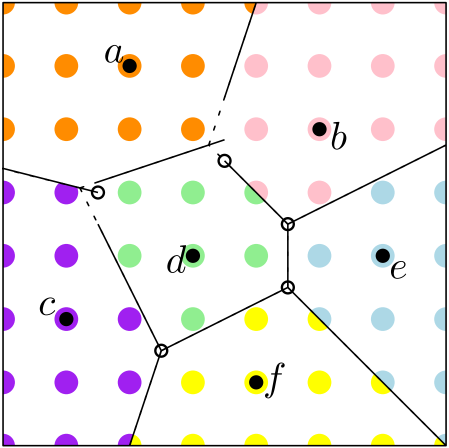


Voronoi diagram

- region
- edge
- vertex – rational degree 3/2

Trapezoid graph for proximity queries

- x-`node()` – degree 3
- y-`node()` – degree 6



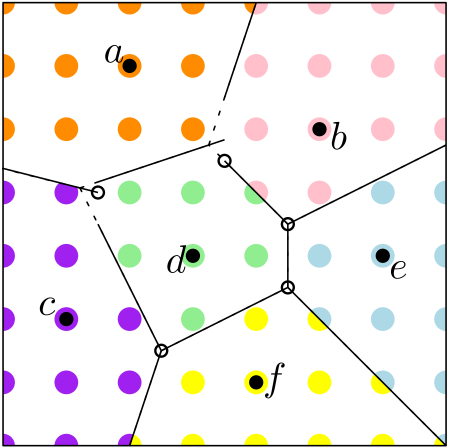
Voronoi diagram

- region
- edge
- vertex – rational degree 3/2

Trapezoid graph for proximity queries [LPT99]

- x-`node()` – degree 1
- y-`node()` – degree 2

Precision of Voronoi Diagram/Trapeziod Graph



Voronoi diagram

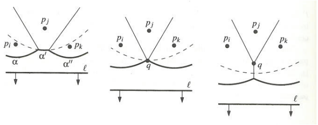
- region
- edge
- vertex – rational degree 3/2

Trapezoid graph for proximity queries [LPT99]

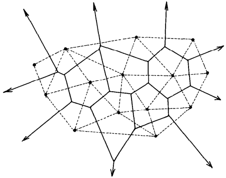
- *x*-`node()` – degree 1
- *y*-`node()` – degree 2

This is a degree 2 trapezoid graph.

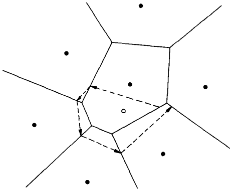
Three well-known Voronoi diagram constructions.



Sweepline[F87]
– degree 6



Divide and Conquer[GS86]
– degree 4



Tracing[SI92]
– degree 4

Precision of Constructing the Voronoi Diagram

Three well-known Voronoi diagram constructions.

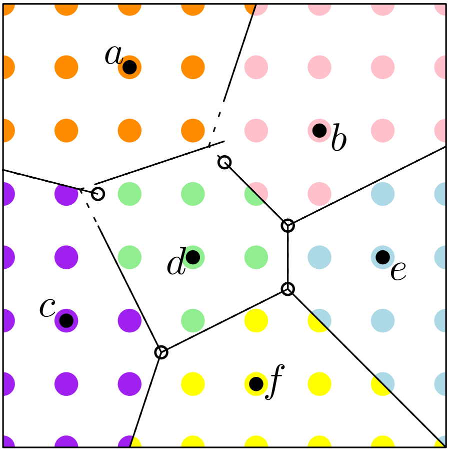
eepline[F87]
degree 6

de and Conquer[GS86]
degree 4

cing[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 [LPT99]



Implicit Voronoi diagram is disconnected.

RP-Voronoi [MS09]

Given n sites in $\mathbb{U}$.

RP-Voronoi

Rand inc construction of the RP-Voronoi of n sites in $\mathbb{U}$.

- Time: $O(n \log(Un))$ expected
- Space: $O(n)$ expected
- Precision: degree 3

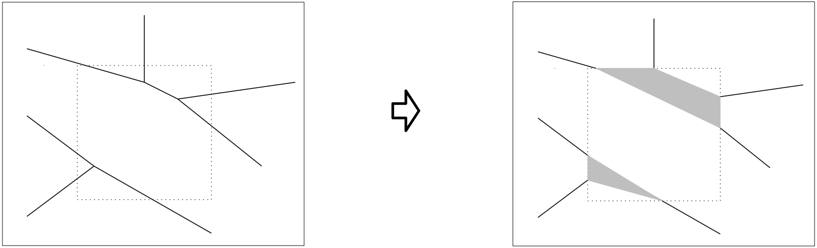
Implicit Voronoi

Construct LPT's implicit Voronoi from RP-Voronoi.

- Time: $O(n)$
- Space: $O(n)$ expected
- Precision: degree 3

RP-Voronoi [MS09]

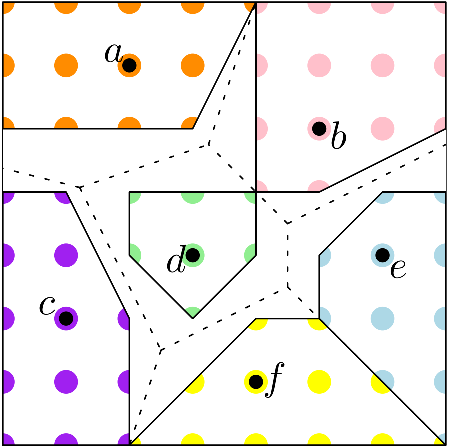
Given *n* sites in $\mathbb{U}$.



Contract trees of Voronoi vertices that occur in the same grid cell into an *rp-vertex*.

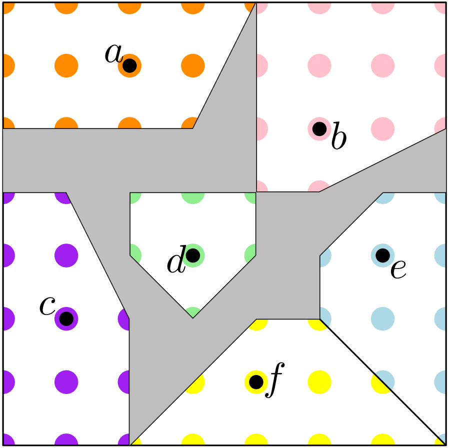
- Precision: degree 3

Voronoi Polygon Set



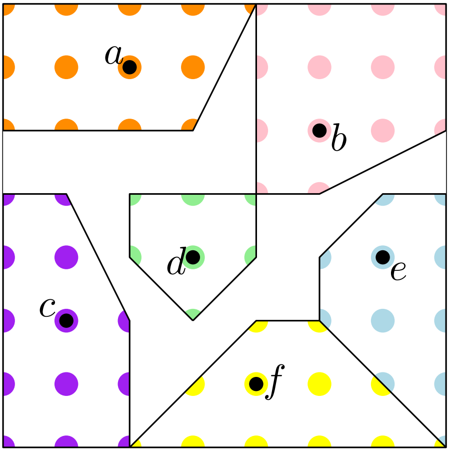
- *Voronoi polygon* is the convex hull of the grid points in a Voronoi cell.

Voronoi Polygon Set



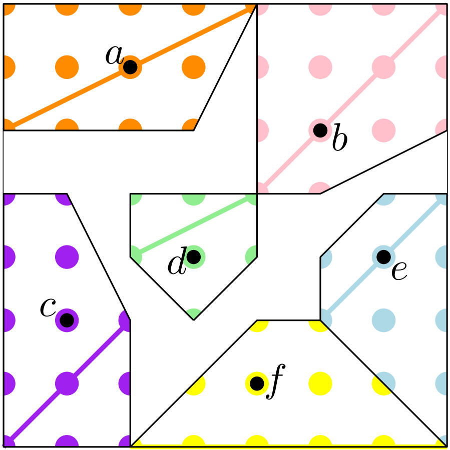
- *Voronoi polygon* is the convex hull of the grid points in a Voronoi cell.
- Gaps

Voronoi Polygon Set



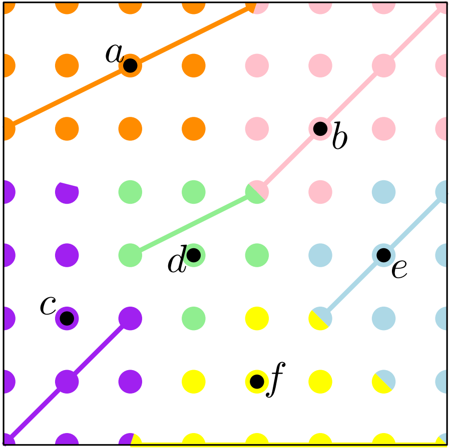
- *Voronoi polygon* is the convex hull of the grid points in a Voronoi cell.
- Gaps
- Total size $\Theta(n \log U)$.

Proxy Segments



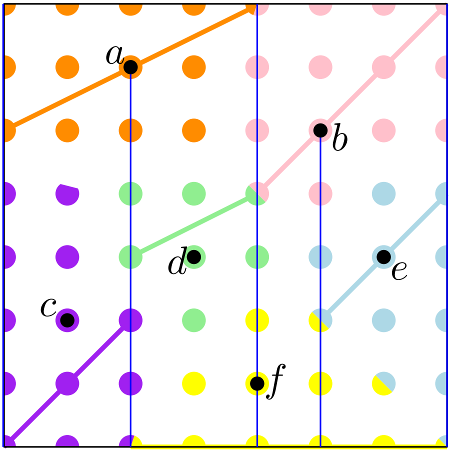
- *Proxy segment* -
  represent Voronoi polygons.

Proxy Segments



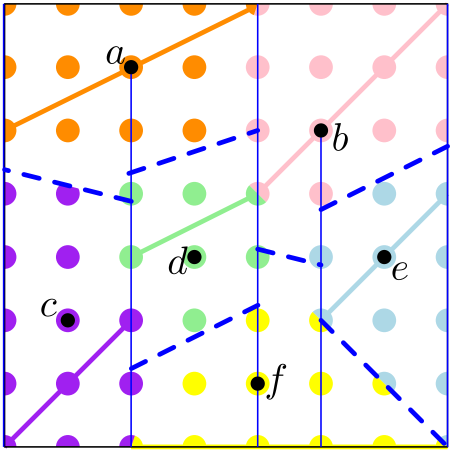
- *Proxy segment* -
  represent Voronoi polygons.

Proxy Segments



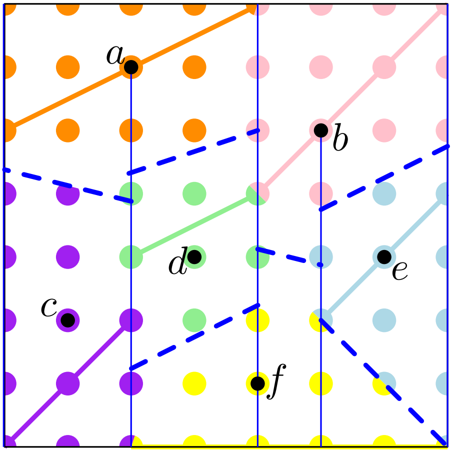
- *Proxy segment* - represent Voronoi polygons.
- *Proxy trapezoidation* - trapezoidation of the proxies.

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 Trapezoidation [MS10]

Given n sites in $\mathbb{U}$.

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

* Analysis of [MS10] is incomplete.

Completing the analysis of [MS10]

Proxy trapezoidation is built with a randomized incremental construction (RIC).

Analysis of [MS10] used the RIC construction framework from the dutch book.

Define: For a grid point g, a set of sites R *certifies* that g is the right end point for the proxy of s if all grid points right g are closer to a site in R than s.

To complete Analysis of [MS10], I need to prove:

Lemma

The maximum number of sites of S required to certify that a grid point is a right end point of a proxy segment is constant.

Goals: Complete the analysis of [MS10], describe the algorithm, provide an implementation, book-quality code and experimental results.

Should I be unable to complete the analysis, I will explore whether a divide-and-conquer algorithm can yield a sub-quadratic time degree 2 construction.

Should that be unsuccessful, I will implement our RIC degree 2 algorithm and observe the experimental running time, implement the degree 3 solution, and provide book-quality code and experimental results for both.

Goals: Complete the analysis of [MS10], describe the algorithm, provide an implementation, book-quality code and experimental results.

Should I be unable to complete the analysis, I will explore whether a divide-and-conquer algorithm can yield a sub-quadratic time degree 2 construction.

Should that be unsuccessful, I will implement our RIC degree 2 algorithm and observe the experimental running time, implement the degree 3 solution, and provide book-quality code and experimental results for both.

Goals: Complete the analysis of [MS10], describe the algorithm, provide an implementation, book-quality code and experimental results.

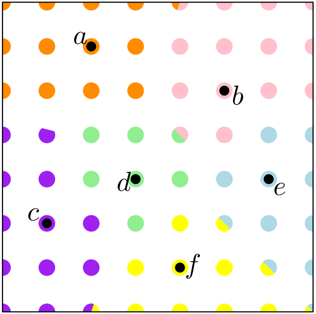
Should I be unable to complete the analysis, I will explore whether a divide-and-conquer algorithm can yield a sub-quadratic time degree 2 construction.

Should that be unsuccessful, I will implement our RIC degree 2 algorithm and observe the experimental running time, implement the degree 3 solution, and provide book-quality code and experimental results for both.

Goals: Complete the analysis of [MS10], describe the algorithm, provide an implementation, book-quality code and experimental results.

Should I be unable to complete the analysis, I will explore whether a divide-and-conquer algorithm can yield a sub-quadratic time degree 2 construction.

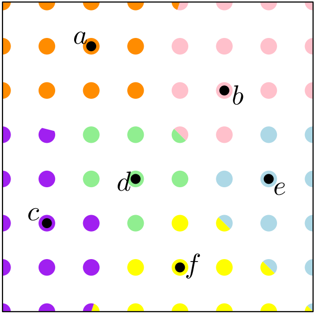
Should that be unsuccessful, I will implement our RIC degree 2 algorithm and observe the experimental running time, implement the degree 3 solution, and provide book-quality code and experimental results for both.



Goal: Compute Nearest Neighbor Transform with degree 2.
I propose to provide:

- degree 2 algorithm
- analysis and implementation
- book-quality code
- experimental results

Nearest Neighbor Transform



Given

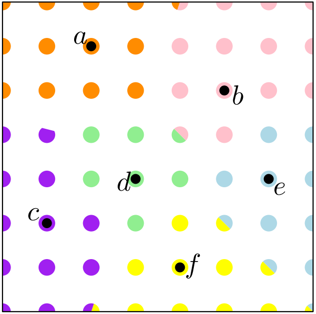
A grid of size U and

Sites $S = \{s_1, \ldots, s_n\} \subset \mathbb{U}$

Label

Each grid point of $\mathbb{U}$ with the closest site of S

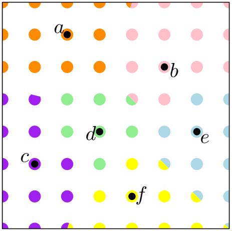
Nearest Neighbor Transform



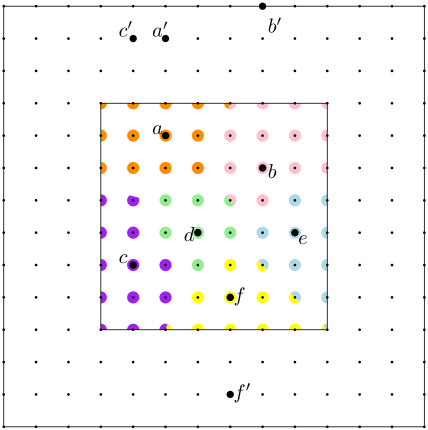
Given
A grid of size U and
Sites $S = \{s_1, \ldots, s_n\} \subset \mathbb{U}$

Label
Each grid point of $\mathbb{U}$ with the closest site of S

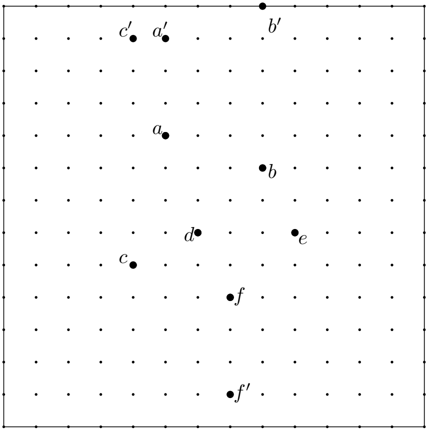
	Alg	Time
Brute Force	deg 2	$O(nU^2)$
Query the Voronoi diagram	deg 4	$O(U^2 \log n)$
Nearest Neighbor Trans. [B90]	deg 4	$O(U^2)$
Dim. Reduction [C06,MQR03]	deg 3	$O(U^2)$
GPU Cone Rendering [H99]	-	$\Theta(nU^2)$
GPU Dim Reduction [CTM*10]	deg 3	$O(U^2)$



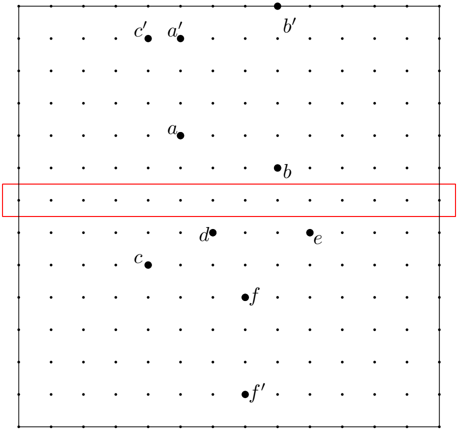
Example of processing one row



Example of processing one row



Example of processing one row

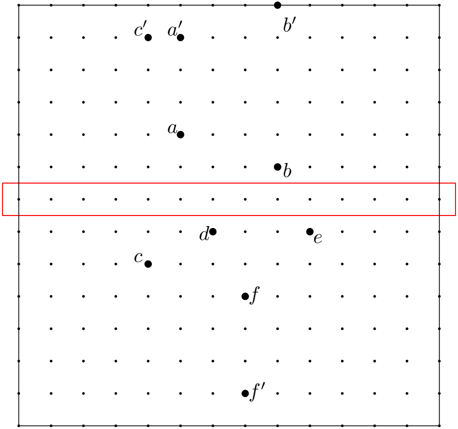


Example of processing one row

Two steps:

(1) Reduce to at most $2U$ sites.

(2) Compute the intersection of the Voronoi diagram of the reduced set of sites with a line though the row.

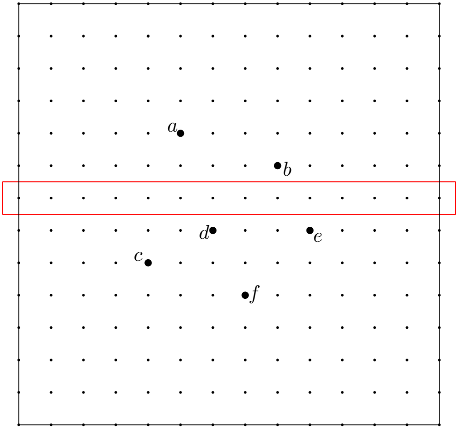


Example of processing one row

Two steps:

(1) Reduce to at most $2U$ sites.

(2) Compute the intersection of the Voronoi diagram of the reduced set of sites with a line though the row.



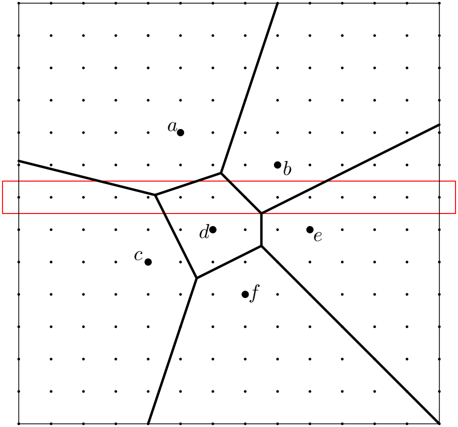
Example of processing one row

Two steps:

(1) Reduce to at most $2U$ sites.

(2) Compute the intersection of the Voronoi diagram of the reduced set of sites with a line though the row.

Dimensional Reduction



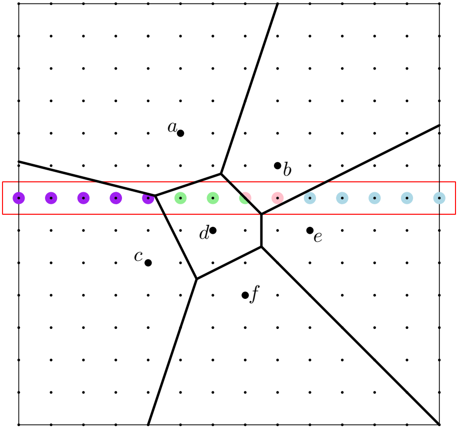
Example of processing one row

Two steps:

(1) Reduce to at most $2U$ sites.

(2) Compute the intersection of the Voronoi diagram of the reduced set of sites with a line though the row.

Dimensional Reduction



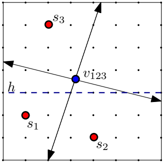
Example of processing one row

Two steps:

(1) Reduce to at most $2U$ sites.

(2) Compute the intersection of the Voronoi diagram of the reduced set of sites with a line though the row.

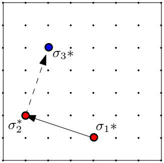
Predicates:



[MQR03, CTM*10]:

`Above`(v_{123}, h) deg 3

`OrderOnLine`(b_{12}, b_{13}, h) deg 3

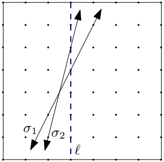


[C06]:

$\sigma_i := y = \textcircled{1}x + \textcircled{2}$

$\sigma_i^* = (\textcircled{1}, \textcircled{2})$

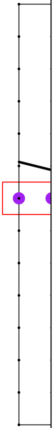
`Orient_d1_d2`$(\sigma_1, \sigma_2, \sigma_3)$ deg 3



[CMS09]:

$\sigma_i = y = \textcircled{1}x + \textcircled{2}$

`OrderOnLine_d1_d2`$(\sigma_1, \sigma_2, \ell)$ deg 2



es.

he

Nearest Neighbor Transform [CMS09]

Compute 2D Nearest Neighbor Transform

- Time: $O(U^2)$ expected time.
- Space: $O(n+U)$
- Precision: degree 2

Assuming $O(n \log n) < O(U^2)$

Compute Voronoi Polygon Set

- Time: $O(U^2)$ expected time
- Space: $O(n \log U)$ and $O(n)$ for proxies only
- Precision: degree 2

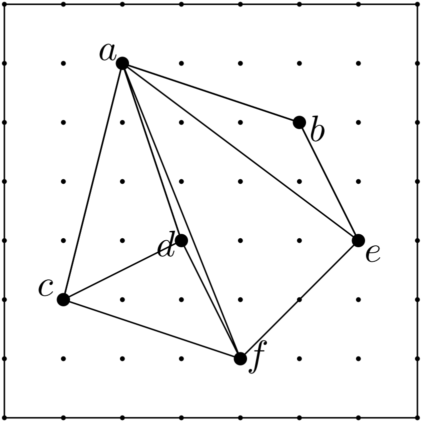
Query Post Office Structure

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

Goal: Our solution, written up in [CMS09], only contains a sketch of the construction, without analysis. In this chapter I propose to provide the details of the construction, analysis, book-quality code, and experimental results for our implementation.

Goal: Our solution, written up in [CMS09], only contains a sketch of the construction, without analysis. In this chapter I propose to provide the details of the construction, analysis, book-quality code, and experimental results for our implementation.

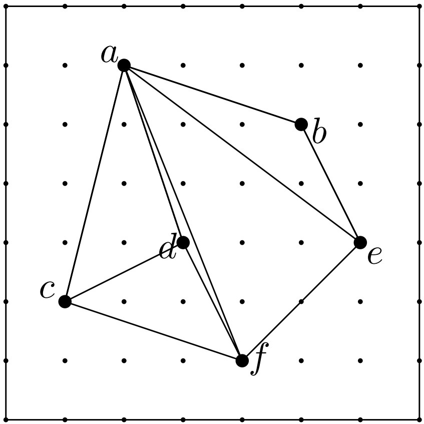
Ch. 5: Triangulations



Goals: Compute Triangulation with degree 2 or degree 3.
I propose to provide:

- deg 2 or deg 3 algorithm
- analysis and implementation
- book-quality code
- experimental results

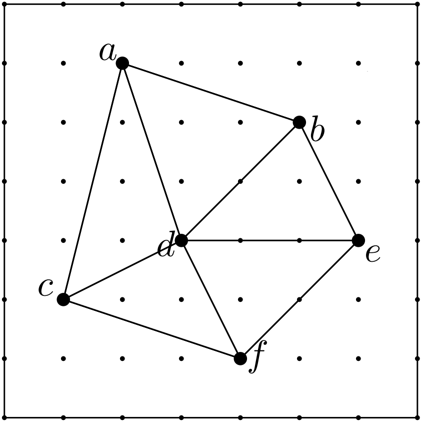
Triangulations



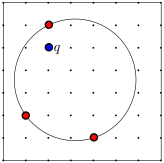
Given
A grid of size U and
sites $S = \{s_1, \ldots, s_n\} \subset \mathbb{U}$

Compute
A planar subdivision with
vertices in S and
edges such that no more edges
can be added without causing the
subdivision to become non-planar

Delaunay Triangulation

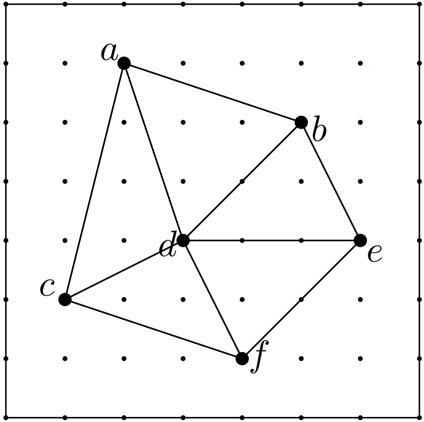


InCircle

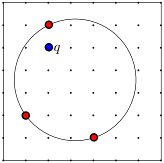


$$Q() = \operatorname{sign}\left(\begin{vmatrix} 1 & 1 & 2 \\ 1 & 1 & 2 \\ 1 & 1 & 2 \end{vmatrix}\right)$$

Delaunay Triangulation



`InCircle`



$$Q() = \mathrm{sign}\left(\begin{vmatrix} 1 & 1 & 2 \\ 1 & 1 & 2 \\ 1 & 1 & 2 \end{vmatrix}\right)$$

degree 4

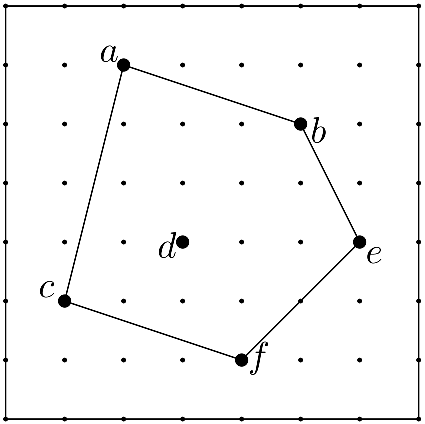
How can we compute
a triangulation with
less than degree 4,
and what are some properties
of this triangulation?

Ideas for Computing a Triangulation

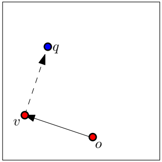
Use low degree algorithms
to compute a subset of known Delaunay edges.

Then, complete add edges to complete a triangulation.

Ideas for Computing a Triangulation

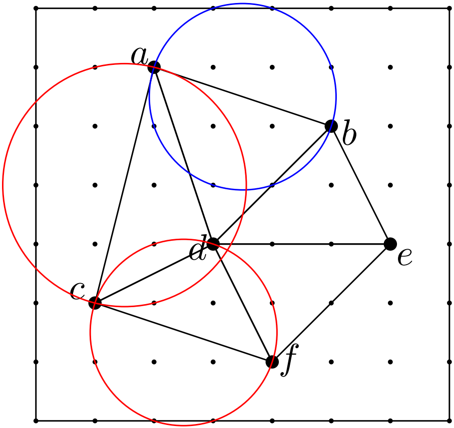


Convex Hull w/ h hull vertices:
- Melkman[M87], $O(n \log n)$, deg 2
- Chan [C96], $O(n \log h)$, deg 2

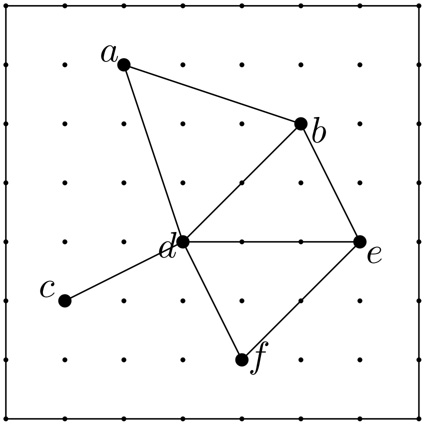


`Orientation` deg 2

Gabriel Graph

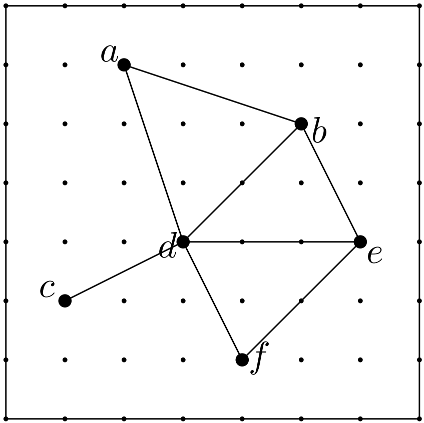


Defn: An edge $\overline{pq}$ is in the Gabriel graph of S if the closed disk centered at the midpoint of $\overline{pq}$ with diameter $|\overline{pq}|$ contains no points other than p and q.



Defn: An edge $\overline{pq}$ is in the Gabriel graph of S if the closed disk centered at the midpoint of $\overline{pq}$ with diameter $|\overline{pq}|$ contains no points other than p and q.

Gabriel Graph



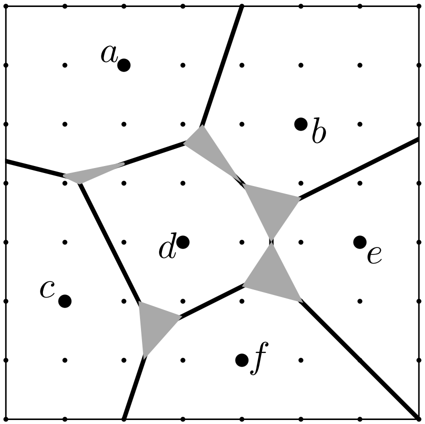
Defn: An edge $\overline{pq}$ is in the Gabriel graph of S if the closed disk centered at the midpoint of $\overline{pq}$ with diameter $|\overline{pq}|$ contains no points other than p and q.

Proposed by:
Gabriel and Sokal [GS69]

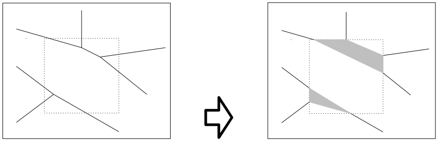
Compute Gabriel from Delaunay:
[MS80] $O(n)$ time, degree 6
[L96] $O(n)$ time, degree 2

Directly compute Gabriel graph:
Brute force, $O(n^3)$ time, degree 2

rp-Voronoi [MS09]

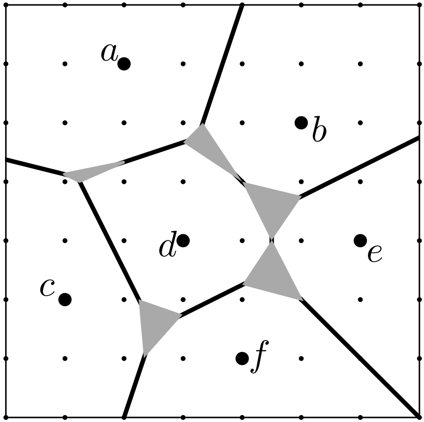


Defn: Replace connected subtrees of Voronoi edges inside a cell with their convex hulls

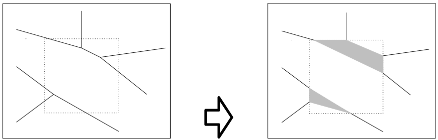


Captures any Voronoi edge longer that $\sqrt{2}$.

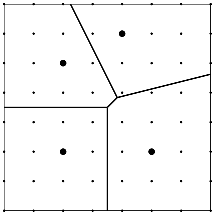
rp-Voronoi [MS09]

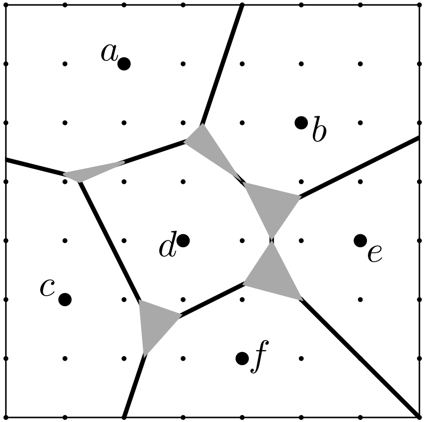


Defn: Replace connected subtrees of Voronoi edges inside a cell with their convex hulls

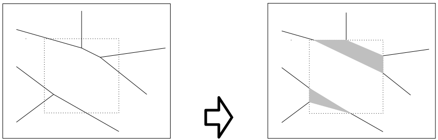


Captures any Voronoi edge longer that $\sqrt{2}$.

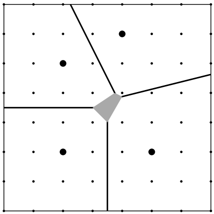




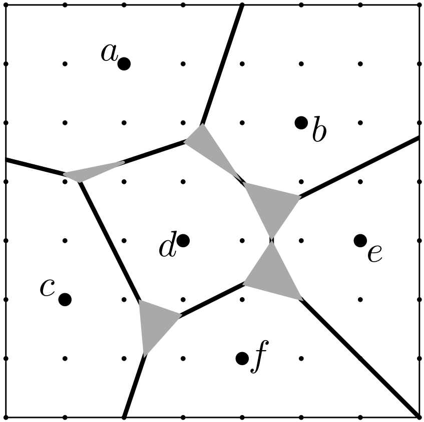
Defn: Replace connected subtrees of Voronoi edges inside a cell with their convex hulls



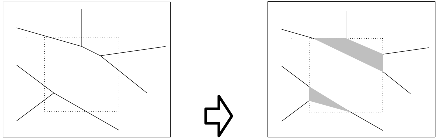
Captures any Voronoi edge longer that $\sqrt{2}$.



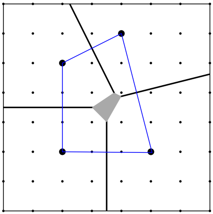
rp-Voronoi [MS09]



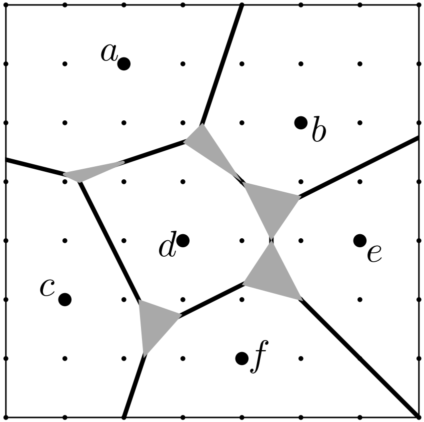
Defn: Replace connected subtrees of Voronoi edges inside a cell with their convex hulls



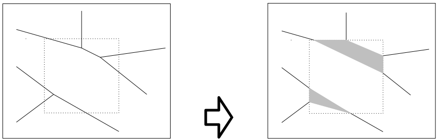
Captures any Voronoi edge longer that $\sqrt{2}$.



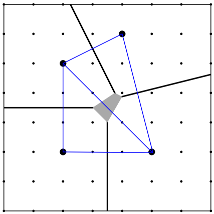
rp-Voronoi [MS09]



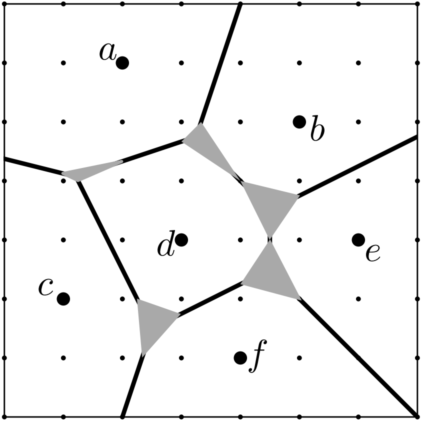
Defn: Replace connected subtrees of Voronoi edges inside a cell with their convex hulls



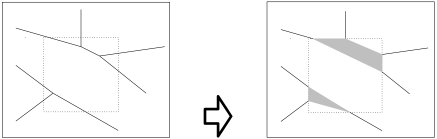
Captures any Voronoi edge longer that $\sqrt{2}$.



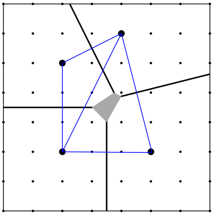
rp-Voronoi [MS09]



Defn: Replace connected subtrees of Voronoi edges inside a cell with their convex hulls



Captures any Voronoi edge longer that $\sqrt{2}$.



Ch. 5: Triangulations

Goals: Describe a triangulation that can be computed sub-quadratic time with two- or three-fold precision, provide book-quality code and experiments for an implementation and some of the properties that the proposed triangulation possesses. Some properties may include angle bounds of the triangulation, or how far it is in the flip graph from the Delaunay.

Should these properties be too difficult to discover, experimental results may be supplied.

Goals: Describe a triangulation that can be computed sub-quadratic time with two- or three-fold precision, provide book-quality code and experiments for an implementation and some of the properties that the proposed triangulation possesses. Some properties may include angle bounds of the triangulation, or how far it is in the flip graph from the Delaunay.

Should these properties be too difficult to discover, experimental results may be supplied.

Goals: Describe a triangulation that can be computed sub-quadratic time with two- or three-fold precision, provide book-quality code and experiments for an implementation and some of the properties that the proposed triangulation possesses. Some properties may include angle bounds of the triangulation, or how far it is in the flip graph from the Delaunay.

Should these properties be too difficult to discover, experimental results may be supplied.

Goals: Describe a triangulation that can be computed sub-quadratic time with two- or three-fold precision, provide book-quality code and experiments for an implementation and some of the properties that the proposed triangulation possesses. Some properties may include angle bounds of the triangulation, or how far it is in the flip graph from the Delaunay.

Should these properties be too difficult to discover, experimental results may be supplied.

Timeline

- **June 1, 2011** Draft chapters of:
  - Ch. 2: Geometric Primitives
  - Ch. 4: Nearest Neighbor Transform
- **Sept 1, 2011** Implementation of:
  - degree 2 and/or degree 3 Voronoi construction.
  - a degree 2 or degree 3 triangulation.

Teach in the Fall:

- **Jan 1, 2012** Drafts of:
  - Ch. 3: PO Queries
  - (status) Ch. 5: Triangulations
- **March 1, 2012** Draft of:
  - Ch. 5: Triangulations
- ~**April 1, 2012** Defense.

Teach in the Spring:

- **Oct 15, 2012** Draft of:
  - Ch. 3: PO Queries
- **Dec 15, 2012** Draft of:
  - Ch. 5: Triangulations
- ~**March 1, 2012** Defense

Conclusion

Thesis Statement: Degree-driven analysis supports the development of new, robust geometric algorithms, as I have demonstrated for computing Post-office query search structures, Nearest Neighbor Transforms, and Triangulations.