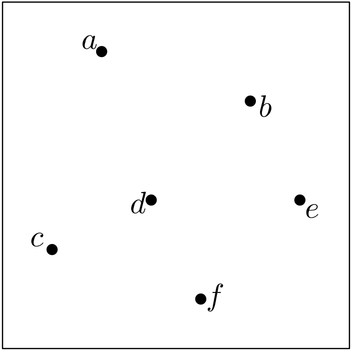
Degree-Driven Design of Geometric Algorithms for Point Location, Proximity, and Volume Calculation
PhD Defense

David L. Millman

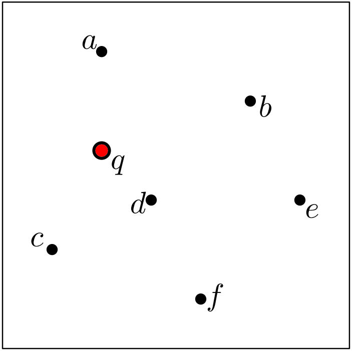
University of North Carolina at Chapel Hill

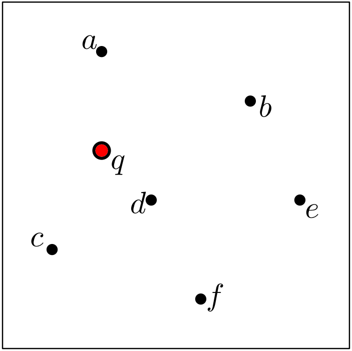
October 10, 2012

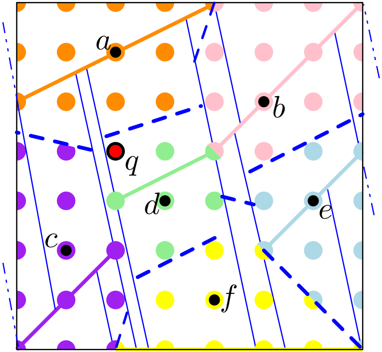
Geometric Algorithms



Geometric Algorithms

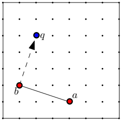




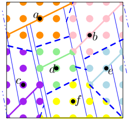


Point location data structure

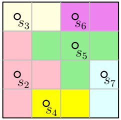
Overview



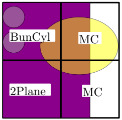
- Derive & upper bdd precision of many common preds
- Show the polys in the common preds are irreducible



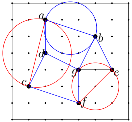
- Compute point location data structure with double & triple precision



- Compute nearest neighbor transform with double precision

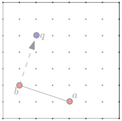


- Compute volumes of CSG models with five-fold precision

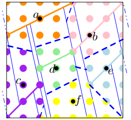


- Compute Gabriel graph with double precision

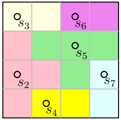
Overview



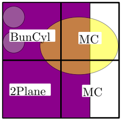
- Derive & upper bdd precision of many common preds
- Show the polys in the common preds are irreducible



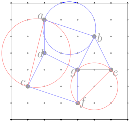
- Compute point location data structure with double & triple precision



- Compute nearest neighbor transform with double precision

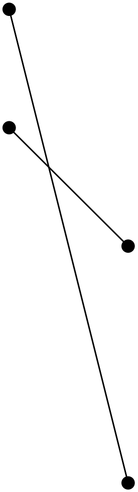


- Compute volumes of CSG models with five-fold precision



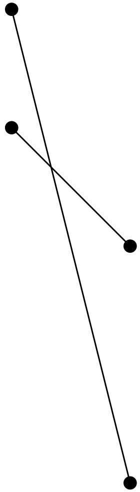
- Compute Gabriel graph with double precision

A Motivational Problem



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

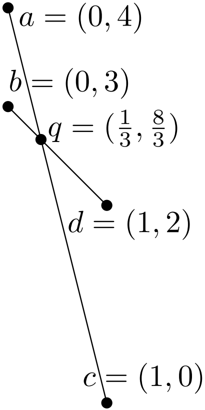
A Motivational Problem



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

How much arithmetic precision is needed to determine this?

Input: Geometric configuration specified by single precision numerical coordinates and relationships between coordinates.



E.g. DoSegsIntersect problem:

Numerical coordinates:

$(0, 4, 0, 3, 1, 0, 1, 2)$

Relationships between coordinates:

$a = (a_x, a_y) = (0, 4)$

$b = (b_x, b_y) = (0, 3)$

$c = (c_x, c_y) = (1, 0)$

$d = (d_x, d_y) = (1, 2)$

$\overline{ac} = (a, c)$

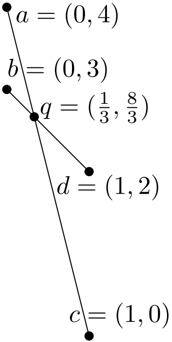
$\overline{bd} = (b, d)$

Solving DoSegsIntersect with Construction

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

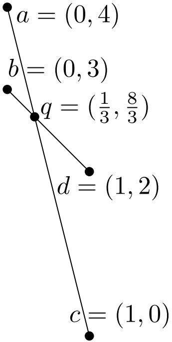
Require: no three points are collinear

```
if ac↔ ∥ bd↔ then
    return NOINTERSECT
end if
Point q = ac↔ ∩ bd↔
Real t1 = (qx − ax)/(cx − ax)
Real t2 = (qx − bx)/(dx − bx)
if t1 ∈ (0, 1) and t2 ∈ (0, 1) then
    return INTERSECT
else
    return NOINTERSECT
end if
```

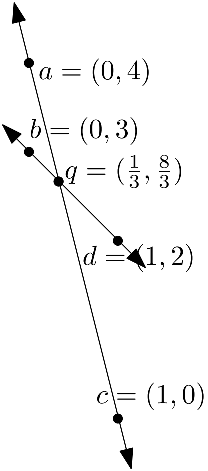


`InterByConstruction`(a, c, b, d):
Determine if $\overline{ac}$ and $\overline{bd}$ intersect;
if so return INTERSECT, if not return NOINTERSECT

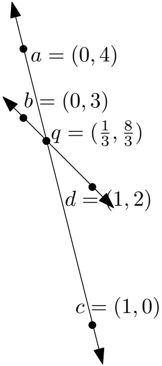
Require: no three points are collinear
1: **if** $\overleftrightarrow{ac} \parallel \overleftrightarrow{bd}$ **then**
2: **return** NOINTERSECT
3: **end if**
4: Point $q = \overleftrightarrow{ac} \cap \overleftrightarrow{bd}$
5: Real $t_1 = (q_x - a_x)/(c_x - a_x)$
6: Real $t_2 = (q_x - b_x)/(d_x - b_x)$
7: **if** $t_1 \in (0, 1)$ and $t_2 \in (0, 1)$ **then**
8: **return** INTERSECT
9: **else**
10: **return** NOINTERSECT
11: **end if**



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



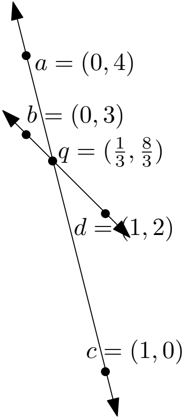
The `Intersect`(a, c, b, d) construction:



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 $\overleftrightarrow{bd}$.

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

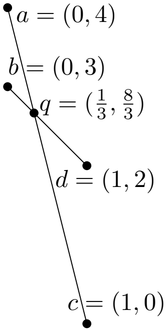
The `Intersect`(a, c, b, d) *construction*:



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 $\overleftrightarrow{bd}$.

$$q_x = 0.\overline{3}$$
$$q_y = 2.\overline{6}$$

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

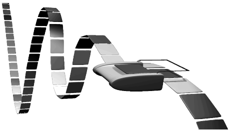


Input: single precision coordinates of a, c, b and d defining non-parallel lines $\overleftrightarrow{ac}$ and $\overleftrightarrow{bd}$.
Construct: the intersection q of $\overleftrightarrow{ac}$ and $\overleftrightarrow{bd}$.
In Python with *numpy.float32* type[a]:

$$\mathtt{fl}(q_x) \approx 0.33333334$$
$$\mathtt{fl}(q_y) \approx 2.66666675$$
$$\mathtt{fl}(q) \notin \mathtt{fl}(\overline{ac}) \;\&\; \mathtt{fl}(q) \notin \overline{ac}$$
$$\mathtt{fl}(q) \notin \mathtt{fl}(\overline{bd}) \;\&\; \mathtt{fl}(q) \notin \overline{bd}$$

[a] Values are the shortest decimal fraction that rounds correctly back to the true binary value.

Thesis Statement

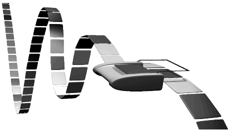


Real-RAM has 3 unbounded quantities.
The number of:

1. steps an algorithm may take
2. memory cells an algorithm may use
3. bits for representing numbers in cells

Image from: `http://en.wikipedia.org/wiki/File:Maquina.png`

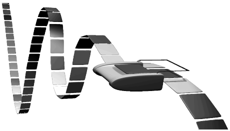
Thesis Statement



Real-RAM has 3 unbounded quantities.
The number of:

1. steps an algorithm may take
2. memory cells an algorithm may use
3. bits for representing numbers in cells

Image from: http://en.wikipedia.org/wiki/File:Maquina.png



Real-RAM has 3 unbounded quantities.
The number of:

1. steps an algorithm may take
2. memory cells an algorithm may use
3. bits for representing numbers in cells

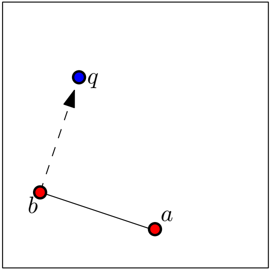
Thesis Statement:

Degree-driven design supports the development of new and robust geometric algorithms.

Image from: `http://en.wikipedia.org/wiki/File:Maquina.png`

Analyzing Precision [LPT99]

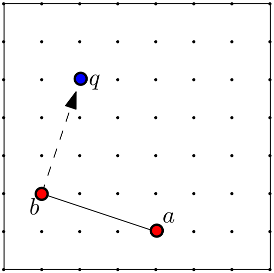
Precision used by the `isRightTurn`:



Input: single precision coordinates of *a*, *b* and *q*.
Return: whether the straight line path from *a* to *b* to *q* forms a right turn.

Analyzing Precision [LPT99]

Precision used by the `isRightTurn`:



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

$a, b, q \in \mathbb{U}$

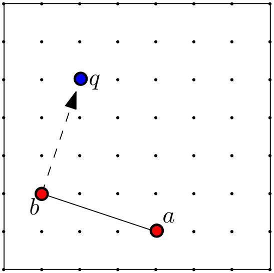
$a = (a_x, a_y)$

$b = (b_x, b_y)$

$q = (q_x, q_y)$

Analyzing Precision [LPT99]

Precision used by the `isRightTurn`:



$$\mathbb{U} = \{1, \ldots, U\}^2$$
$$a, b, q \in \mathbb{U}$$
$$a = (a_x, a_y)$$
$$b = (b_x, b_y)$$
$$q = (q_x, q_y)$$

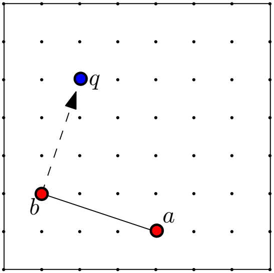
A *predicate* is a test of the sign of a multivariate polynomial with variables from the input coordinates.

$$\texttt{Orientation}(a, b, q) = \mathrm{sign}(b_x q_y - b_x a_y - a_x q_y - q_x b_y + q_x a_y + a_x b_y)$$

$\texttt{Orientation} < 0$ Right turn
$\texttt{Orientation} > 0$ Left turn
$\texttt{Orientation} = 0$ Collinear

Analyzing Precision [LPT99]

Precision used by the `isRightTurn`:



$$\mathbb{U} = \{1, \ldots, U\}^2$$
$$a, b, q \in \mathbb{U}$$
$$a = (a_x, a_y)$$
$$b = (b_x, b_y)$$
$$q = (q_x, q_y)$$

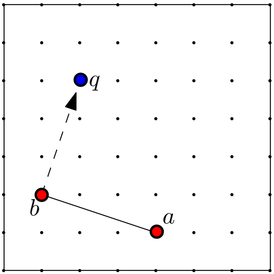
A *predicate* is a test of the sign of a multivariate polynomial with variables from the input coordinates.

$$\texttt{Orientation}(a, b, q) = \operatorname{sign}(b_x q_y - b_x a_y - a_x q_y - q_x b_y + q_x a_y + a_x b_y)$$
$$= \operatorname{sign}(\textcircled{2})$$

`Orientation` < 0 Right turn
`Orientation` > 0 Left turn
`Orientation` $= 0$ Collinear

Analyzing Precision [LPT99]

Precision used by the `isRightTurn`:



$\mathbb{U} = \{1, \ldots, U\}^2$
$a, b, q \in \mathbb{U}$
$a = (a_x, a_y)$
$b = (b_x, b_y)$
$q = (q_x, q_y)$

`Orientation` is degree 2

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

$$\texttt{Orientation}(a, b, q) = \text{sign}(b_x q_y - b_x a_y - a_x q_y - q_x b_y + q_x a_y + a_x b_y) = \text{sign}(②)$$

`Orientation` < 0 Right turn
`Orientation` > 0 Left turn
`Orientation` $= 0$ Collinear

Analyzing Precision [LPT99]

How the degree relates to precision:

Consider multivariate poly $Q(x_1, \ldots, x_n)$ of deg k and s monomials (for simplicity, assume that coefficient of each monomial is 1).

Analyzing Precision [LPT99]

How the degree relates to precision:

Consider multivariate poly $Q(x_1, \ldots, x_n)$ of deg k and s monomials (for simplicity, assume that coefficient of each monomial is 1).

Let each x_i be an ℓ-bit integer $\implies x_i \in \{-2^\ell, \ldots, 2^\ell\}$.

Analyzing Precision [LPT99]

How the degree relates to precision:

Consider multivariate poly $Q(x_1, \ldots, x_n)$ of deg k and s monomials (for simplicity, assume that coefficient of each monomial is 1).

Let each x_i be an ℓ-bit integer $\implies x_i \in \{-2^\ell, \ldots, 2^\ell\}$.

Each monomial is in $\{-2^{\ell k}, \ldots, 2^{\ell k}\}$.

Analyzing Precision [LPT99]

How the degree relates to precision:

Consider multivariate poly $Q(x_1, \ldots, x_n)$ of deg k and s monomials (for simplicity, assume that coefficient of each monomial is 1).
Let each x_i be an ℓ-bit integer $\implies x_i \in \{-2^\ell, \ldots, 2^\ell\}$.
Each monomial is in $\{-2^{\ell k}, \ldots, 2^{\ell k}\}$.
The value of $Q(x_1, \ldots, x_n)$ is in $\{-s2^{\ell k}, \ldots, s2^{\ell k}\}$.

Analyzing Precision [LPT99]

How the degree relates to precision:

Consider multivariate poly $Q(x_1, \ldots, x_n)$ of deg k and s monomials (for simplicity, assume that coefficient of each monomial is 1).
Let each x_i be an ℓ-bit integer $\implies x_i \in \{-2^\ell, \ldots, 2^\ell\}$.
Each monomial is in $\{-2^{\ell k}, \ldots, 2^{\ell k}\}$.
The value of $Q(x_1, \ldots, x_n)$ is in $\{-s2^{\ell k}, \ldots, s2^{\ell k}\}$.
$\implies \ell k + \log(s) + O(1)$ bits are enough to evaluate Q.

Analyzing Precision [LPT99]

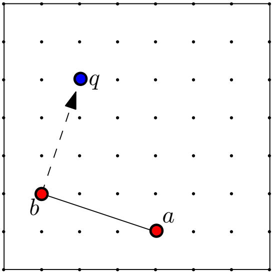
How the degree relates to precision:

Consider multivariate poly $Q(x_1, \ldots, x_n)$ of deg k and s monomials (for simplicity, assume that coefficient of each monomial is 1).
Let each x_i be an ℓ-bit integer $\implies x_i \in \{-2^\ell, \ldots, 2^\ell\}$.
Each monomial is in $\{-2^{\ell k}, \ldots, 2^{\ell k}\}$.
The value of $Q(x_1, \ldots, x_n)$ is in $\{-s2^{\ell k}, \ldots, s2^{\ell k}\}$.
$\implies \ell k + \log(s) + O(1)$ bits are enough to evaluate Q.

Note that ℓk bits is enough to evaluate the sign.

Analyzing Precision [LPT99]

Precision used by the `isRightTurn`:

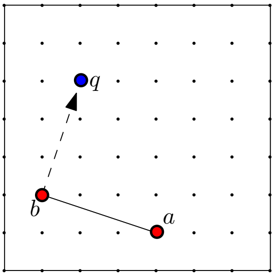


$\mathbb{U} = \{1, \ldots, U\}^2$
$a, b, q \in \mathbb{U}$
$a = (a_x, a_y)$
$b = (b_x, b_y)$
$q = (q_x, q_y)$

`Orientation` is degree 2

Analyzing Precision [LPT99]

Precision used by the `isRightTurn`:



$\mathbb{U} = \{1, \ldots, U\}^2$
$a, b, q \in \mathbb{U}$
$a = (a_x, a_y)$
$b = (b_x, b_y)$
$q = (q_x, q_y)$

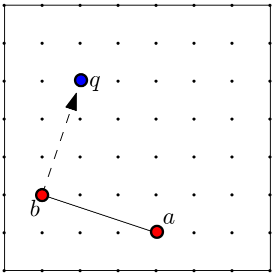
`Orientation` is degree 2

`isRightTurn`(a, b, q):

1: **if** `Orientation`$(a, b, q) < 0$ **then**
2: **return** TRUE
3: **else**
4: **return** FALSE
5: **end if**

Analyzing Precision [LPT99]

Precision used by the `isRightTurn`:



$$\mathbb{U} = \{1, \ldots, U\}^2$$
$$a, b, q \in \mathbb{U}$$
$$a = (a_x, a_y)$$
$$b = (b_x, b_y)$$
$$q = (q_x, q_y)$$

`Orientation` is degree 2
`isRightTurn` is degree 2

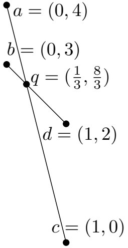
`isRightTurn`(a, b, q):

```
if Orientation(a, b, q) < 0 then
    return TRUE
else
    return FALSE
end if
```

Solving DoSegsIntersect without Construction

`InterByOrientation`(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
1: **if** `Orientation`$(a, c, b) \neq$ `Orientation`(a, c, d) and `Orientation`$(b, d, a) \neq$ `Orientation`(b, d, c) **then**
2: **return** INTERSECT
3: **else**
4: **return** NOINTERSECT
5: **end if**



Solving DoSegsIntersect without Construction

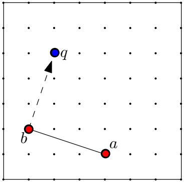
`InterByOrientation`(a, c, b, d):
Determine if $\overline{ac}$ and $\overline{bd}$ intersect;
if so return INTERSECT, if not return NOINTERSECT

Require: no three points are collinear
1: **if** `Orientation`$(a, c, b) \neq$ `Orientation`(a, c, d) and `Orientation`$(b, d, a) \neq$ `Orientation`(b, d, c) **then**
2: **return** INTERSECT

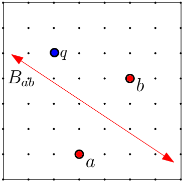
In summary:
`Orientation` predicate is degree 2
`InterByOrientation` algorithm is degree 2
`InterByConstruction` algorithm is degree 3

More Predicates

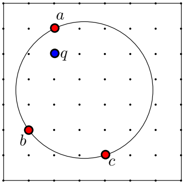
Some other well known predicates:



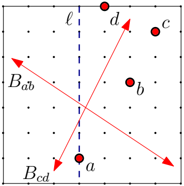
`Orientation`(a, b, q)
degree 2



`SideOfBisector`(B_{ab}, q)
degree 2



`InCircle`(a, b, c, q)
degree 4



`OrderOnLine`(B_{ab}, B_{cd}, ℓ)
degree 3

Precision/Robust Techniques

Techniques for implementing geometric algorithms
using finite precision computer arithmetic:

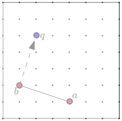
- Rely on machine precision ($+\epsilon$) [NAT90,LTH86,KMP*08]
- Topological Consistency [S99, S01, SI90, SI92, SII*00]
- Exact Geometric Computation [Y97]
  - Software based arithmetic [CORE, LEDA, GMP, MPFR]
  - Predicate eval schemes [ABO*97, FW93, BBP01, S97]
  - Degree-driven algorithm design [LPT99]
    and [BP00,BS00,C00,MS01,MS09,MS10,MV11,MLC*12]

Precision/Robust Techniques

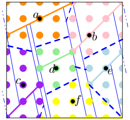
Techniques for implementing geometric algorithms
using finite precision computer arithmetic:

- Rely on machine precision ($+\epsilon$) [NAT90,LTH86,KMP*08]
- Topological Consistency [S99, S01, SI90, SI92, SII*00]
- Exact Geometric Computation [Y97]
  - Software based arithmetic [CORE, LEDA, GMP, MPFR]
  - Predicate eval schemes [ABO*97, FW93, BBP01, S97]
  - Degree-driven algorithm design [LPT99]
    and [BP00,BS00,C00,MS01,MS09,MS10,MV11,MLC*12]

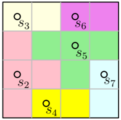
Overview



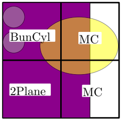
- Derive & upper bdd precision of many common preds
- Show the polys in the common preds are irreducible



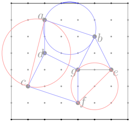
- Compute point location data structure with double & triple precision



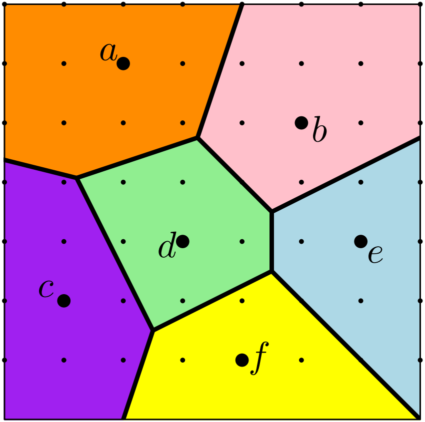
- Compute nearest neighbor transform with double precision



- Compute volumes of CSG models with five-fold precision



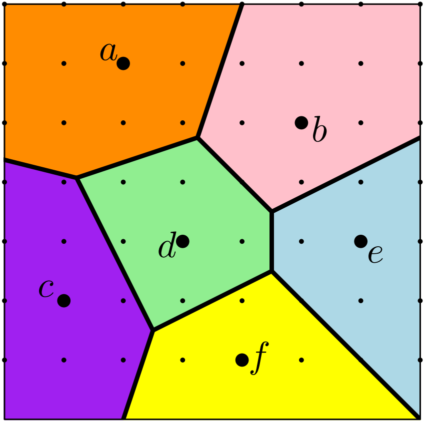
- Compute Gabriel graph with double precision



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 closest $s_i \in S$ to a
query point $q \in \mathbb{U}$ in $O(\log n)$ time

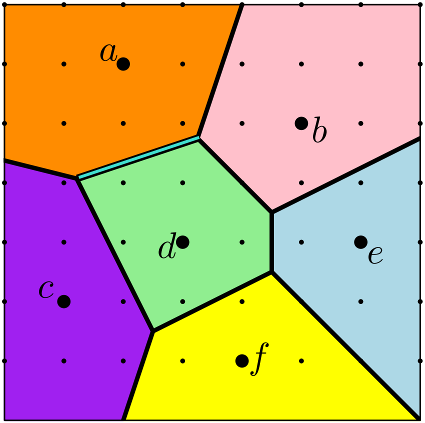
Precision of Voronoi Diagram/Trapezoid Graph



Voronoi diagram

- region

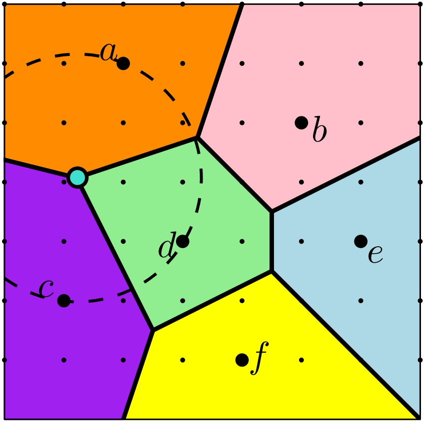
Precision of Voronoi Diagram/Trapezoid Graph



Voronoi diagram

- region
- edge

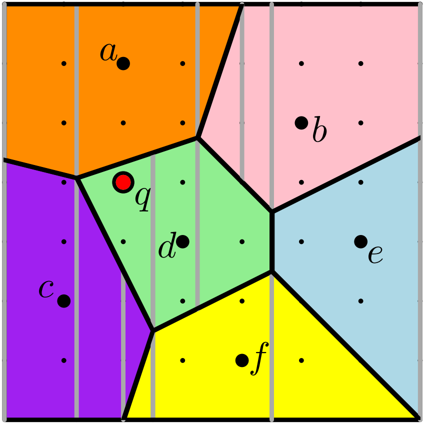
Precision of Voronoi Diagram/Trapezoid Graph



Voronoi diagram

- region
- edge
- vertex

Precision of Voronoi Diagram/Trapezoid Graph

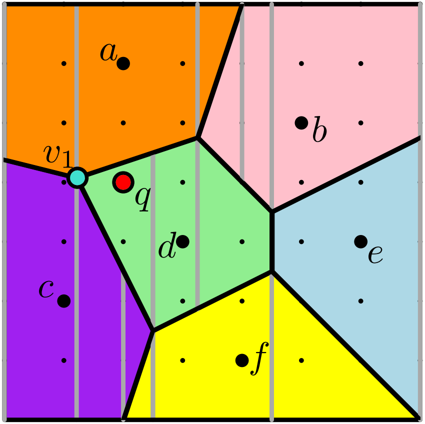


Voronoi diagram

- region
- edge
- vertex

Trapezoid graph for proximity queries

Precision of Voronoi Diagram/Trapezoid Graph



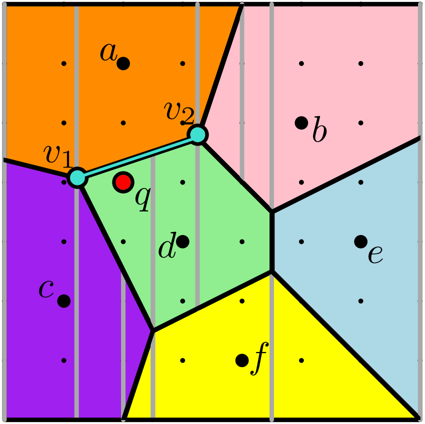
Voronoi diagram

- region
- edge
- vertex

Trapezoid graph for proximity queries

- *x*-`node`() – degree 3

Precision of Voronoi Diagram/Trapezoid Graph



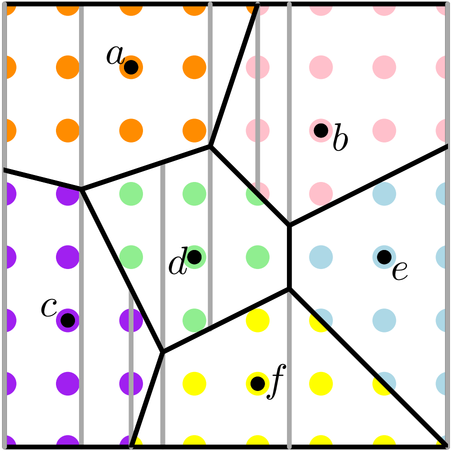
Voronoi diagram

- region
- edge
- vertex

Trapezoid graph for proximity queries

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

Precision of Voronoi Diagram/Trapezoid Graph



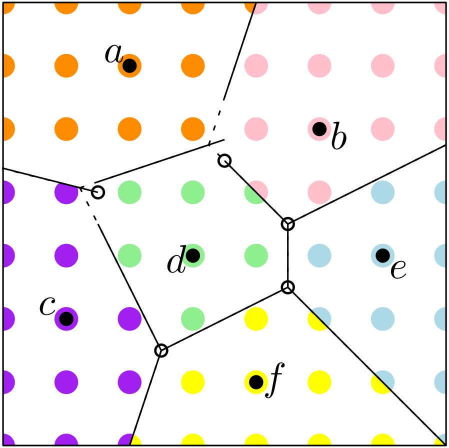
Voronoi diagram

- region
- edge
- vertex

Trapezoid graph for proximity queries

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

Precision of Voronoi Diagram/Trapezoid Graph



Voronoi diagram

- region
- edge
- vertex

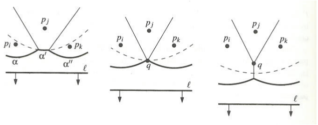
Trapezoid graph for proximity queries [LPT99]

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

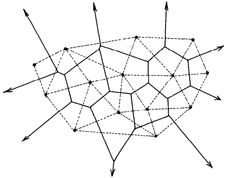
The *Implicit Voronoi diagram* is a degree 2 trapezoid graph.

Precision of Constructing the Voronoi Diagram

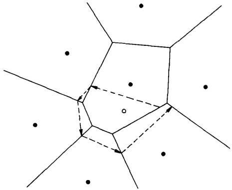
Three well-known Voronoi diagram constructions.



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.

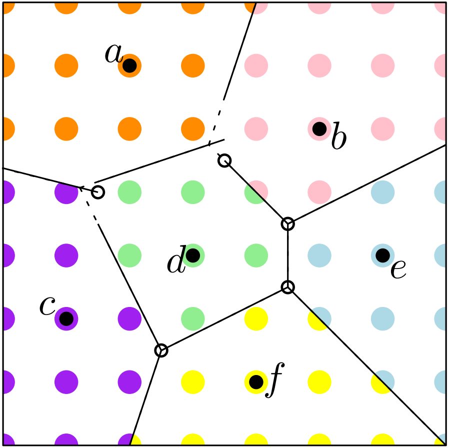
How do we build
a degree 2 trapezoid graph
for proximity queries
when we can't even construct
a Voronoi vertex?

eepline[F87]
degree 6

de and Conquer[GS86]
degree 4

ing[SI92]
degree 4

Implicit Voronoi Diagram [LPT99]



Implicit Voronoi diagram
is disconnected.

Reduced Precision Voronoi [MS09]

Given n sites in $\mathbb{U}$

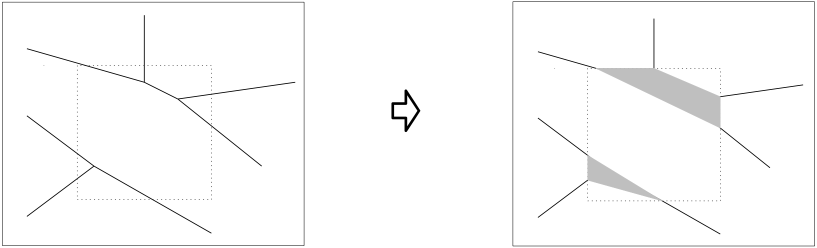
RP-Voronoi randomized incremental construction

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

LPT's Implicit Voronoi constructed from RP-Voronoi

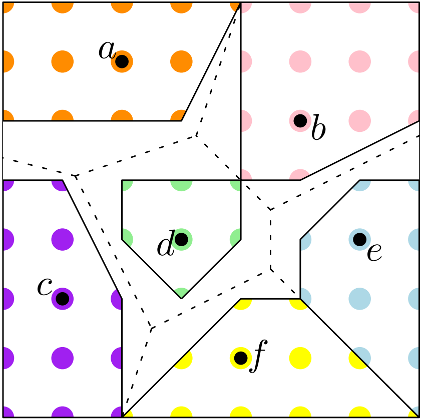
- Time: $O(n)$
- Space: $O(n)$
- Precision: degree 3

Reduced Precision Voronoi [MS09]



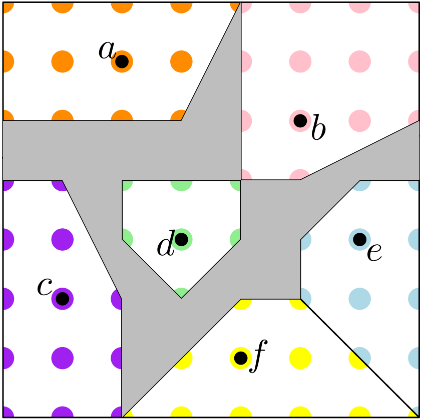
Contract trees of Voronoi vertices that occur in the same grid cell into an *rp-vertex*.

Voronoi Polygon Set



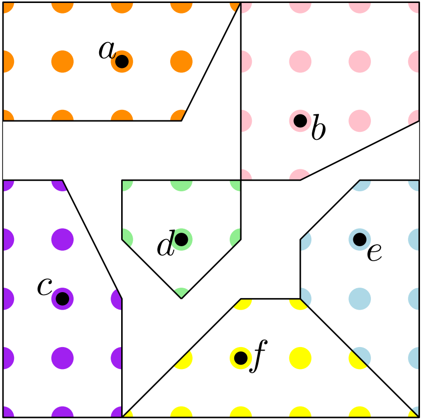
- *Voronoi polygon* is the convex hull of the grid points in a Voronoi cell.

Voronoi Polygon Set



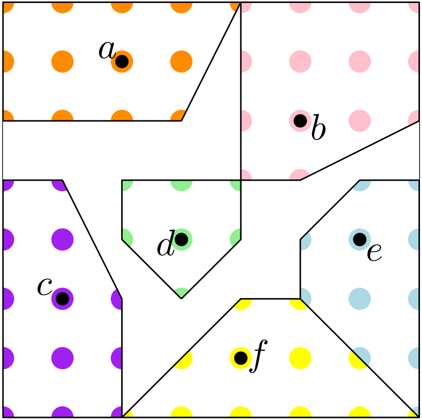
- *Voronoi polygon* is the convex hull of the grid points in a Voronoi cell.
- Gaps

Voronoi Polygon Set



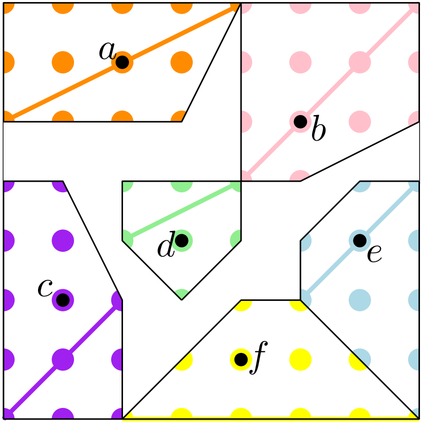
- *Voronoi polygon* is the convex hull of the grid points in a Voronoi cell.
- Gaps
- *Voronoi polygon set* is the collection of the *n* Voronoi polygons.

Voronoi Polygon Set



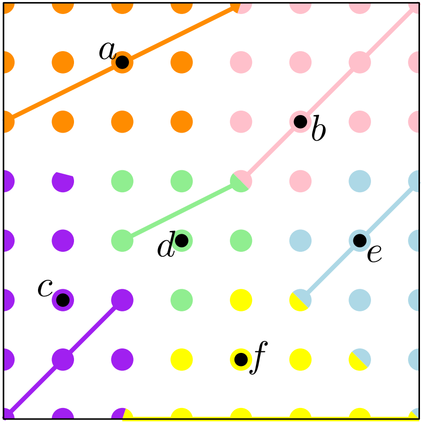
- *Voronoi polygon* is the convex hull of the grid points in a Voronoi cell.
- Gaps
- *Voronoi polygon set* is the collection of the n Voronoi polygons.
- Total size of the Voronoi polygon set is $\Theta(n \log U)$.

Proxy Segments



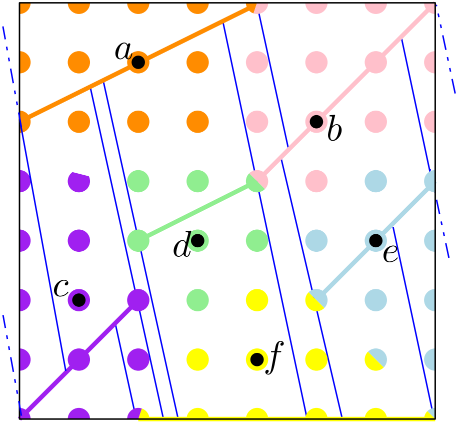
- *Proxy segment* -
  represent Voronoi polygons

Proxy Segments



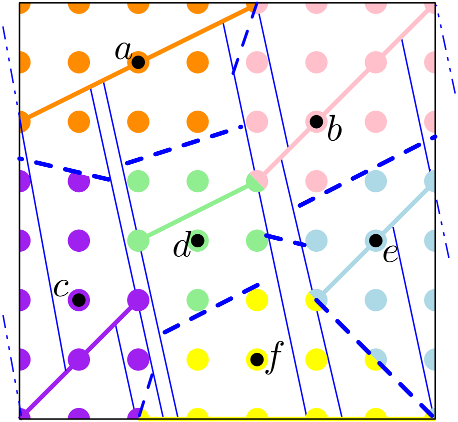
- *Proxy segment* -
  represent Voronoi polygons

Proxy Segments



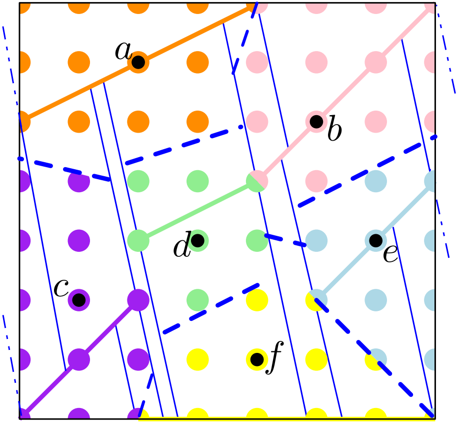
- *Proxy segment* -
  represent Voronoi polygons
- *Proxy trapezoidation* -
  trapezoidation of the proxies

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 supporting $O(\log n)$ time and degree 2 queries.

Point Location[MS09,MS10]

Given n sites in $\mathbb{U}$

RP-Voronoi randomized incremental construction

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

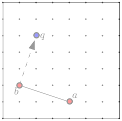
LPT's Implicit Voronoi constructed from RP-Voronoi

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

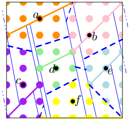
Queries on Proxy Trapezoidation

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

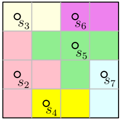
Overview



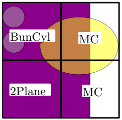
- Derive & upper bdd precision of many common preds
- Show the polys in the common preds are irreducible



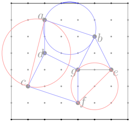
- Compute point location data structure with double & triple precision



- Compute nearest neighbor transform with double precision

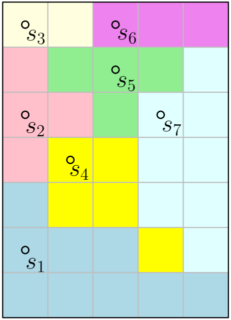


- Compute volumes of CSG models with five-fold precision



- Compute Gabriel graph with double precision

Nearest Neighbor Transform



Given

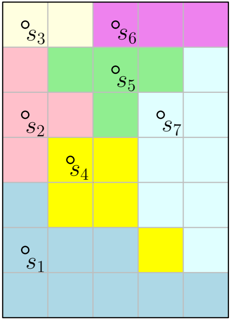
A grid of size U and

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

Algorithm	Precision	Time
Brute Force	degree 2	$O(nU^2)$
Nearest Neighbor Trans. [B90]	degree 5	$O(U^2)$
Discrete Voronoi diagram [C06, MQR03]	degree 3	$O(U^2)$
GPU Hardware [H99]	-	$\Theta(nU^2)$

Problem Transformations: Part 1

Problem (NNTrans-min)

For each pixel $q \in U^2$, find the site with lowest index $s_i \in S$ minimizing $\|q - s_i\|$.

Problem Transformations: Part 1

Problem (NNTrans-min)

For each pixel $q \in U^2$, find the site with lowest index $s_i \in S$ minimizing $\|q - s_i\|$.

$$\|q - s_i\|^2 < \|q - s_j\|^2$$
$$q \cdot q - 2q \cdot s_i + s_i \cdot s_i < q \cdot q - 2q \cdot s_j + s_j \cdot s_j$$
$$2x_i x_q + 2y_i y_q - x_i^2 - y_i^2 > 2x_j x_q + 2y_j y_q - x_j^2 - y_j^2.$$

Problem Transformations: Part 1

Problem (NNTrans-min)

For each pixel $q \in U^2$, find the site with lowest index $s_i \in S$ minimizing $\|q - s_i\|$.

$$\|q - s_i\|^2 < \|q - s_j\|^2$$
$$q \cdot q - 2q \cdot s_i + s_i \cdot s_i < q \cdot q - 2q \cdot s_j + s_j \cdot s_j$$
$$2x_i x_q + 2y_i y_q - x_i^2 - y_i^2 > 2x_j x_q + 2y_j y_q - x_j^2 - y_j^2.$$

Problem (NNTrans-max)

For each pixel q, find the site with lowest index $s_i \in S$ maximizing $2x_i x_q + 2y_i y_q - x_i^2 - y_i^2$.

Problem Transformations: Part 2

Problem (NNTrans-max)

For each pixel q, find the site with lowest index $s_i \in S$ *maximizing* $2x_ix_q + 2y_iy_q - x_i^2 - y_i^2$.

Problem Transformations: Part 2

Problem (NNTrans-max)

For each pixel q, find the site with lowest index $s_i \in S$ maximizing $2x_ix_q + 2y_iy_q - x_i^2 - y_i^2$.

For a fixed, $y_q = Y$

$$2x_ix_q + 2y_iy_q - x_i^2 - y_i^2 > 2x_jx_q + 2y_jy_q - x_j^2 - y_j^2$$
$$2x_ix_q + (2y_iY - x_i^2 - y_i^2) > 2x_jx_q + (2y_jY - x_j^2 - y_j^2)$$
$$①x_q + ② > ①x_q + ②$$

Problem Transformations: Part 2

Problem (NNTrans-max)

For each pixel q, find the site with lowest index $s_i \in S$ maximizing $2x_ix_q + 2y_iy_q - x_i^2 - y_i^2$.

For a fixed, $y_q = Y$

$$2x_ix_q + 2y_iy_q - x_i^2 - y_i^2 > 2x_jx_q + 2y_jy_q - x_j^2 - y_j^2$$
$$2x_ix_q + (2y_iY - x_i^2 - y_i^2) > 2x_jx_q + (2y_jY - x_j^2 - y_j^2)$$
$$① x_q + ② > ① x_q + ②$$

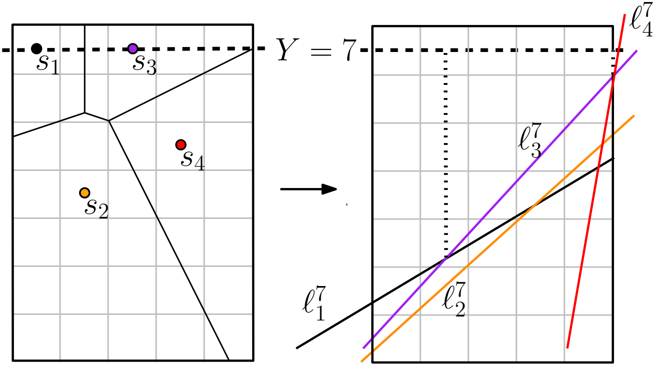
Problem (DUE-Y)

For a fixed $1 \leq Y \leq U$, and for each $1 \leq X \leq U$, find the smallest index of a line of L_Y with maximum y coordinate at $x = X$.

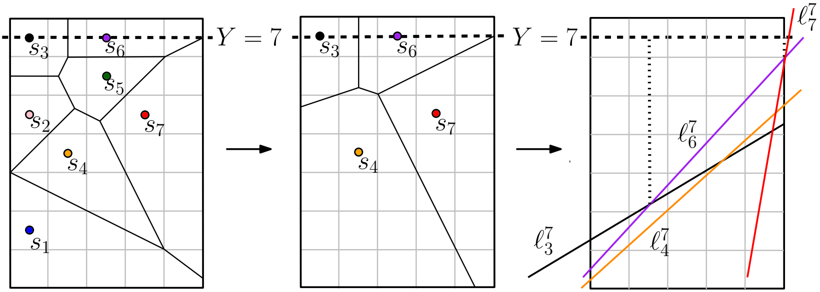
Problem Transformations: Part 2

Problem (DUE-Y)

For a fixed $1 \leq Y \leq U$, and for each $1 \leq X \leq U$, find the smallest index of a line of L_Y with maximum y coordinate at $x = X$.



Sketch of NNTransform Algorithm



Three Algorithms for Computing the DUE [MLCS12]

Given m lines of the form $y = ①x + ②$

Discrete Upper Envelope construction

- `DUE-DEG3`: $O(m + U)$ time and degree 3
- `DUE-ULgU`: $O(m + U \log U)$ time and degree 2
- `DUE-U`: $O(m + U)$ expected time and degree 2

Three Algorithms for Computing the DUE [MLCS12]

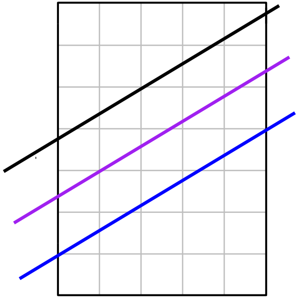
Given m lines of the form $y = ①x + ②$

Discrete Upper Envelope construction

- `DUE-DEG3`: $O(m+U)$ time and degree 3
- `DUE-ULgU`: $O(m+U\log U)$ time and degree 2
- `DUE-U`: $O(m+U)$ expected time and degree 2

For each algorithm:

1. Reduce to at most $O(U)$ lines.
2. Compute DUE of lines.



Three Algs for Computing the NNTransform [MLCS12]

Given n sites from $\mathbb{U}$

Nearest Neighbor Transform construction

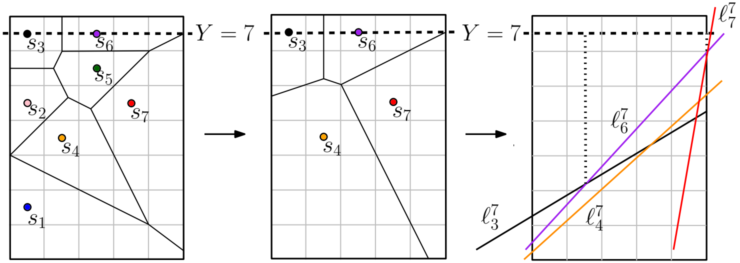
- `Deg3`: $O(U^2)$ time and degree 3
- `UsqLgU`: $O(U^2 \log U)$ time and degree 2
- `Usq`: $O(U^2)$ expected time and degree 2

Three Algs for Computing the NNTransform [MLCS12]

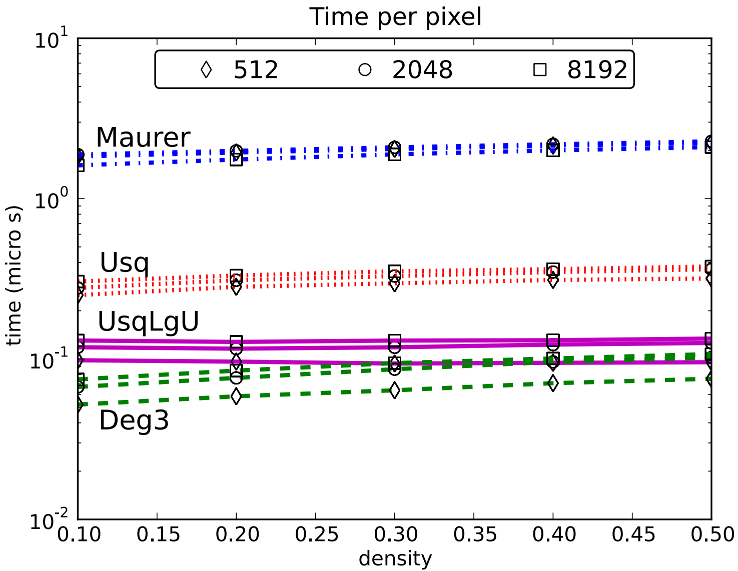
Given n sites from $\mathbb{U}$

Nearest Neighbor Transform construction

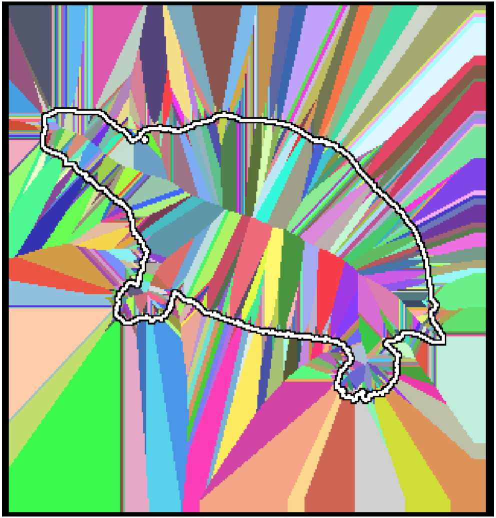
- `Deg3`: $O(U^2)$ time and degree 3
- `UsqLgU`: $O(U^2 \log U)$ time and degree 2
- `Usq`: $O(U^2)$ expected time and degree 2



Experiments: Part 1

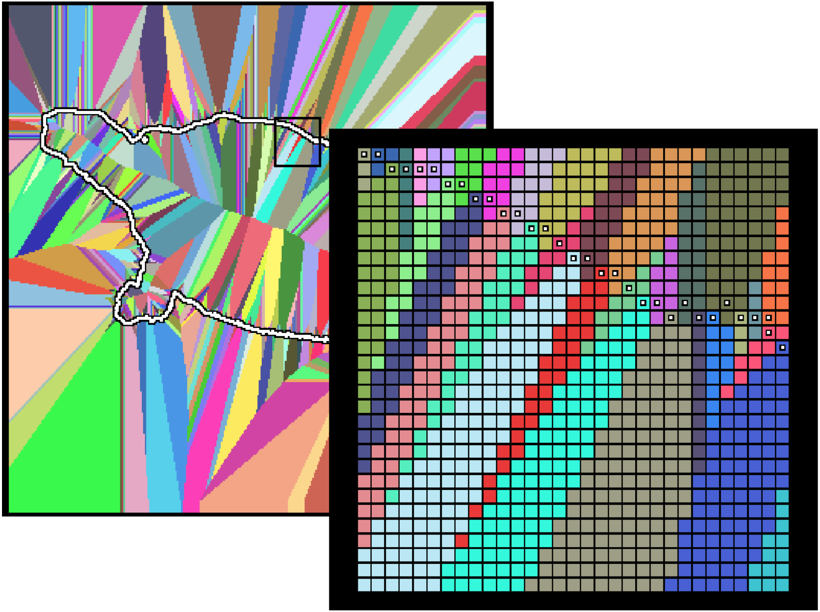


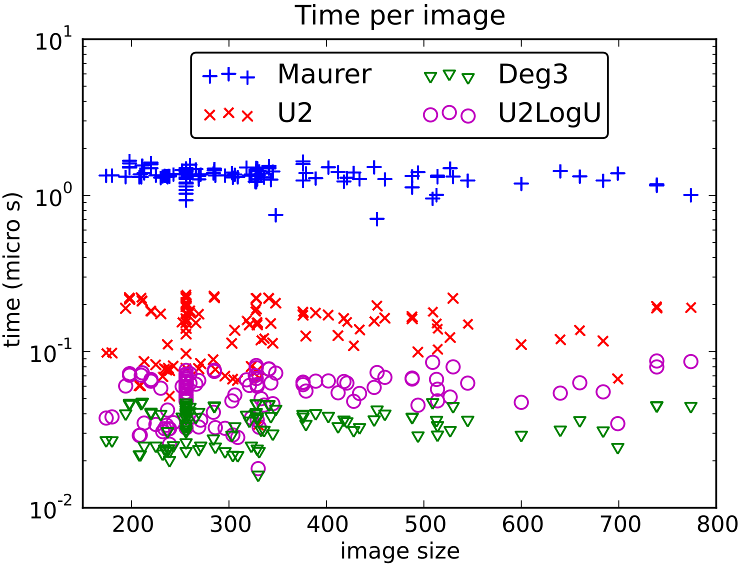
Experiments: Part 2



Boundaries extracted from 120 images of the MPEG 7 CE Shape-1 Part B data set.

Experiments: Part 2




Time per image
Maurer
Deg3
U2
U2LogU
time (micro s)
10^{1}
10^{0}
10^{-1}
10^{-2}
200
300
400
500
600
700
800
image size

NNTransform [MLCS12]

Given m lines of the form $y = ① x + ②$

Discrete Upper Envelope construction

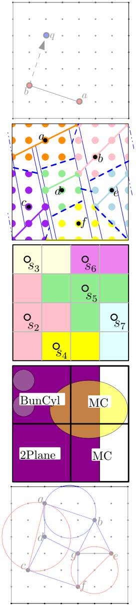
- `DUE-DEG3`: $O(m + U)$ time and degree 3
- `DUE-ULgU`: $O(m + U \log U)$ time and degree 2
- `DUE-U`:$O(m + U)$ expected time and degree 2

Given n sites from $\mathbb{U}$

Nearest Neighbor Transform construction

- `Deg3`: $O(U^2)$ time and degree 3
- `UsqLgU`: $O(U^2 \log U)$ time and degree 2
- `Usq`: $O(U^2)$ expected time and degree 2

Overview



- Derive & upper bdd precision of many common preds
- Show the polys in the common preds are irreducible
- Compute point location data structure with double & triple precision
- Compute nearest neighbor transform with double precision
- Compute volumes of CSG models with five-fold precision
- Compute Gabriel graph with double precision

Motivation and Background

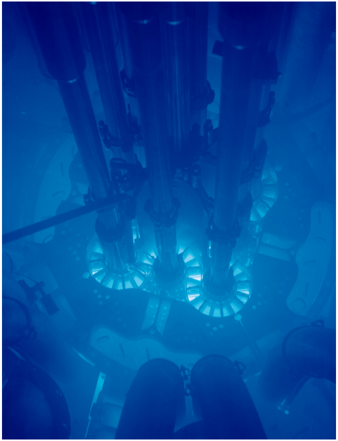


Image from Idaho National Lab, Flickr

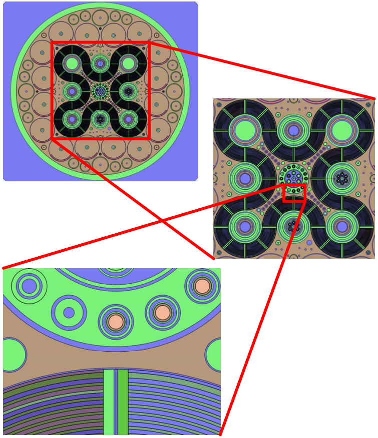
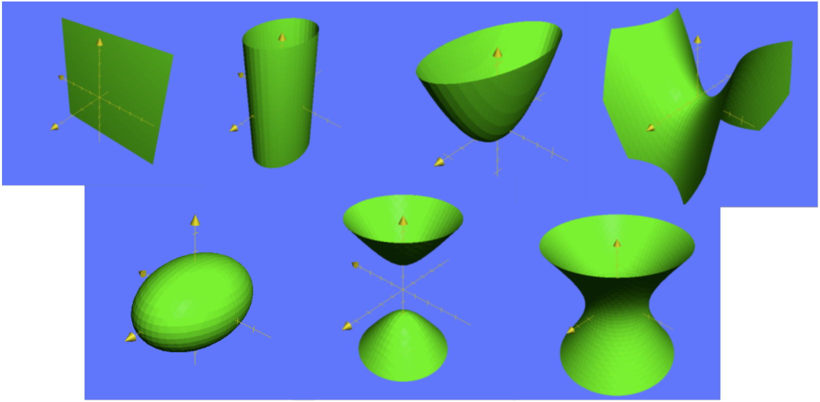


Image from: T.M. Sutton, et al.,
The MC21 Monte Carlo Transport Code,
Proceedings of M&C + SNA 2007

Primitives: Signed Quadratic Surfaces

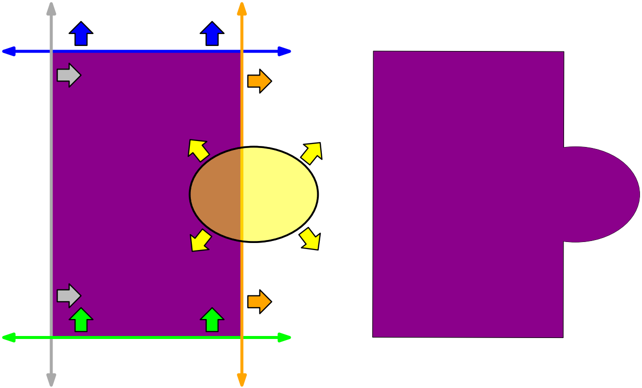


$$f(x, y, z) < A_1 x^2 + A_2 y^2 + A_3 z^2 + A_4 xy + A_5 xz + A_6 yz + A_7 x + A_8 y + A_9 z + A_{10}$$

Model Representation
Basic Component: Boolean Formula

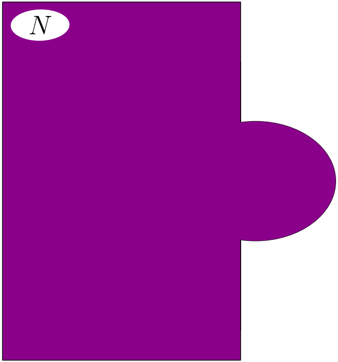
A *basic component* defined by
intersections and unions of signed surfaces.

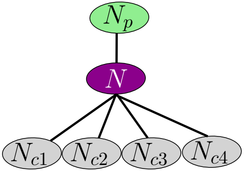
$$(-s_{blue} \cap s_{grey} \cap s_{green} \cap -s_{orange}) \cup -s_{yellow}$$



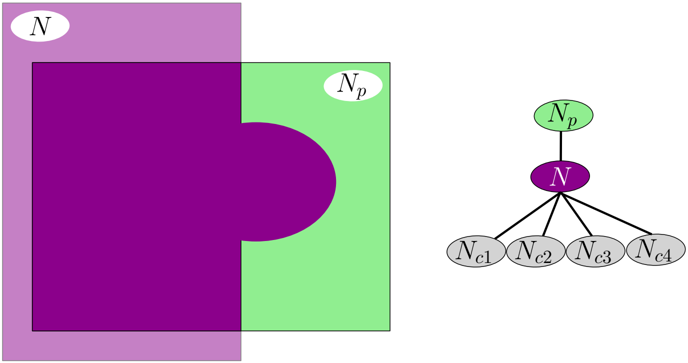
Model Representation
Component Hierarchy: Boolean Formulae

Basic comp: $B(N)$, ∪ and ∩ of signed surfaces



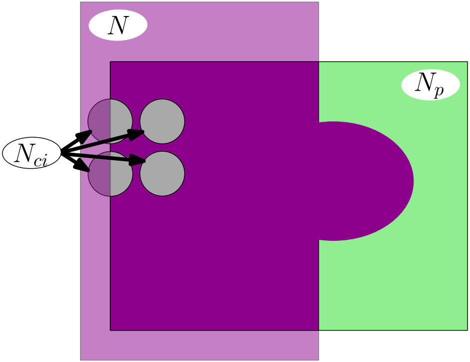


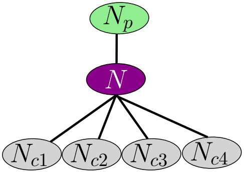
Basic comp: $B(N)$, $\cup$ and $\cap$ of signed surfaces
Restricted comp: $R(N) = B(N) \cap R(N_p)$



Model Representation
Component Hierarchy: Boolean Formulae

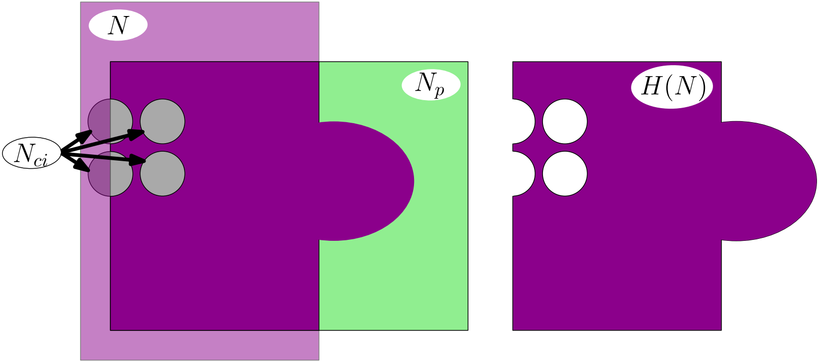
Basic comp: $B(N)$, $\cup$ and $\cap$ of signed surfaces
Restricted comp: $R(N) = B(N) \cap R(N_p)$
Hierarchical comp: $H(N) = R(N) \setminus (\bigcup_i R(N_{ci}))$

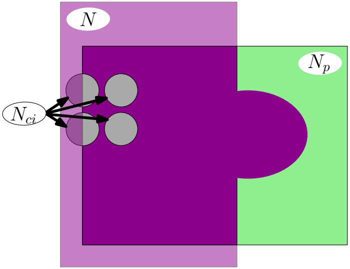


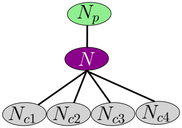


Model Representation
Component Hierarchy: Boolean Formulae

Basic comp: $B(N)$, $\cup$ and $\cap$ of signed surfaces
Restricted comp: $R(N) = B(N) \cap R(N_p)$
Hierarchical comp: $H(N) = R(N) \setminus (\bigcup_i R(N_{ci}))$







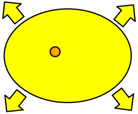
Given
A component hierarchy and an accuracy

Compute
The volume of each hierarchical component to accuracy

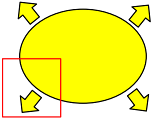
Operations on Surfaces

Operations on signed surfaces *s* with a query point *q* or an axis-aligned box *b*:

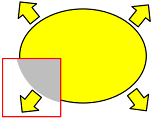
- `Inside`(s, q) – return if *q* is inside *s*.



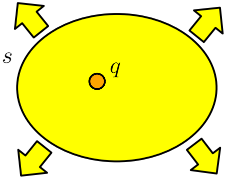
- `Classify`(s, b) – return if the points in *b* are inside, outside or both with respect to *s*.



- `Integrate`(s, b) – return the volume of $s \cap b$.
.



Inside(s, q) – return if query point q is inside signed surface s.

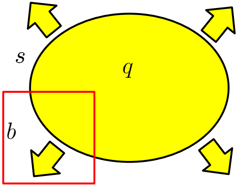


$$q = (q_1, q_2, q_3)$$
$$s = (s_1, s_2, \ldots, s_{10})$$
$$p_i, s_i \in \{-U, \ldots, U\}$$

$$\begin{aligned}\texttt{PointInside}(s, q) &= s_1 q_1^2 + s_2 q_2^2 + s_3 q_3^2 \\ &+ s_4 q_1 q_2 + s_5 q_1 q_3 + s_6 q_2 q_3 \\ &+ s_7 q_1 + s_8 q_2 + s_9 q_3 + s_{10} \\ &= \mathrm{sign}(③)\end{aligned}$$

`Classify` Test

`Classify`(s, q) – return if the points in axis-aligned box b are inside, outside or both with respect to signed surface s.



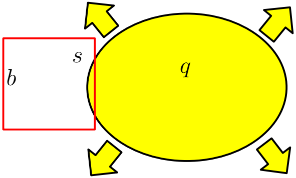
$$b = (b_1, b_2, \ldots, b_6)$$
$$s = (s_1, s_2, \ldots, s_{10})$$
$$b_i, s_i \in \{-U, \ldots, U\}$$

`Classify`(s, b), check if:

1. any `Vertices` of b are on different sides of s. – Degree 3
2. any `Edge` of b intersects s. – Degree 4
3. any `Face` b intersects s. – Degree 5

`Classify` Test

`Classify`(s, q) – return if the points in axis-aligned box b are inside, outside or both with respect to signed surface s.

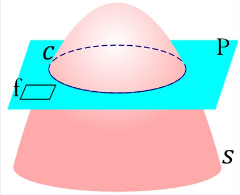


$$b = (b_1, b_2, \ldots, b_6)$$
$$s = (s_1, s_2, \ldots, s_{10})$$
$$b_i, s_i \in \{-U, \ldots, U\}$$

`Classify`(s, b), check if:

1. any `Vertices` of b are on different sides of s. – Degree 3
2. any `Edge` of b intersects s. – Degree 4
3. any `Face` b intersects s. – Degree 5

Test if a face f intersects s.

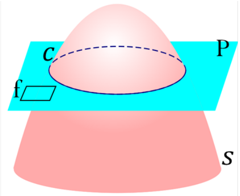


Let c be the intersection curve of the plane P containing f and s.

$$c(x,y) = \begin{pmatrix} x & y & 1 \end{pmatrix} \begin{pmatrix} ① & ① & ② \\ ① & ① & ② \\ ② & ② & ③ \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Face Test

Test if a face f intersects s.



Let c be the intersection curve of the plane P containing f and s.

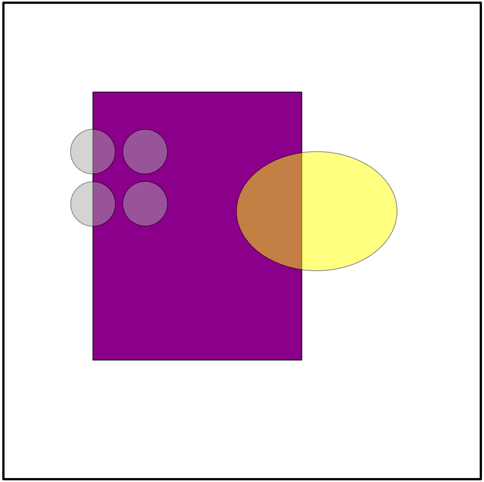
$$c(x,y) = \begin{pmatrix} x & y & 1 \end{pmatrix} \begin{pmatrix} \textcircled{1} & \textcircled{1} & \textcircled{2} \\ \textcircled{1} & \textcircled{1} & \textcircled{2} \\ \textcircled{2} & \textcircled{2} & \textcircled{3} \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

To determine if s intersects f, test properties of the matrix.

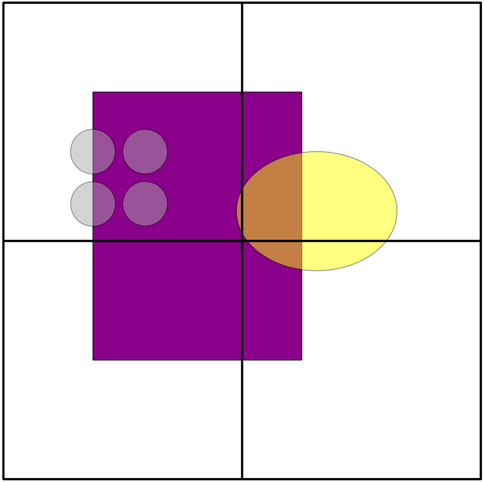
Test if c is an ellipse: $$\operatorname{sign}\left(\begin{vmatrix} \textcircled{1} & \textcircled{1} \\ \textcircled{1} & \textcircled{1} \end{vmatrix} \right) = \operatorname{sign}(\textcircled{2})$$

Test if c is real or img: $$\operatorname{sign}\left(\begin{vmatrix} \textcircled{1} & \textcircled{1} & \textcircled{2} \\ \textcircled{1} & \textcircled{1} & \textcircled{2} \\ \textcircled{2} & \textcircled{2} & \textcircled{3} \end{vmatrix} \right) = \operatorname{sign}(\textcircled{5})$$

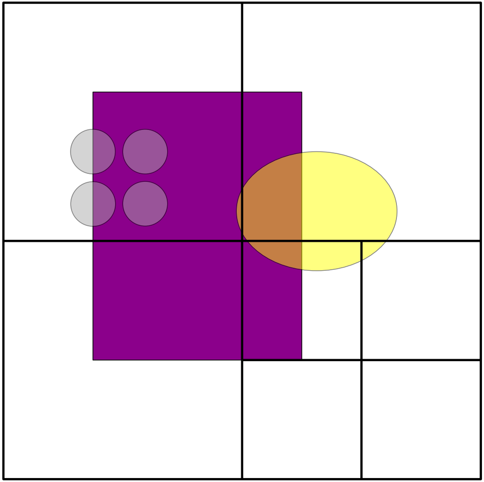
Algorithm Animation



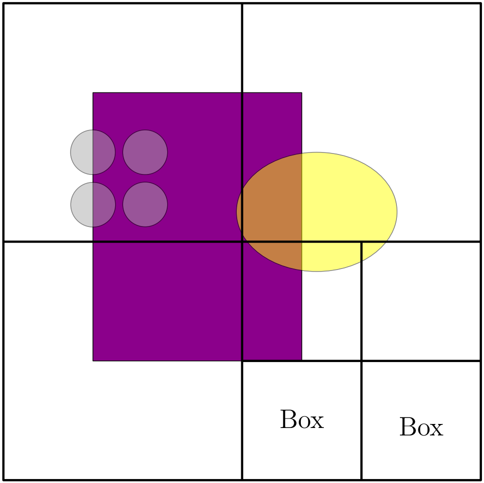
Algorithm Animation



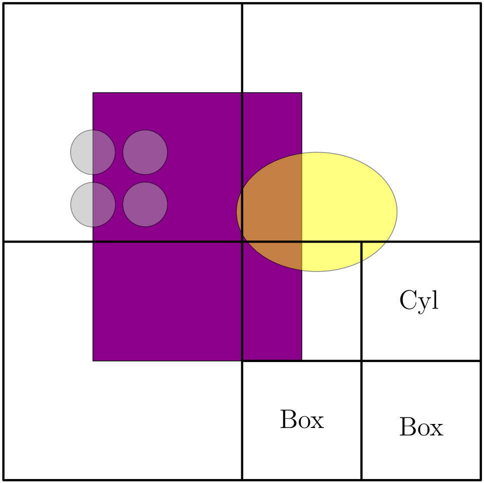
Algorithm Animation



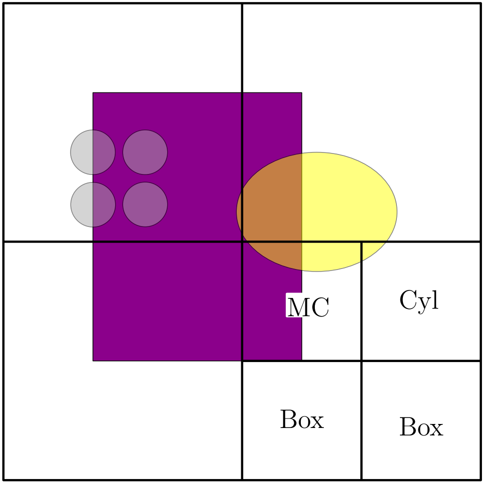
Algorithm Animation



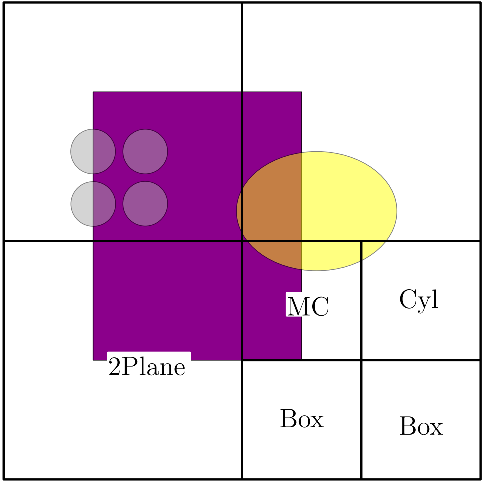
Algorithm Animation



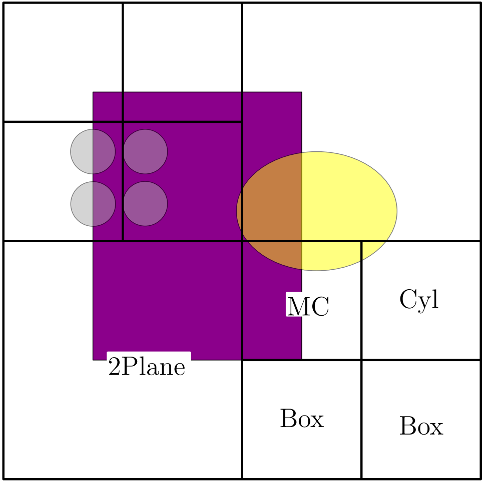
Algorithm Animation



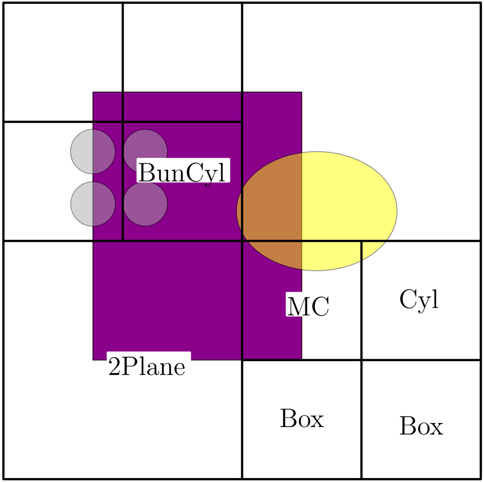
Algorithm Animation



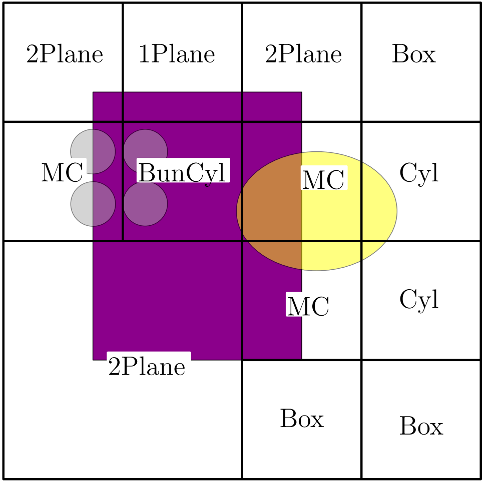
Algorithm Animation



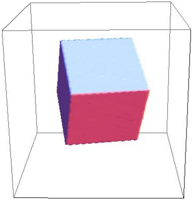
Algorithm Animation



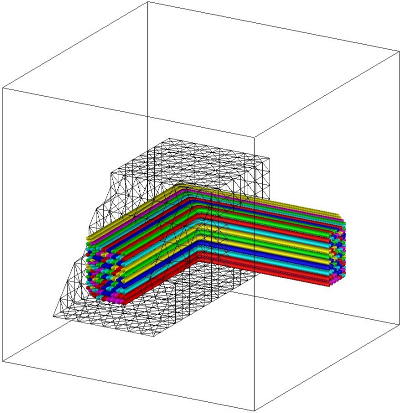
Algorithm Animation



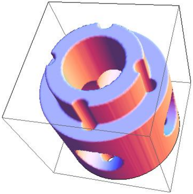
Experiments: Models



Cube

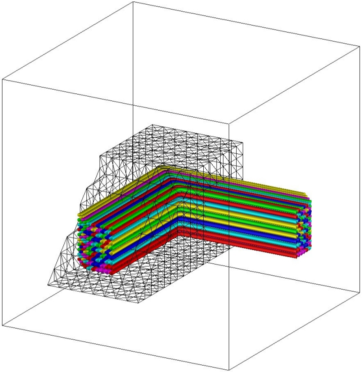


L-Pipe12,
L-Pipe100, and L-Pipe10k



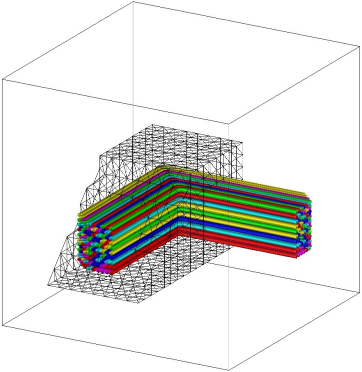
DrillCyl

Experiments: Accuracy and Time for L-Pipe100



Algorithm	Requested Accuracy	Error	Time (sec)
MC	1e-4	$<$1e-4	790.28
New	1e-4	$<$1e-6	1.41

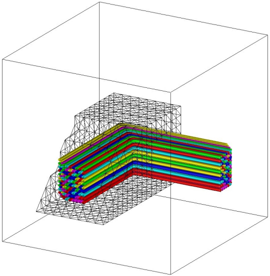
Experiments: Accuracy and Time for L-Pipe100



Algorithm	Requested Accuracy	Error	Time (sec)
MC	1e-4	$<$1e-4	790.28
New	1e-4	$<$1e-6	1.41

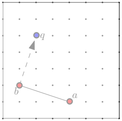
Experiments: Larger Model for L-Pipe10k

L-Pipe10k is similar to L-Pipe100 but defined by over 40k surfaces.

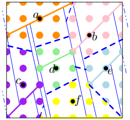


Algorithm	Requested Accuracy	Error	Time
MC	1e-4	-	$>$ 12.00h*
New	1e-4	$<$1e-6	9.43s

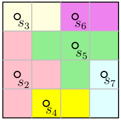
*Halted after 12 hours. Extrapolating from other experiments, 76 hours.



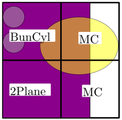
- Derive & upper bdd precision of many common preds
- Show the polys in the common preds are irreducible



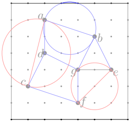
- Compute point location data structure with double & triple precision



- Compute nearest neighbor transform with double precision



- Compute volumes of CSG models with five-fold precision



- Compute Gabriel graph with double precision

Acknowledgements

Adviser: Jack Snoeyink

Committee: David Griesheimer, Ming Lin, Dinesh Manocha, and Chee Yap

Collaborators and research group

UNC CS tech and admin support

Funding: DOE, Bettis, NSF, and Google

Acknowledgements

Adviser: Jack Snoeyink

Committee: David Griesheimer, Ming Lin, Dinesh Manocha, and Chee Yap

Collaborators and research group

UNC CS tech and admin support

Funding: DOE, Bettis, NSF, and Google

My fiancée: Brittany Fasy

Acknowledgements

Adviser: Jack Snoeyink

Committee: David Griesheimer, Ming Lin, Dinesh Manocha, and Chee Yap

Collaborators and research group

UNC CS tech and admin support

Funding: DOE, Bettis, NSF, and Google

My fiancée: Brittany Fasy

My friends and family

Acknowledgements

Adviser: Jack Snoeyink

Committee: David Griesheimer, Ming Lin, Dinesh Manocha, and Chee Yap

Collaborators and research group

UNC CS tech and admin support

Funding: DOE, Bettis, NSF, and Google

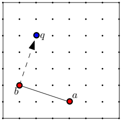
My fiancée: Brittany Fasy

My friends and family

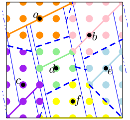
Everyone here today!!

THANK YOU!!!

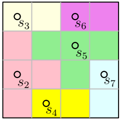
Contributions



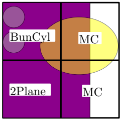
- Derive & upper bdd precision of many common preds
- Show the polys in the common preds are irreducible



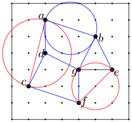
- Compute point location data structure with double & triple precision



- Compute nearest neighbor transform with double precision



- Compute volumes of CSG models with five-fold precision



- Compute Gabriel graph with double precision

Contact

David L. Millman
dave@cs.unc.edu
http://cs.unc.edu/~dave