Robustness

Outline

1. Introduction
   - (Simplified) history
   - Robustness: Two main issues
2. Arithmetic issues
   - Reminder: floating-point arithmetic
   - Consequences
   - Exact Geometric Computing
3. Degenerate cases

Introduction

Introduction — (Simplified) history

Early algorithms [70's]

- Incremental
- Gift wrapping

non-optimal
"simple"
actually coded
motivated by applications (meshes, reconstruction)

Optimal algorithms [80's]

- Divide & conquer
- Plane sweep

worst-case optimal

~~simple~~

~~actually coded~~

~~motivated by applications (meshes, reconstruction)~~

computational geometry known as a purely theoretical field

A more pragmatic point of view

Randomized algorithms [90's]

- Delaunay tree
- Conflict graph
- History graph
- Delaunay hierarchy
- Spatial sorting (BRIO)

non-optimal
"simple"

A more pragmatic point of view

Randomized algorithms [90's]

- Delaunay tree
- Conflict graph
- History graph
- Delaunay hierarchy
- Spatial sorting (BRIO)

non-optimal
"simple"

Software development

- VRONI
- TRIANGLE
- LEDA
- CGAL

A more pragmatic point of view

Randomized algorithms [90's]

- Delaunay tree
- Conflict graph
- History graph
- Delaunay hierarchy
- Spatial sorting (BRIO)

non-optimal
"simple"

Software development $\longrightarrow$ Robustness issues [95's]

- VRONI
- TRIANGLE
- LEDA
- CGAL

Introduction — Robustness: Two main issues

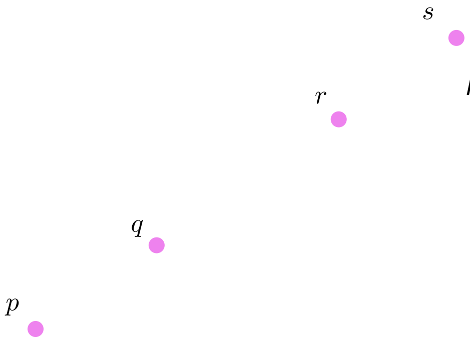
- Arithmetic issues
- Degenerate cases

Arithmetic issues

- Algorithms designed in Real RAM model
- `float, double` are not reals

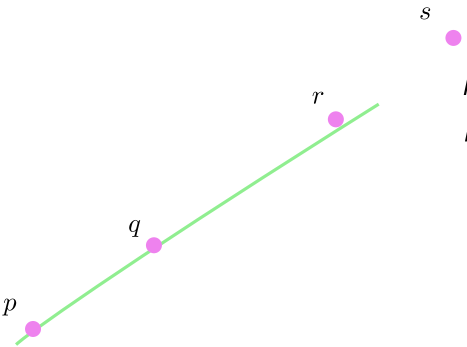
Rounding errors

Arithmetic issues



$p <_x q <_x r <_x s$

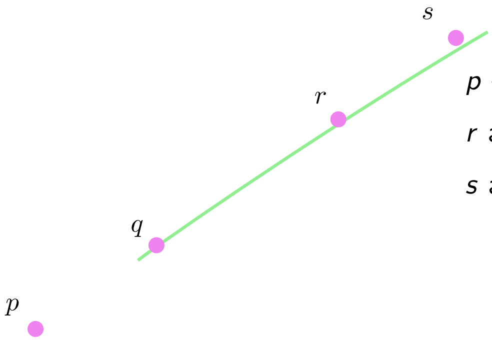
Arithmetic issues



$p <_x q <_x r <_x s$

r above (pq)

Arithmetic issues

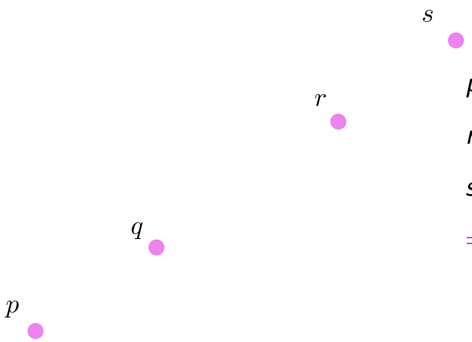


$p <_x q <_x r <_x s$

r above (pq)

s above (qr)

Arithmetic issues



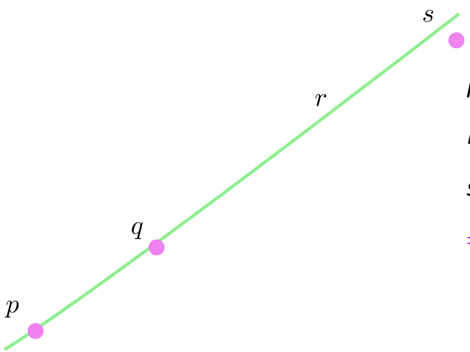
$p <_x q <_x r <_x s$

r above (pq)

s above (qr)

$\Longrightarrow$ s above (pq)

Arithmetic issues



$p <_x q <_x r <_x s$

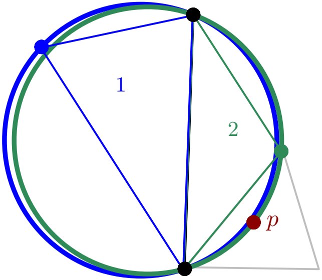
r above (pq)

s above (qr)

$\Longrightarrow s$ above (pq)

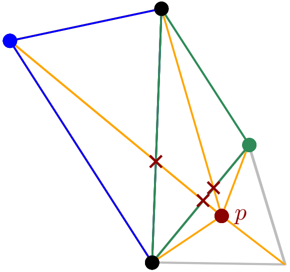
$\longrightarrow$ inconsistency

Arithmetic issues



$\text{in\_disk}_1(p) = \text{true}$
$\text{in\_disk}_2(p) = \text{false}$

Arithmetic issues



$\text{in\_disk}_1(p) = \text{true}$
$\text{in\_disk}_2(p) = \text{false}$

inconsistencies!
algorithm fails

Arithmetic issues

- Algorithms designed in Real RAM model
- `float, double` are not reals

Rounding errors

$\Longrightarrow$ inconsistencies

not just imprecisions

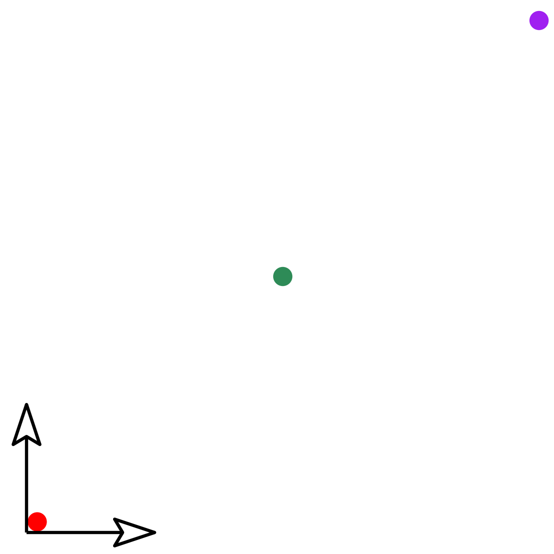
$\Longrightarrow$ failure

see course convex hull

Convex hull

Orientation predicate

Rounding errors possible



$p = (\frac{1}{2} + x.u, \frac{1}{2} + y.u)$

$0 \leq x, y \leq 256, \; u = 2^{-53}$

$q = (12, 12)$

$r = (24, 24)$

Teaser robustness lecture

$orientation(p, q, r)$

evaluated with double

Convex hull

Orientation predicate

Rounding errors possible

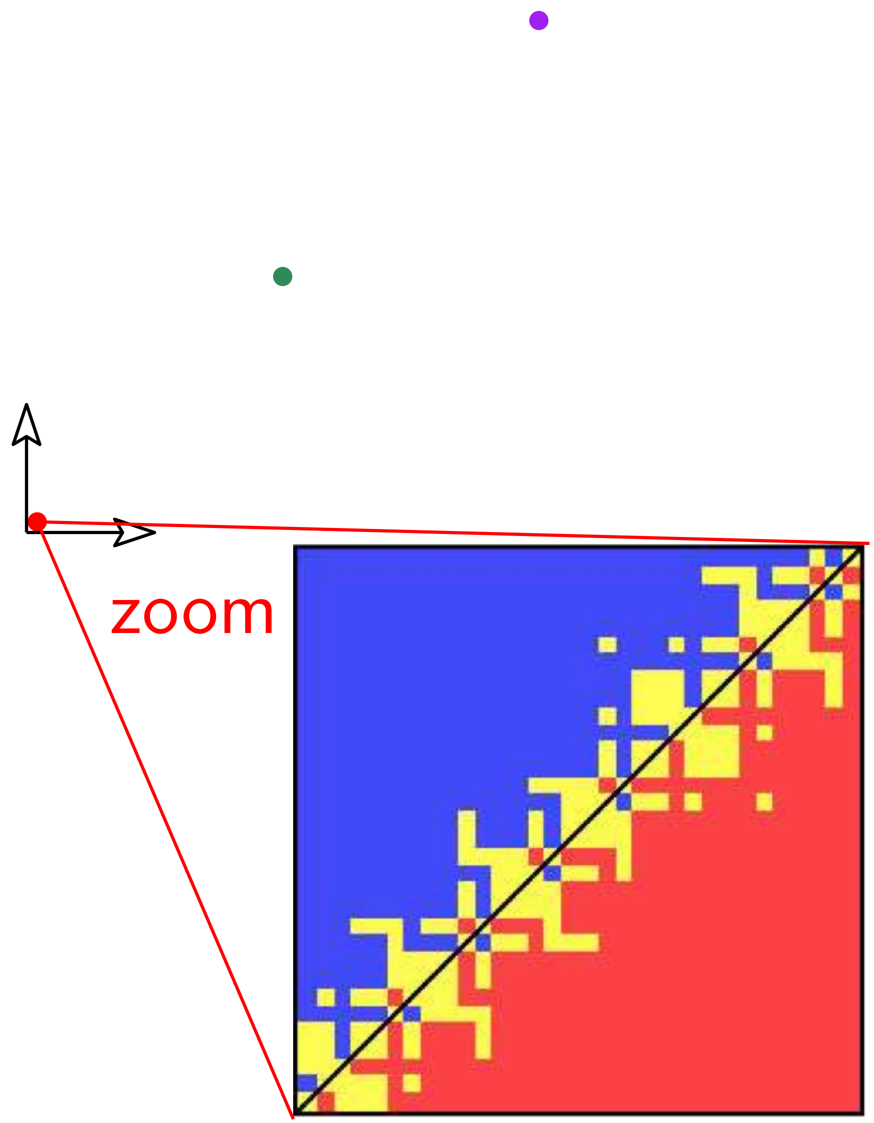
$p = (\frac{1}{2} + x.u, \frac{1}{2} + y.u)$

$0 \leq x, y \leq 256, \; u = 2^{-53}$

$q = (12, 12)$

$r = (24, 24)$

Teaser robustness lecture



$orientation(p, q, r)$

evaluated with double

≤ 0

0

≥ 0

Convex hull

Teaser robustness lecture

Buggy degenerate example
(single precision)

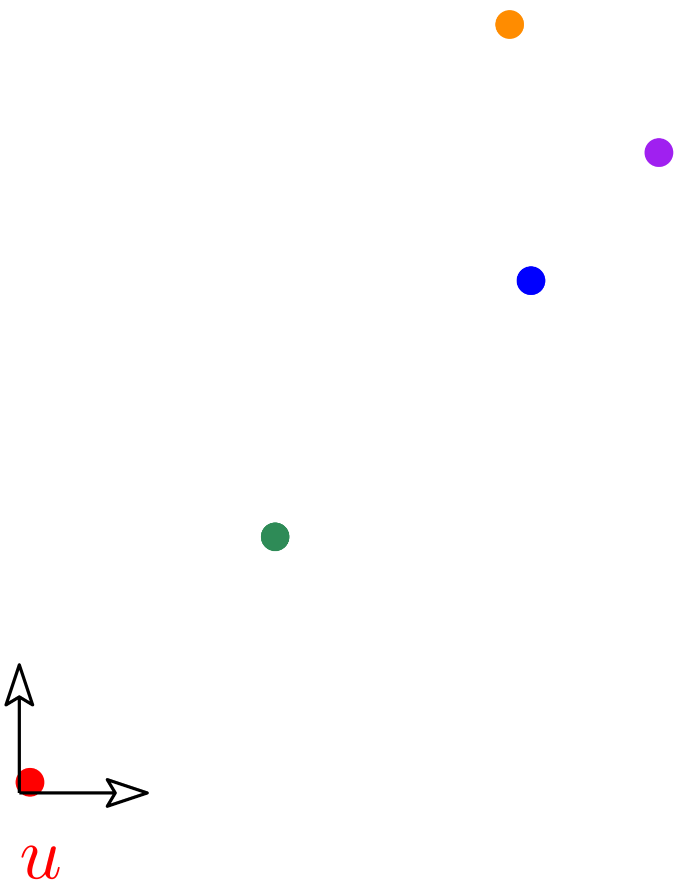
$w_1 = (12, 12)$

$w_2 = (24, 24)$

$w_3 = (30, 30.000001)$

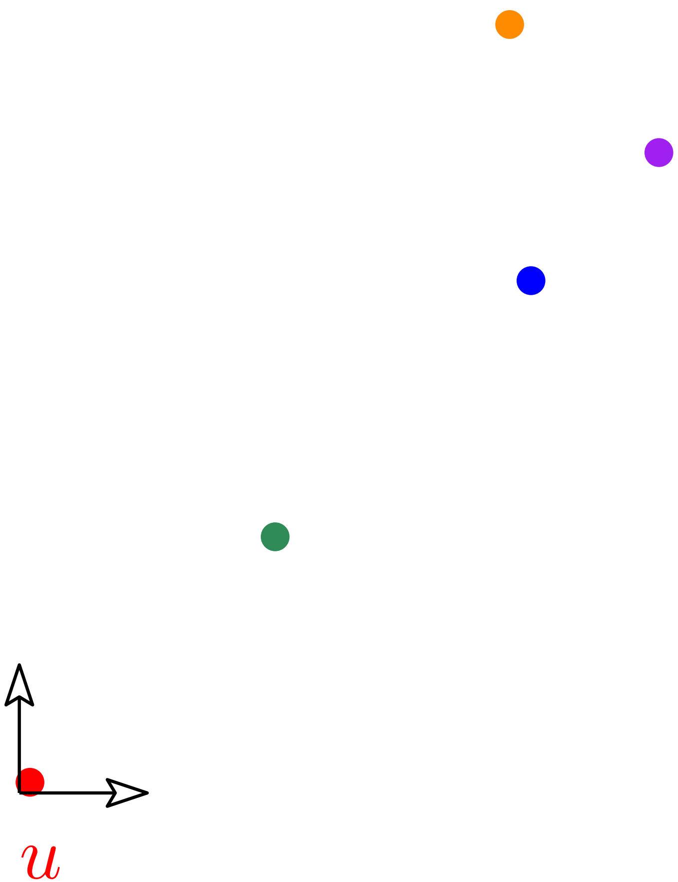
$w_4 = (23, 36)$

$w_5 = (0.5000029, 0.5000027)$



Convex hull

Teaser robustness lecture



Buggy degenerate example
(single precision)

$w_1 = (12, 12)$

$w_2 = (24, 24)$

$w_3 = (30, 30.000001)$

$w_4 = (23, 36)$

$w_5 = (0.5000029, 0.5000027)$

Input : point set S
$u = v =$ lowest point in S;

Jarvis

Convex hull

Teaser robustness lecture

Buggy degenerate example
(single precision)

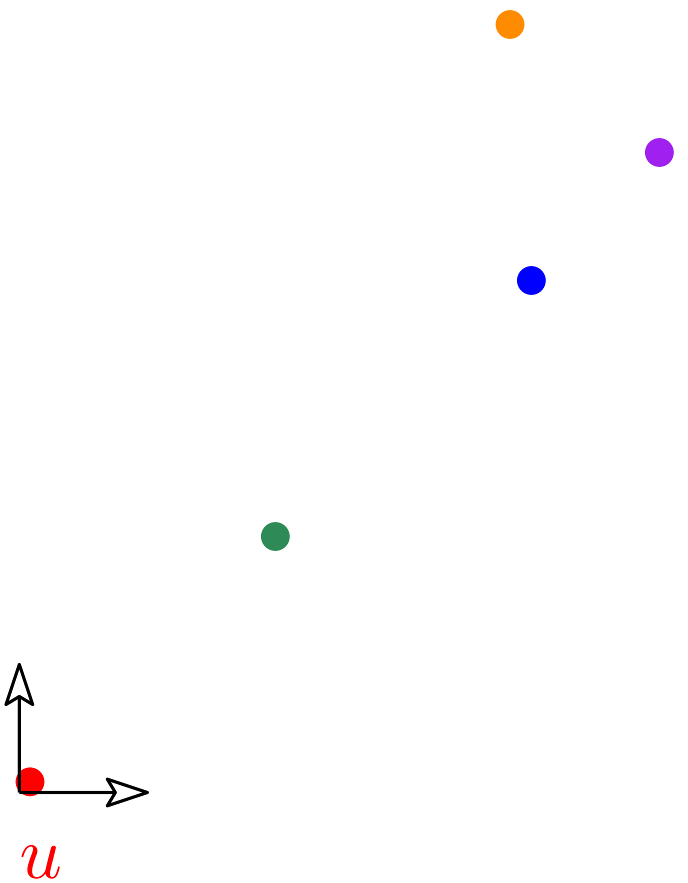
$w_1 = (12, 12)$

$w_2 = (24, 24)$

$w_3 = (30, 30.000001)$

$w_4 = (23, 36)$

$w_5 = (0.5000029, 0.5000027)$



Do
  $n =$ first in S;
  For each $w \in S$
    if vwn positive
     then $n = w$;
  $v.next = n$; $v = n$;
  $S = S \setminus \{v\}$
While $v \neq u$

Convex hull

Teaser robustness lecture

Buggy degenerate example
(single precision)

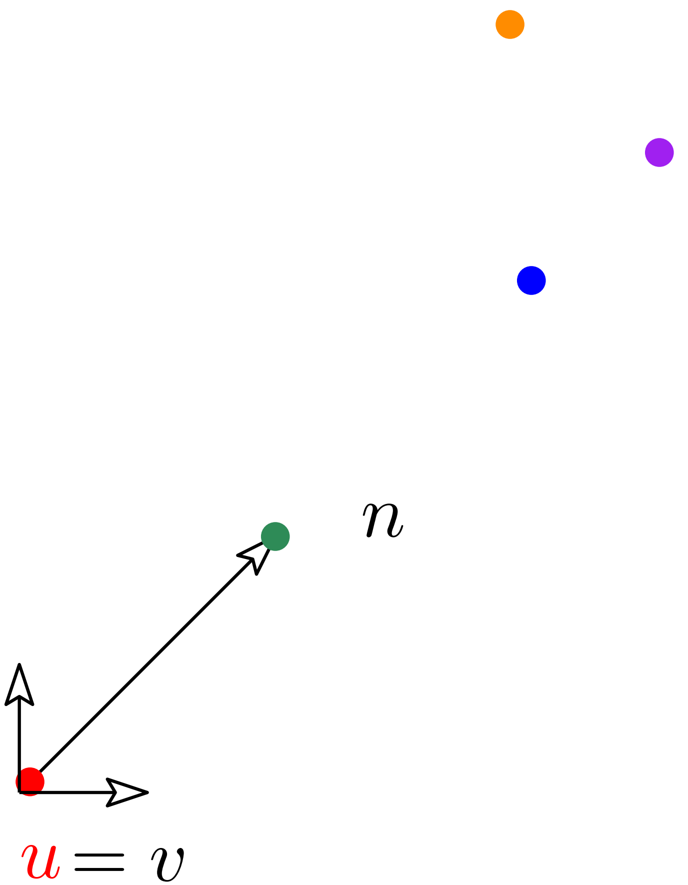
$w_1 = (12, 12)$

$w_2 = (24, 24)$

$w_3 = (30, 30.000001)$

$w_4 = (23, 36)$

$w_5 = (0.5000029, 0.5000027)$



Do

  $n =$ first in S;

  For each $w \in S$

    if vwn positive

     then $n = w$;

  $v.next = n$; $v = n$;

  $S = S \setminus \{v\}$

While $v \neq u$

Convex hull

Teaser robustness lecture

Buggy degenerate example (single precision)

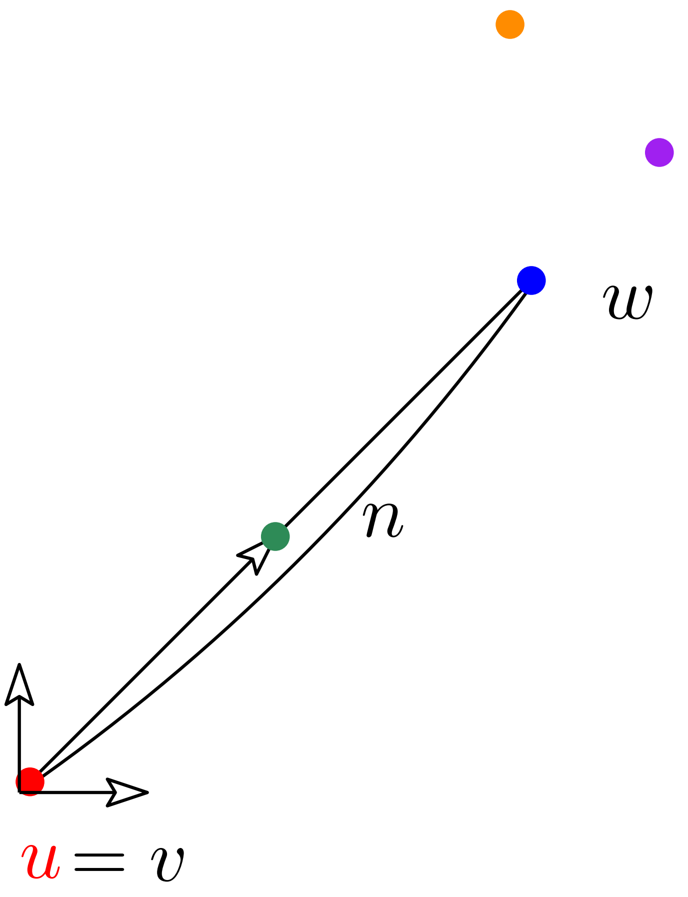
$w_1 = (12, 12)$

$w_2 = (24, 24)$

$w_3 = (30, 30.000001)$

$w_4 = (23, 36)$

$w_5 = (0.5000029, 0.5000027)$



Do

  $n =$ first in S;

  For each $w \in S$

    if vwn positive

     then $n = w$;

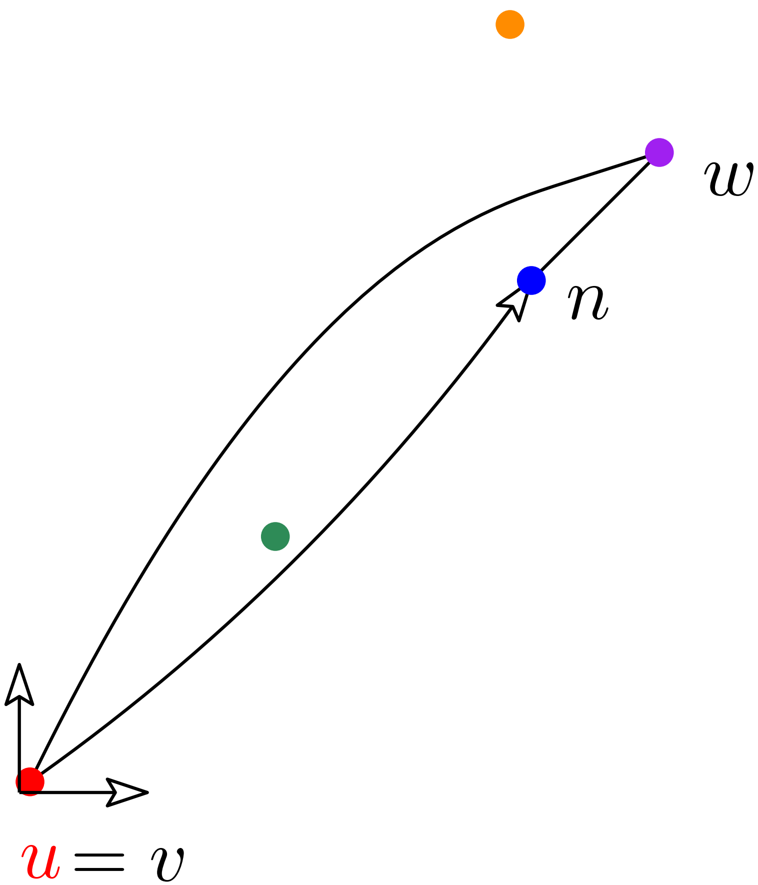
  $v.next = n$; $v = n$;

  $S = S \setminus \{v\}$

While $v \neq u$

Convex hull

Teaser robustness lecture



Buggy degenerate example
(single precision)

$w_1 = (12, 12)$

$w_2 = (24, 24)$

$w_3 = (30, 30.000001)$

$w_4 = (23, 36)$

$w_5 = (0.5000029, 0.5000027)$

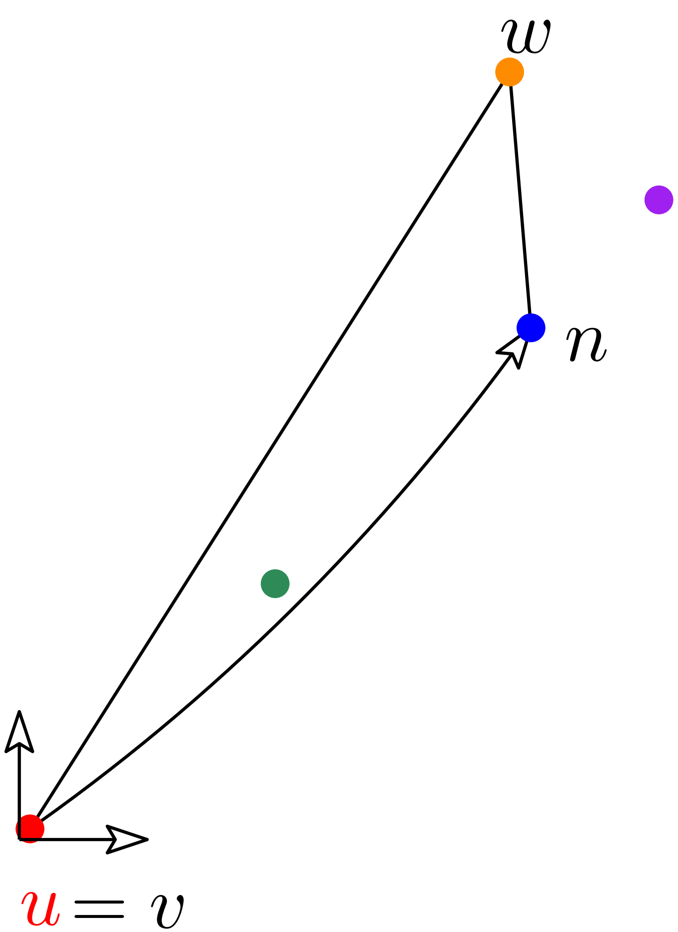
Do
- $n =$ first in S;
- For each $w \in S$
  - if vwn positive
    - then $n = w$;
- $v.next = n$; $v = n$;
- $S = S \setminus \{v\}$

While $v \neq u$

Convex hull

Teaser robustness lecture

Buggy degenerate example
(single precision)



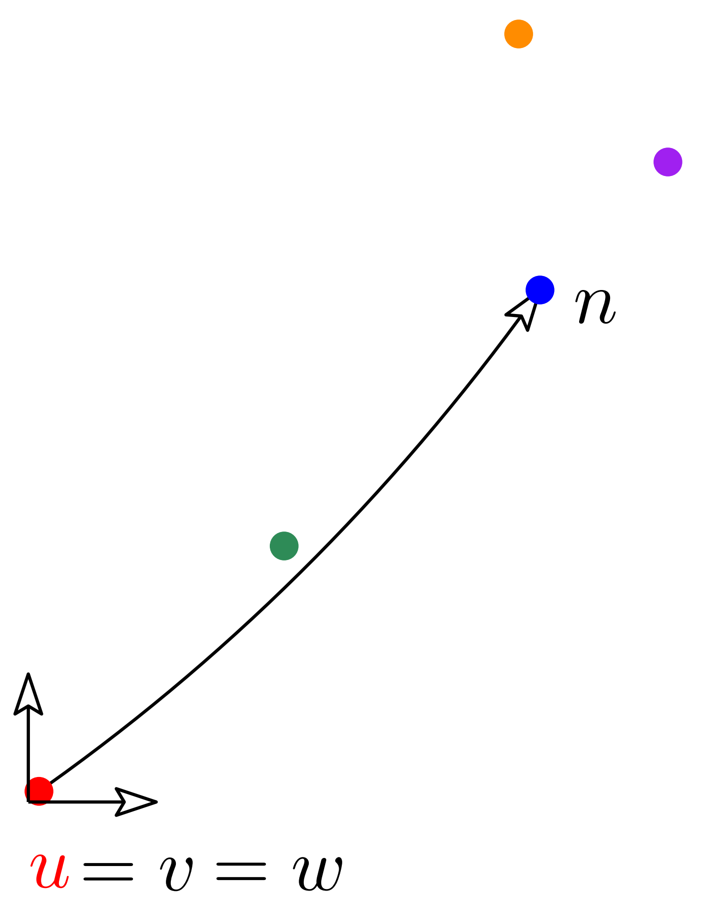
$$w_1 = (12, 12)$$
$$w_2 = (24, 24)$$
$$w_3 = (30, 30.000001)$$
$$w_4 = (23, 36)$$
$$w_5 = (0.5000029, 0.5000027)$$

Do
  $n =$ first in S;
  For each $w \in S$
    if vwn positive
      then $n = w$;
  $v.next = n$; $v = n$;
  $S = S \setminus \{v\}$
While $v \neq u$

Convex hull

Teaser robustness lecture

Buggy degenerate example (single precision)



$w_1 = (12, 12)$

$w_2 = (24, 24)$

$w_3 = (30, 30.000001)$

$w_4 = (23, 36)$

$w_5 = (0.5000029, 0.5000027)$

Do
  $n =$ first in S;
  For each $w \in S$
    if vwn positive
     then $n = w$;
  $v.next = n$; $v = n$;
  $S = S \setminus \{v\}$
While $v \neq u$

Convex hull

Teaser robustness lecture

Buggy degenerate example
(single precision)

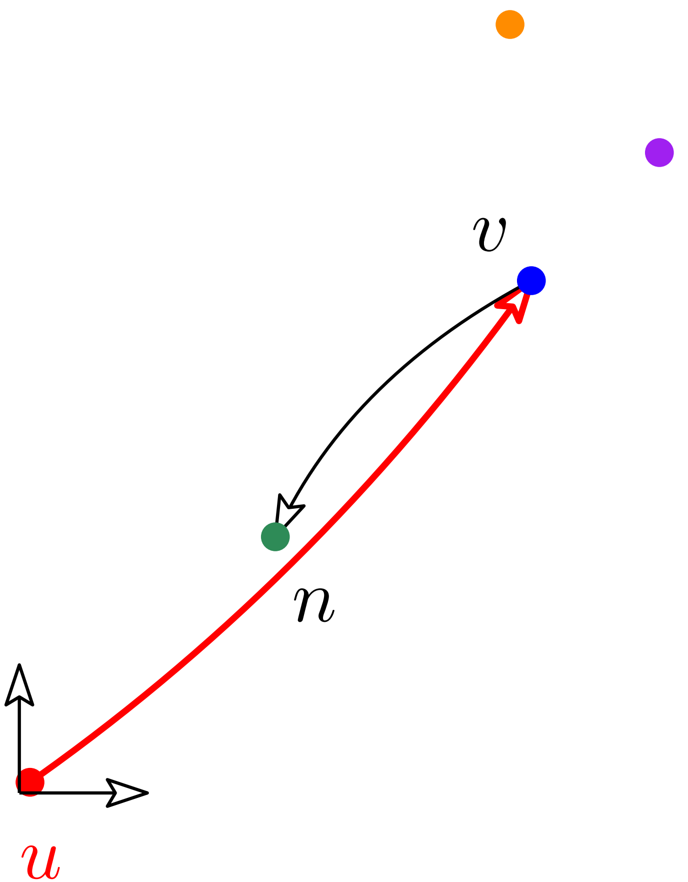
$w_1 = (12, 12)$

~~$w_2 = (24, 24)$~~

$w_3 = (30, 30.000001)$

$w_4 = (23, 36)$

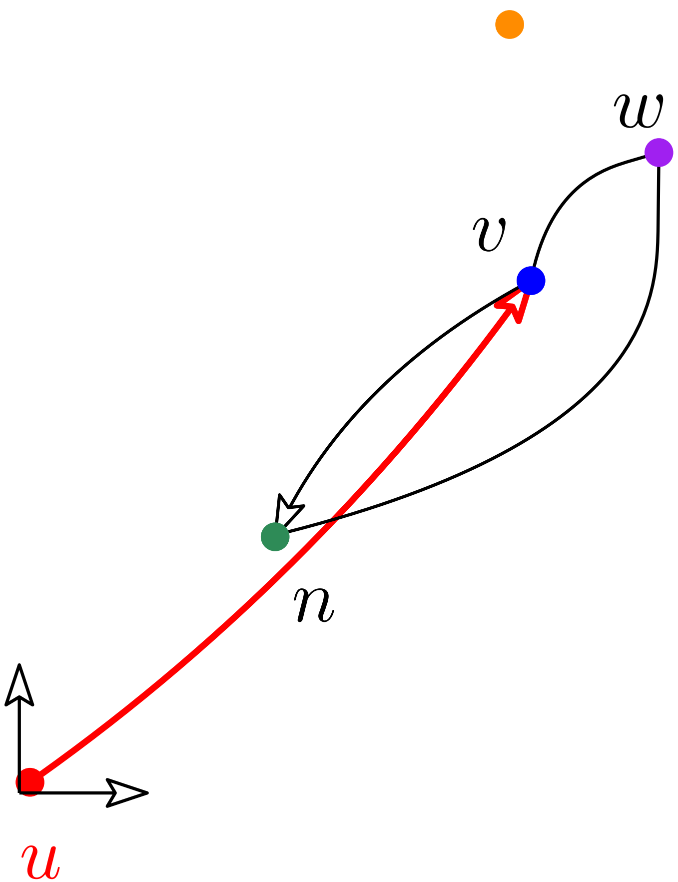
$w_5 = (0.5000029, 0.5000027)$



Do
$n =$ first in S;
For each $w \in S$
if vwn positive
then $n = w$;
$v.next = n$; $v = n$;
$S = S \setminus \{v\}$
While $v \neq u$

Convex hull

Teaser robustness lecture



Buggy degenerate example
(single precision)

$w_1 = (12, 12)$
~~$w_2 = (24, 24)$~~
$w_3 = (30, 30.000001)$
$w_4 = (23, 36)$
$w_5 = (0.5000029, 0.5000027)$

Do
  $n =$ first in S;
  For each $w \in S$
    if vwn positive
     then $n = w$;
  $v.next = n$; $v = n$;
  $S = S \setminus \{v\}$
While $v \neq u$

Convex hull

Teaser robustness lecture

Buggy degenerate example
(single precision)

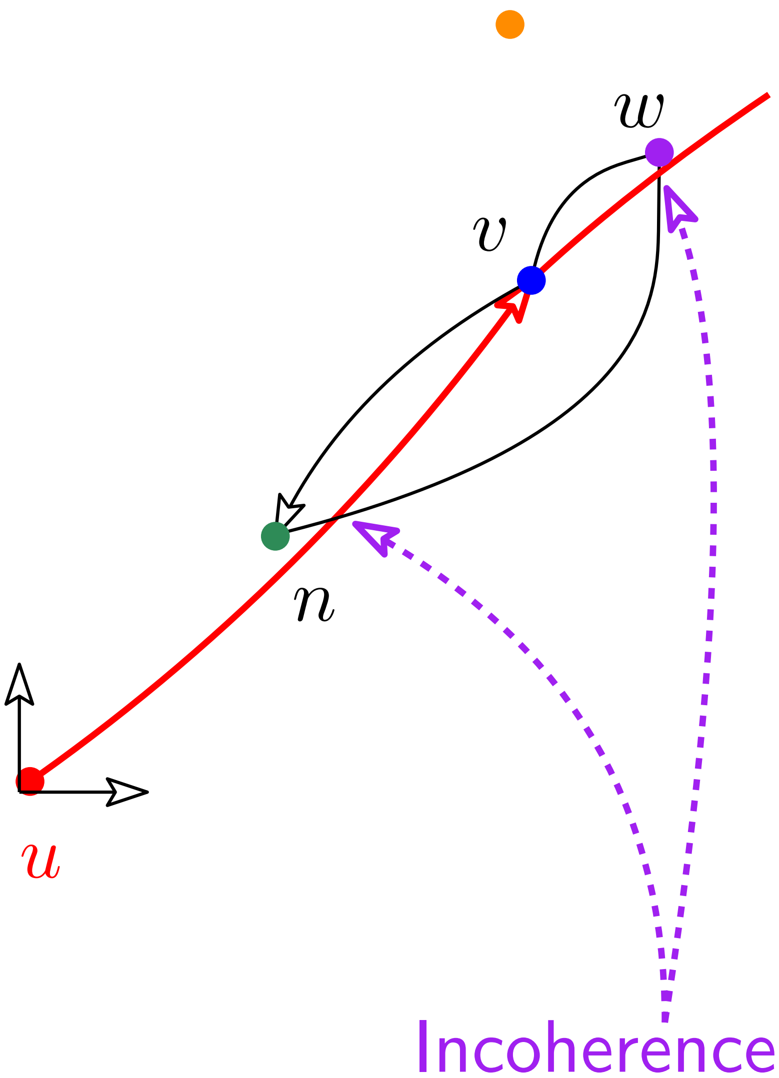
$w_1 = (12, 12)$

~~$w_2 = (24, 24)$~~

$w_3 = (30, 30.000001)$

$w_4 = (23, 36)$

$w_5 = (0.5000029, 0.5000027)$



Do

  $n =$ first in S;

  For each $w \in S$

    if vwn positive

      then $n = w$;

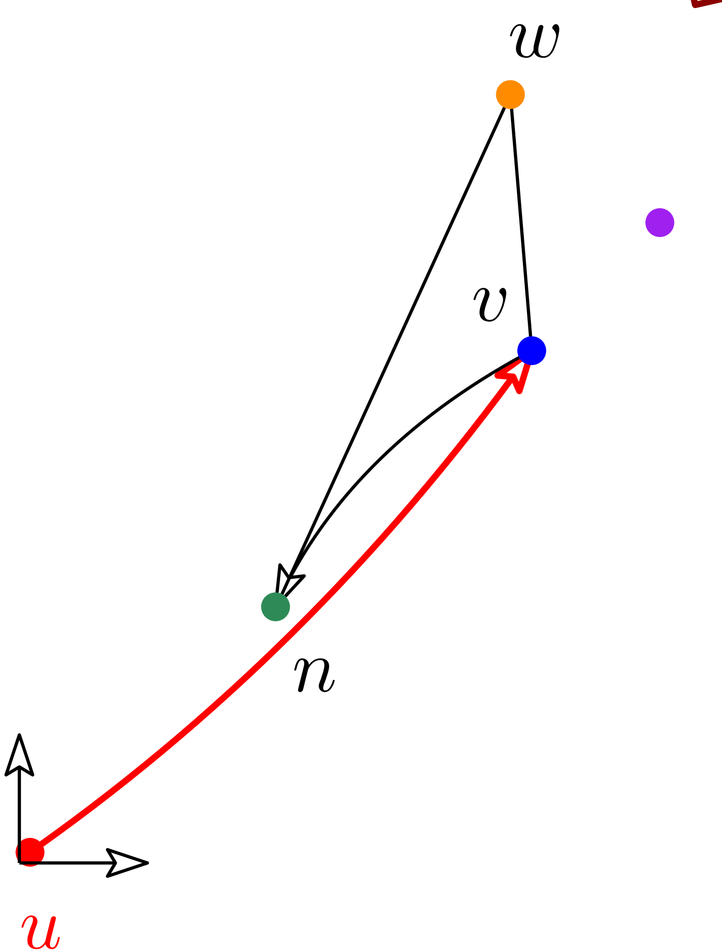
  $v.next = n$; $v = n$;

  $S = S \setminus \{v\}$

While $v \neq u$

Convex hull

Teaser robustness lecture



Buggy degenerate example (single precision)

$w_1 = (12, 12)$

~~$w_2 = (24, 24)$~~

$w_3 = (30, 30.000001)$

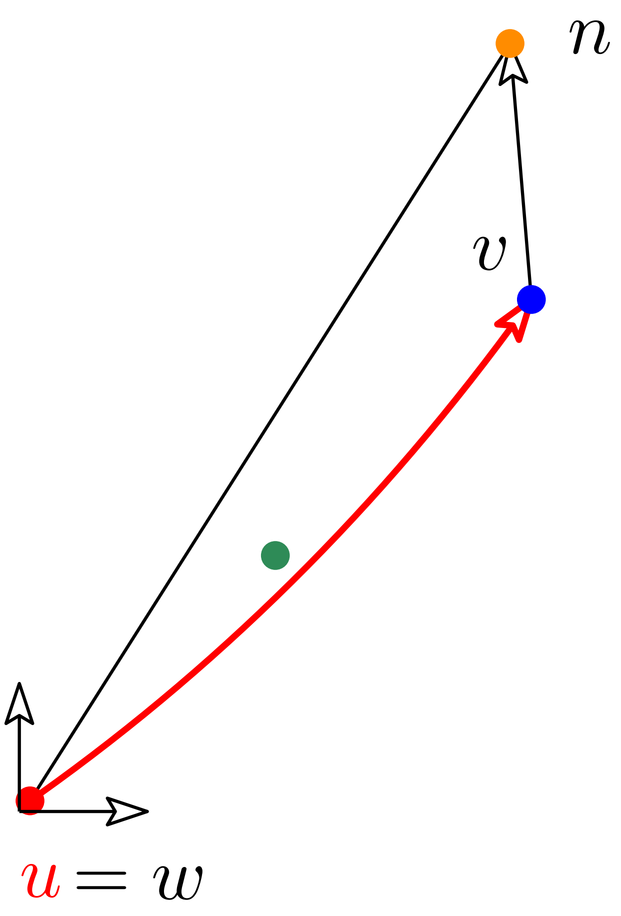
$w_4 = (23, 36)$

$w_5 = (0.5000029, 0.5000027)$

Do
  $n =$ first in S;
  For each $w \in S$
    if vwn positive
      then $n = w$;
  $v.next = n$; $v = n$;
  $S = S \setminus \{v\}$
While $v \neq u$

Convex hull

Teaser robustness lecture



Buggy degenerate example (single precision)

$w_1 = (12, 12)$

~~$w_2 = (24, 24)$~~

$w_3 = (30, 30.000001)$

$w_4 = (23, 36)$

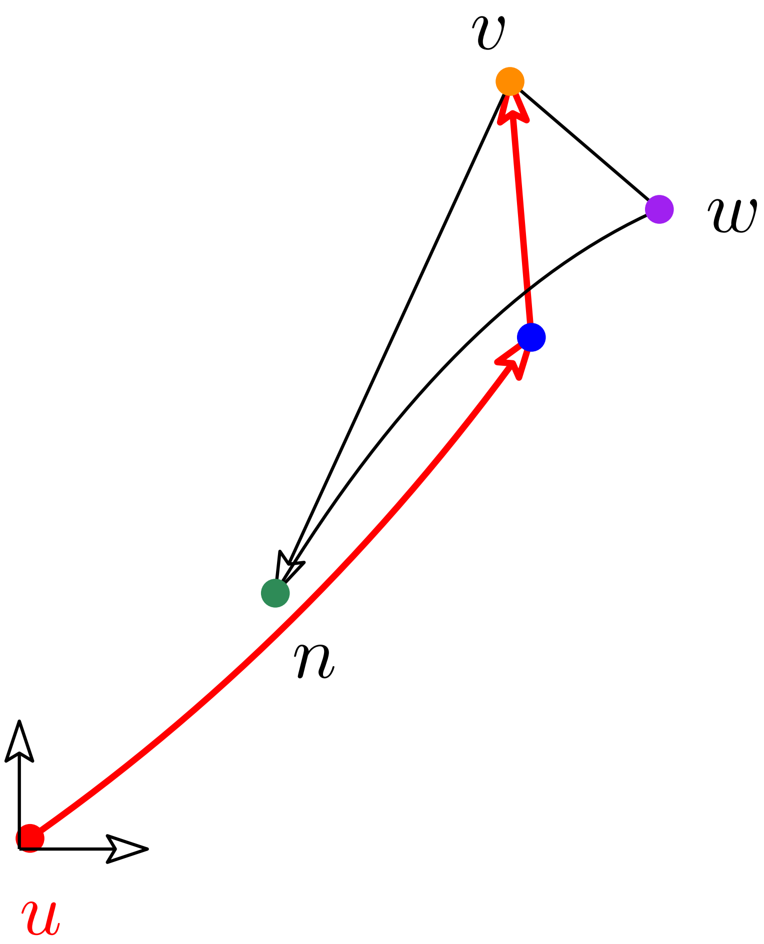
$w_5 = (0.5000029, 0.5000027)$

Do
- $n =$ first in S;
- For each $w \in S$
  - if vwn positive
    - then $n = w$;
- $v.next = n$; $v = n$;
- $S = S \setminus \{v\}$

While $v \neq u$

Convex hull

Teaser robustness lecture



Buggy degenerate example (single precision)

$w_1 = (12, 12)$

~~$w_2 = (24, 24)$~~

$w_3 = (30, 30.000001)$

~~$w_4 = (23, 36)$~~

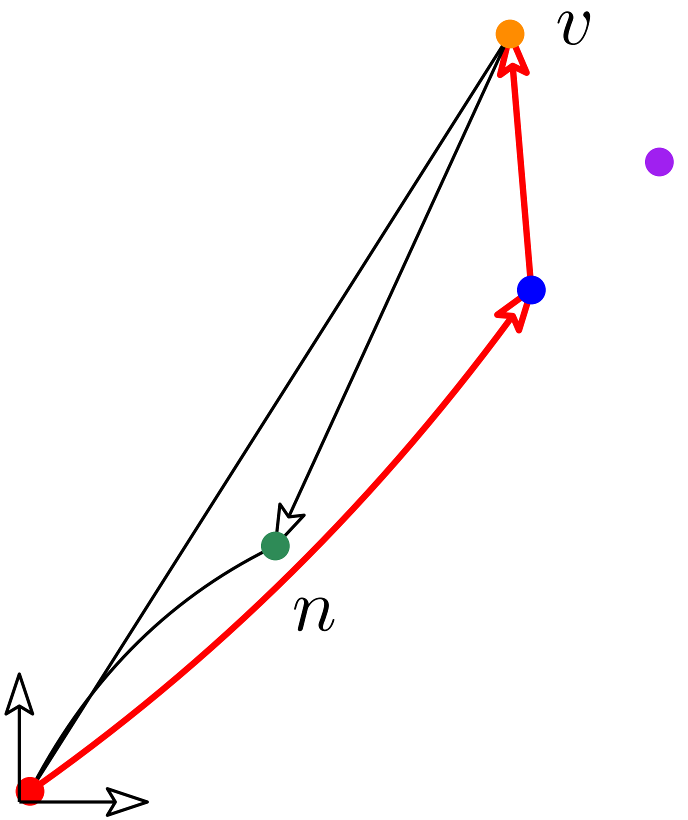
$w_5 = (0.5000029, 0.5000027)$

Do
  $n =$ first in S;
  For each $w \in S$
    if vwn positive
     then $n = w$;
  $v.next = n$; $v = n$;
  $S = S \setminus \{v\}$
While $v \neq u$

Convex hull

Teaser robustness lecture

Buggy degenerate example (single precision)



$w_1 = (12, 12)$

~~$w_2 = (24, 24)$~~

$w_3 = (30, 30.000001)$

~~$w_4 = (23, 36)$~~

$w_5 = (0.5000029, 0.5000027)$

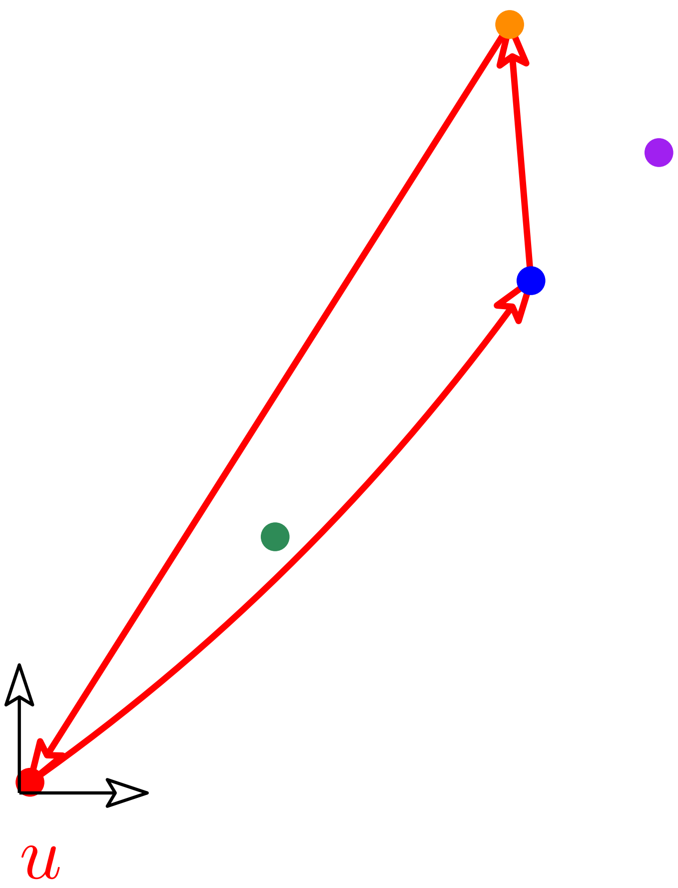
Do
- $n =$ first in S;
- For each $w \in S$
  - if vwn positive
    - then $n = w$;
- $v.next = n$; $v = n$;
- $S = S \setminus \{v\}$

While $v \neq u$

Convex hull

Teaser robustness lecture

Buggy degenerate example (single precision)



~~$w_1 = (12, 12)$~~

~~$w_2 = (24, 24)$~~

$w_3 = (30, 30.000001)$

~~$w_4 = (23, 36)$~~

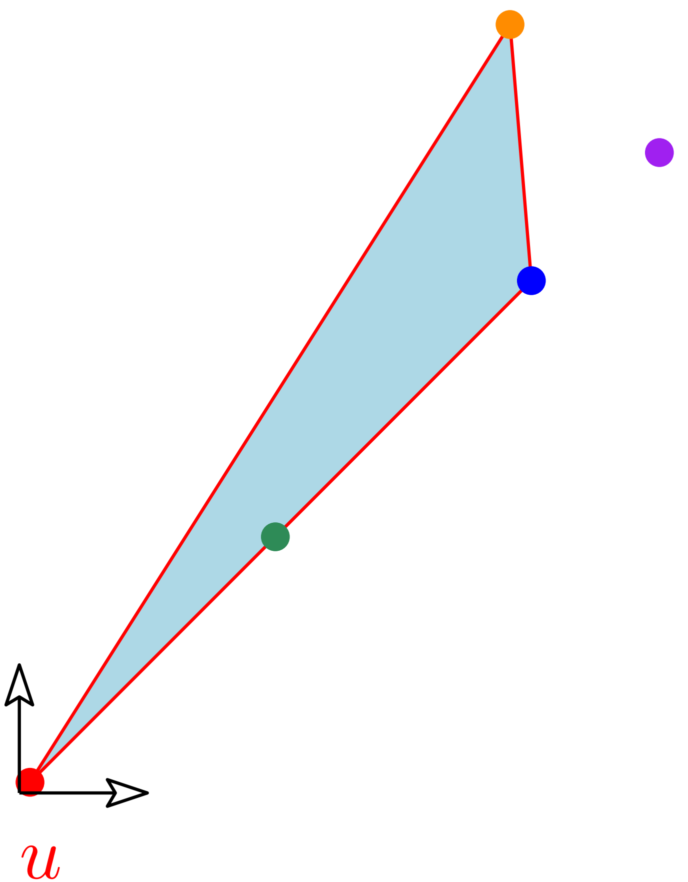
$u = \quad w_5 = (0.5000029, 0.5000027)$

Do
  $n =$ first in S;
  For each $w \in S$
    if vwn positive
     then $n = w$;
  $v.next = n$; $v = n$;
  $S = S \setminus \{v\}$
While $v \neq u$

Convex hull

Teaser robustness lecture

Buggy degenerate example
(single precision)



$w_1 = (12, 12)$

$w_2 = (24, 24)$

$w_3 = (30, 30.000001)$

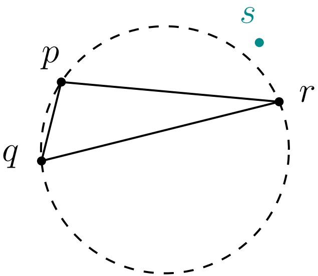
$w_4 = (23, 36)$

$w_5 = (0.5000029, 0.5000027)$

Result is really wrong

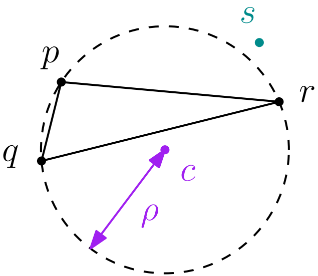
Arithmetic issues

in_disk predicate



Arithmetic issues

in_disk predicate



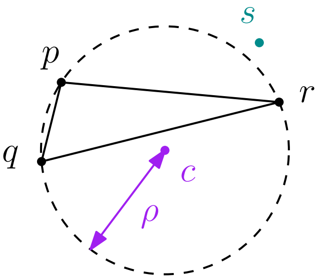
circle $\mathcal{C}$ through p, q, r

unknowns c, ρ
solve $\longrightarrow$

- center c
- radius ρ

Arithmetic issues

in_disk predicate



circle $\mathcal{C}$ through p, q, r

unknowns c, ρ
solve $\longrightarrow$

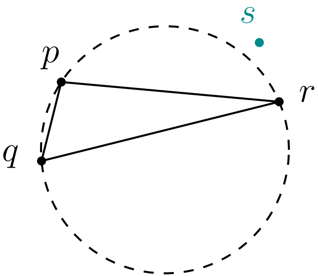
- center c
- radius ρ

Bad idea. . .
rounding errors $\hookrightarrow$ $p, q, r \notin \mathcal{C}(c, \rho)$

“random” result for s

Arithmetic issues

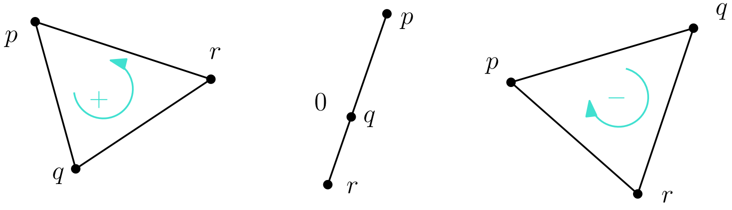
in_disk predicate



$$\text{in\_disk}(p, q, r, s) = \text{sign} \begin{vmatrix} 1 & 1 & 1 & 1 \\ p_x & q_x & r_x & s_x \\ p_y & q_y & r_y & s_y \\ p_x^2 + p_y^2 & q_x^2 + q_y^2 & r_x^2 + r_y^2 & s_x^2 + s_y^2 \end{vmatrix}$$

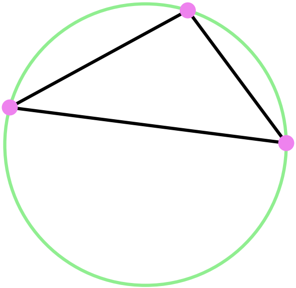
Arithmetic issues

orientation predicate



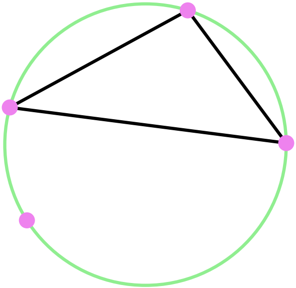
$$\text{orient}(p, q, r) = \text{sign} \begin{vmatrix} 1 & 1 & 1 \\ p_x & q_x & r_x \\ p_y & q_y & r_y \end{vmatrix}$$

Degenerate cases



Choices

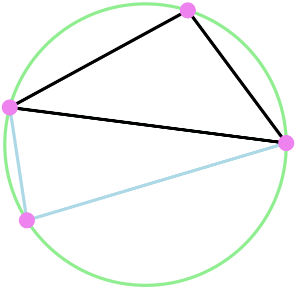
Degenerate cases



Choices

?

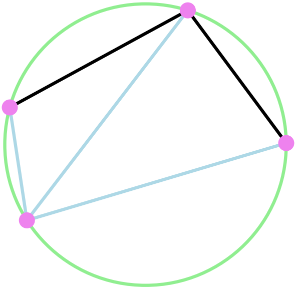
Degenerate cases



Choices

? new point “outside disk”

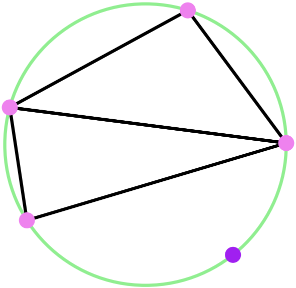
Degenerate cases



Choices

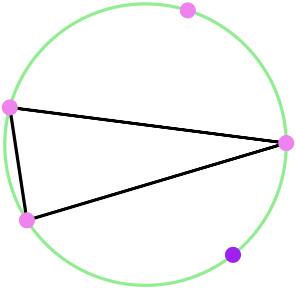
? new point “in disk”

Degenerate cases



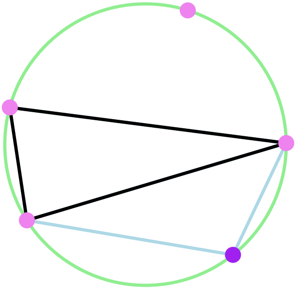
?

Degenerate cases



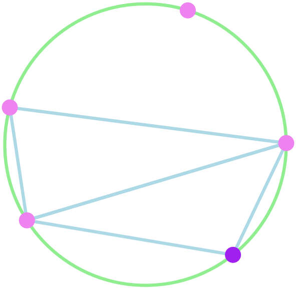
?

Degenerate cases



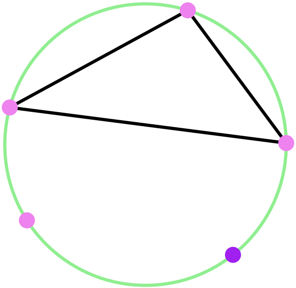
choice: new point “outside”

Degenerate cases



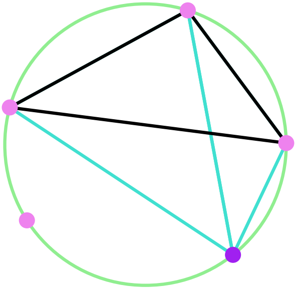
choice: new point "outside"

Degenerate cases



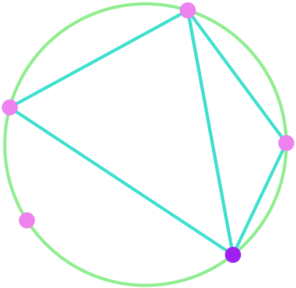
?

Degenerate cases



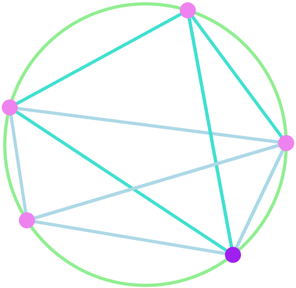
choice: new point "inside"

Degenerate cases



choice: new point “inside”

Degenerate cases



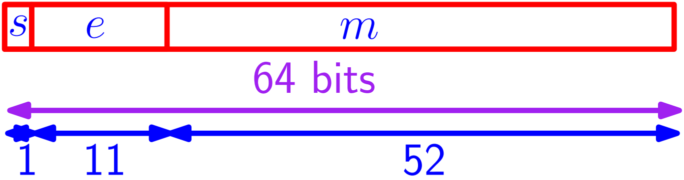
inconsistency

decisions must be made
in a consistent way

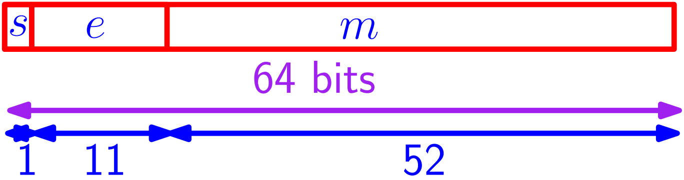
Arithmetic issues

Arithmetic issues — Reminder: floating-point arithmetic

`double` IEEE 754

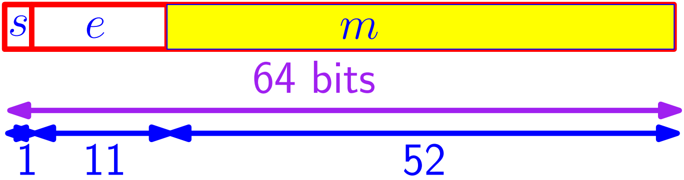


`double` IEEE 754

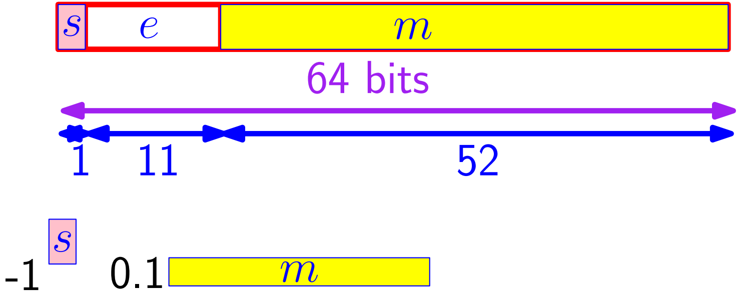


0.1

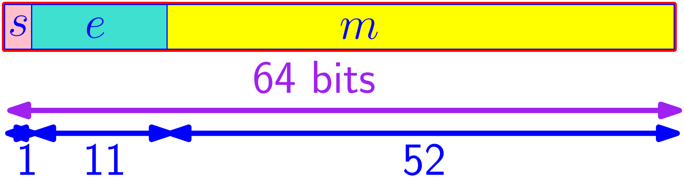
`double` IEEE 754



`double` IEEE 754

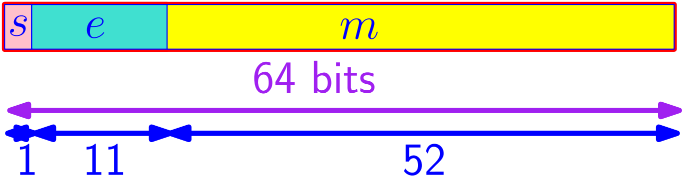


`double` IEEE 754



$$-1^{s}\quad 0.1\,m\quad 2^{-1024+e}$$

`double` IEEE 754



$$-1^{s} \; 0.1\,m \; 2^{-1024+e}$$

normalized numbers

Representable Numbers

Representable Numbers

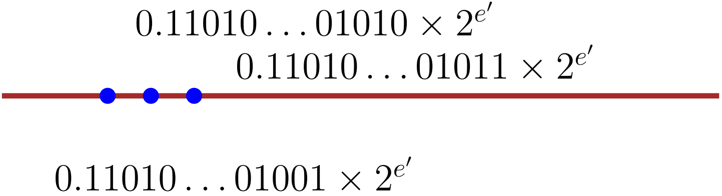
$$0.11010\ldots01001 \times 2^{e'}$$

Representable Numbers

$$0.11010\ldots01010 \times 2^{e'}$$

$$0.11010\ldots01001 \times 2^{e'}$$

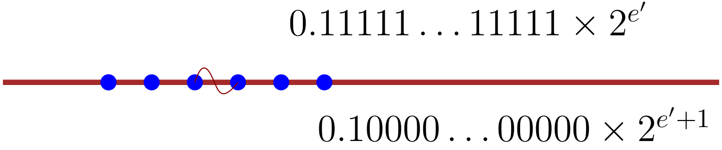
Representable Numbers



Representable Numbers

$$0.11111\ldots11111 \times 2^{e'}$$

Representable Numbers



Representable Numbers

$$0.11111\ldots11111 \times 2^{e'}$$

$$0.10000\ldots00000 \times 2^{e'+1}$$

$$0.10000\ldots00001 \times 2^{e'+1}$$

Representable Numbers

$$0.11111\ldots11111 \times 2^{e'}$$

$$0.10000\ldots00000 \times 2^{e'+1}$$

$$0.10000\ldots00001 \times 2^{e'+1}$$

$$0.10000\ldots00010 \times 2^{e'+1}$$

$$0.10000\ldots00011 \times 2^{e'+1}$$

Smallest Representable Number

$$0.00000\ldots00000 \times 2^{-1024}$$

$$0.10000\ldots00000 \times 2^{-1024}$$

Smallest Representable Number

$$0.00000\ldots00000 \times 2^{-1024}$$

$$0.10000\ldots00000 \times 2^{-1024}$$
$$0.10000\ldots00001 \times 2^{-1024}$$
$$0.10000\ldots00010 \times 2^{-1024}$$

Smallest Representable Number

$$0.11111\ldots11111 \times 2^{-1024}$$

$$0.00000\ldots00000 \times 2^{-1024}$$

$$0.10000\ldots00000 \times 2^{-1024}$$

$$0.10000\ldots00001 \times 2^{-1024}$$

$$0.10000\ldots00010 \times 2^{-1024}$$

Smallest Representable Number

$$0.10000\ldots00000 \times 2^{-1023}$$

$$0.11111\ldots11111 \times 2^{-1024}$$

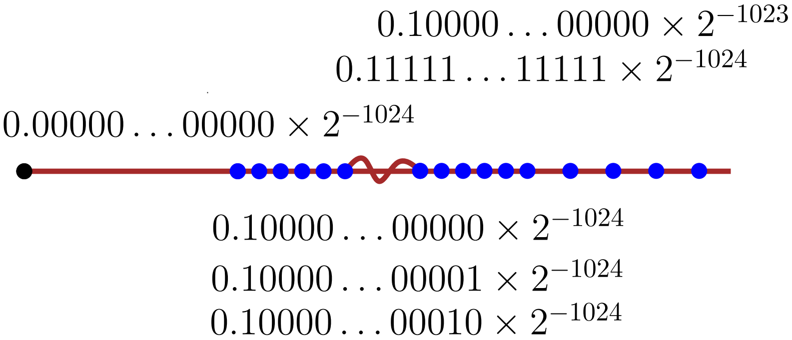
$$0.00000\ldots00000 \times 2^{-1024}$$

$$0.10000\ldots00000 \times 2^{-1024}$$

$$0.10000\ldots00001 \times 2^{-1024}$$

$$0.10000\ldots00010 \times 2^{-1024}$$

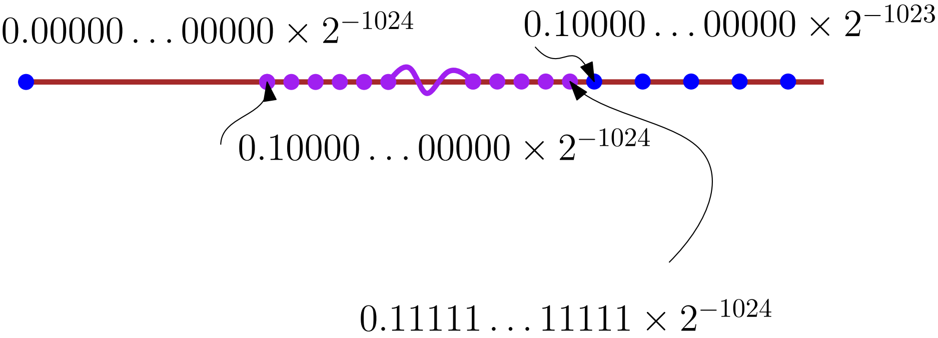
Smallest Representable Number



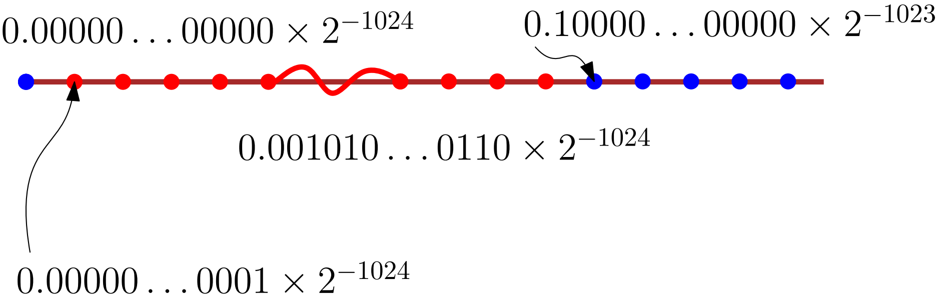
Denormalized Numbers

Denormalized Numbers

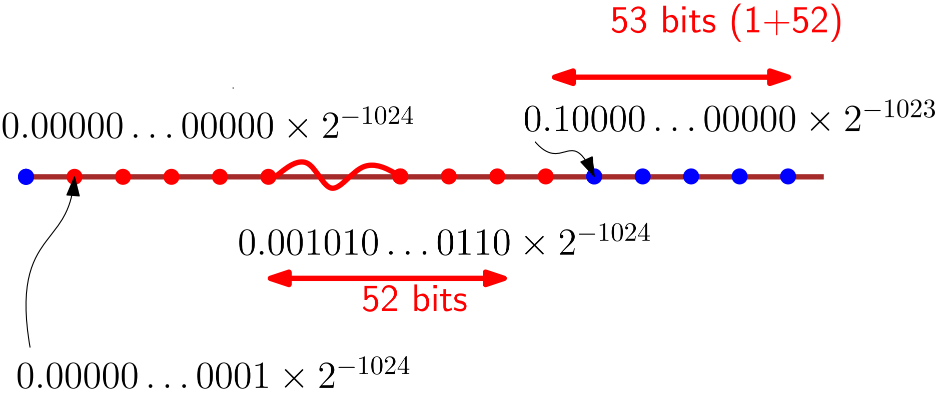
Denormalized Numbers



Denormalized Numbers

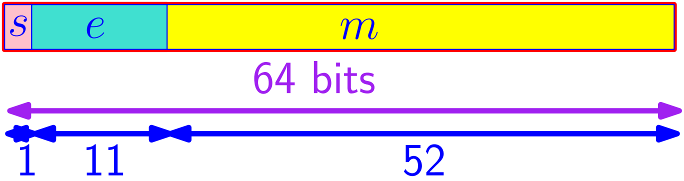


Denormalized Numbers



Denormalized Numbers

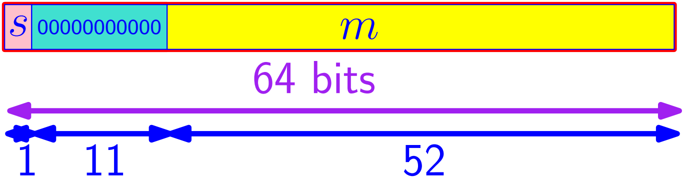
`double` → normalized



$$-1^{s} \; 0.1\,m \; 2^{-1024+e}$$

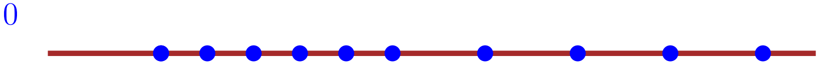
Denormalized Numbers

`double` → de-normalized

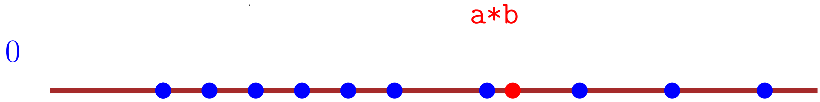


$-1^{s}\ 0.m\ 2^{-1024}$

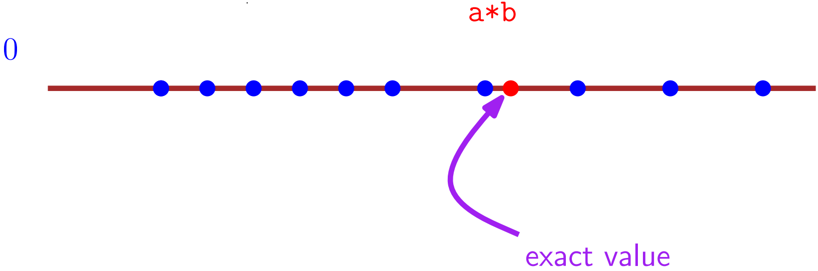
Rounding modes



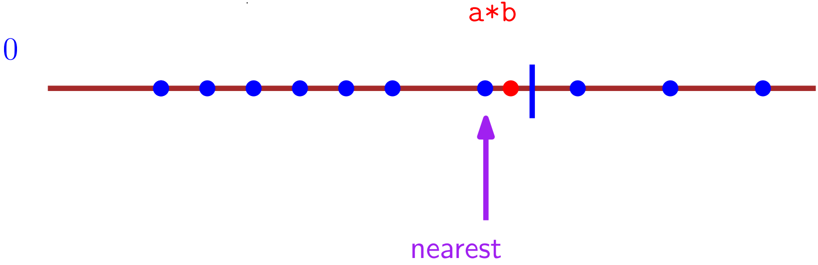
Rounding modes



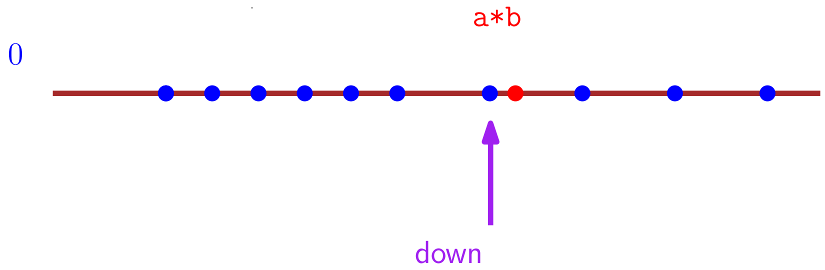
Rounding modes



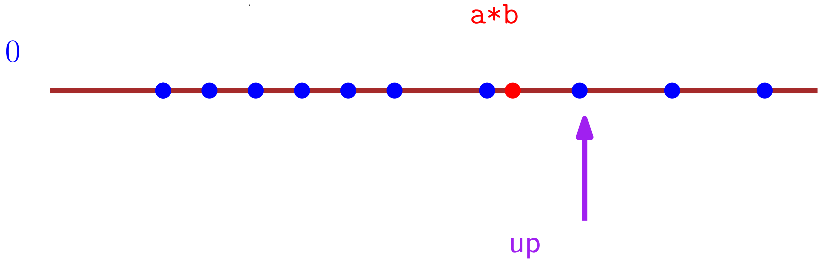
Rounding modes



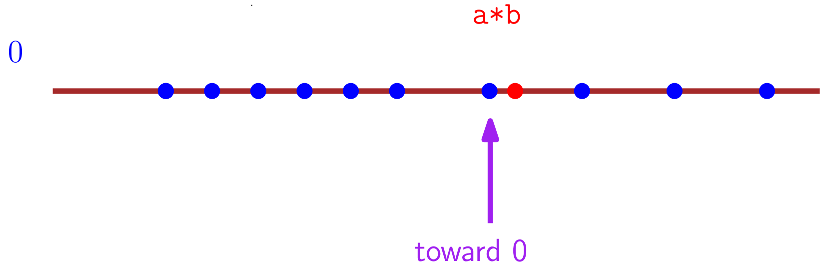
Rounding modes



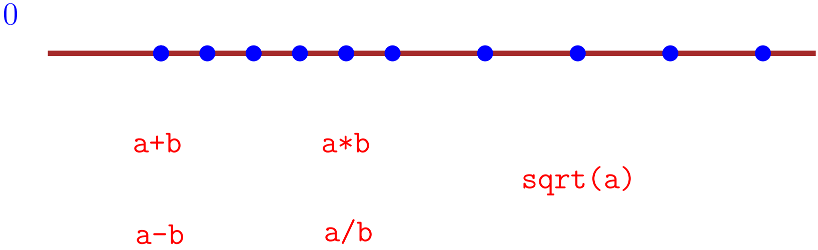
Rounding modes



Rounding modes

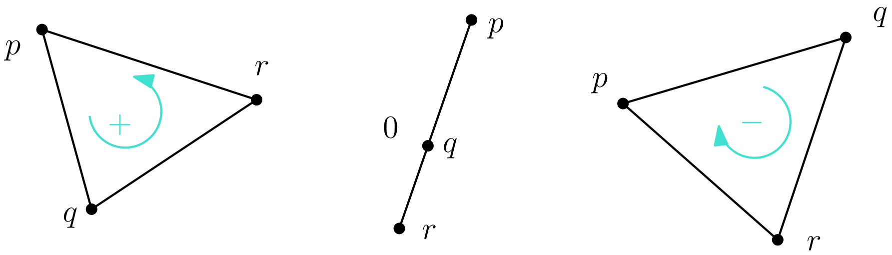


Rounding modes



Arithmetic issues — Consequences

Orientation predicate



$$\operatorname{sign}\begin{vmatrix} x_p & x_q & x_r \\ y_p & y_q & y_r \\ 1 & 1 & 1 \end{vmatrix} = \operatorname{sign}\begin{vmatrix} x_q - x_p & x_r - x_p \\ y_q - y_p & y_r - y_p \end{vmatrix}$$

Toy model

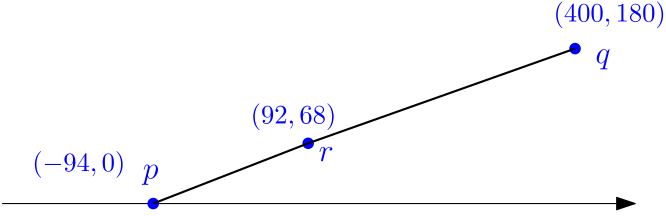
`double`	53 binary digits
`float`	24 binary digits
Toy model	2 decimal digits

$$35 + 3.7 = 38.7 \quad \textit{exact}$$

$$(35 + 3.3) + 0.4 \simeq 38$$
$$35 + (3.3 + 0.4) \simeq 39$$

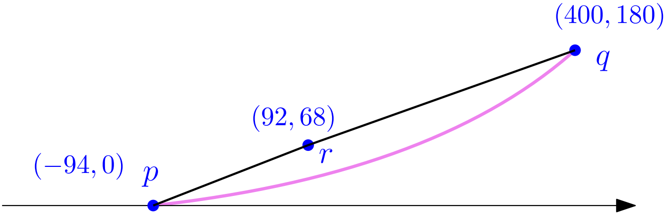
Orientation predicate in the toy model

$\text{sign}((x_q - x_p) \times (y_r - y_p) \; - \; (y_q - y_p) \times (x_r - x_p))$



Orientation predicate in the toy model

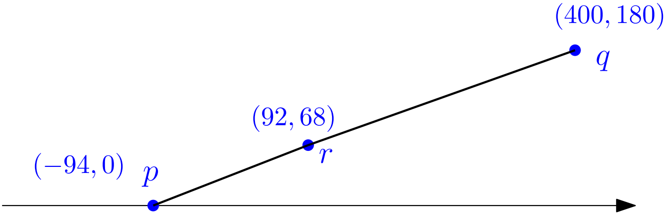
$\text{sign}((x_q - x_p) \times (y_r - y_p) \; - \; (y_q - y_p) \times (x_r - x_p))$



$$(400 + 94) \quad \times \quad 68 \quad - \quad (92 + 94) \quad \times \quad 180 \quad = \quad 112 \text{ exact}$$

Orientation predicate in the toy model

$\text{sign}((x_q - x_p) \times (y_r - y_p) \; - \; (y_q - y_p) \times (x_r - x_p))$

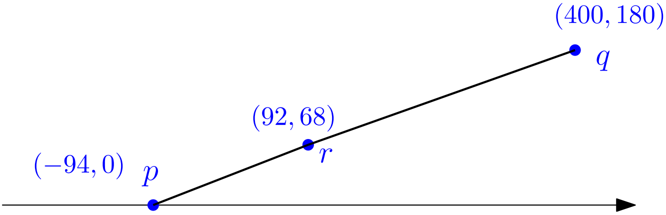


$(400 + 94) \times 68 - (92 + 94) \times 180 =$ 112 exact

$494 \times 68 - 186 \times 180 \simeq$

Orientation predicate in the toy model

$\text{sign}((x_q - x_p) \times (y_r - y_p) \; - \; (y_q - y_p) \times (x_r - x_p))$



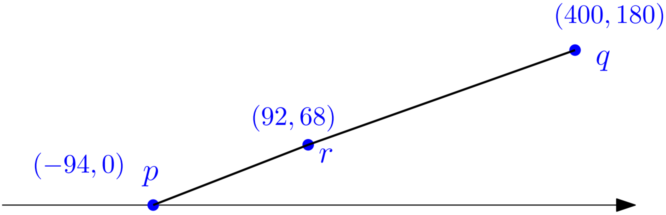
$(400 + 94) \times 68 - (92 + 94) \times 180 = 112$ exact

$494 \times 68 - 186 \times 180 \simeq$

$490 \times 68 - 190 \times 180 =$

Orientation predicate in the toy model

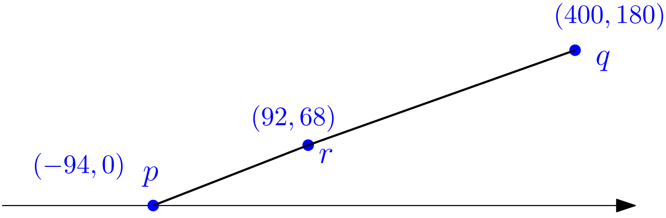
$\mathrm{sign}((x_q - x_p) \times (y_r - y_p) \; - \; (y_q - y_p) \times (x_r - x_p))$



(400 + 94)	×	68	−	(92 + 94)	×	180	=	112 exact
494	×	68	−	186	×	180	≃	
490	×	68	−	190	×	180	=	
33320			−	34200			≃	

Orientation predicate in the toy model

$\text{sign}((x_q - x_p) \times (y_r - y_p) \; - \; (y_q - y_p) \times (x_r - x_p))$

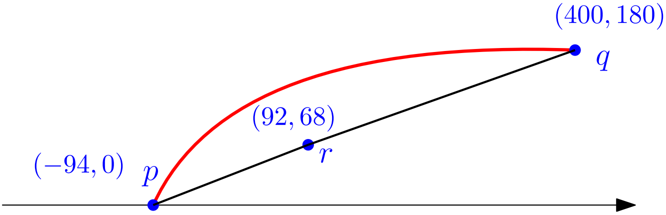


$$(400 + 94) \times 68 - (92 + 94) \times 180 = 112 \text{ exact}$$

$$\begin{array}{ccccccc}
494 & \times & 68 & - & 186 & \times & 180 \simeq \\
490 & \times & 68 & - & 190 & \times & 180 = \\
33320 & & & - & 34200 & & \simeq \\
33000 & & & - & 34000 & & =
\end{array}$$

Orientation predicate in the toy model

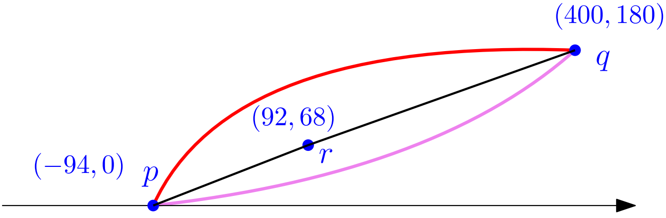
$\text{sign}((x_q - x_p) \times (y_r - y_p) \; - \; (y_q - y_p) \times (x_r - x_p))$



$$
\begin{array}{ccccccccc}
(400+94) & \times & 68 & - & (92+94) & \times & 180 & = & 112 \text{ exact} \\
494 & \times & 68 & - & 186 & \times & 180 & \simeq & \\
490 & \times & 68 & - & 190 & \times & 180 & = & \\
33320 & & & - & 34200 & & & \simeq & \\
33000 & & & - & 34000 & & & = & \\
 & & & & & & & = & -1000 \;\; !!
\end{array}
$$

Orientation predicate in the toy model

$$\text{sign}((x_q - x_p) \times (y_r - y_p) \; - \; (y_q - y_p) \times (x_r - x_p))$$



$$(400 + 94) \times 68 - (92 + 94) \times 180 = 112 \text{ exact}$$

$$494 \times 68 - 186 \times 180 \simeq$$
$$490 \times 68 - 190 \times 180 =$$
$$33320 - 34200 \simeq$$
$$33000 - 34000 =$$
$$= -1000 \;\; !!$$

Orientation predicate with `double`

$p = (0.5 + x.u,\ 0.5 + y.u)$
$0 \leq\ x, y\ < 256,\ \ u = 2^{-53}$
$q = (12, 12)$
$r = (24, 24)$

Orientation predicate with `double`

$p = (0.5 + x.u,\ 0.5 + y.u)$
$0 \leq\ x, y\ < 256,\ \ u = 2^{-53}$
$q = (12, 12)$
$r = (24, 24)$

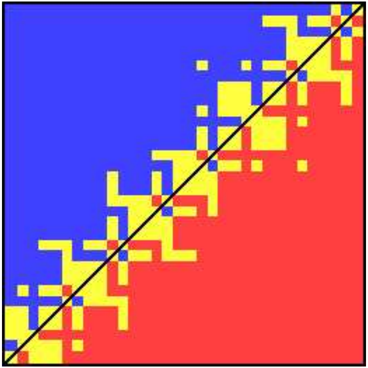
orientation(p, q, r)
evaluated with `double`

Orientation predicate with `double`

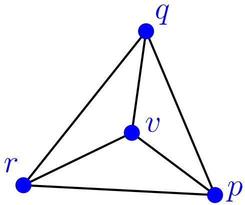
$p = (0.5 + x.u,\ 0.5 + y.u)$
$0 \leq\ x, y\ < 256, \quad u = 2^{-53}$
$q = (12, 12)$
$r = (24, 24)$

$orientation(p, q, r)$
evaluated with `double`

256 x 256 pixel image
> 0, $= 0$, < 0



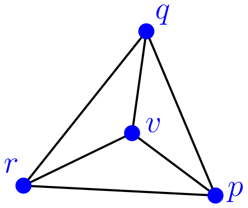
Failure of geometric theorems



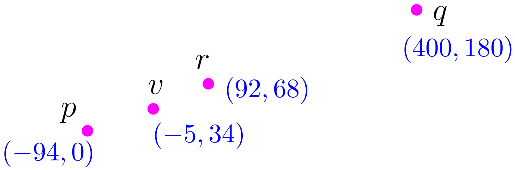
pqv, qrv, rpv `ccw`

$\Longrightarrow$ pqr `ccw`

Failure of geometric theorems



Toy model

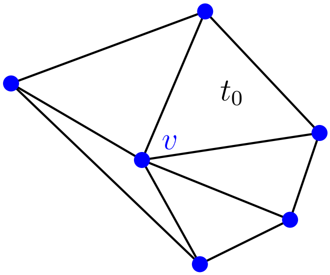


pqv, qrv, rpv `ccw`
$\Longrightarrow$ pqr `ccw`

pqv, qrv, rpv `ccw`
$\longrightarrow$ pqr `cw`

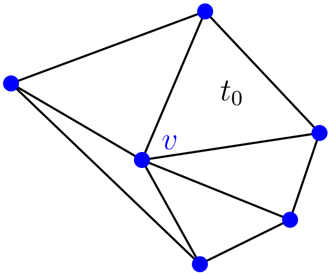
Failure of geometric algorithms

Turn around a vertex



Failure of geometric algorithms

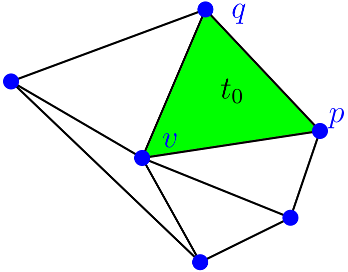
Turn around a vertex




```
do
      // vpq ccw triangle
      go to neighbor through qv
while triangle ≠ t0
```

Failure of geometric algorithms

Turn around a vertex



`do`

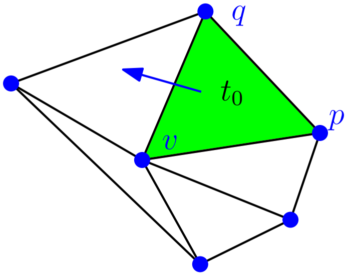
  // *vpq* `ccw` *triangle*

  `go to neighbor through` *qv*

`while triangle` $\neq t_0$

Failure of geometric algorithms

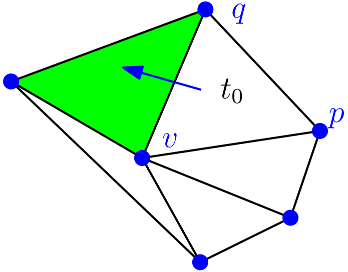
Turn around a vertex




```
do
      // vpq ccw triangle
      go to neighbor through qv
while triangle ≠ t0
```

Failure of geometric algorithms

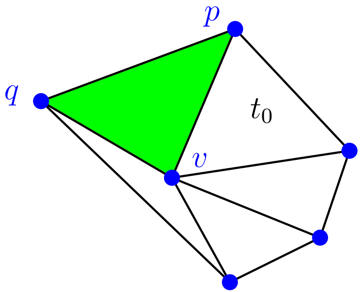
Turn around a vertex




```
do
      // vpq ccw triangle
      go to neighbor through qv
while triangle ≠ t0
```

Failure of geometric algorithms

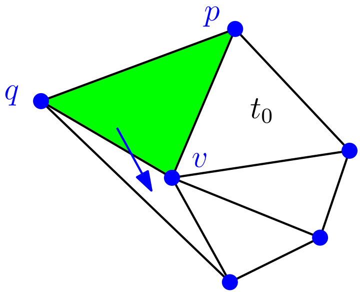
Turn around a vertex




```
do
      // vpq ccw triangle
      go to neighbor through qv
while triangle ≠ t0
```

Failure of geometric algorithms

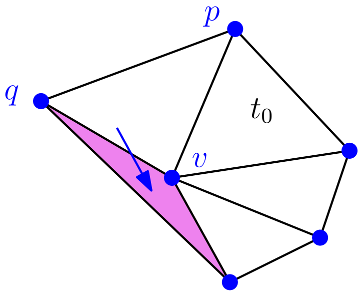
Turn around a vertex




```
do
      // vpq ccw triangle
      go to neighbor through qv
while triangle ≠ t0
```

Failure of geometric algorithms

Turn around a vertex



`do`

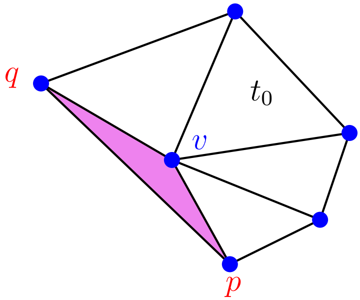
    // *vpq* `ccw` *triangle*

    `go to neighbor through` *qv*

`while triangle` $\neq t_0$

Failure of geometric algorithms

Turn around a vertex



do

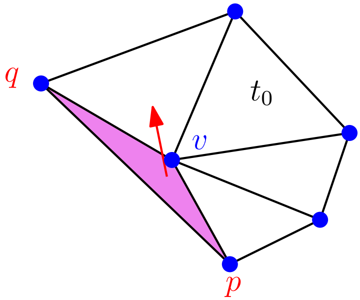
    // *vpq* `ccw` *triangle*

    `go to neighbor through` *qv*

`while triangle` $\neq t_0$

Failure of geometric algorithms

Turn around a vertex

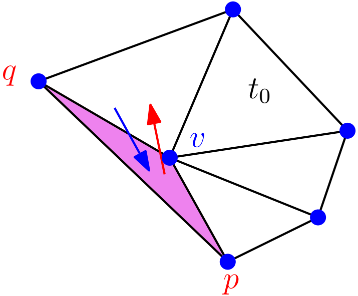



```
do
      // vpq ccw triangle
      go to neighbor through qv
while triangle ≠ t0
```

LOOP

Failure of geometric algorithms

Turn around a vertex

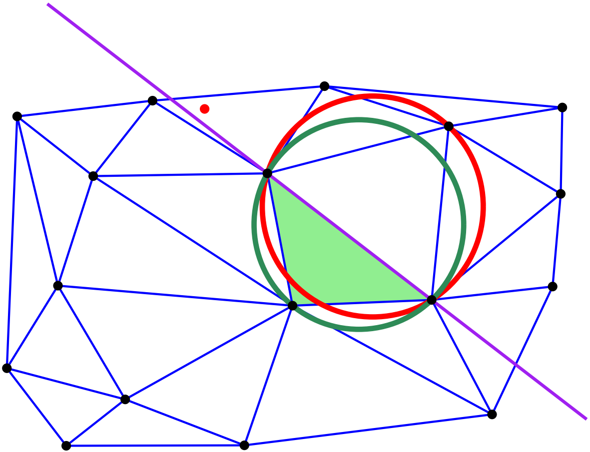



```
do
      // vpq ccw triangle
      go to neighbor through qv
while triangle ≠ t0
```

LOOP

Failure of geometric algorithms

Visibility walk

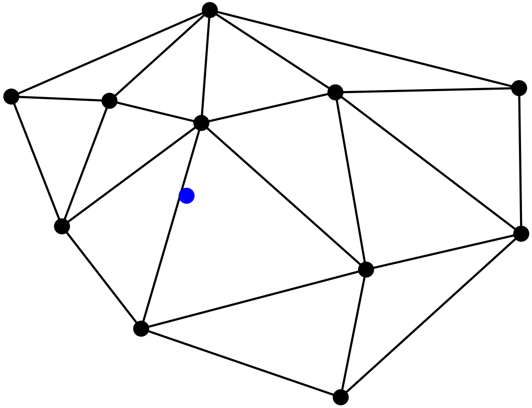


may have cycles even in a Delaunay triangulation

see course Delaunay triangulation

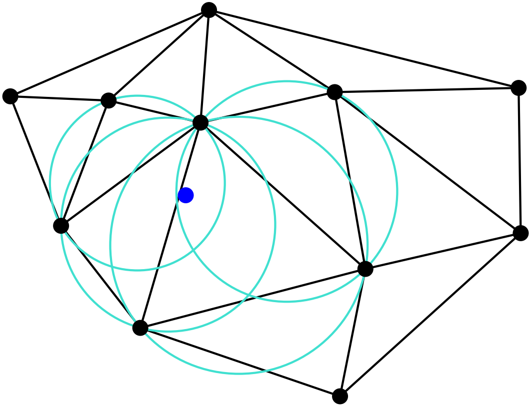
Solution 1 - Forget about geometric theorems

Insertion in a Delaunay triangulation



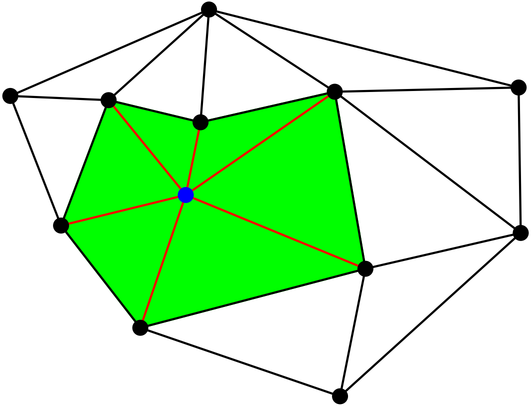
Solution 1 - Forget about geometric theorems

Insertion in a Delaunay triangulation



Solution 1 - Forget about geometric theorems

Insertion in a Delaunay triangulation

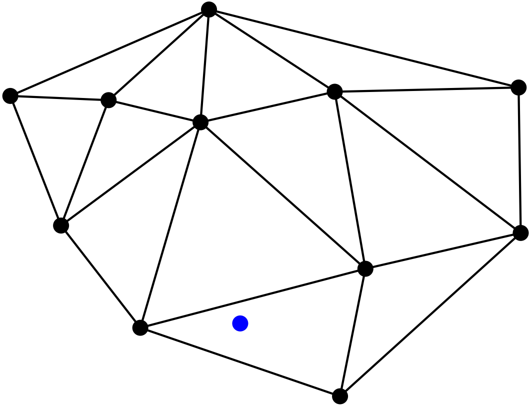


the conflict region is star-shaped

Solution 1 - Forget about geometric theorems

Insertion in a Delaunay triangulation

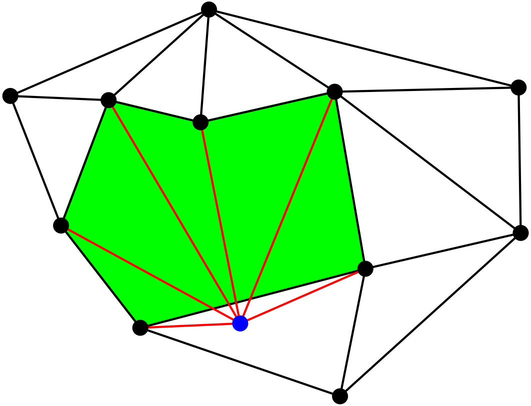
with inexact predicates



Solution 1 - Forget about geometric theorems

Insertion in a Delaunay triangulation

with inexact predicates

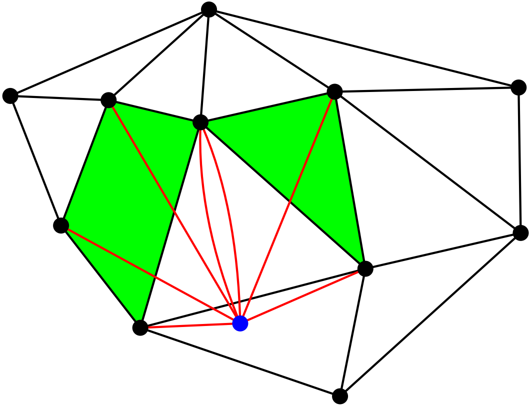


the conflict region is not always star-shaped

Solution 1 - Forget about geometric theorems

Insertion in a Delaunay triangulation

with inexact predicates

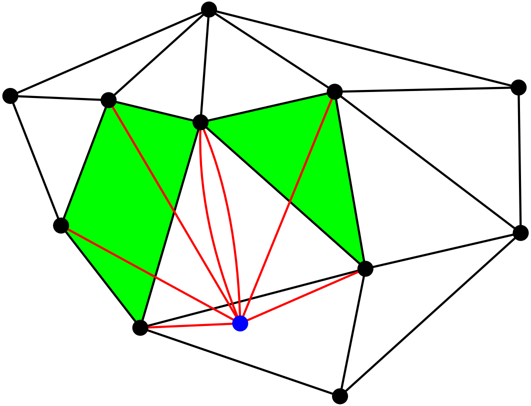


the conflict region is not always star-shaped

Solution 1 - Forget about geometric theorems

Insertion in a Delaunay triangulation

with inexact predicates

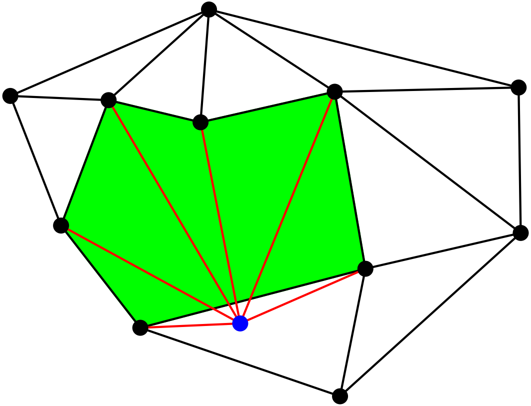


not always combinatorially correct

Solution 1 - Forget about geometric theorems

Insertion in a Delaunay triangulation

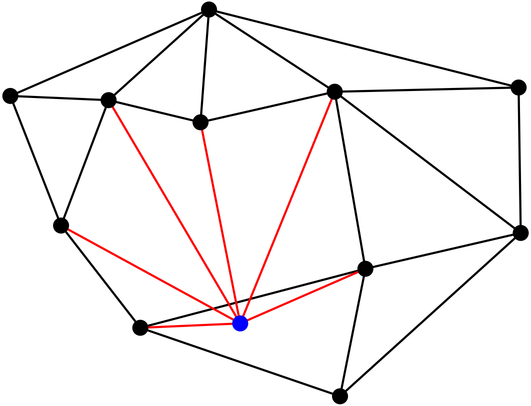
Check combinatorial consistency



Solution 1 - Forget about geometric theorems

Insertion in a Delaunay triangulation

Check combinatorial consistency

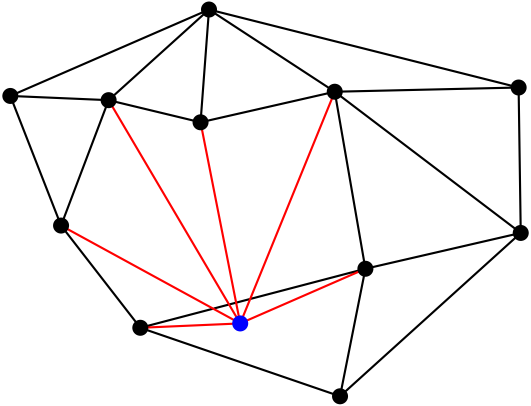


The result is a combinatorial triangulation

Solution 1 - Forget about geometric theorems

Insertion in a Delaunay triangulation

Check combinatorial consistency

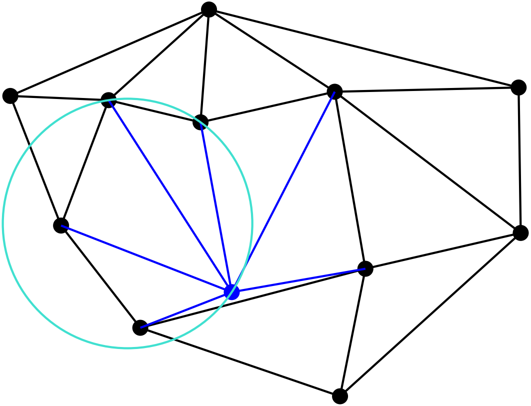


The result is a combinatorial triangulation
but the geometric embedding may have intersecting edges

Solution 1 - Forget about geometric theorems

Insertion in a Delaunay triangulation

Check combinatorial consistency

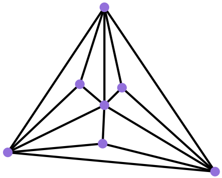


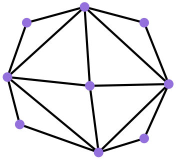
It may not be Delaunay

Solution 1 - Forget about geometric theorems

Insertion in a Delaunay triangulation

Check combinatorial consistency





It may be non combinatorially (strictly) Delaunay

Solution 1 - Forget about geometric theorems

- hardly ever used
- not used any more

Solution 2 - Pretend that we can compute on real numbers

Maybe use a subset of the real numbers

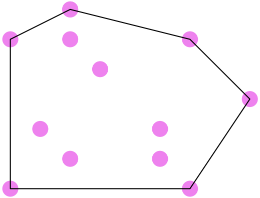
- rational numbers
- algebraic numbers
- ...?

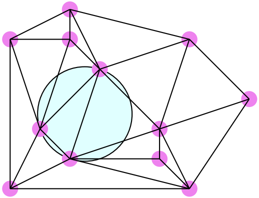
Exact geometric computing

Arithmetic issues — Exact Geometric Computing

Predicates

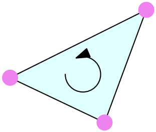
Result of a combinatorial algorithm

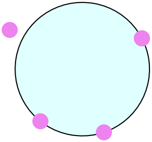




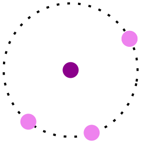
depends on decisions based on the evaluation of predicates:

set of points $\mapsto \{-1, 0, 1\}$





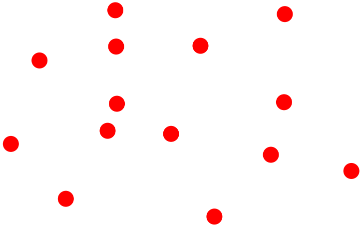
Predicates vs. constructions



construction of a new geometric object

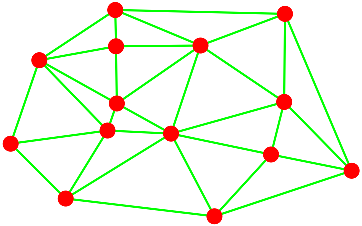
Predicates vs. constructions

Algorithms



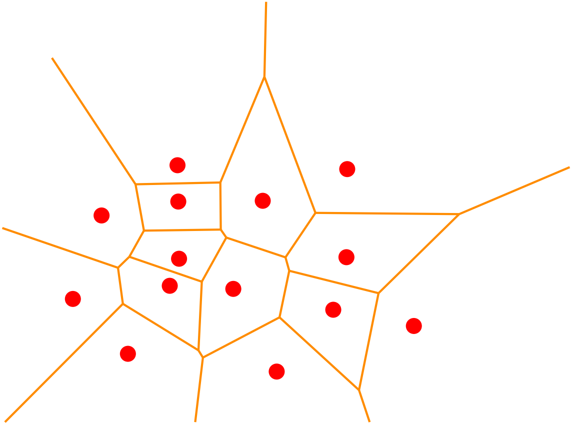
Predicates vs. constructions

Algorithms
based on predicates



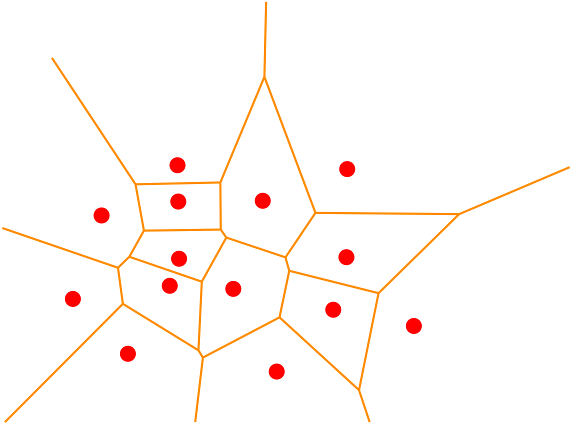
Predicates vs. constructions

Algorithms
based on constructions



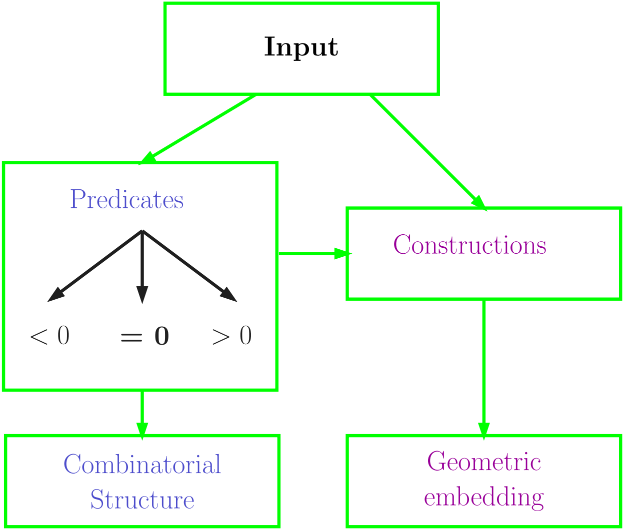
Predicates vs. constructions

Algorithms
based on constructions



⟶ the underlying combinatorial structure only needs predicates

Predicates vs. constructions

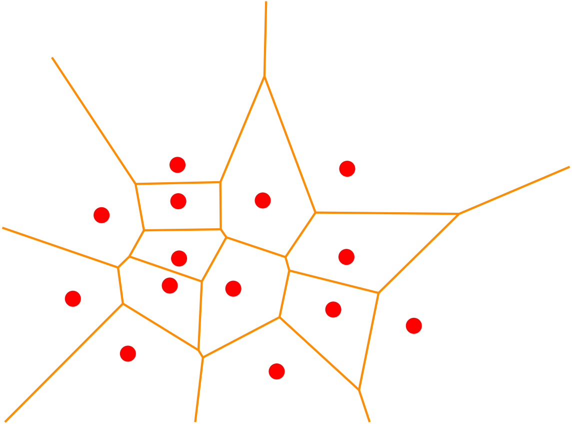


Exact geometric computing

Exact arithmetics ?

Exact geometric computing

~~Exact arithmetics~~ $\longrightarrow$ Exact predicates

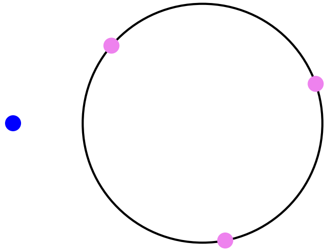


sufficient to get a correct underlying combinatorial structure
constructions can be approximate

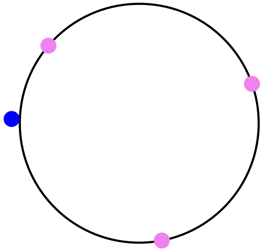
Exact geometric computing

~~Exact arithmetics~~ ⟶ Exact predicates

filtered computations



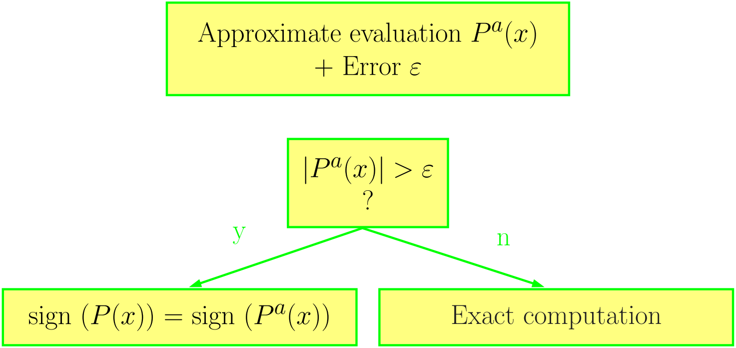
easy cases
exact computation
unnecessary



difficult cases
exact computation
necessary

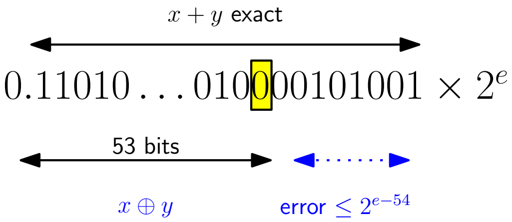
Filtering

Predicate = sign of a polynomial expression $P(x)$
(x = coordinates)



easy cases are more frequent
$\Longrightarrow$ cost $\simeq$ cost of approximate (`double`) computation

Controlling the error



$x + y = x \oplus y + \varepsilon_{x+y}$

$2^{e-1} \leq |x + y| < 2^e \leq 2|x + y| \Longrightarrow$

$$\varepsilon_{x+y} \leq 2^{e-54} \leq |x + y|\ 2^{-53}$$

and similarly for other operations

Static filtering

Known bound on all input coordinates: $|x_i|, |y_i| \leq M$

Static filtering

Known bound on all input coordinates: $|x_i|, |y_i| \leq M$

Example: orientation predicate

$$\begin{vmatrix} x_q - x_p & x_r - x_p \\ y_q - y_p & y_r - y_p \end{vmatrix}$$

Static filtering

Known bound on all input coordinates: $|x_i|, |y_i| \leq M$

Example: orientation predicate

$$\begin{vmatrix} x_q - x_p & x_r - x_p \\ y_q - y_p & y_r - y_p \end{vmatrix}$$

$x_i - x_p = x_i \ominus x_p + \varepsilon_{x_i - x_p}$
$|x_i - x_p| \leq 2M$
$\varepsilon_{x_i - x_p} \leq |x_i - x_p| \, 2^{-53} \leq 2M \, 2^{-53} \leq 2^{-52} M$

Static filtering

Known bound on all input coordinates: $|x_i|, |y_i| \leq M$

Example: orientation predicate

$$\begin{vmatrix} x_q - x_p & x_r - x_p \\ y_q - y_p & y_r - y_p \end{vmatrix}$$

$x_i - x_p = x_i \ominus x_p + \varepsilon_{x_i - x_p}$
$|x_i - x_p| \leq 2M$
$\varepsilon_{x_i - x_p} \leq |x_i - x_p|\, 2^{-53} \leq 2M\, 2^{-53} \leq 2^{-52} M$

$$\begin{aligned}
(x_i - x_p)(y_i - y_p) &= (x_i - x_p) \otimes (y_i - y_p) + \varepsilon_\otimes \\
&= (x_i \ominus x_p + \varepsilon_{x_i - x_p}) \otimes (y_i \ominus y_p + \varepsilon_{y_i - y_p}) + \varepsilon_\otimes \\
&= \ldots \\
|(x_i - x_p)(y_i - y_p)| &\leq 4M^2 \\
\varepsilon_{(x_i - x_p)(y_i - y_p)} &\leq 2(2M\, 2^{-52} M) + 2^{-53} 4M^2 \\
&= 3 \cdot 2^{-51} M^2
\end{aligned}$$

Static filtering

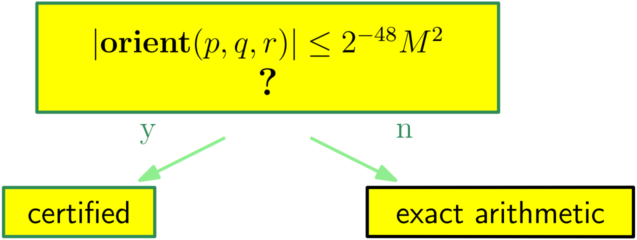
$|(x_i - x_p)(y_i - y_p)| \leq 4M^2$
$\varepsilon_{(x_i - x_p)(y_i - y_p)} \leq 3 \cdot 2^{-51} M^2$

$$
\begin{aligned}
\text{orient}(p,q,r) &= \begin{vmatrix} x_q - x_p & x_r - x_p \\ y_q - y_p & y_r - y_p \end{vmatrix} \\
|\text{orient}(p,q,r)| &\leq 8M^2 \\
\varepsilon_{\text{orient}(p,q,r)} &\leq 2^{-53} 8M^2 + 2 \cdot 3 \cdot 2^{-51} M^2 \leq 2^{-48} M^2
\end{aligned}
$$

Static filtering

$|(x_i - x_p)(y_i - y_p)| \leq 4M^2$
$\varepsilon_{(x_i - x_p)(y_i - y_p)} \leq 3 \cdot 2^{-51} M^2$

$$\begin{aligned} \text{orient}(p,q,r) &= \begin{vmatrix} x_q - x_p & x_r - x_p \\ y_q - y_p & y_r - y_p \end{vmatrix} \\ |\text{orient}(p,q,r)| &\leq 8M^2 \\ \varepsilon_{\text{orient}(p,q,r)} &\leq 2^{-53} 8M^2 + 2 \cdot 3 \cdot 2^{-51} M^2 \leq 2^{-48} M^2 \end{aligned}$$



Static filtering

—

- hypotheses on input data
- restricted set of operations
- error bounds computed manually(?)

Static filtering

−

- hypotheses on input data
- restricted set of operations
- error bounds computed manually(?)

+

- error bounds pre-computed $\longrightarrow$ very fast
- reasonable success rate

Dynamic filtering

Interval arithmetic

a non representable $\quad\mapsto\quad [\,\underline{a}, \overline{a}\,]$

$$[\,\underline{a}, \overline{a}\,] + [\,\underline{b}, \overline{b}\,] = [\,\underline{a} \,\underline{+}\, \underline{a}\,,\ \overline{a} \,\overline{+}\, \overline{b}\,]$$

... error propagation ...

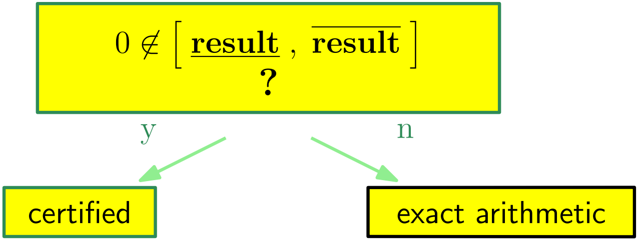
Dynamic filtering

Interval arithmetic

a non representable $\mapsto$ $[\,\underline{a}, \overline{a}\,]$

$$[\,\underline{a}, \overline{a}\,] + [\,\underline{b}, \overline{b}\,] = [\,\underline{a}\ \underline{+}\ \underline{a}\ ,\ \overline{a}\ \overline{+}\ \overline{b}\,]$$

. . . error propagation . . .



Dynamic filtering

Programming

```
template <class FT>
Orientation orientation
     ( FT px, FT py, FT qx, FT qy, FT rx, FT ry)
{
   return sign( (qx-px)*(ry-py)-(rx-px)*(qy-py) );
}
```

`FT` = “any” number type, including `interval`

Dynamic filtering

—

- slower
  - error computed at runtime
  - 2 changes of rounding mode for each predicate call

Dynamic filtering

−

- slower
  - error computed at runtime
  - 2 changes of rounding mode for each predicate call

+

- no hypotheses on input data
- excellent success rate
- large set of operations (if intervals can be computed)

Remarks

- Static and dynamic filtering can be combined

Remarks

- Static and dynamic filtering can be combined
- # of arithmetic operations
  $\longrightarrow$ constant in $O(\ )$

Remarks

- Static and dynamic filtering can be combined
- # of arithmetic operations
  $\longrightarrow$ constant in $O(\ )$
- Degree of predicate
  $\longrightarrow$ size of numbers for exact arithmetic
  $\longrightarrow$ precision of rounding

Conclusion

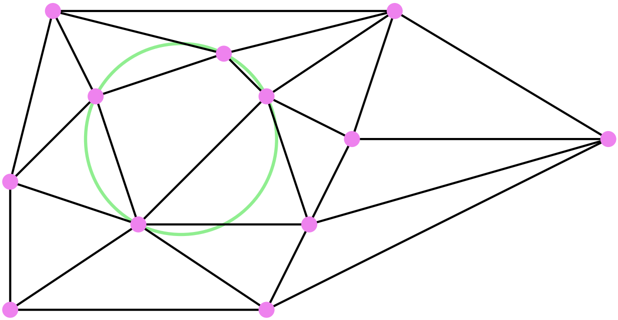
Exact predicates

- a way to ensure consistency
- does not mean exact arithmetic

Degenerate cases

“The” Delaunay “triangulation”?

A polygonization?

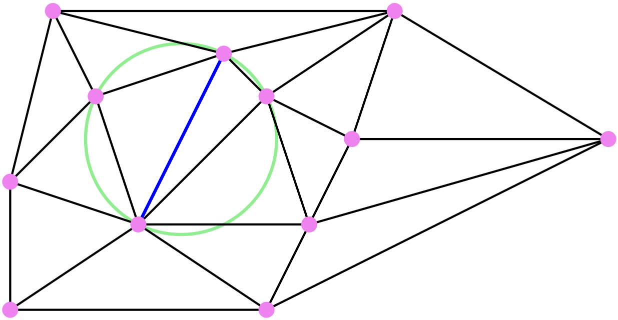


would require to treat all degenerate cases explicitly
and more general data structures ⇝ non-constant access

"The" Delaunay "triangulation"?

A polygonization?

Triangulation not uniquely defined

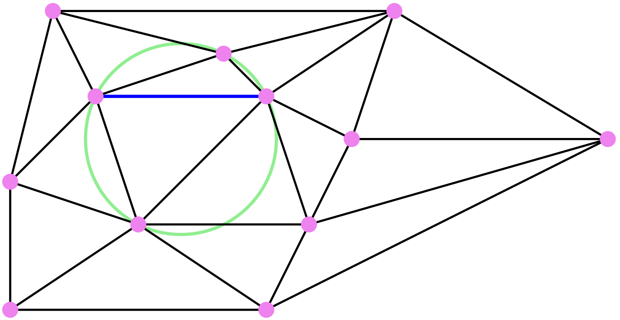


requires consistent choices

"The" Delaunay "triangulation"?

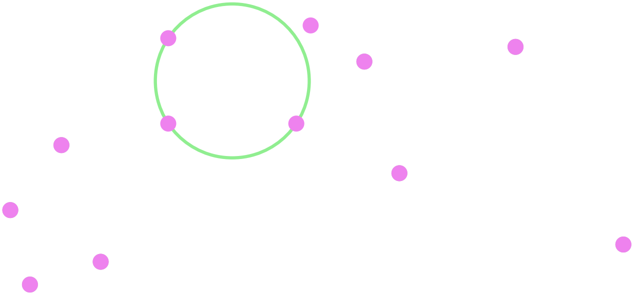
A polygonization?

Triangulation not uniquely defined



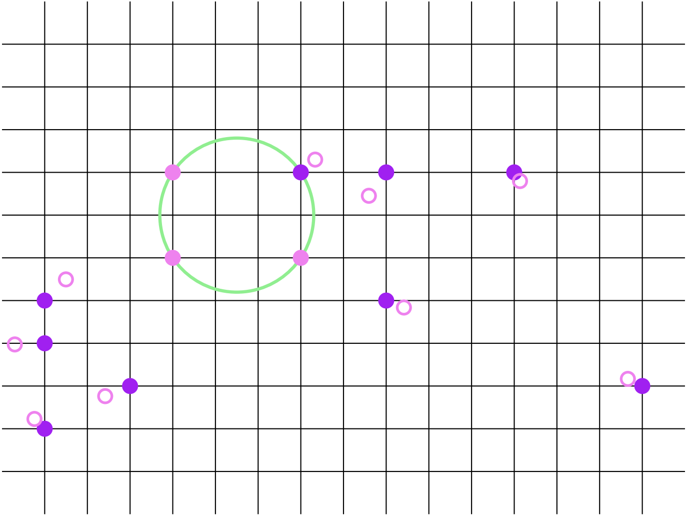
requires consistent choices

It never happens in practice

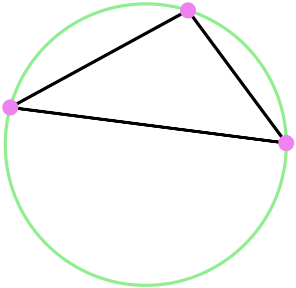


It ~~never~~ happens in practice

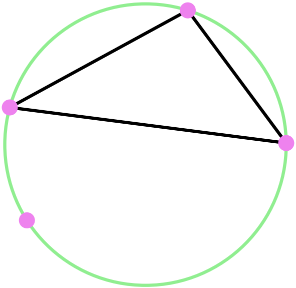
input data are discrete



Simulating the absence of degeneracies

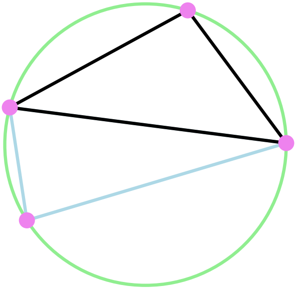


Simulating the absence of degeneracies



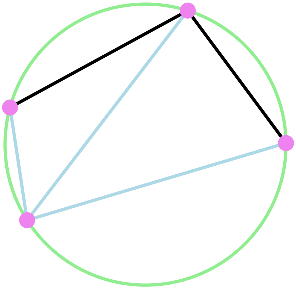
?

Simulating the absence of degeneracies



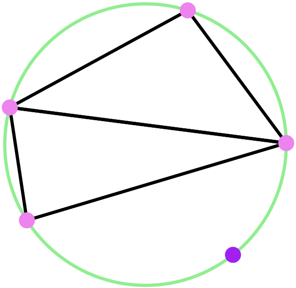
? *as if* new point "outside disk"

Simulating the absence of degeneracies



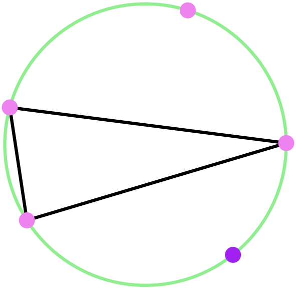
? *as if* new point "in disk"

Random Choices?



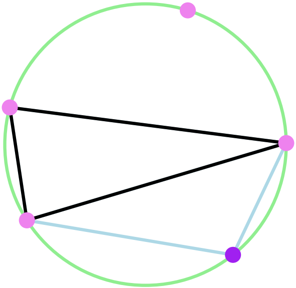
?

Random Choices?



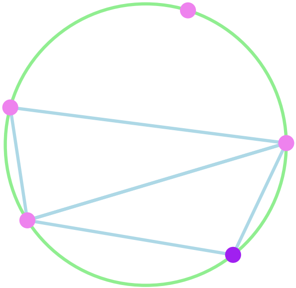
?

Random Choices?



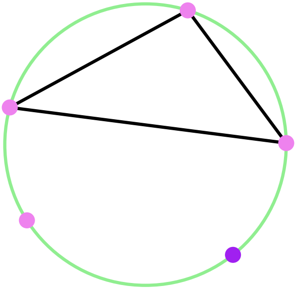
random choice: new point “outside”

Random Choices?



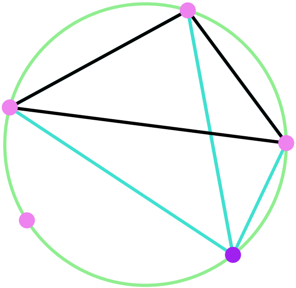
random choice: new point “outside”

Random Choices?



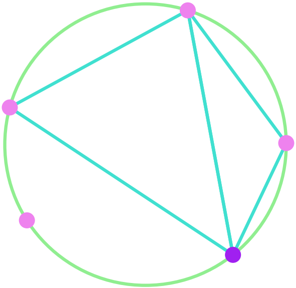
?

Random Choices?



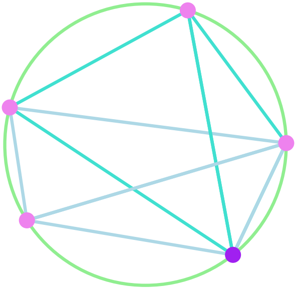
random choice: new point “inside”

Random Choices?



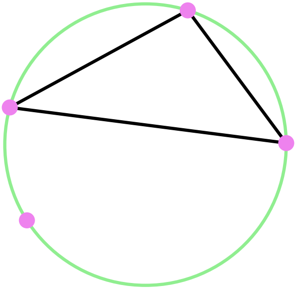
random choice: new point “inside”

Random Choices?



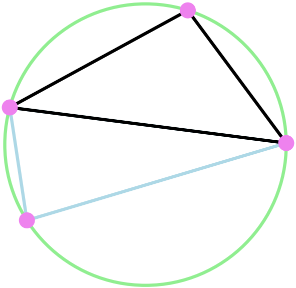
inconsistency

A strategy?



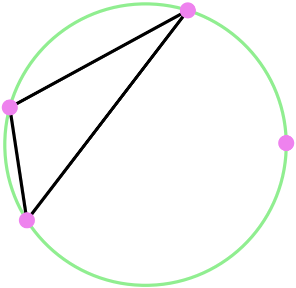
?

A strategy?



strategy: new point “outside”

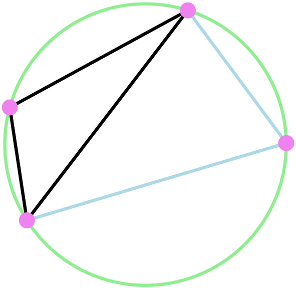
A strategy?



strategy: new point “outside”

same points, different order

A strategy?



strategy: new point “outside”

same points, different order

$\longrightarrow$ different results

(and still sometimes inconsistencies)

Controlled perturbations

A way to ensure consistent choices

Degenerate input

Data perturbed by small epsilons

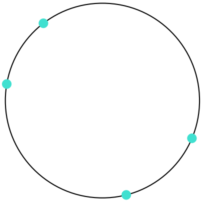
Requires careful control of epsilons
to ensure that perturbed data is non-degenerate

Symbolic perturbations

A way to ensure consistent choices

Input data $\mapsto$ data depending on a symbolic parameter ε

- $\varepsilon = 0$: (maybe) degenerate problem

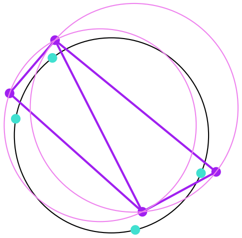


Symbolic perturbations

A way to ensure consistent choices

Input data $\mapsto$ data depending on a symbolic parameter ε

- $\varepsilon = 0$: (maybe) degenerate problem
- $\varepsilon \neq 0$: non-degenerate problem $\mapsto$ Result(ε)

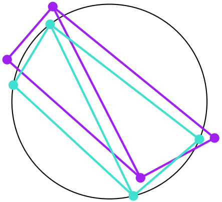


Symbolic perturbations

A way to ensure consistent choices

Input data $\mapsto$ data depending on a symbolic parameter ε

- $\varepsilon = 0$: (maybe) degenerate problem
- $\varepsilon \neq 0$: non-degenerate problem $\mapsto$ Result(ε)



Final result = $\lim_{\varepsilon \to 0^+}$ Result(ε)

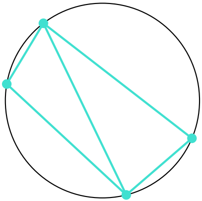
Result is discrete
constant for small ε

Symbolic perturbations

A way to ensure consistent choices

Input data $\mapsto$ data depending on a symbolic parameter ε

- $\varepsilon = 0$: (maybe) degenerate problem
- $\varepsilon \neq 0$: non-degenerate problem $\mapsto$ Result(ε)



Final result $= \lim_{\varepsilon \to 0^+}$ Result(ε)

Result is discrete
constant for small ε

SoS = Simulation of Simplicity

Input: n points $p_i = (x_i, y_i), i = 1, \ldots, n$

Perturb data: $\forall i, (x_i, y_i) \mapsto (x_i, y_i) + \varepsilon^{2^i}(i, i^2)$

SoS = Simulation of Simplicity

Input: n points $p_i = (x_i, y_i), i = 1, \ldots, n$

Perturb data: $\forall i, (x_i, y_i) \mapsto (x_i, y_i) + \varepsilon^{2^i}(i, i^2)$

Example: $\text{orient}(O, p_i, p_j) = \text{sign}\begin{vmatrix} x_i & x_j \\ y_i & y_j \end{vmatrix}$

$i = 3, j = 1,\ \text{sign}\begin{vmatrix} x_3 & x_1 \\ y_3 & y_1 \end{vmatrix}$

$$\begin{vmatrix} x_3 + 3\varepsilon^8 & x_1 + \varepsilon^2 \\ y_3 + 9\varepsilon^8 & y_1 + \varepsilon^2 \end{vmatrix} = \begin{vmatrix} x_3 & x_1 \\ y_3 & y_1 \end{vmatrix} + \varepsilon^2 \begin{vmatrix} x_3 & 1 \\ y_3 & 1 \end{vmatrix} + \varepsilon^8 \begin{vmatrix} 3 & x_1 \\ 9 & y_1 \end{vmatrix} + \varepsilon^{10} \begin{vmatrix} 3 & 1 \\ 9 & 1 \end{vmatrix}$$

non-null polynomial

sign = sign of first non-null coefficient

SoS = Simulation of Simplicity

+

- general method (?)

SoS = Simulation of Simplicity

$+$

- general method (?)

$-$

- requires to check that it works for each necessary predicate
- heavy
- depends on an indexing of the points

Perturbing the world

Example: comparison of abcissae
$\mathrm{sign}(x_2 - x_1)$

$$(x, y) \mapsto (x + \varepsilon y, y)$$

$$x_2 - x_1 \mapsto (x_2 + \varepsilon y_2) - (x_1 + \varepsilon y_1) =$$

$$(x_2 - x_1) + \varepsilon(y_2 - y_1)$$

non-null polynomial (except for equal points)

Perturbing the world for Delaunay

Orientation predicate

$$(x, y) \mapsto (x + \varepsilon y, y + \varepsilon^2 x + \varepsilon^3(x^2 + y^2))$$

$$\begin{vmatrix} 1 & x_p & y_p \\ 1 & x_q & y_q \\ 1 & x_r & y_r \end{vmatrix}$$

$$\mapsto$$

$$(1 - \varepsilon^3) \begin{vmatrix} 1 & x_p & y_p \\ 1 & x_q & y_q \\ 1 & x_r & y_r \end{vmatrix} + \varepsilon^3 \begin{vmatrix} 1 & x_p & x_p^2 + y_p^2 \\ 1 & x_q & x_q^2 + y_q^2 \\ 1 & x_r & x_r^2 + y_r^2 \end{vmatrix} + \varepsilon^4 \begin{vmatrix} 1 & y_p & x_p^2 + y_p^2 \\ 1 & y_q & x_q^2 + y_q^2 \\ 1 & y_r & x_r^2 + y_r^2 \end{vmatrix}$$

Perturbing the world for Delaunay

Orientation predicate

$p^\star, q^\star, r^\star$: projections of p, q, r on the unit paraboloid

null polynomial $\Longrightarrow$ the projections of $p^\star, q^\star, r^\star$

- on the (x, y)-plane
- on the (x, z)-plane
- on the (y, z)-plane

are collinear $\Longrightarrow$ $p^\star, q^\star, r^\star$ are collinear

$\Longrightarrow$ 2 points among $p^\star, q^\star, r^\star$ are equal

$$(1-\varepsilon^3)\begin{vmatrix} 1 & x_p & y_p \\ 1 & x_q & y_q \\ 1 & x_r & y_r \end{vmatrix} + \varepsilon^3 \begin{vmatrix} 1 & x_p & x_p^2+y_p^2 \\ 1 & x_q & x_q^2+y_q^2 \\ 1 & x_r & x_r^2+y_r^2 \end{vmatrix} + \varepsilon^4 \begin{vmatrix} 1 & y_p & x_p^2+y_p^2 \\ 1 & y_q & x_q^2+y_q^2 \\ 1 & y_r & x_r^2+y_r^2 \end{vmatrix}$$

Perturbing the world for Delaunay

in_disk predicate

$$(x, y) \mapsto (x + \varepsilon y, y + \varepsilon^2 x + \varepsilon^3 (x^2 + y^2))$$

$$\begin{vmatrix} 1 & x_p & y_p & x_p^2 + y_p^2 \\ 1 & x_q & y_q & x_q^2 + y_q^2 \\ 1 & x_r & y_r & x_r^2 + y_r^2 \\ 1 & x_s & y_s & x_s^2 + y_s^2 \end{vmatrix}$$

$\mapsto$

$$\begin{vmatrix} 1 & x_p & y_p & x_p^2 + y_p^2 \\ 1 & x_q & y_q & x_q^2 + y_q^2 \\ 1 & x_r & y_r & x_r^2 + y_r^2 \\ 1 & x_s & y_s & x_s^2 + y_s^2 \end{vmatrix} + 2\varepsilon \begin{vmatrix} 1 & x_p & y_p & x_p y_p \\ 1 & x_q & y_q & x_q y_q \\ 1 & x_r & y_r & x_r y_r \\ 1 & x_s & y_s & x_s y_s \end{vmatrix}$$

$$+\varepsilon^2 \begin{vmatrix} 1 & x_p & y_p & y_p^2 \\ 1 & x_q & y_q & y_q^2 \\ 1 & x_r & y_r & y_r^2 \\ 1 & x_s & y_s & y_s^2 \end{vmatrix} + 2\varepsilon^2 \begin{vmatrix} 1 & x_p & y_p & x_p y_p \\ 1 & x_q & y_q & x_q y_q \\ 1 & x_r & y_r & x_r y_r \\ 1 & x_s & y_s & x_s y_s \end{vmatrix} + \varepsilon^3 D_{pqrs}(\varepsilon)$$

Perturbing the world for Delaunay

in_disk predicate

$$(x, y) \mapsto (x + \varepsilon y, y + \varepsilon^2 x + \varepsilon^3 (x^2 + y^2))$$

Assume null polynomial

$$\begin{vmatrix} 1 & x_p & y_p & x_p^2 + y_p^2 \\ 1 & x_q & y_q & x_q^2 + y_q^2 \\ 1 & x_r & y_r & x_r^2 + y_r^2 \\ 1 & x_s & y_s & x_s^2 + y_s^2 \end{vmatrix} + 2\varepsilon \begin{vmatrix} 1 & x_p & y_p & x_p y_p \\ 1 & x_q & y_q & x_q y_q \\ 1 & x_r & y_r & x_r y_r \\ 1 & x_s & y_s & x_s y_s \end{vmatrix}$$

$$+\varepsilon^2 \begin{vmatrix} 1 & x_p & y_p & y_p^2 \\ 1 & x_q & y_q & y_q^2 \\ 1 & x_r & y_r & y_r^2 \\ 1 & x_s & y_s & y_s^2 \end{vmatrix} + 2\varepsilon^2 \begin{vmatrix} 1 & x_p & y_p & x_p y_p \\ 1 & x_q & y_q & x_q y_q \\ 1 & x_r & y_r & x_r y_r \\ 1 & x_s & y_s & x_s y_s \end{vmatrix} + \varepsilon^3 D_{pqrs}(\varepsilon)$$

Perturbing the world for Delaunay

in_disk predicate

Let $\mathcal{C}$ be a conic through p, q, r

$$\mathcal{C}(x, y) = ax^2 + by^2 + cxy + dx + ey + f = 0$$

$$\begin{vmatrix} 1 & x_p & y_p & \mathcal{C}(p) \\ 1 & x_q & y_q & \mathcal{C}(q) \\ 1 & x_r & y_r & \mathcal{C}(r) \\ 1 & x_s & y_s & \mathcal{C}(s) \end{vmatrix}$$

Perturbing the world for Delaunay

in_disk predicate

Let $\mathcal{C}$ be a conic through p, q, r

$$\mathcal{C}(x, y) = ax^2 + by^2 + cxy + dx + ey + f = 0$$

$$\begin{vmatrix} 1 & x_p & y_p & \mathcal{C}(p) \\ 1 & x_q & y_q & \mathcal{C}(q) \\ 1 & x_r & y_r & \mathcal{C}(r) \\ 1 & x_s & y_s & \mathcal{C}(s) \end{vmatrix}$$

$$= a \begin{vmatrix} 1 & x_p & y_p & x_p^2 \\ 1 & x_q & y_q & x_q^2 \\ 1 & x_r & y_r & x_r^2 \\ 1 & x_s & y_s & x_s^2 \end{vmatrix} + b \begin{vmatrix} 1 & x_p & y_p & y_p^2 \\ 1 & x_q & y_q & y_q^2 \\ 1 & x_r & y_r & y_r^2 \\ 1 & x_s & y_s & y_s^2 \end{vmatrix} + c \begin{vmatrix} 1 & x_p & y_p & x_p y_p \\ 1 & x_q & y_q & x_q y_q \\ 1 & x_r & y_r & x_r y_r \\ 1 & x_s & y_s & x_s y_s \end{vmatrix}$$

$$+d \cdot 0 + e \cdot 0 + f \cdot 0$$

Perturbing the world for Delaunay

in_disk predicate

Let $\mathcal{C}$ be a conic through p, q, r

$$\mathcal{C}(x,y) = ax^2 + by^2 + cxy + dx + ey + f = 0$$

$$\begin{vmatrix} 1 & x_p & y_p & \mathcal{C}(p) \\ 1 & x_q & y_q & \mathcal{C}(q) \\ 1 & x_r & y_r & \mathcal{C}(r) \\ 1 & x_s & y_s & \mathcal{C}(s) \end{vmatrix} = 0$$

$$= a \begin{vmatrix} 1 & x_p & y_p & x_p^2 \\ 1 & x_q & y_q & x_q^2 \\ 1 & x_r & y_r & x_r^2 \\ 1 & x_s & y_s & x_s^2 \end{vmatrix} + b \begin{vmatrix} 1 & x_p & y_p & y_p^2 \\ 1 & x_q & y_q & y_q^2 \\ 1 & x_r & y_r & y_r^2 \\ 1 & x_s & y_s & y_s^2 \end{vmatrix} + c \begin{vmatrix} 1 & x_p & y_p & x_p y_p \\ 1 & x_q & y_q & x_q y_q \\ 1 & x_r & y_r & x_r y_r \\ 1 & x_s & y_s & x_s y_s \end{vmatrix}$$

$$+ d \cdot 0 + e \cdot 0 + f \cdot 0$$

Perturbing the world for Delaunay

in_disk predicate

Let $\mathcal{C}$ be a conic through p, q, r

$$\mathcal{C}(x,y) = ax^2 + by^2 + cxy + dx + ey + f = 0$$

$$\begin{vmatrix} 1 & x_p & y_p & \mathcal{C}(p) \\ 1 & x_q & y_q & \mathcal{C}(q) \\ 1 & x_r & y_r & \mathcal{C}(r) \\ 1 & x_s & y_s & \mathcal{C}(s) \end{vmatrix} = 0$$

$$= \begin{vmatrix} 1 & x_p & y_p & 0 \\ 1 & x_q & y_q & 0 \\ 1 & x_r & y_r & 0 \\ 1 & x_s & y_s & \mathcal{C}(s) \end{vmatrix} = \mathcal{C}(s) \begin{vmatrix} 1 & x_p & y_p \\ 1 & x_q & y_q \\ 1 & x_r & y_r \end{vmatrix} \implies$$

- $s \in \mathcal{C} \ \forall \mathcal{C}$
- or p, q, r collinear

Perturbing the world for Delaunay

in_disk predicate

Let $\mathcal{C}$ be a conic through p, q, r

$$\mathcal{C}(x,y) = ax^2 + by^2 + cxy + dx + ey + f = 0$$

$$\begin{vmatrix} 1 & x_p & y_p & \mathcal{C}(p) \\ 1 & x_q & y_q & \mathcal{C}(q) \\ 1 & x_r & y_r & \mathcal{C}(r) \\ 1 & x_s & y_s & \mathcal{C}(s) \end{vmatrix} = 0$$

$$= \begin{vmatrix} 1 & x_p & y_p & 0 \\ 1 & x_q & y_q & 0 \\ 1 & x_r & y_r & 0 \\ 1 & x_s & y_s & \mathcal{C}(s) \end{vmatrix} = \mathcal{C}(s) \begin{vmatrix} 1 & x_p & y_p \\ 1 & x_q & y_q \\ 1 & x_r & y_r \end{vmatrix} \Longrightarrow$$

- $s \in \mathcal{C} \ \forall \mathcal{C}$
- or p, q, r collinear

$\Longrightarrow p, q, r, s$ collinear

Perturbing the world for Delaunay

in_disk predicate

Let $\mathcal{C}$ be a conic through p, q, r

$$\mathcal{C}(x, y) = ax^2 + by^2 + cxy + dx + ey + f = 0$$

$$\begin{vmatrix} 1 & x_p & y_p & \mathcal{C}(p) \\ 1 & x_q & y_q & \mathcal{C}(q) \\ 1 & x_r & y_r & \mathcal{C}(r) \\ 1 & x_s & y_s & \mathcal{C}(s) \end{vmatrix} = 0$$

p, q, r, s cannot be both collinear and cocircular

$\longrightarrow$ contradiction

the perturbed in_disk polynomial is non-null

Perturbing the world

+

- general method (?)
- does not need points to be indexed
- lighter computations

Perturbing the world

+

- general method (?)
- does not need points to be indexed
- lighter computations

—

- requires to check that it works for each necessary predicate

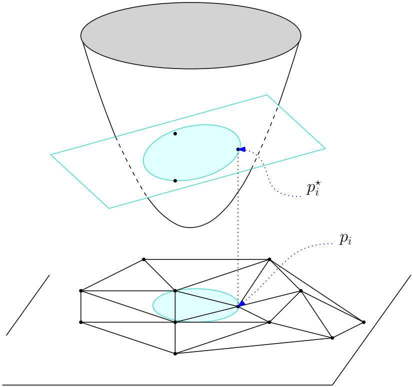
Main drawback for Delaunay

All predicates are perturbed by all the above perturbations

including orientation

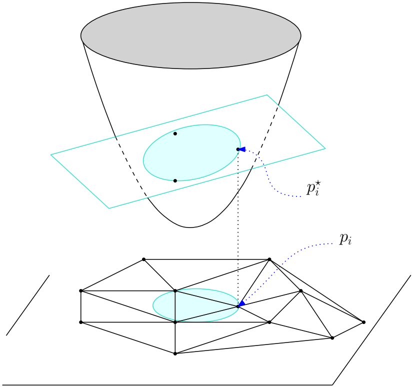
$\mapsto$ these perturbations can lead to flat tetrahedra

Perturbing points in the $d + 1^{th}$ dimension



$$p_i = (x_i, y_i)$$
$$p_i^\star = (x_i, y_i, t_i = x_i^2 + y_i^2)$$

Perturbing points in the $d+1^{th}$ dimension

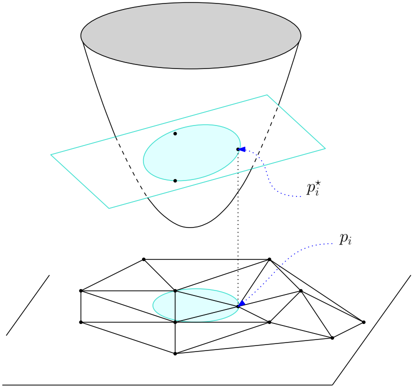


$$p_i = (x_i, y_i)$$
$$p_i^\star = (x_i, y_i, t_i = x_i^2 + y_i^2)$$

$$\text{in\_disk}(p_i, p_j, p_k, p_l) = \frac{\mathcal{D}(p_i,p_j,p_k,p_l)}{\text{orient}(p_i,p_j,p_k)}$$

$$\mathcal{D}(p_i, p_j, p_k, p_l) = \text{orient}(p_i^\star, p_j^\star, p_k^\star, p_l^\star)$$

Perturbing points in the $d + 1^{th}$ dimension



$p_i = (x_i, y_i)$
$p_i^\star = (x_i, y_i, t_i = x_i^2 + y_i^2)$

$\mapsto p^{\star_i\varepsilon} = (x_i, y_i, t_i + \varepsilon^{n-i})$

$\text{in\_disk}(p_i, p_j, p_k, p_l) = \frac{\mathcal{D}(p_i,p_j,p_k,p_l)}{\text{orient}(p_i,p_j,p_k)}$

$\mathcal{D}(p_i, p_j, p_k, p_l) = \text{orient}(p_i^\star, p_j^\star, p_k^\star, p_l^\star)$

$\mapsto \text{orient}(p^{\star_i\varepsilon}, p^{\star_j\varepsilon}, p^{\star_k\varepsilon}, p^{\star_l\varepsilon})$

Perturbing points in the $d+1^{th}$ dimension

$$\text{orient}(p^{\star_i\varepsilon}, p^{\star_j\varepsilon}, p^{\star_k\varepsilon}, p^{\star_l\varepsilon}) = \begin{vmatrix} 1 & 1 & 1 & 1 \\ x_i & x_j & x_k & x_l \\ y_i & y_j & y_k & y_l \\ z_i & z_j & z_k & z_l \\ t_i+\varepsilon^{n-i} & t_j+\varepsilon^{n-j} & t_k+\varepsilon^{n-k} & t_l+\varepsilon^{n-l} \end{vmatrix}$$

$$\begin{aligned} = \mathcal{D}(p_i, p_j, p_k, p_l) \\ & -\text{orient}(p_i, p_j, p_k)\varepsilon^{n-l} \\ & +\text{orient}(p_i, p_j, p_l)\varepsilon^{n-k} \\ & -\text{orient}(p_i, p_k, p_l)\varepsilon^{n-j} \\ & +\text{orient}(p_j, p_k, p_l)\varepsilon^{n-i} \end{aligned}$$

4 cocircular points $\longrightarrow$ non-null polynomial in ε

Perturbing points in the $d+1^{th}$ dimension

$$\text{orient}(p^{\star_i\varepsilon}, p^{\star_j\varepsilon}, p^{\star_k\varepsilon}, p^{\star_l\varepsilon}) = \begin{vmatrix} 1 & 1 & 1 & 1 \\ x_i & x_j & x_k & x_l \\ y_i & y_j & y_k & y_l \\ z_i & z_j & z_k & z_l \\ t_i+\varepsilon^{n-i} & t_j+\varepsilon^{n-j} & t_k+\varepsilon^{n-k} & t_l+\varepsilon^{n-l} \end{vmatrix}$$

$$\begin{aligned} = \mathcal{D}(p_i, p_j, p_k, p_l) & \\ & -\text{orient}(p_i, p_j, p_k)\varepsilon^{n-l} \\ & +\text{orient}(p_i, p_j, p_l)\varepsilon^{n-k} \\ & -\text{orient}(p_i, p_k, p_l)\varepsilon^{n-j} \\ & +\text{orient}(p_j, p_k, p_l)\varepsilon^{n-i} \end{aligned}$$

4 cocircular points $\longrightarrow$ non-null polynomial in ε

point with highest index
in the disk of the other 3

Perturbing points in the $d+1^{th}$ dimension

global indexing = lexicographic order

Delaunay triangulation uniquely defined

Perturbing points in the $d+1^{th}$ dimension

global indexing = lexicographic order

Delaunay triangulation uniquely defined

+

- NO flat simplex
- easy to implement
- does not need points to be globally sorted
- lighter computations

Perturbing points in the $d + 1^{th}$ dimension

global indexing = lexicographic order

Delaunay triangulation uniquely defined

$+$

- NO flat simplex
- easy to implement
- does not need points to be globally sorted
- lighter computations

$-$

- specific for (weighted) Delaunay triangulations

Perturbing points in the $d + 1^{th}$ dimension

global indexing = lexicographic order

Delaunay triangulation uniquely defined

+

- NO flat simplex
- easy to implement
- does not need points to be globally sorted
- lighter computations

—

- specific for (weighted) Delaunay triangulations
- result is not (always) the Delaunay triangulation of a non-degenerate set of points
  weighted Delaunay triangulation instead