

# Teaching Computational Geometry, II

Raimund Seidel

Saarland University

Invited Talk at the  
21st Annual Canadian Conference  
on Computational Geometry, CCCG09,  
August 19th, 2009,  
University of British Columbia, Vancouver, Canada



1978 Shamos's PhD thesis

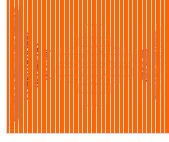
1978 Shamos's PhD thesis

~1980 first CG courses

1978 Shamos's PhD thesis

~1980 first CG courses

1985 1<sup>st</sup> ACM SOCG

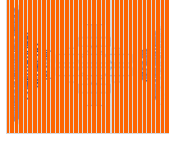


1978 Shamos's PhD thesis

~1980 first CG courses

1985 1<sup>st</sup> ACM SOCG

1986 D & CG

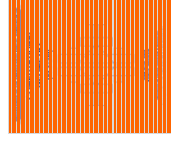


1978 Shamos's PhD thesis

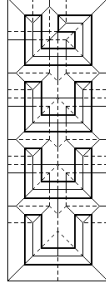
~1980 first CG courses

1985 1<sup>st</sup> ACM SOCG

1986 D & CG



1989 1<sup>st</sup> CCCG

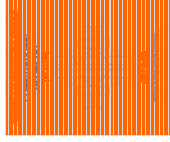


1978 Shamos's PhD thesis

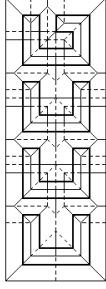
~1980 first CG courses

1985 1<sup>st</sup> ACM SOCG

1986 D & CG



1989 1<sup>st</sup> CCCG



1991 CGTA



IJCGA



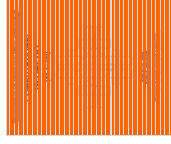


1978 Shamos's PhD thesis

~1980 first CG courses

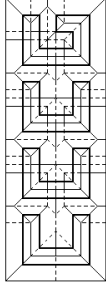
1985 1<sup>st</sup> ACM SOCG

1986 D & CG



1989

1<sup>st</sup> CCCG



1991

CGTA



IJCGA



2009

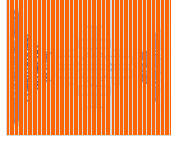
Journal of Computational Geometry

1978 Shamos's PhD thesis

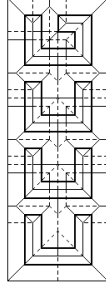
~1980 first CG courses

1985 1<sup>st</sup> ACM SOCG

1986 D & CG



1989 1<sup>st</sup> CCCG



1991 CGTA



IJCGA



1993 Teaching Computational Geometry, I

2009 Journal of Computational Geometry

# TEACHING COMPUTATIONAL GEOMETRY

*Raimund Seidel*

*U.C. Berkeley*



# TEACHING COMPUTATIONAL GEOMETRY

*Raimund Seidel*

*U.C. Berkeley*

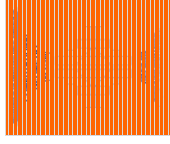
CCCG 1993  
Waterloo

1978 Shamos's PhD thesis

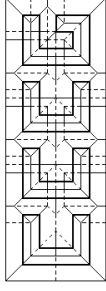
~1980 first CG courses

1985 1<sup>st</sup> ACM SOCG

1986 D & CG



1989 1<sup>st</sup> CCCG



1991 CGTA



IJCGA



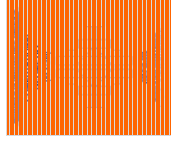
1993 Teaching Computational Geometry, I

2009 Journal of Computational Geometry

1978 Shamos's PhD thesis

~1980 first CG courses

1984



1985 1<sup>st</sup> ACM SOCG

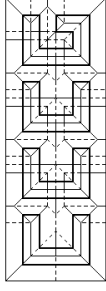


1986 D & CG



Mehlhorn

1989



1<sup>st</sup> CCCG

1991



CGTA



IJCGA

1993 Teaching Computational Geometry, I

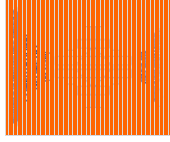
2009

Journal of Computational Geometry

1978 Shamos's PhD thesis

~1980 first CG courses

1984

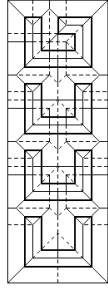


1985 1<sup>st</sup> ACM SOCG



1986 D & CG

1989



1<sup>st</sup> CCCG

1991



CGTA

IJCGA



1993 Teaching Computational Geometry, I



Mehlhorn

Preparata Shamos



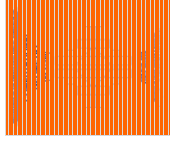
2009

Journal of Computational Geometry

1978 Shamos's PhD thesis

~1980 first CG courses

1984

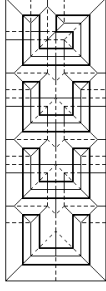


1985 1<sup>st</sup> ACM SOCG



1986 D & CG

1987



1989 1<sup>st</sup> CCCG

1991



CGTA

IJCGA



1993 Teaching Computational Geometry, I



Mehlhorn



Preparata Shamos



Edelsbrunner

2009

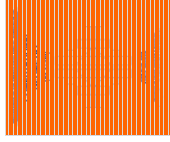
Journal of Computational Geometry



1978 Shamos's PhD thesis

~1980 first CG courses

1984

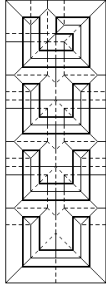


1985 1<sup>st</sup> ACM SOCG



1986 D & CG

1987



1989 1<sup>st</sup> CCCG

1991



CGTA

IJCGA



1993 Teaching Computational Geometry, I



Mulmuley



O'Rourke



Mehlhorn



Preparata Shamos



Edelsbrunner

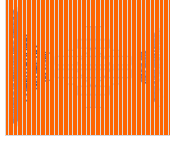
2009

Journal of Computational Geometry

1978 Shamos's PhD thesis

~1980 first CG courses

1984

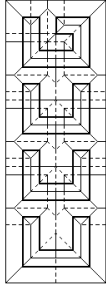


1985 1<sup>st</sup> ACM SOCG



1986 D & CG

1987



1989 1<sup>st</sup> CCCG

1991



CGTA

IJCGA



1993 Teaching Computational Geometry, I

1995



Mulmuley



O'Rourke



Boissonnat Yvinec



Mehlhorn



Preparata Shamos

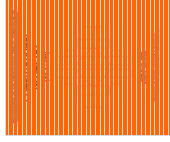


Edelsbrunner

1978 Shamos's PhD thesis

~1980 first CG courses

1984

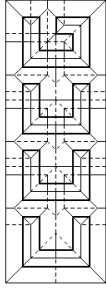


1985 1<sup>st</sup> ACM SOCG



1986 D & CG

1987



1989 1<sup>st</sup> CCCG

1991



CGTA

IJCGA



1993 Teaching Computational Geometry, I

1995

1997



"Dutch Book"



Boissonnat Yvinec

Mehlhorn



Preparata Shamos



Edelsbrunner



Mulmuley



O'Rourke



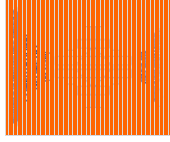
2009

Journal of Computational Geometry

1978 Shamos's PhD thesis

~1980 first CG courses

1984

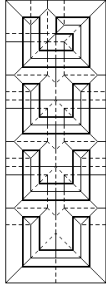


1985 1<sup>st</sup> ACM SOCG



1986 D & CG

1987



1989 1<sup>st</sup> CCCG



1991 CGTA



IJCGA

1993 Teaching Computational Geometry, I

1995

1997



"Dutch Book"



Boissonnat Yvinec



Klein



2009

Journal of Computational Geometry



Mehlhorn

Preparata Shamos



Edelsbrunner

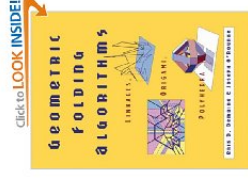


Mulmuley

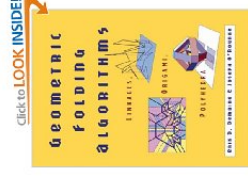
O'Rourke



# Monographs in Computational Geometry



# Monographs in Computational Geometry



incomplete

# Why do we teach computational geometry?

# Why do we teach computational geometry?

- to convey facts



# Why do we teach computational geometry?

- to convey facts
- to convey knowledge

# Why do we teach computational geometry?

- to convey facts
- to convey knowledge
- to convey understanding

# Why do we teach computational geometry?

- to convey facts
- to convey knowledge
- to convey understanding
- to provide tools

# Why do we teach computational geometry?

- to convey facts
- to convey knowledge
- to convey understanding
- to provide tools
- to convey beauty

# Why do we teach computational geometry?

- to convey facts
- to convey knowledge
- to convey understanding
- to provide tools
- to convey beauty
- for mental workout

# Why do we teach computational geometry?

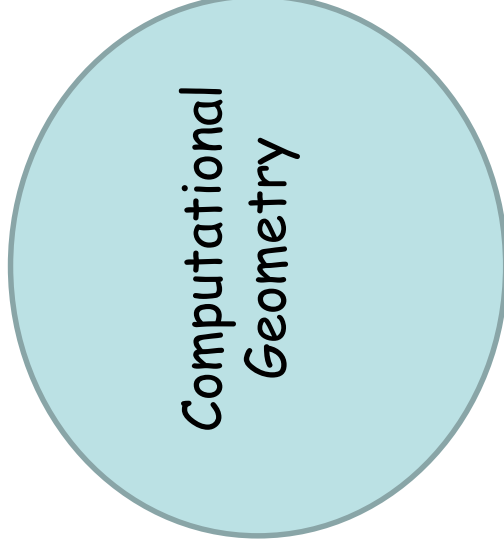
- to convey facts
- to convey knowledge
- to convey understanding
- to provide tools
- to convey beauty
- for mental workout
- .....

# Fundamentals

Computational  
Geometry

# Fundamentals

Euclidean Geometry

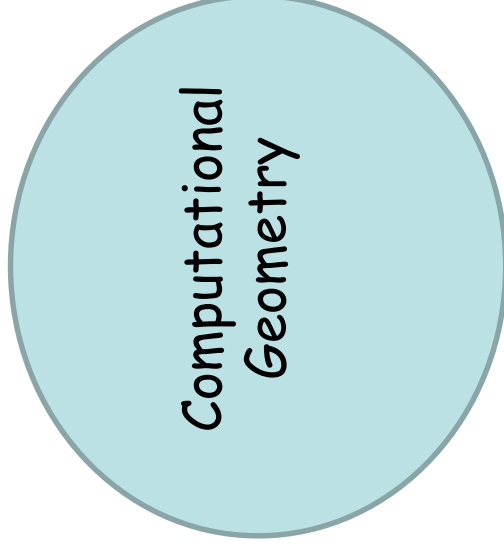




# Fundamentals

Euclidean Geometry

Discrete Geometry

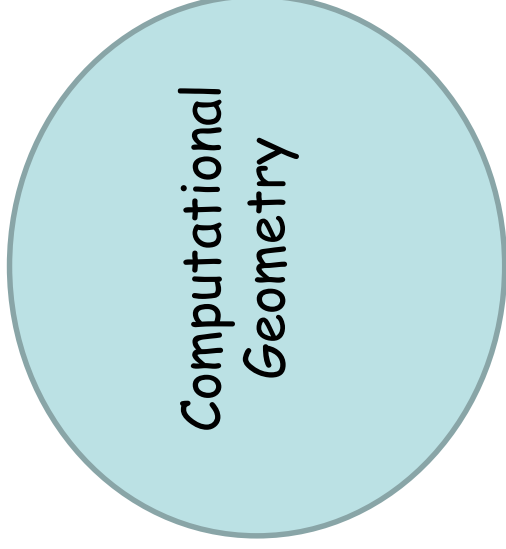


# Fundamentals

Euclidean Geometry

Algorithmics

Discrete Geometry



# Fundamentals

*Euclidean Geometry*

*Algorithmics*

*Discrete Geometry*



*Combinatorial Geometry*

# Fundamentals

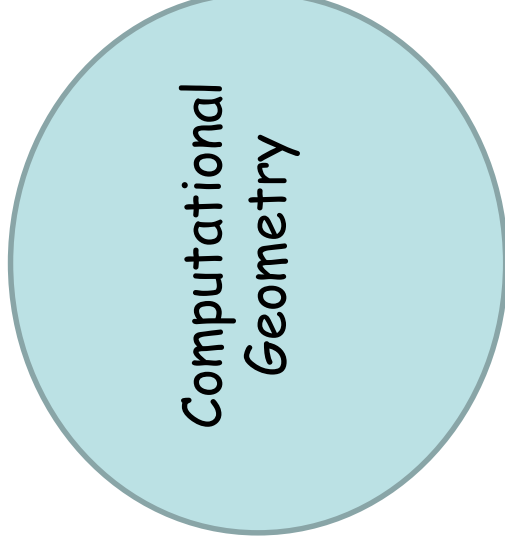
Euclidean Geometry

Discrete Geometry

Combinatorial Geometry

Algorithmics

Probability Theory



# Fundamentals

Euclidean Geometry

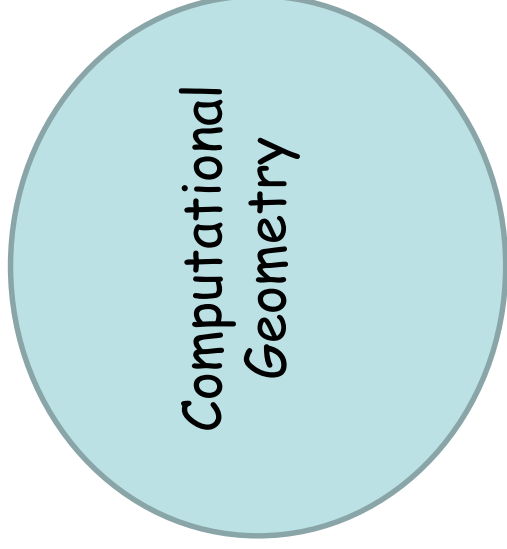
Discrete Geometry

Combinatorial Geometry

Coordinates

Algorithmics

Probability Theory



# Fundamentals

Euclidean Geometry

Discrete Geometry

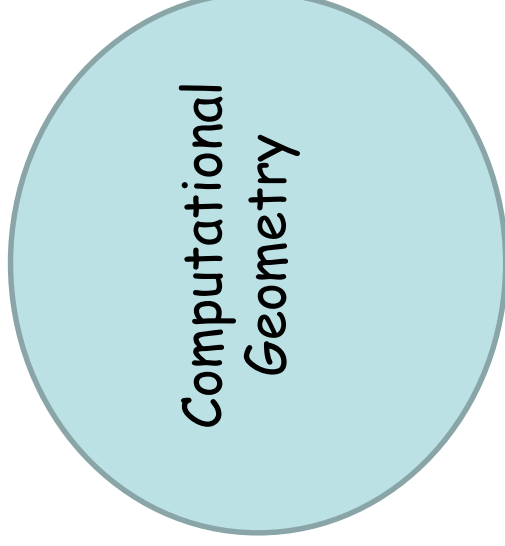
Combinatorial Geometry

Coordinates

Numerical Analysis

Algorithmics

Probability Theory



# Fundamentals

Euclidean Geometry

Discrete Geometry

Combinatorial Geometry

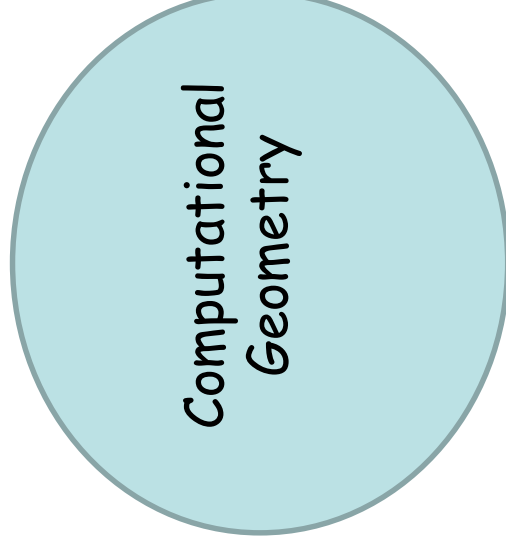
Coordinates

Numerical Analysis

Algorithmics

Probability Theory

Applications



# Fundamentals 1993

Euclidean Geometry

Discrete Geometry

Combinatorial Geometry

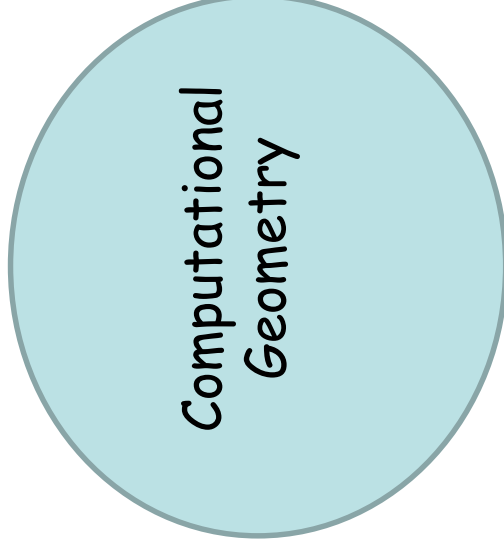
Coordinates

Numerical Analysis

Algorithmics

Probability Theory

Applications





# CG Introductory Course Topics

- 2d convex hulls
- segment intersection
- sweep algorithms
- proximity problems
- polygons
- triangulations
- planar subdivisions
- Voronoi Diagrams
- Delaunay Triangulations
- arrangements
- geometric transforms / duality
- planar point location
- orthogonal range searching
- Linear Programming
- 3d convex hulls

80 - 90 %

# CG Introductory Course Topics

- 2d convex hulls
- segment intersection
- sweep algorithms
- proximity problems
- polygons
- triangulations
- planar subdivisions
- Voronoi Diagrams
- Delaunay Triangulations
- arrangements
- geometric transforms / duality
- planar point location
- orthogonal range searching
- Linear Programming
- 3d convex hulls
- motion planning
- visibility graphs
- binary space partitions
- higher dimensional  
convex hulls, VoDs,  
DTs, arrangements
- simplex range searching
- combinatorial geometry  
(k-sets, upperbound thm,  
DS-sequences)
- spanners
- kinetic algorithms
- random sampling
- perturbations
- robust computation

80 - 90 %

rest

# CG Introductory Course Topics 1993

- 2d convex hulls
- segment intersection
- sweep algorithms
- proximity problems
- polygons
- triangulations
- planar subdivisions
- Voronoi Diagrams
- Delaunay Triangulations
- arrangements
- geometric transforms / duality
- planar point location
- orthogonal range searching
- Linear Programming
- 3d convex hulls

80 - 90 %

# CG Introductory Course Topics 1993

- 2d convex hulls
- segment intersection
- sweep algorithms
- proximity problems
- polygons
- triangulations
- planar subdivisions
- Voronoi Diagrams
- Delaunay Triangulations
- arrangements
- geometric transforms / duality
- planar point location
- orthogonal range searching
- Linear Programming
- 3d convex hulls
- motion planning
- visibility graphs
- binary space partitions
- higher dimensional convex hulls, VoDs, DTs, arrangements
- simplex range searching
- combinatorial geometry (k-sets, upperbound thm, DS-sequences)
- spanners
- kinetic algorithms
- random sampling
- perturbations
- robust computation

80 - 90 %

rest

# 1993 Teaching Computational Geometry, I

## Topics

- Randomized algorithms as first class objects
- The importance of invariants for sweep algorithms
- Representation of planar subdivisions
- Planar point location
- Linear Programming
- 3d Convex hulls
- Perturbations
- Upper bound theorem for polytopes

# • Randomized Algorithms as First class Objects

1993

2009

# • Randomized Algorithms as First class Objects

1993

2009

No randomized methods in  
the available textbooks

Preparata & Shamos

O'Rourke (1<sup>st</sup> edition)

# • Randomized Algorithms as First class Objects

1993

No randomized methods in  
the available textbooks

Preparata & Shamos

O'Rourke (1<sup>st</sup> edition)

2009

Randomized methods in  
virtually all textbooks.



# • Randomized Algorithms as First class Objects

1993

No randomized methods in  
the available textbooks

Preparata & Shamos

O'Rourke (1<sup>st</sup> edition)

2009

Randomized methods in  
virtually all textbooks.

Some topics are covered  
**exclusively** via randomized  
methods:

e.g. in "Dutch Book"  
planar point location  
linear programming  
3d convex hulls

# 1993 Teaching Computational Geometry, I

## Topics

- Randomized algorithms as first class objects
- The importance of invariants for sweep algorithms
- Representation of planar subdivisions
- Planar point location
- Linear Programming
- 3d Convex hulls
- Perturbations
- Upper bound theorem for polytopes

SWEEPING

1. INVARIANT

2. EVENTS

3. DATA STRUCTURES



# FORTUNE'S SWEEP ALGORITHM (SIMPLIFIED)

Reformulation of Problem:

$T$  ... set of sites  
each site  $p \in T$  is associated with a  
unique color

Goal: Color each point of the plane  
with the color of its closest site.

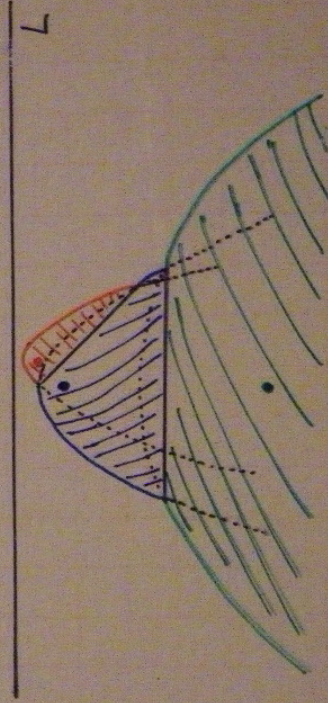
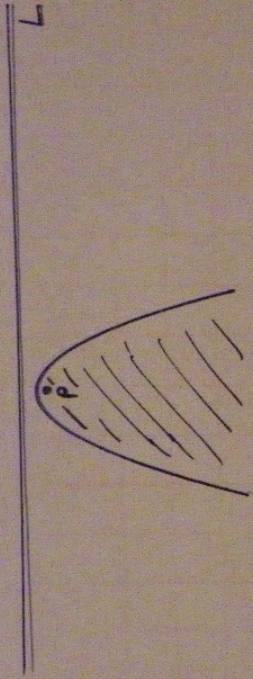
Method: Sweep horizontal line  $L$  over plane  
from "bottom" to "top" and maintain

INVARIANT:

Each point "below"  $L$  for which one  
can be sure of its color has been  
colored correctly.

$p \in T$  below  $L$   $\left. \vphantom{\begin{matrix} p \in T \\ \text{below } L \end{matrix}} \right\} \Rightarrow x$  has been colored  
 $d(x, p) < d(x, L)$





Significant Changes in "Parabolic Wavefront":

Events:

- (i) L encounters new site p & T
- (ii) One of the arcs in the wavefront collapses to a point.

$\Rightarrow$  "Simple"  $O(n \log n)$  algorithm.

- **Sweeping**

- Sweeping

# 1. INVARIANT

- Sweeping

# 1. INVARIANT

## 2. Events



- Sweeping

# 1. INVARIANT

2. Events

3. Data Structures

- Sweeping

# 1. INVARIANT

## 2. Events

## 3. Data Structures

event queue

sweep line structures

auxiliary structures

# Invariants:

# Invariants:

- 1) main geometric invariant

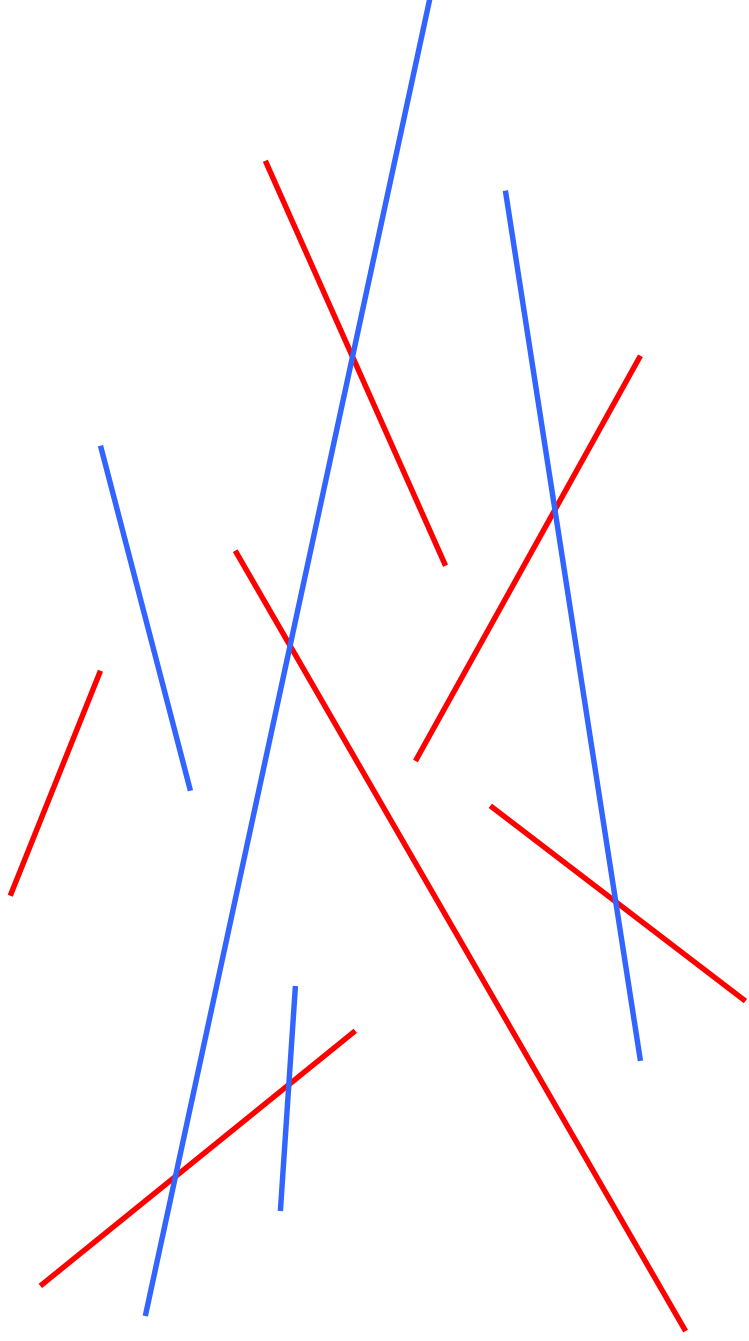
# Invariants:

- 1) main geometric invariant  
true initially  
implies desired result finally

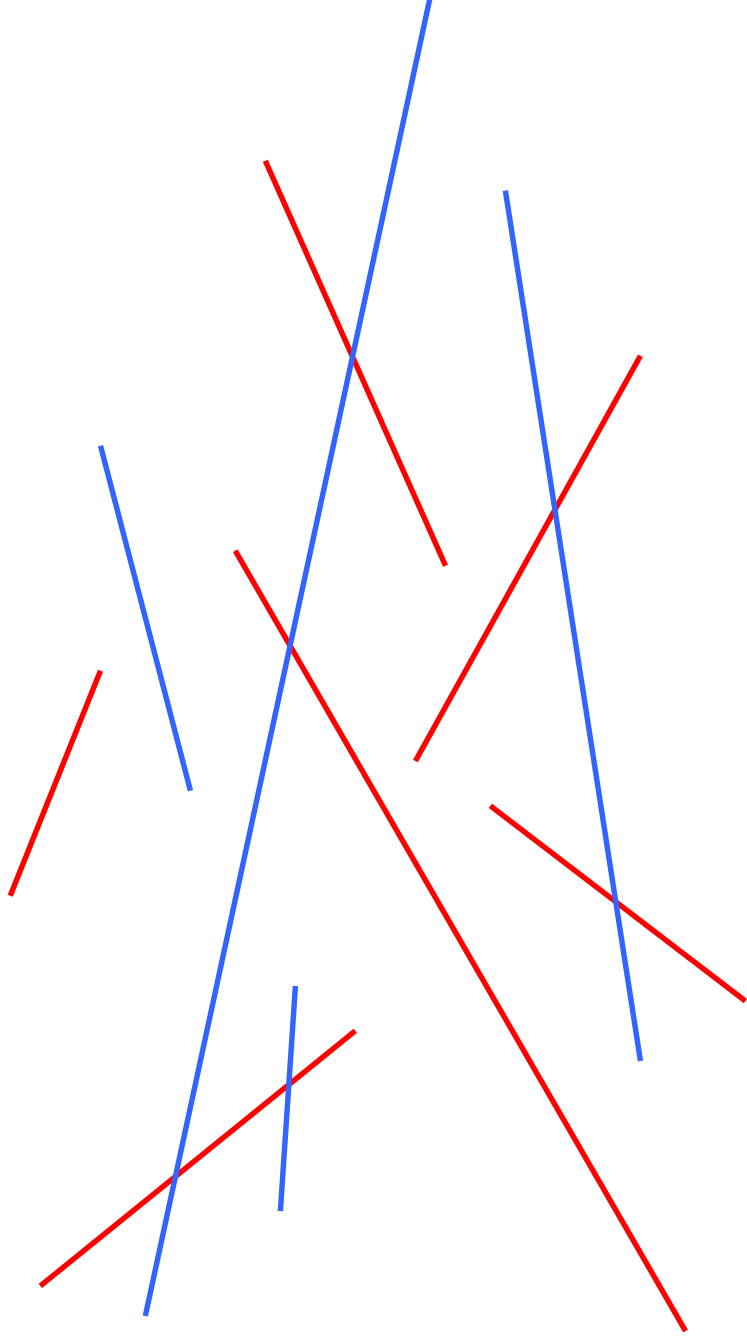
# Invariants:

- 1) main geometric invariant  
true initially  
implies desired result finally
- 2) invariants about contents of  
the data structures

# Red - Blue - Segment Intersection Counting



## Red - Blue - Segment Intersection Counting



Determine the **number** of red-blue intersecting pairs of segments.



# INVARIANT (Don Hatch)

Maintain

# of red-blue intersections below the sweepline  
plus

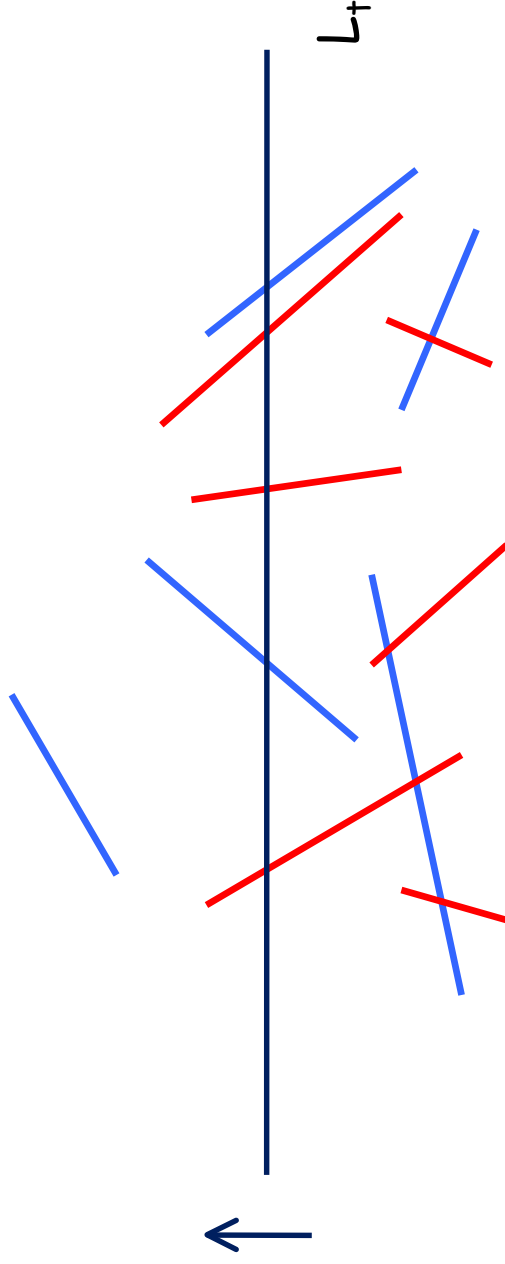
# of red-blue intersections between "rays"  
above the sweepline

# INVARIANT (Don Hatch)

Maintain

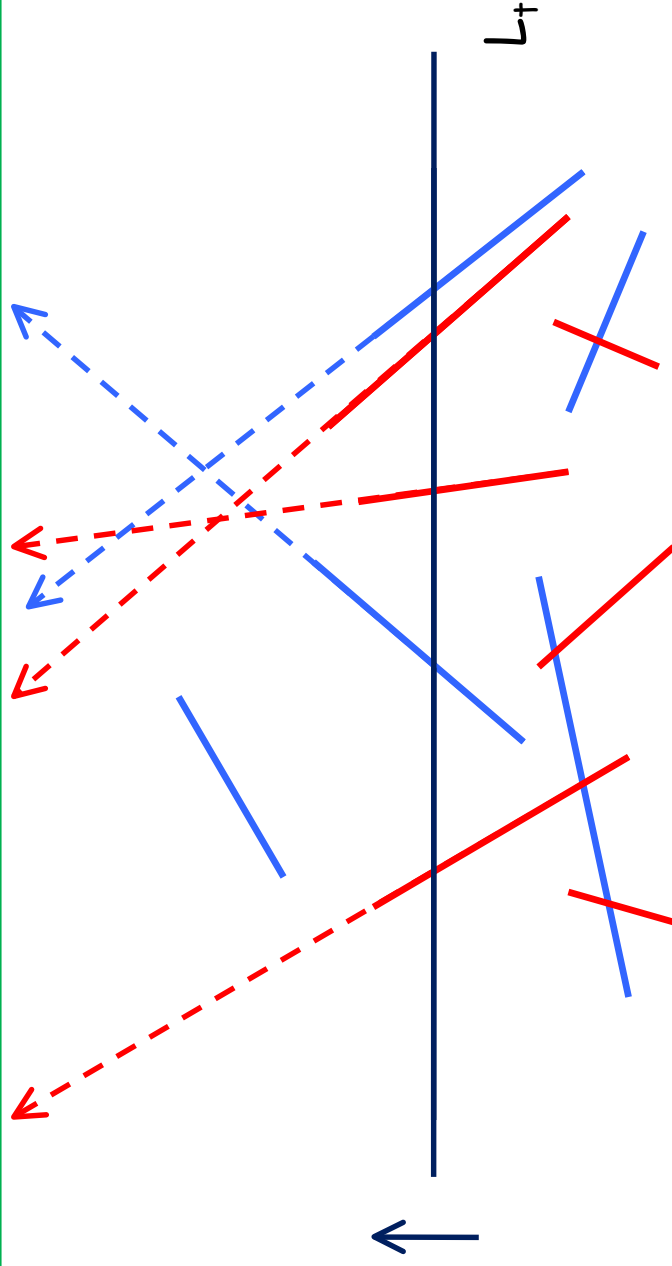
# of red-blue intersections below the sweepline  
plus

# of red-blue intersections between "rays"  
above the sweepline



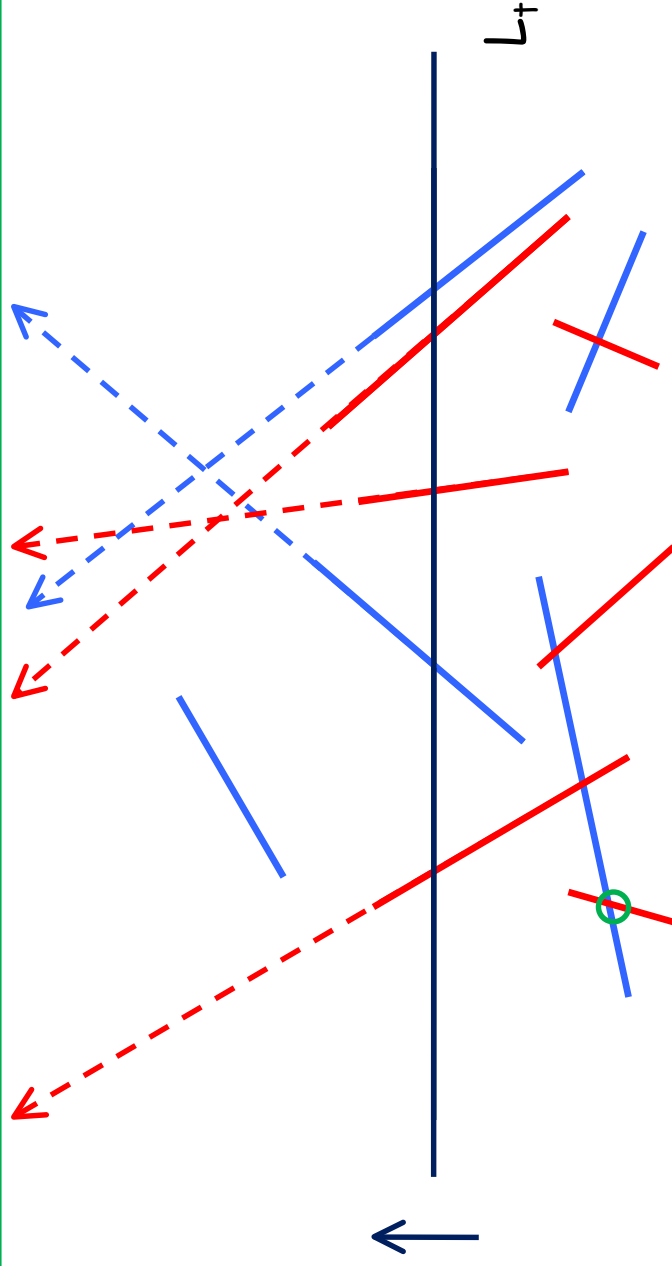
# INVARIANT (Don Hatch)

Maintain  
# of red-blue intersections below the sweepline  
plus  
# of red-blue intersections between "rays"  
above the sweepline



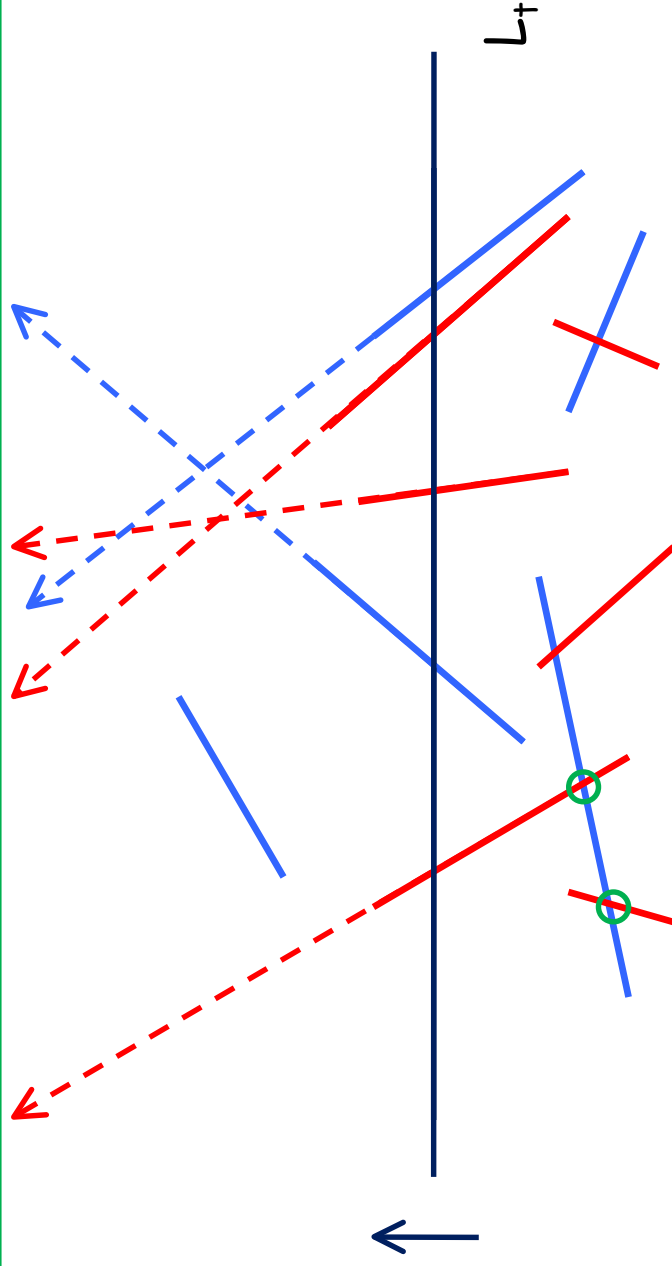
# INVARIANT (Don Hatch)

Maintain  
# of red-blue intersections below the sweepline  
plus  
# of red-blue intersections between "rays"  
above the sweepline



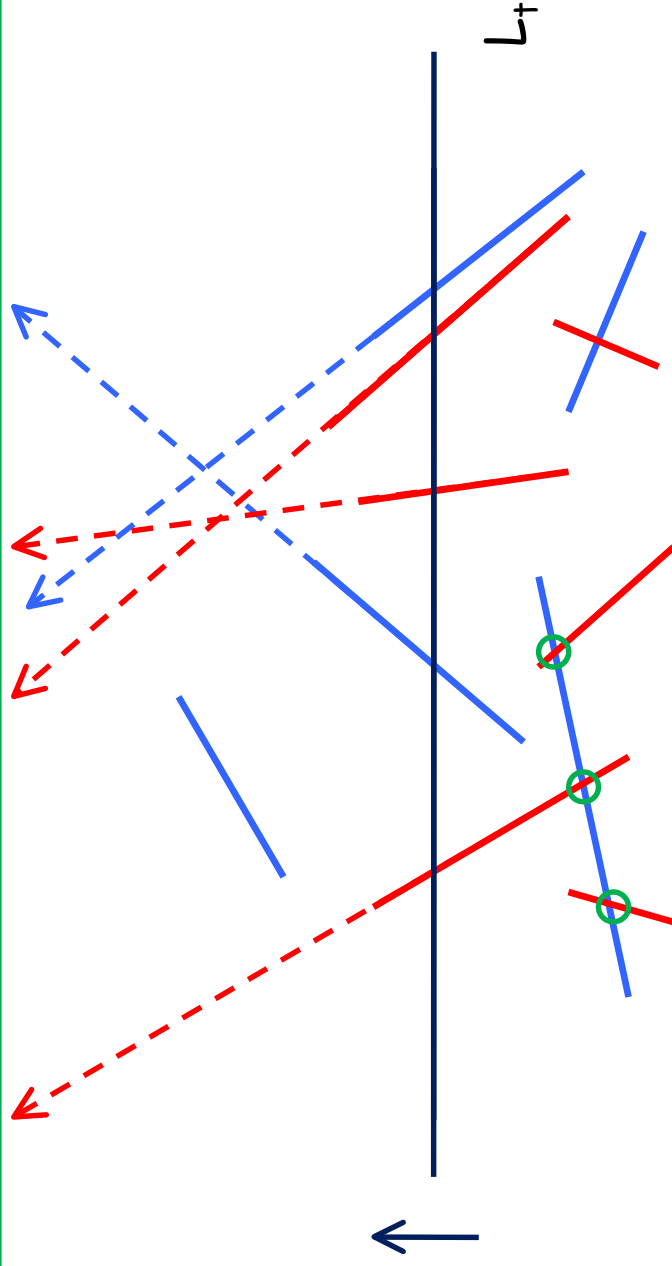
# INVARIANT (Don Hatch)

Maintain  
# of red-blue intersections below the sweepline  
plus  
# of red-blue intersections between "rays"  
above the sweepline



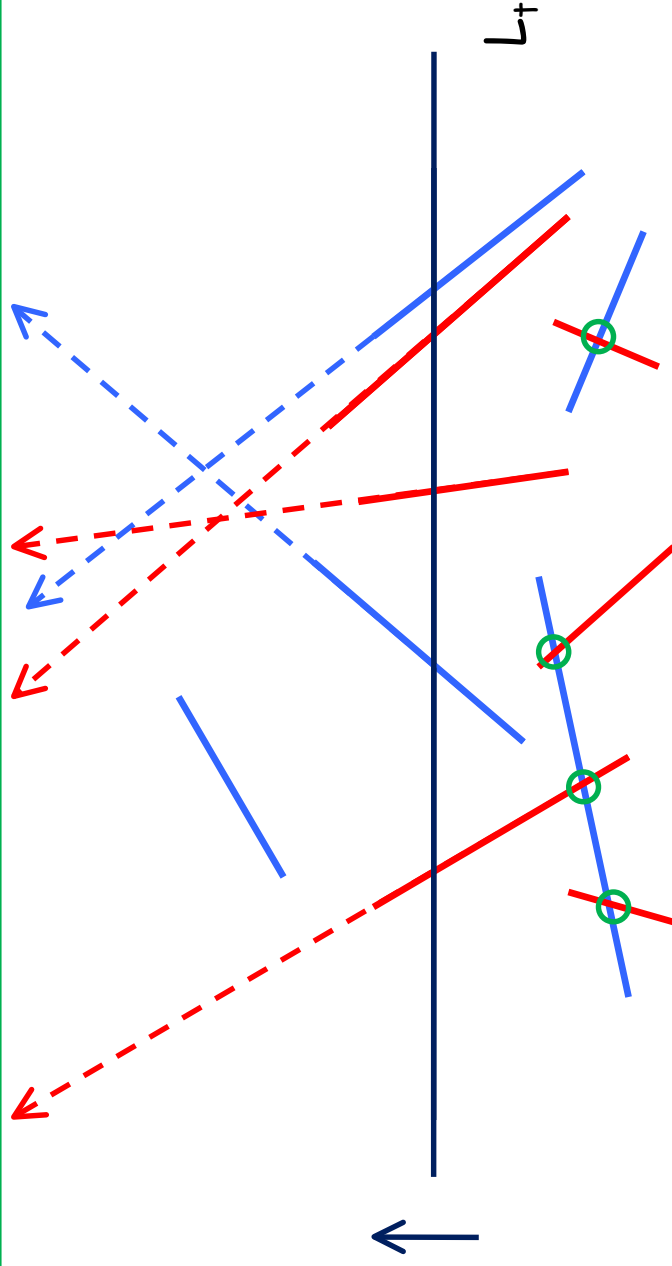
# INVARIANT (Don Hatch)

Maintain  
# of red-blue intersections below the sweepline  
plus  
# of red-blue intersections between "rays"  
above the sweepline



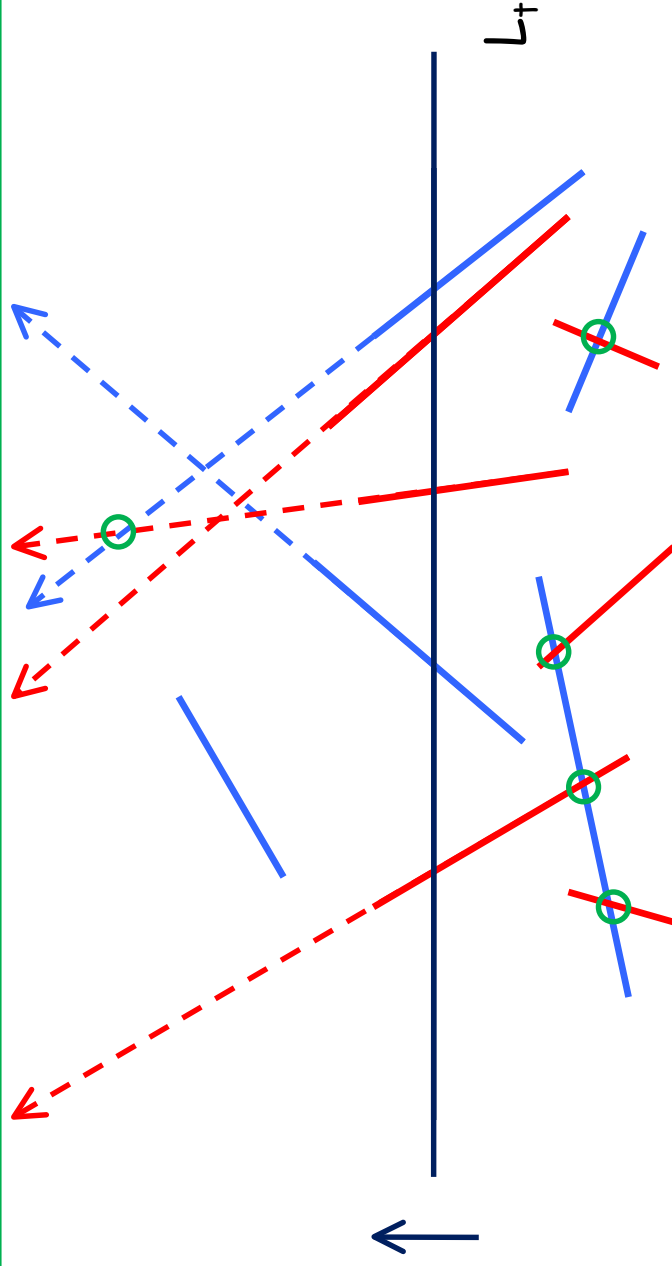
# INVARIANT (Don Hatch)

Maintain  
# of red-blue intersections below the sweepline  
plus  
# of red-blue intersections between "rays"  
above the sweepline



# INVARIANT (Don Hatch)

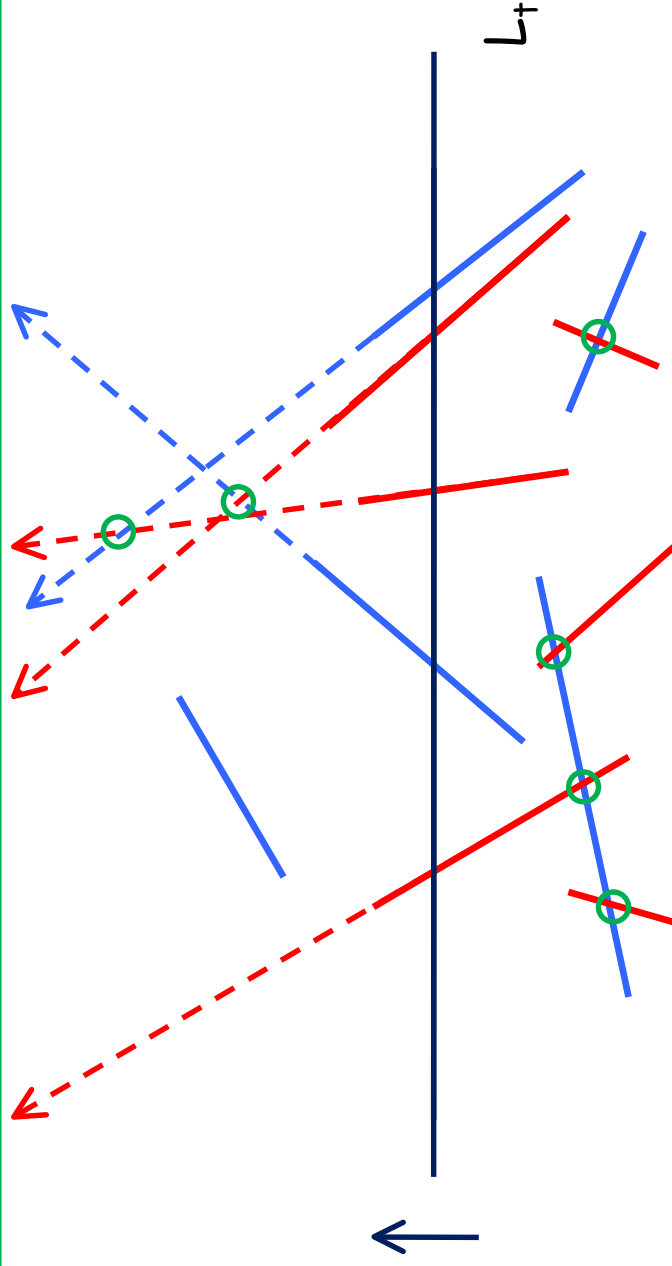
Maintain  
# of red-blue intersections below the sweepline  
plus  
# of red-blue intersections between "rays"  
above the sweepline





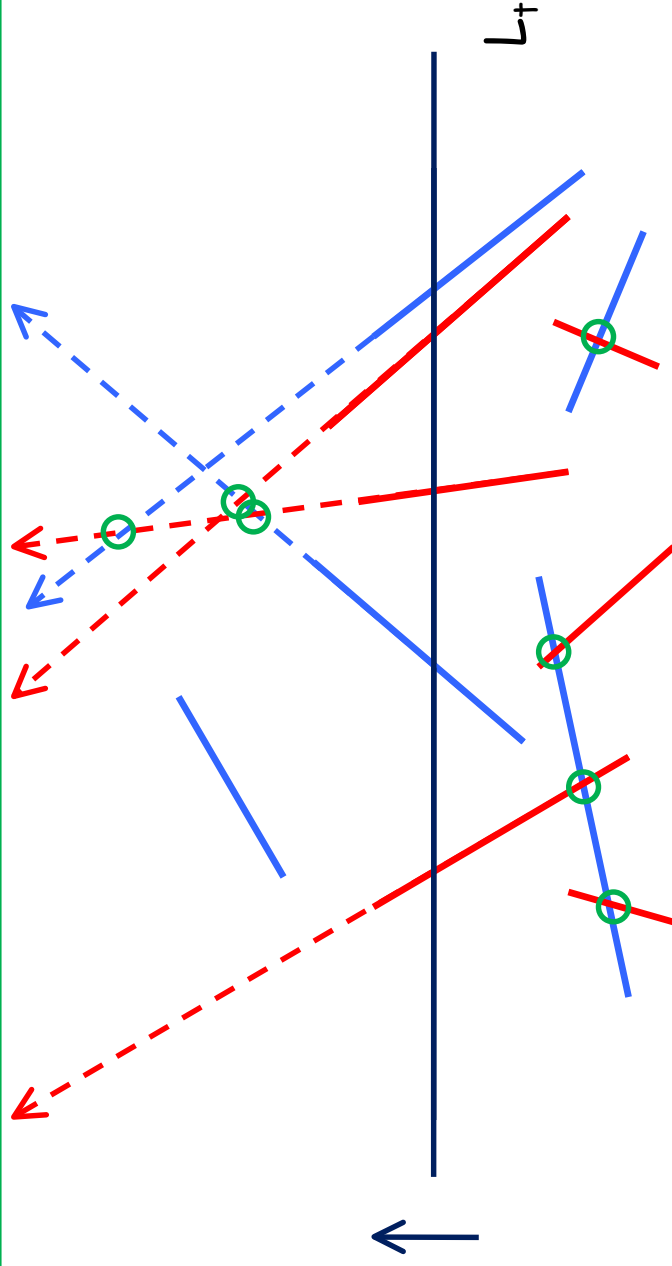
# INVARIANT (Don Hatch)

Maintain  
# of red-blue intersections below the sweepline  
plus  
# of red-blue intersections between "rays"  
above the sweepline



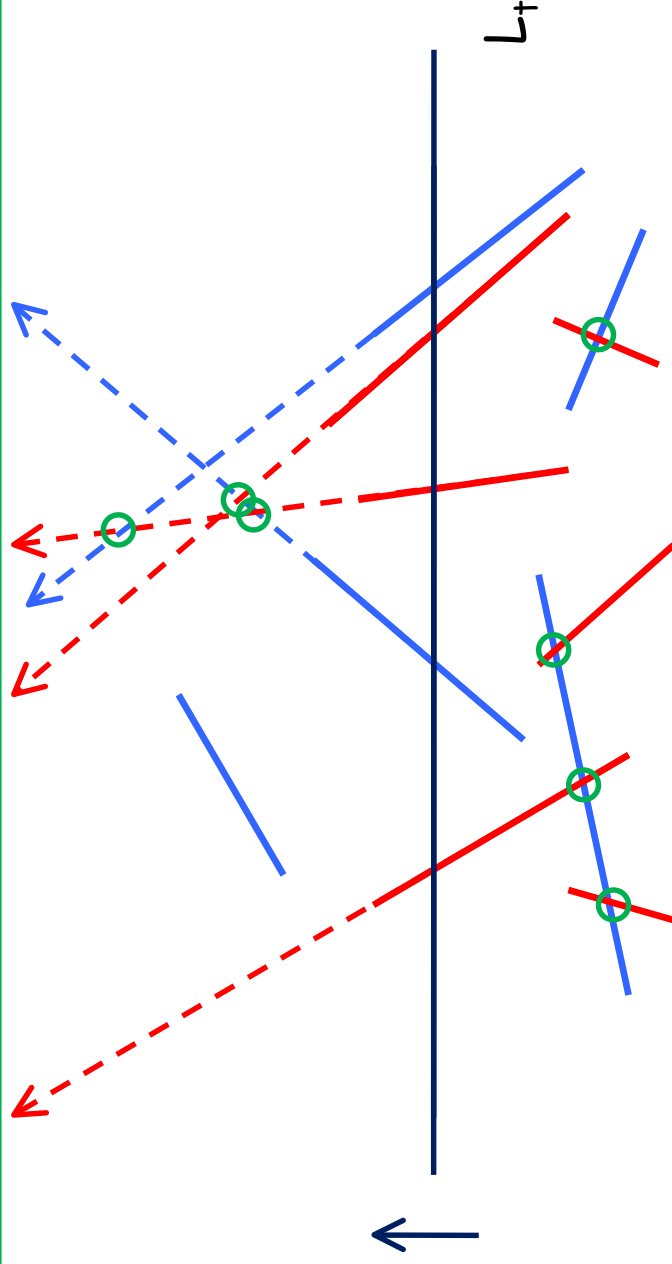
# INVARIANT (Don Hatch)

Maintain  
# of red-blue intersections below the sweepline  
plus  
# of red-blue intersections between "rays"  
above the sweepline



# INVARIANT (Don Hatch)

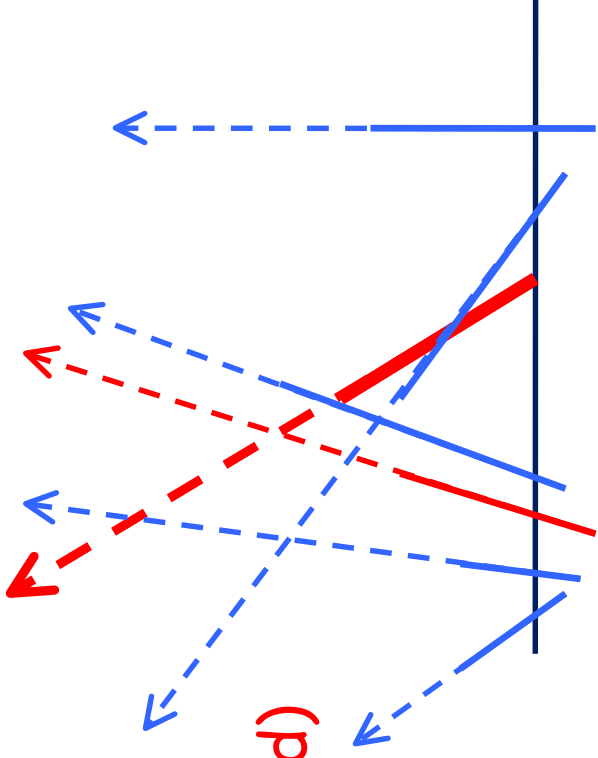
Maintain  
# of red-blue intersections below the sweepline  
plus  
# of red-blue intersections between "rays"  
above the sweepline



7

## Events:

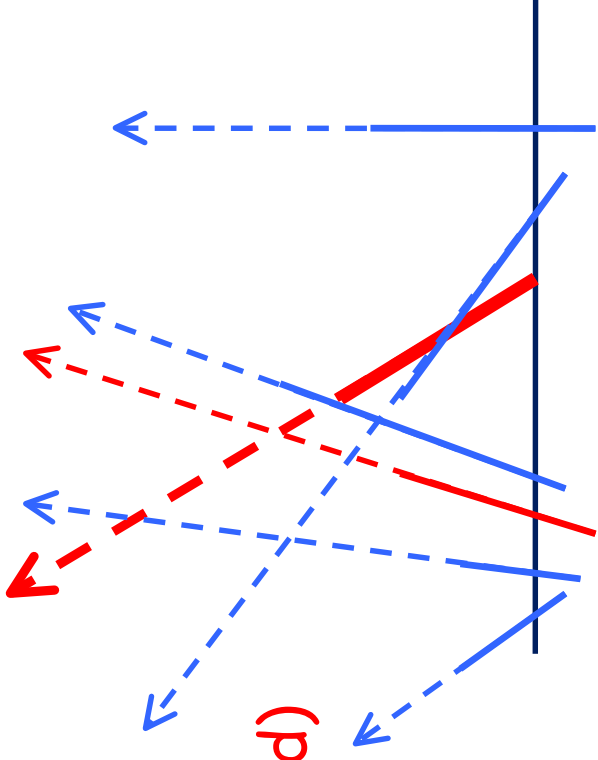
1) Lower segment endpoint (red)



## Events:

### 1) Lower segment endpoint (red)

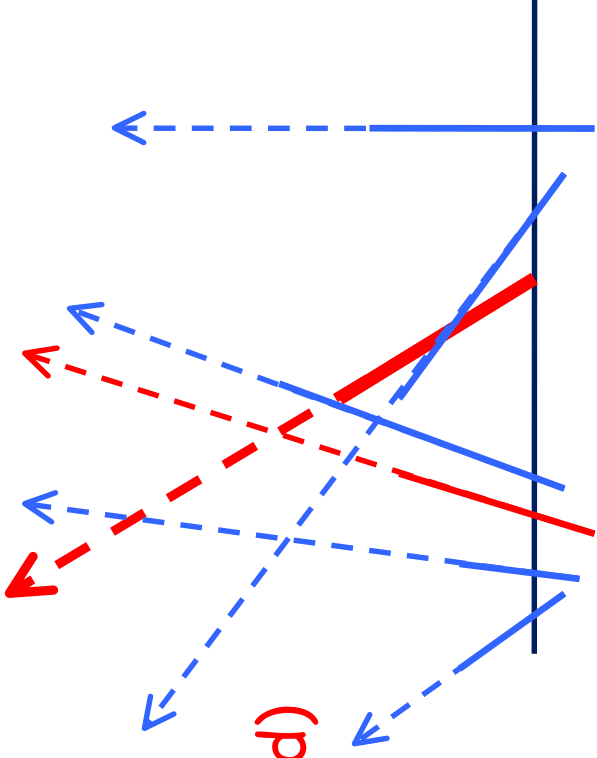
count # of intersections of  
**new red ray** with **blue rays**  
and add to the running total



## Events:

### 1) Lower segment endpoint (red)

count # of intersections of  
**new red ray** with **blue rays**  
and add to the running total

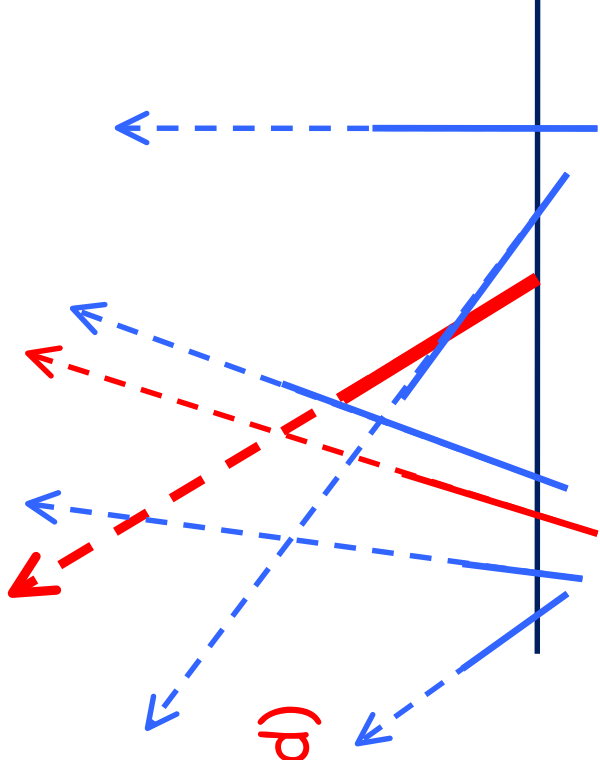


### 2) Upper segment endpoint (red)

## Events:

### 1) Lower segment endpoint (red)

count # of intersections of  
**new red ray** with **blue rays**  
and add to the running total



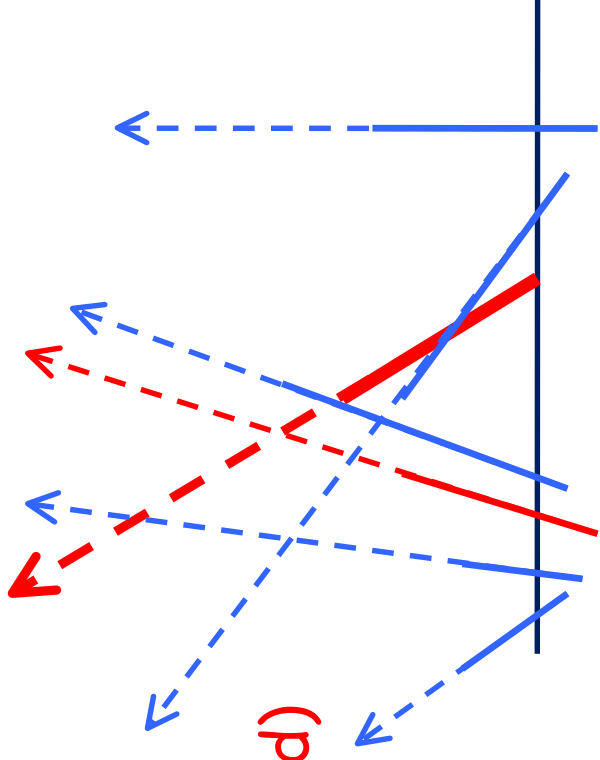
### 2) Upper segment endpoint (red)

count # of intersections of  
**disappearing red ray** with **blue rays**  
and subtract from the running total

## Events:

### 1) Lower segment endpoint (red)

count # of intersections of  
**new red ray** with **blue rays**  
and add to the running total



### 2) Upper segment endpoint (red)

count # of intersections of  
**disappearing red ray** with **blue rays**  
and subtract from the running total

### 3) and 4) analogous for blue



# Data structures:

## 1) Event queue

**Invariant:** contains upper and lower endpoints of segments above  $L_t$  and upper endpoints of segments intersected by  $L_t$

## 2) Red structure

**Invariant:** contains all red segments that currently intersect  $L_t$

## 2) Blue structure

**Invariant:** contains all blue segments that currently intersect  $L_t$

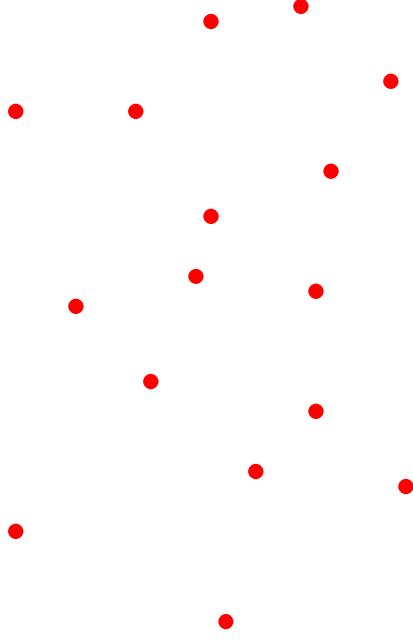
# Red Structure:

# Red Structure:

supports queries of the form:  
given segment  $q$   
produce the number of segments that  
intersect  $L_+$  further to the left of  $q$  and have slope  
"larger" than  $q$   
plus the number of segments that  
intersect  $L_+$  further to the right of  $q$  and have slope  
"smaller" than  $q$

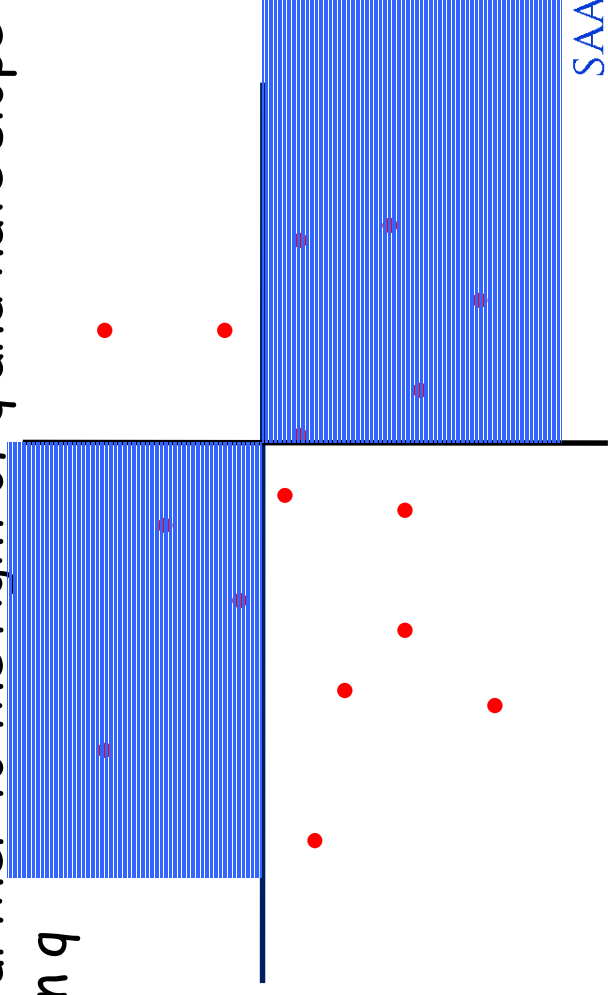
# Red Structure:

supports queries of the form:  
given segment  $q$   
produce the number of segments that  
intersect  $L_+$  further to the left of  $q$  and have slope  
"larger" than  $q$   
plus the number of segments that  
intersect  $L_+$  further to the right of  $q$  and have slope  
"smaller" than  $q$



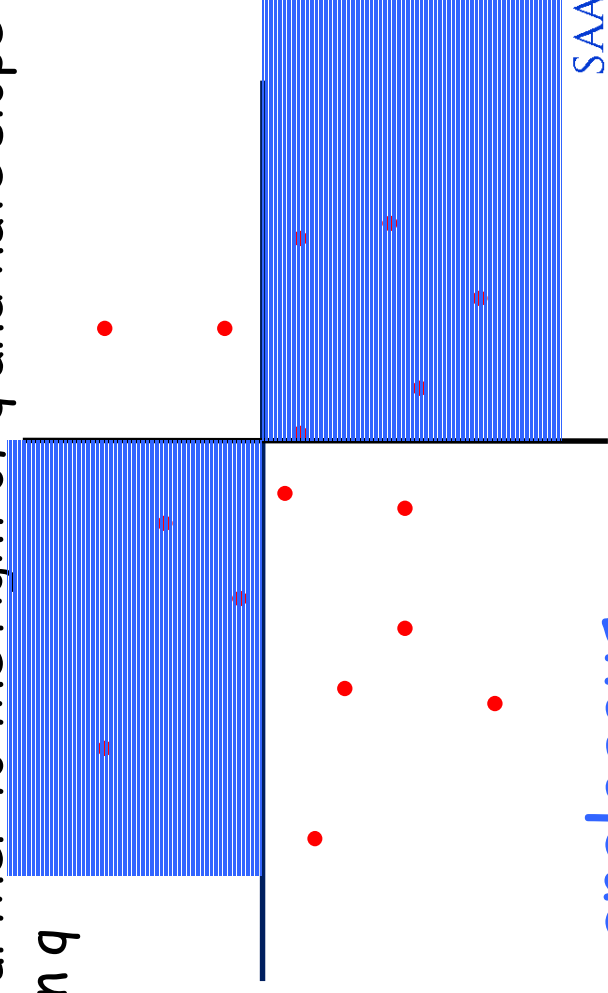
# Red Structure:

supports queries of the form:  
given segment  $q$   
produce the number of segments that  
intersect  $L_+$  further to the left of  $q$  and have slope  
"larger" than  $q$   
plus the number of segments that  
intersect  $L_+$  further to the right of  $q$  and have slope  
"smaller" than  $q$



# Red Structure:

supports queries of the form:  
given segment  $q$   
produce the number of segments that  
intersect  $L_+$  further to the left of  $q$  and have slope  
"larger" than  $q$   
plus the number of segments that  
intersect  $L_+$  further to the right of  $q$  and have slope  
"smaller" than  $q$



Blue Structure: analogous

2d range trees for red and blue structure

Straightforward  $O(n \log^2 n)$  implementation.

# 1993 Teaching Computational Geometry, I

## Topics

- Randomized algorithms as first class objects
- The importance of invariants for sweep algorithms
- **Representation of planar subdivisions**
- Planar point location
- Linear Programming
- 3d Convex hulls
- Perturbations
- Upper bound theorem for polytopes



- **Representation of planar subdivisions**

# • Representation of planar subdivisions

- \* Doubly Connected Edge Lists
- \* Winged Edge Data Structure
- \* Quadedges

# • Representation of planar subdivisions

- \* Doubly Connected Edge Lists
- \* Winged Edge Data Structure
- \* Quadedges



# • Representation of planar subdivisions

- \* Doubly Connected Edge Lists
- \* Winged Edge Data Structure
- \* Quadedges



# • Representation of planar subdivisions

- \* Doubly Connected **E**dge **L**ists
- \* Winged **E**dge **D**ata **S**tructure
- \* **Q**uadedges



---

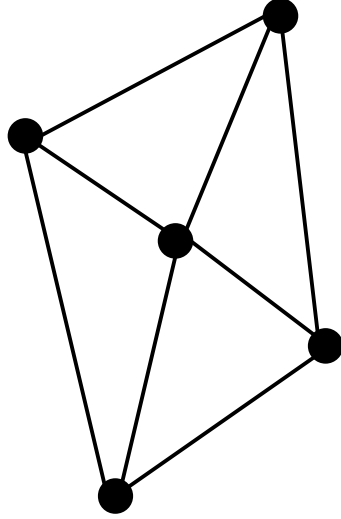
**Observation:** Triangulations are easy to represent :  
trivalent graph plus additional info at “nodes”

# • Representation of planar subdivisions

- \* Doubly Connected **E**dge **L**ists
- \* Winged **E**dge **D**ata **S**tructure
- \* **Q**uadedges



**Observation:** Triangulations are easy to represent :  
trivalent graph plus additional info at “nodes”

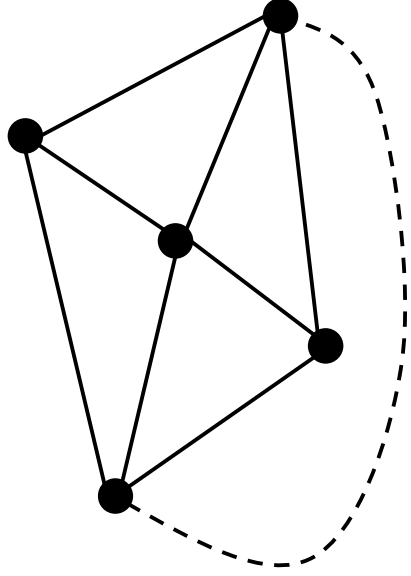


# • Representation of planar subdivisions

- \* Doubly Connected **E**dge **L**ists
- \* Winged **E**dge **D**ata **S**tructure
- \* **Q**uadedges



**Observation:** Triangulations are easy to represent :  
trivalent graph plus additional info at “nodes”

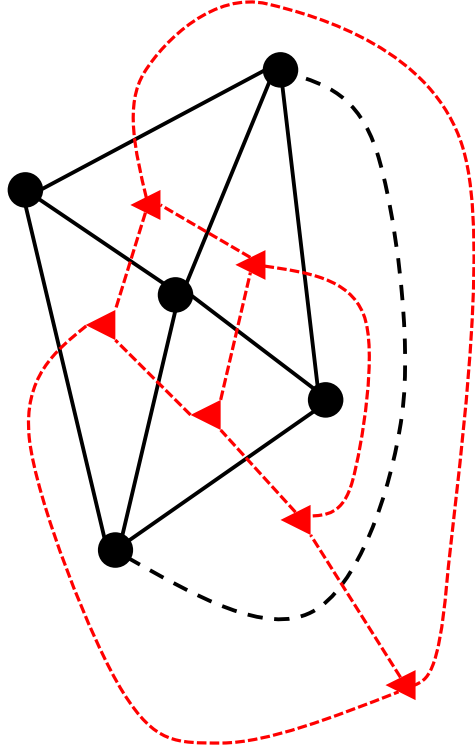


# • Representation of planar subdivisions

- \* Doubly Connected **E**dge **L**ists
- \* Winged **E**dge **D**ata **S**tructure
- \* **Q**uadedges



**Observation:** Triangulations are easy to represent :  
trivalent graph plus additional info at "nodes"



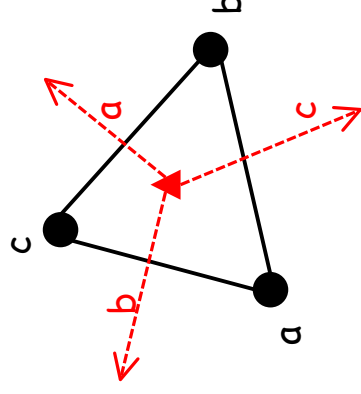
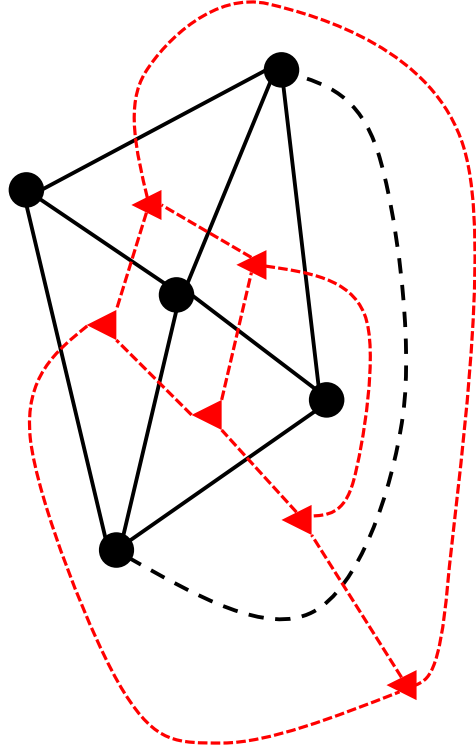


# • Representation of planar subdivisions

- \* Doubly Connected **E**dge **L**ists
- \* Winged **E**dge **D**ata **S**tructure
- \* **Q**uadedges



**Observation:** Triangulations are easy to represent :  
trivalent graph plus additional info at "nodes"

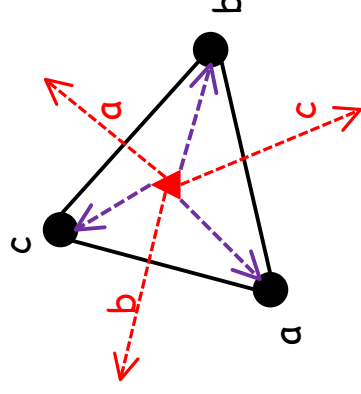
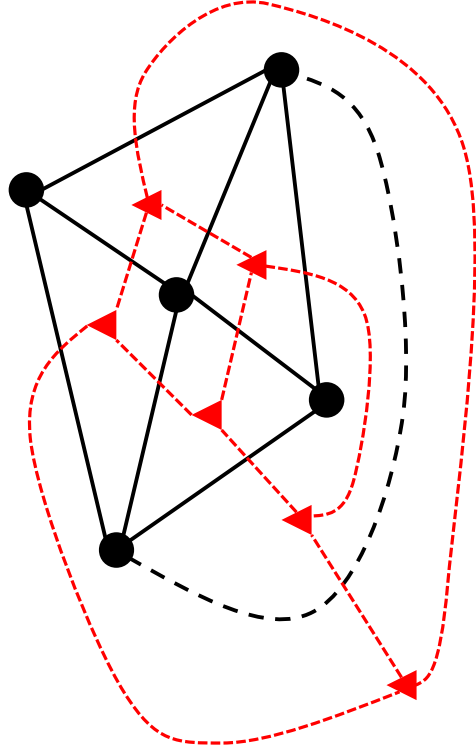


# • Representation of planar subdivisions

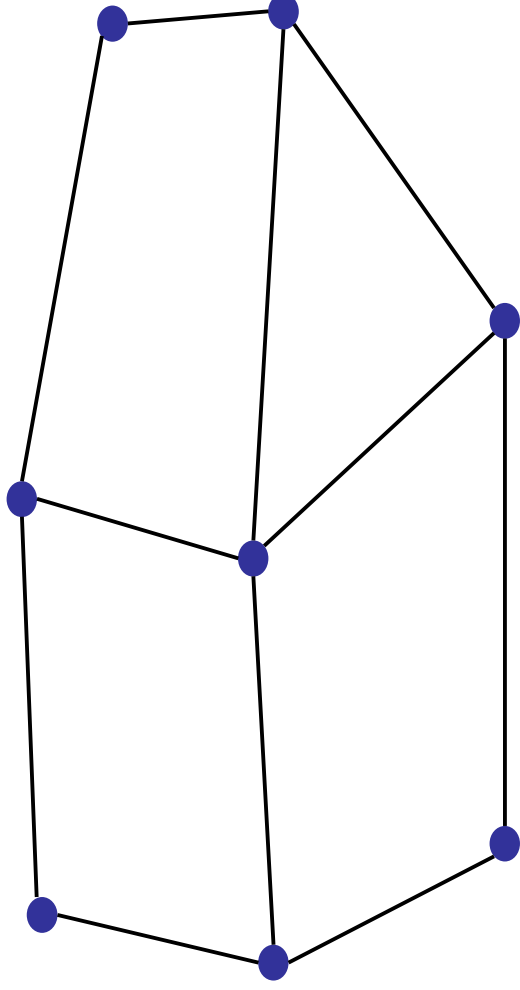
- \* Doubly Connected **E**dge **L**ists
- \* Winged **E**dge **D**ata **S**tructure
- \* **Q**uadedges



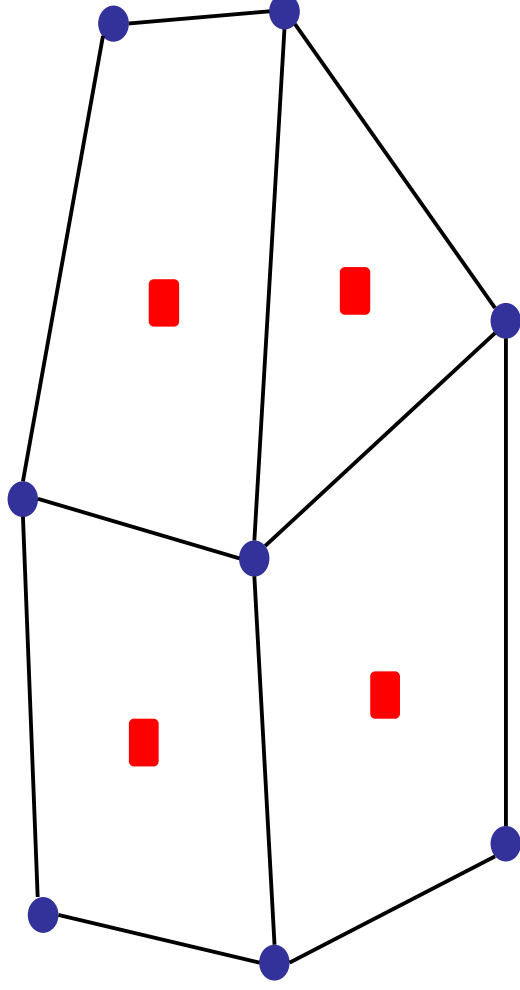
**Observation:** Triangulations are easy to represent :  
trivalent graph plus additional info at "nodes"



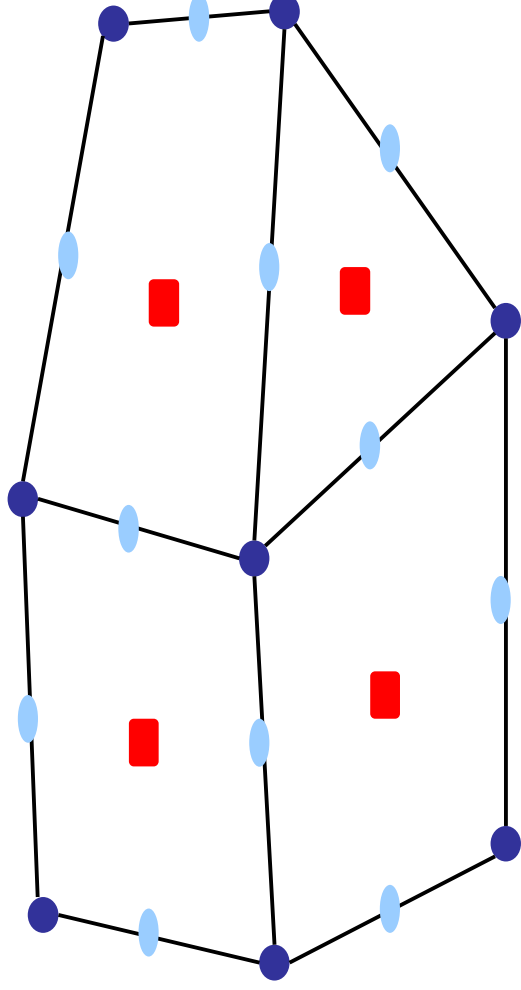
**Idea:** Make general planar subdivision into triangulation



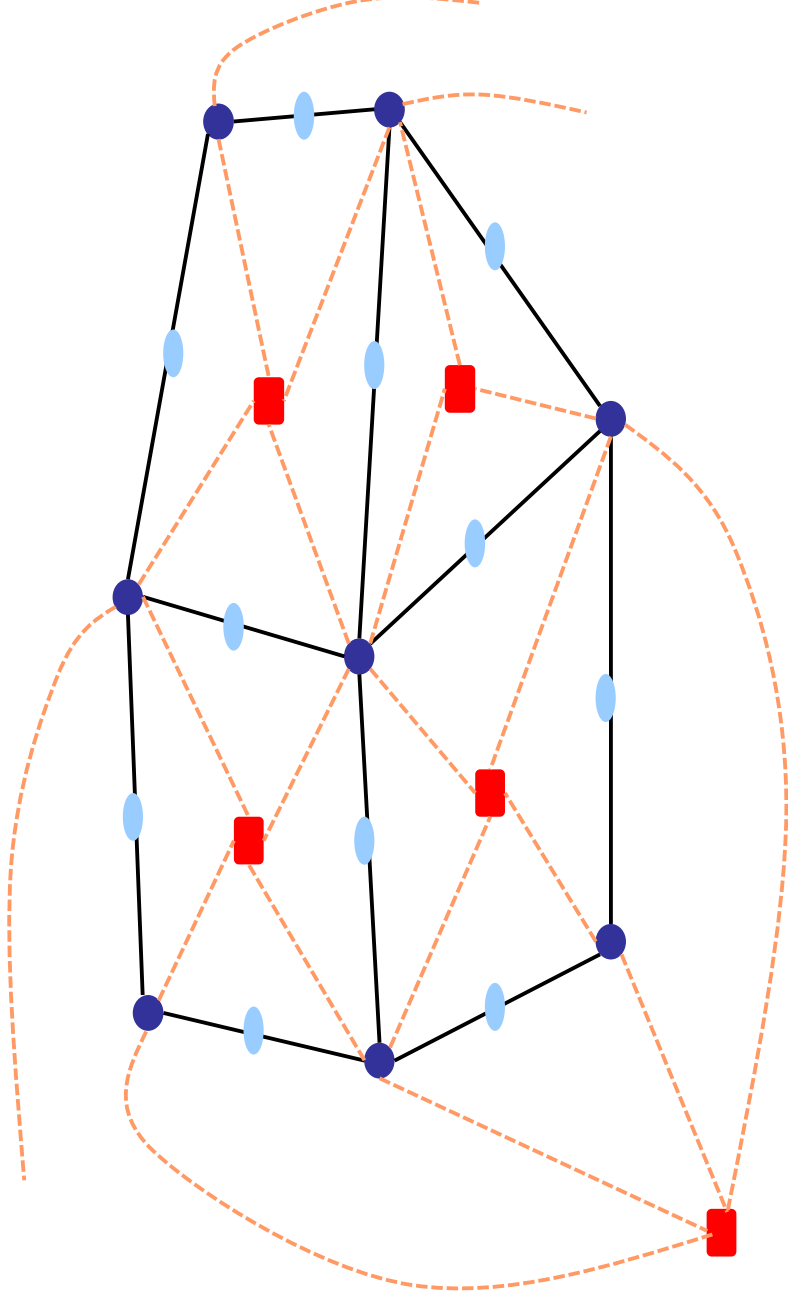
**Idea:** Make general planar subdivision into triangulation



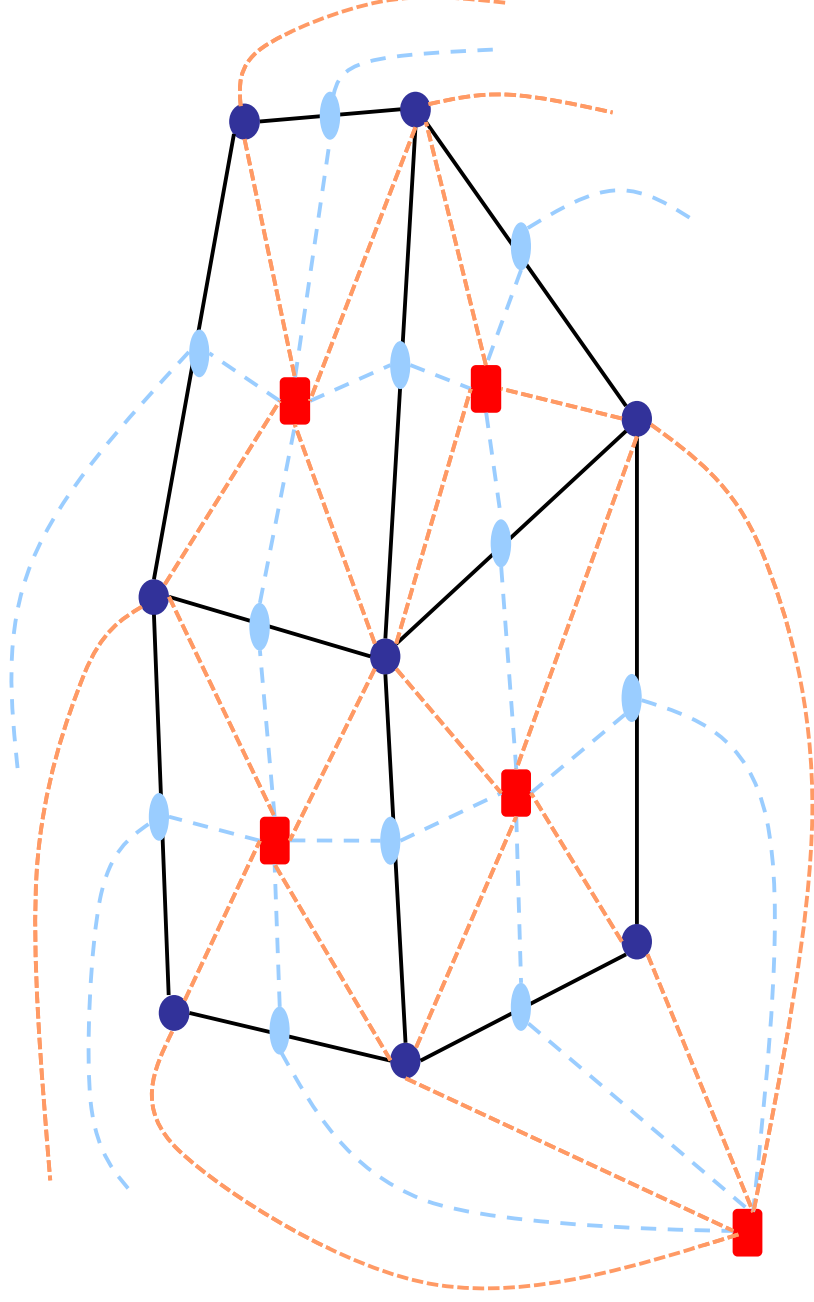
**Idea:** Make general planar subdivision into triangulation



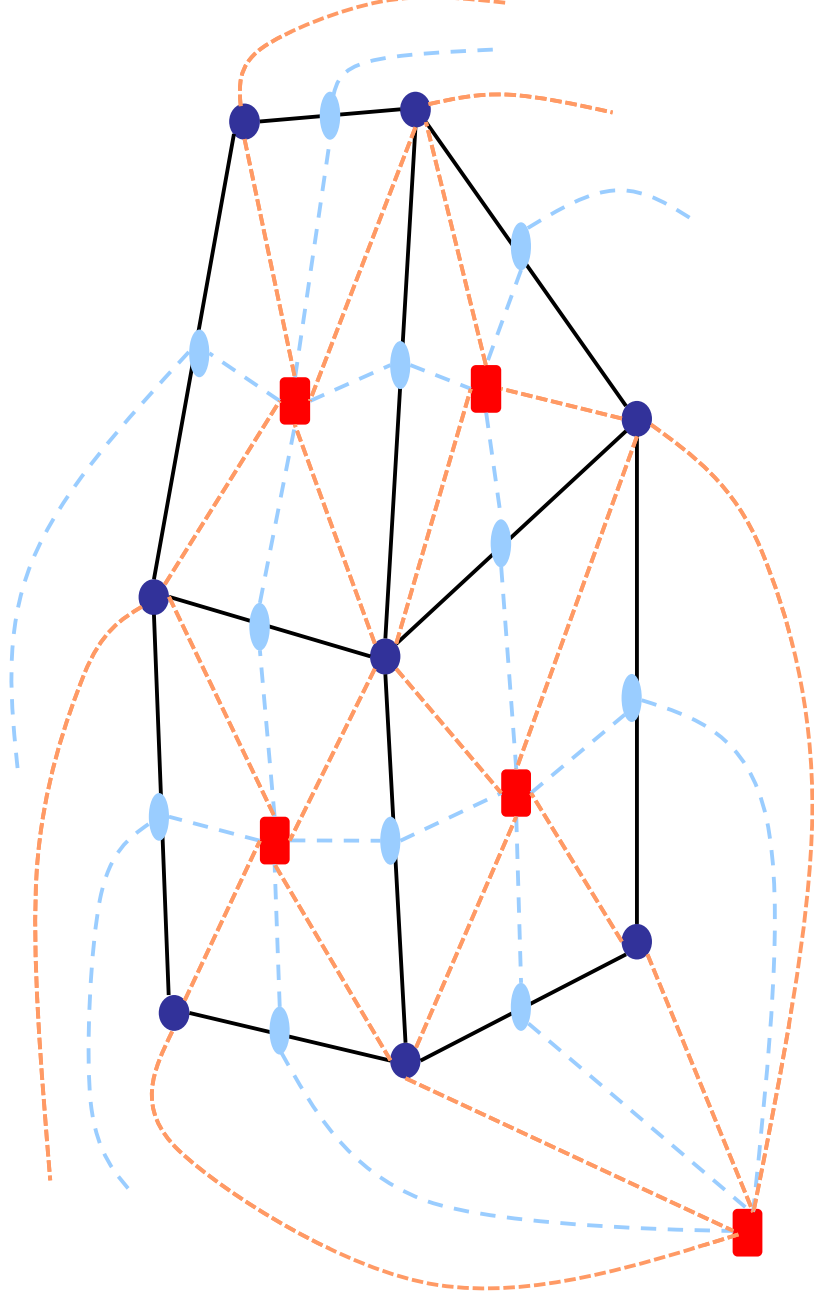
**Idea:** Make general planar subdivision into triangulation



**Idea:** Make general planar subdivision into triangulation



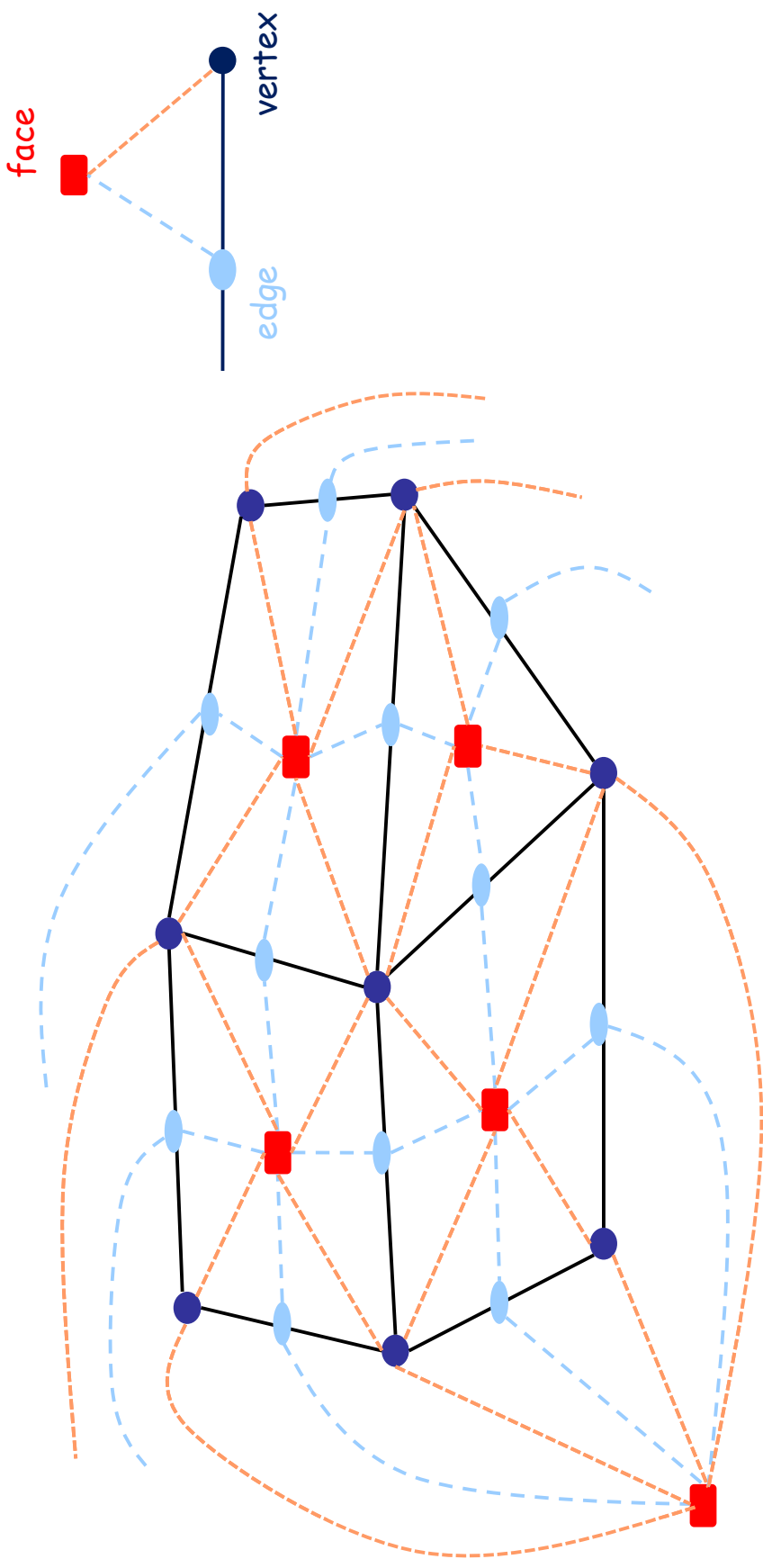
**Idea:** Make general planar subdivision into triangulation



**stellar subdivision**

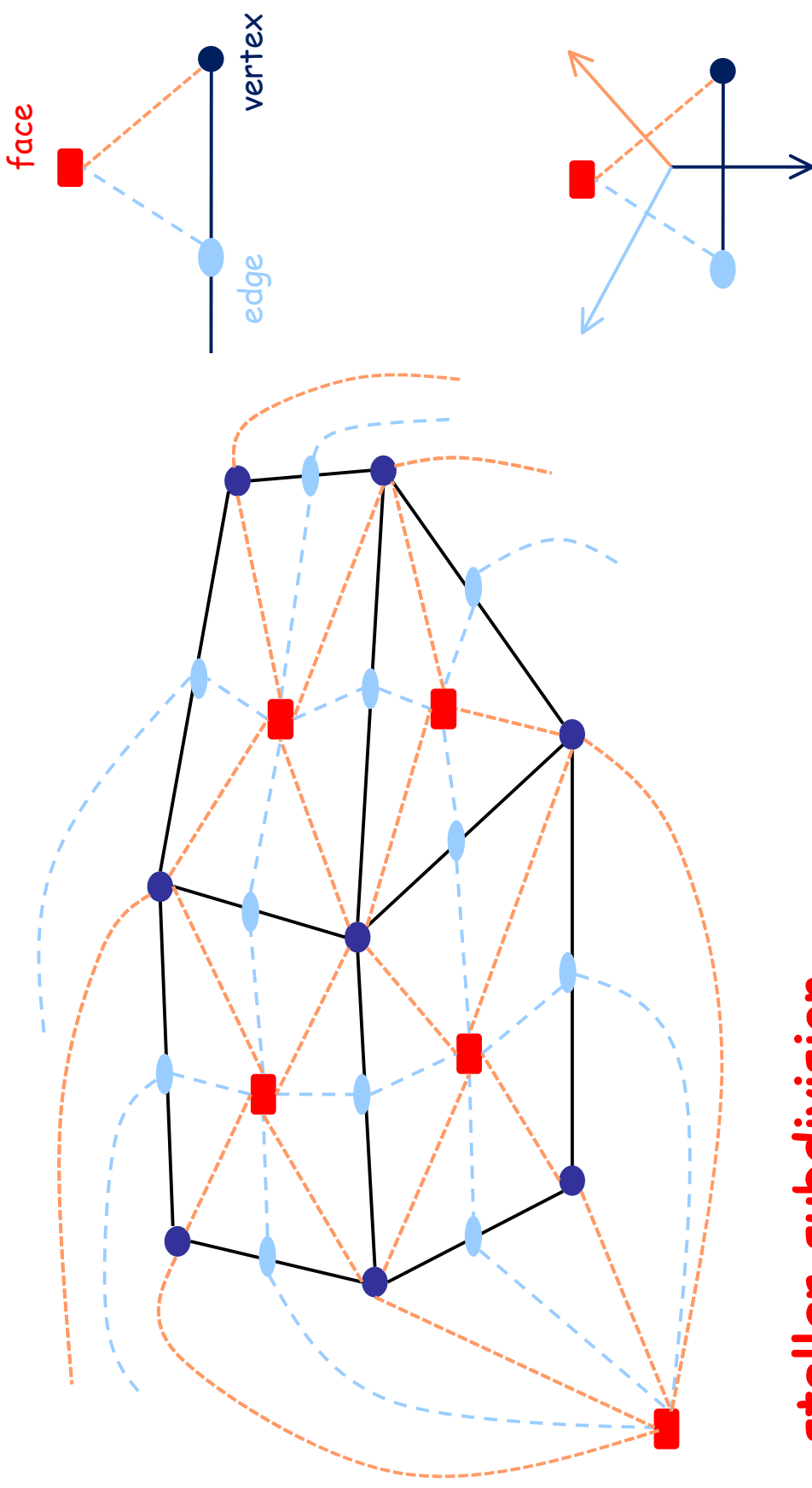


**Idea:** Make general planar subdivision into triangulation



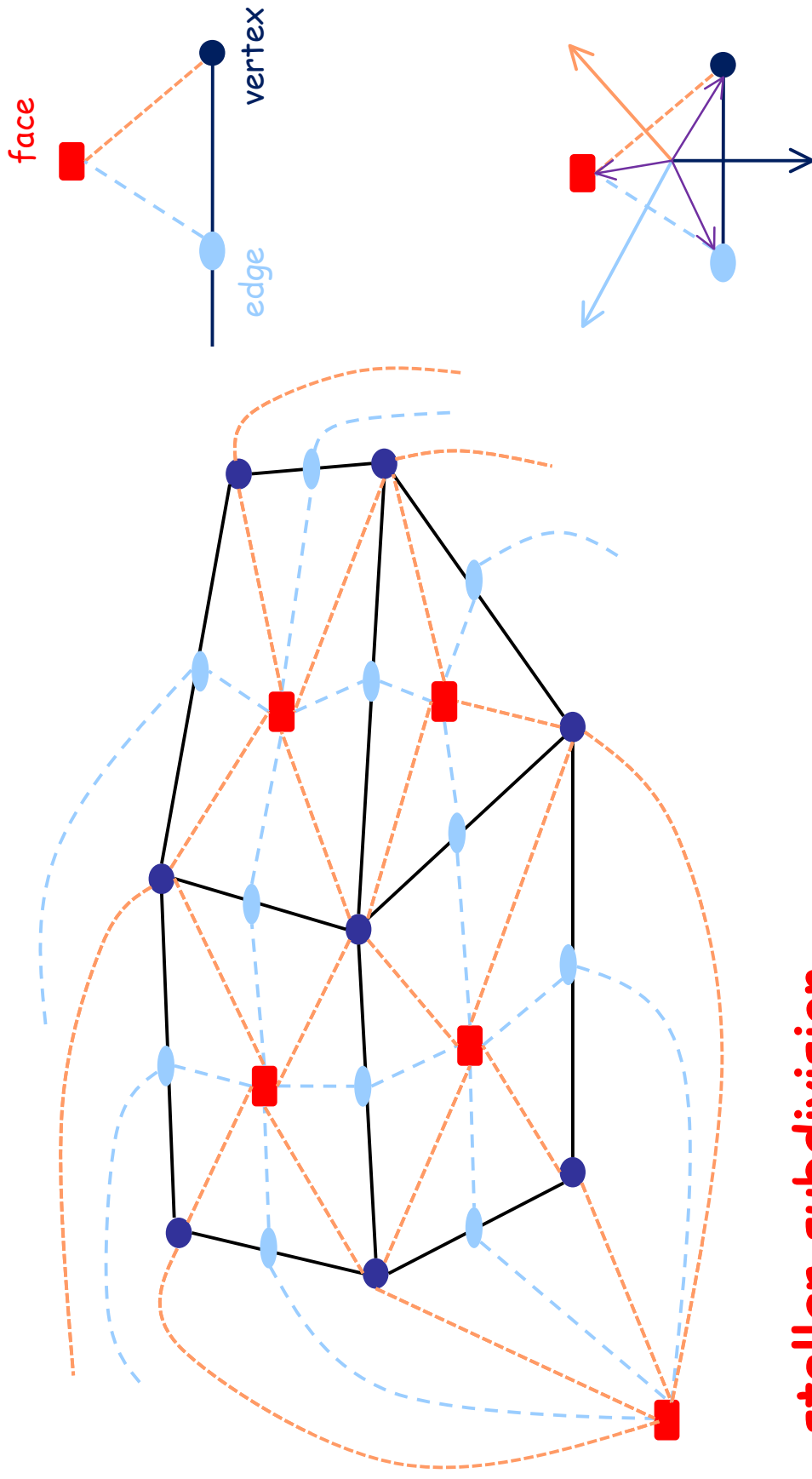
**stellar subdivision**

**Idea:** Make general planar subdivision into triangulation



**stellar subdivision**

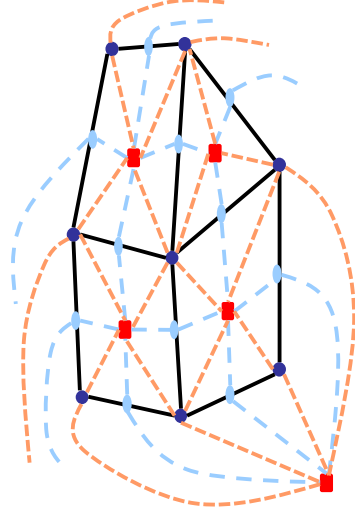
**Idea:** Make general planar subdivision into triangulation



**stellar subdivision**

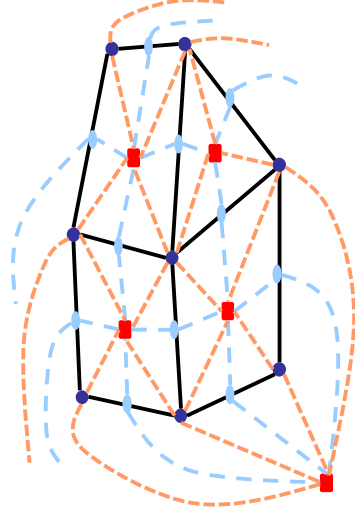
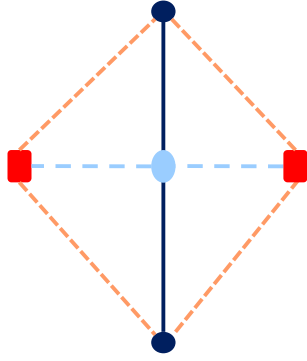
Triangles occur in groups of four (around each edge)

$\Rightarrow$  some pointers can be stored implicitly



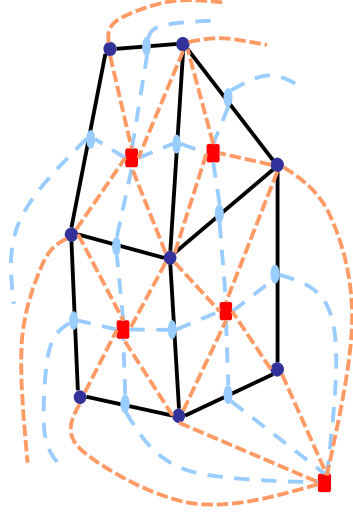
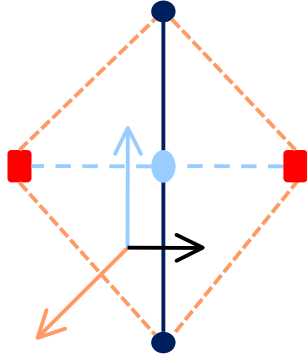
Triangles occur in groups of four (around each edge)

$\Rightarrow$  some pointers can be stored implicitly



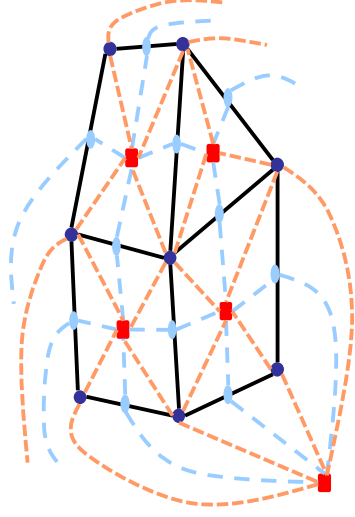
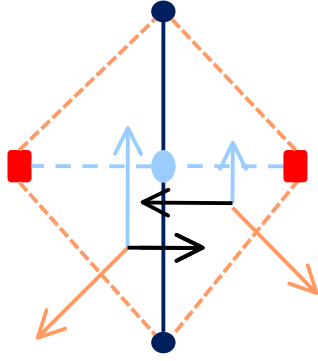
Triangles occur in groups of four (around each edge)

$\Rightarrow$  some pointers can be stored implicitly



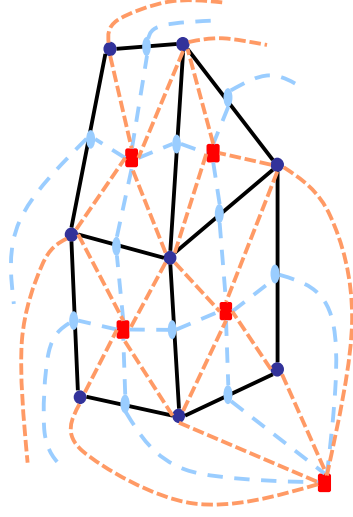
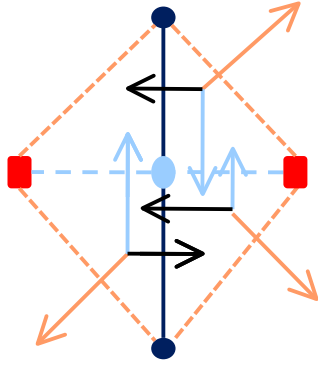
Triangles occur in groups of four (around each edge)

$\Rightarrow$  some pointers can be stored implicitly



Triangles occur in groups of four (around each edge)

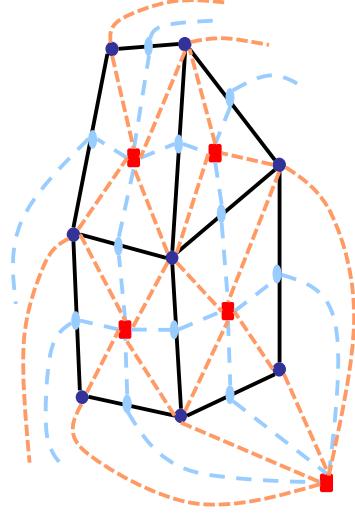
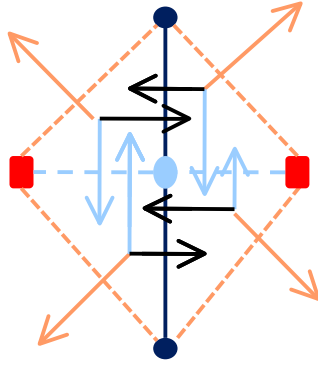
$\Rightarrow$  some pointers can be stored implicitly





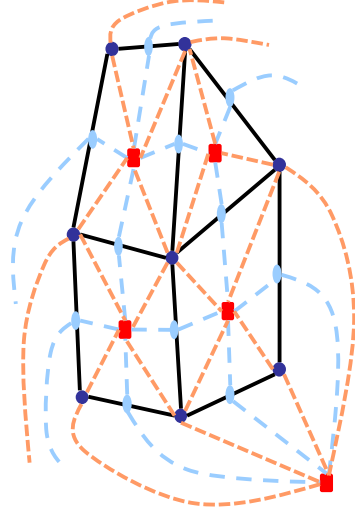
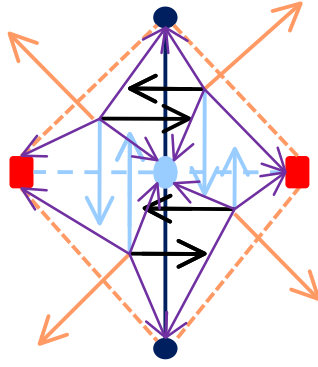
Triangles occur in groups of four (around each edge)

$\Rightarrow$  some pointers can be stored implicitly



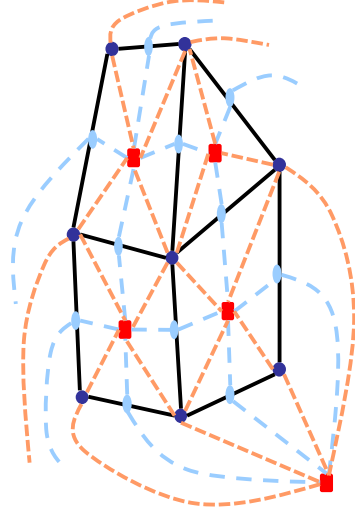
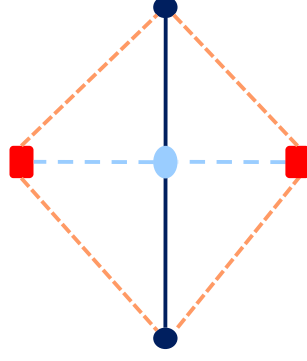
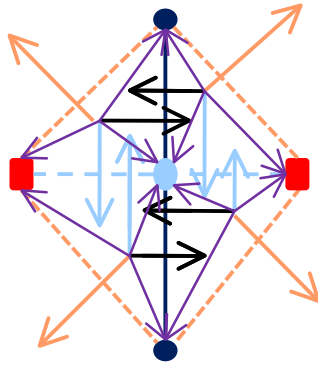
Triangles occur in groups of four (around each edge)

$\Rightarrow$  some pointers can be stored implicitly



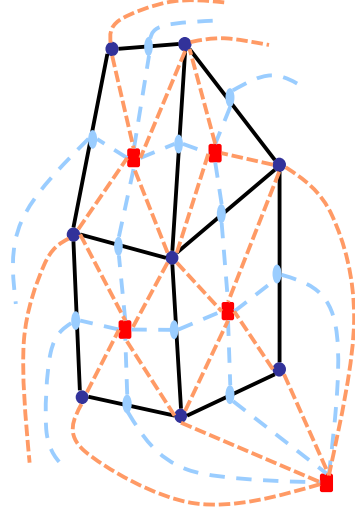
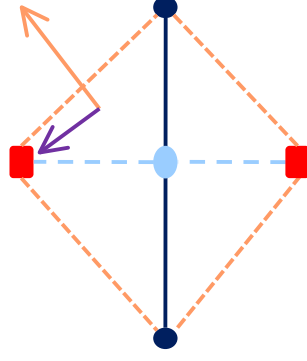
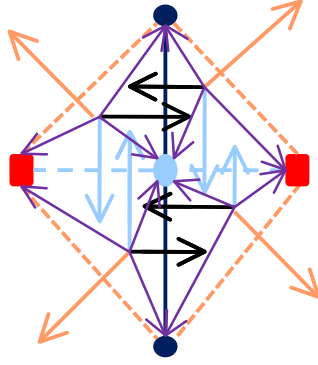
Triangles occur in groups of four (around each edge)

$\Rightarrow$  some pointers can be stored implicitly



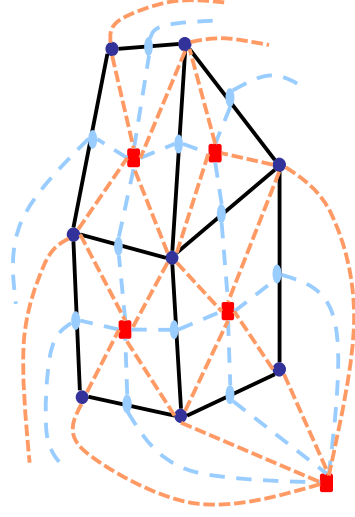
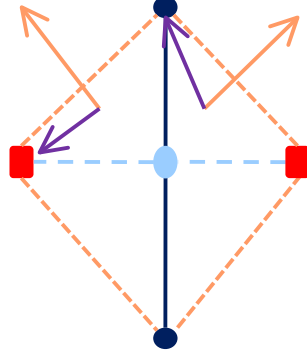
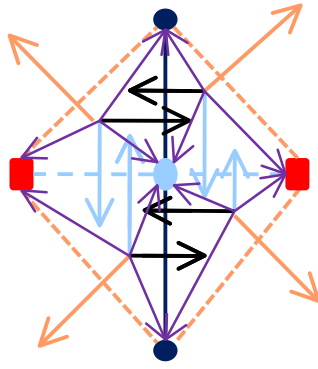
Triangles occur in groups of four (around each edge)

$\Rightarrow$  some pointers can be stored implicitly



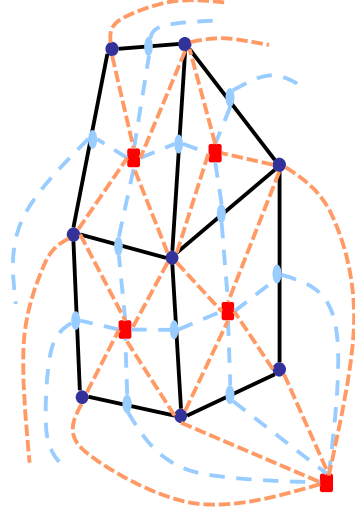
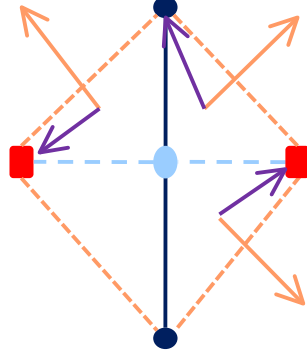
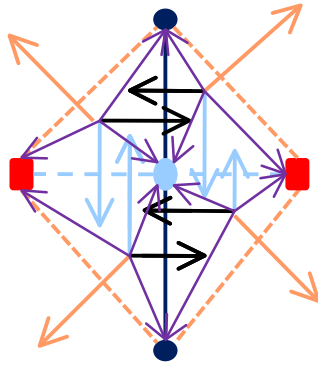
Triangles occur in groups of four (around each edge)

$\Rightarrow$  some pointers can be stored implicitly



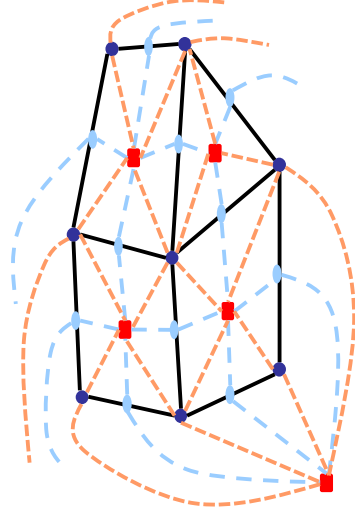
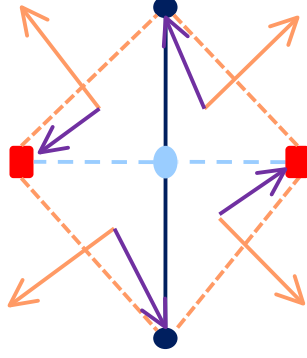
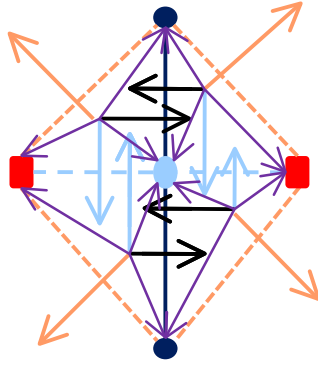
Triangles occur in groups of four (around each edge)

$\Rightarrow$  some pointers can be stored implicitly



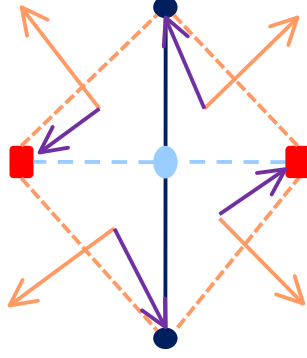
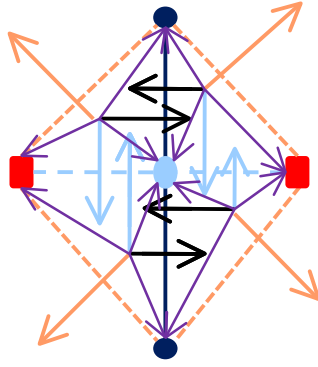
Triangles occur in groups of four (around each edge)

$\Rightarrow$  some pointers can be stored implicitly

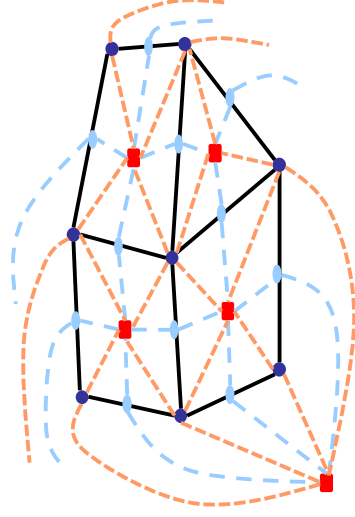


Triangles occur in groups of four (around each edge)

$\Rightarrow$  some pointers can be stored implicitly



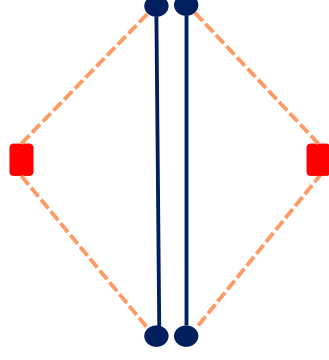
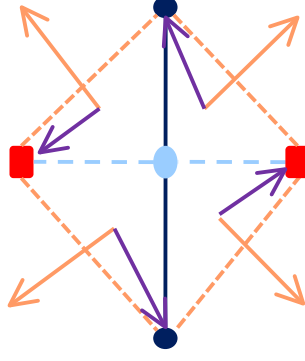
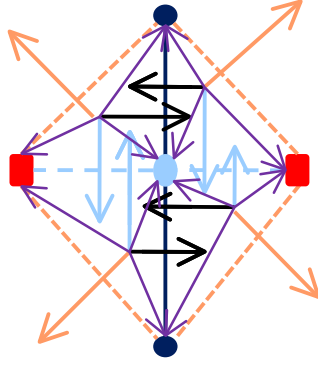
quadedge



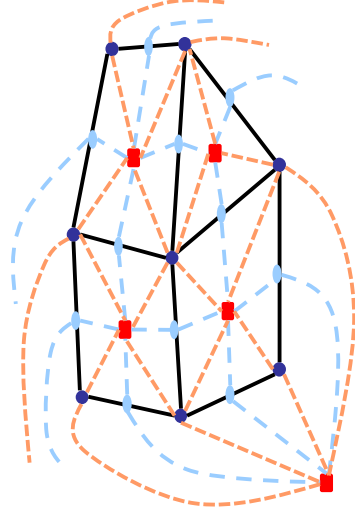


Triangles occur in groups of four (around each edge)

$\Rightarrow$  some pointers can be stored implicitly

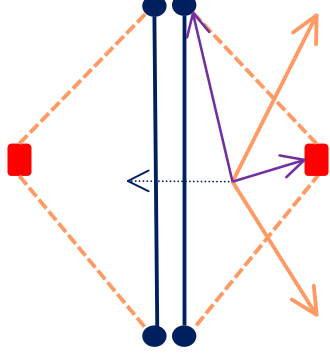
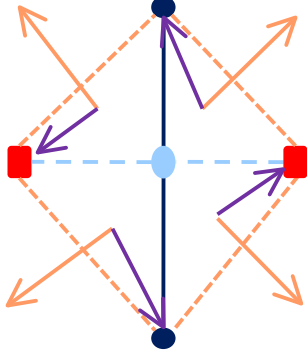
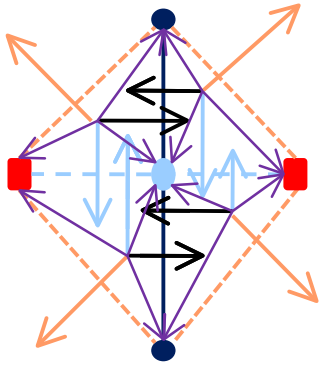


quadedge

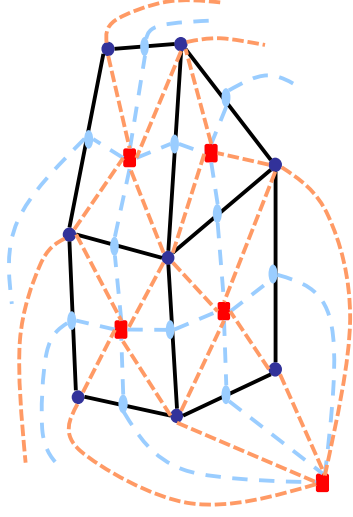


Triangles occur in groups of four (around each edge)

$\Rightarrow$  some pointers can be stored implicitly

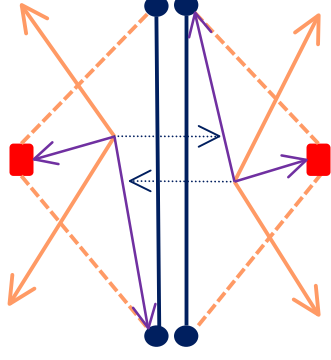
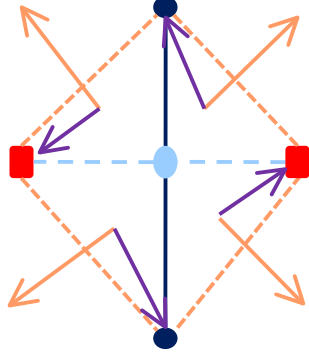
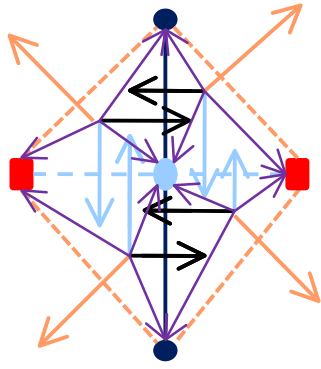


quadedge

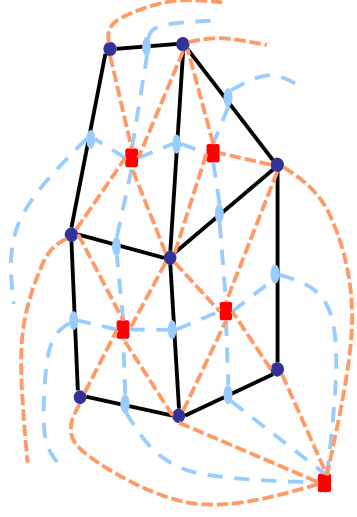


Triangles occur in groups of four (around each edge)

$\Rightarrow$  some pointers can be stored implicitly

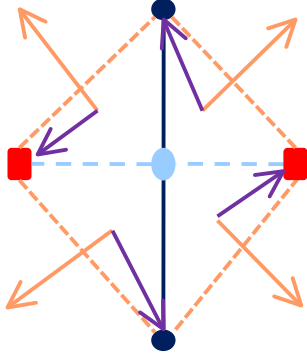
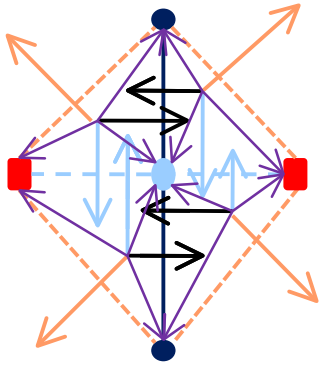


quadedge

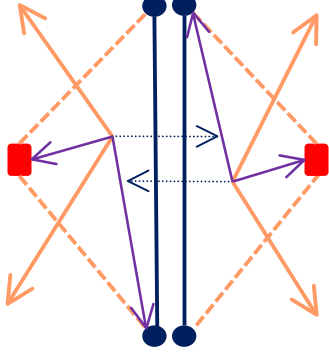


# Triangles occur in groups of four (around each edge)

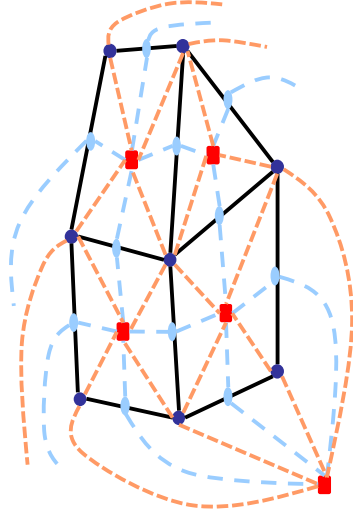
⇒ some pointers can be stored implicitly



quadedge

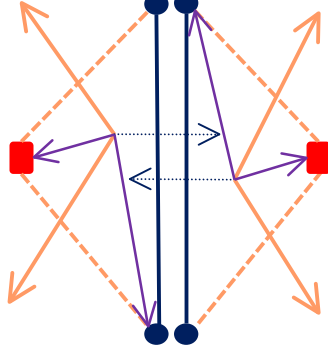
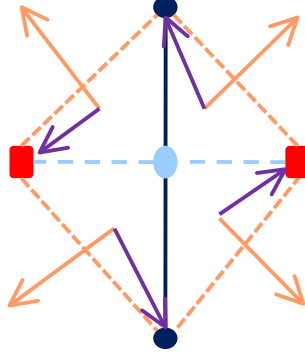
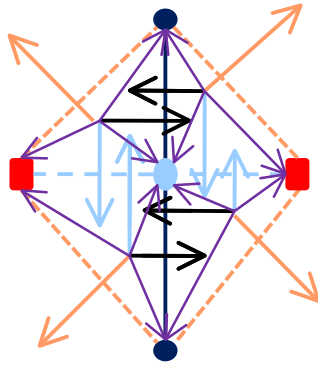


DCEL  
winged edge



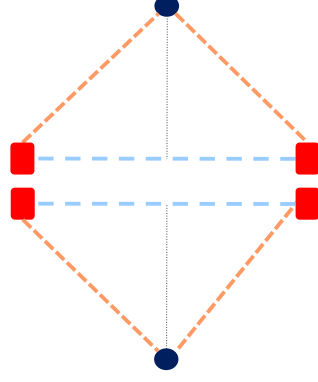
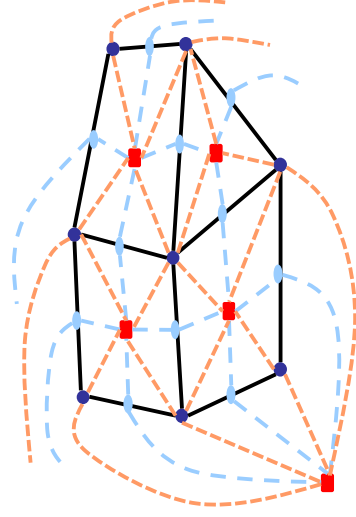
# Triangles occur in groups of four (around each edge)

⇒ some pointers can be stored implicitly



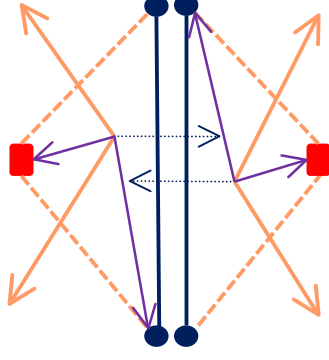
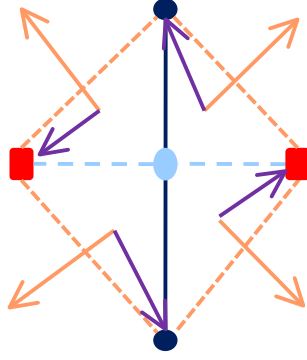
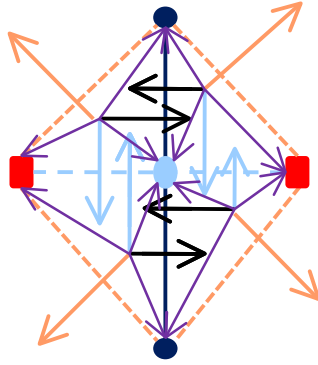
quadedge

DCEL  
winged edge



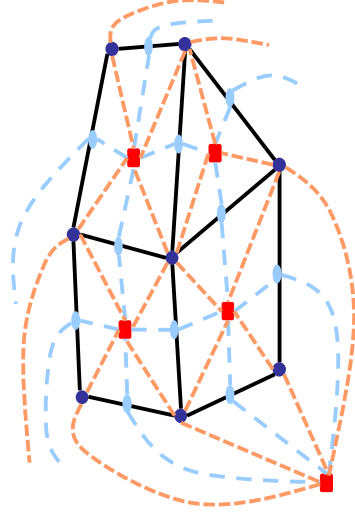
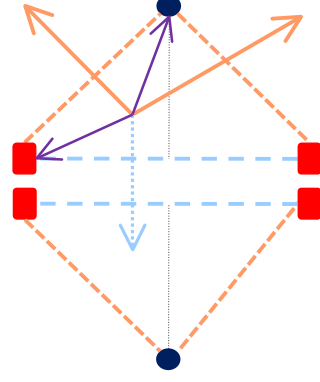
# Triangles occur in groups of four (around each edge)

$\Rightarrow$  some pointers can be stored implicitly



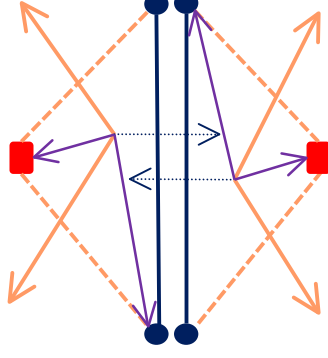
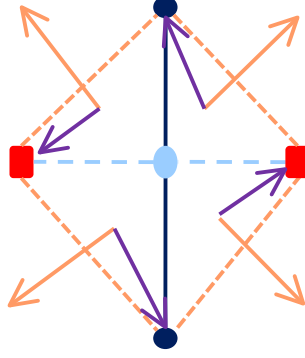
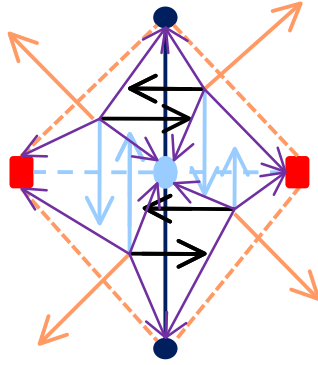
quadedge

DCEL  
winged edge



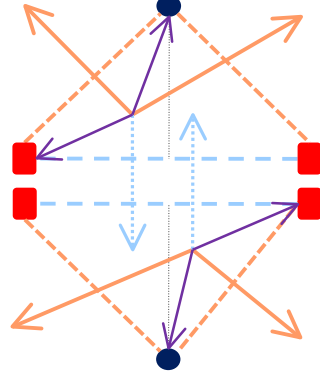
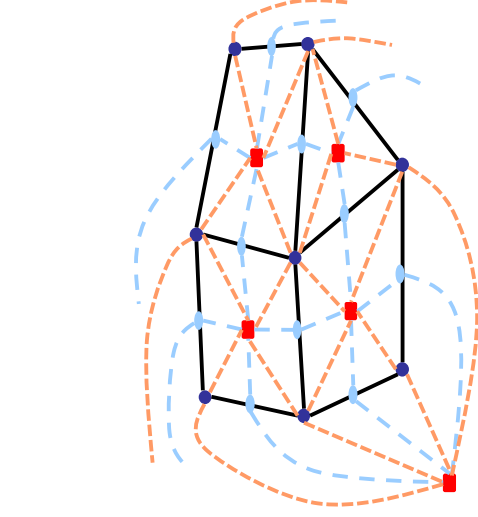
Triangles occur in groups of four (around each edge)

$\Rightarrow$  some pointers can be stored implicitly



quadedge

DCEL  
winged edge



# 1993 Teaching Computational Geometry, I

## Topics

- Randomized algorithms as first class objects
- The importance of invariants for sweep algorithms
- Representation of planar subdivisions
- Planar point location
- Linear Programming
- 3d Convex hulls
- Perturbations
- Upper bound theorem for polytopes



- **Planar point location**

## • Planar point location

### Optimal methods:

- Lipton - Tarjan
- Kirkpatrick
- Edelsbrunner - Guibas - Stolfi
- Cole
- Sarnak - Tarjan
- randomized

## • Planar point location

### Optimal methods:

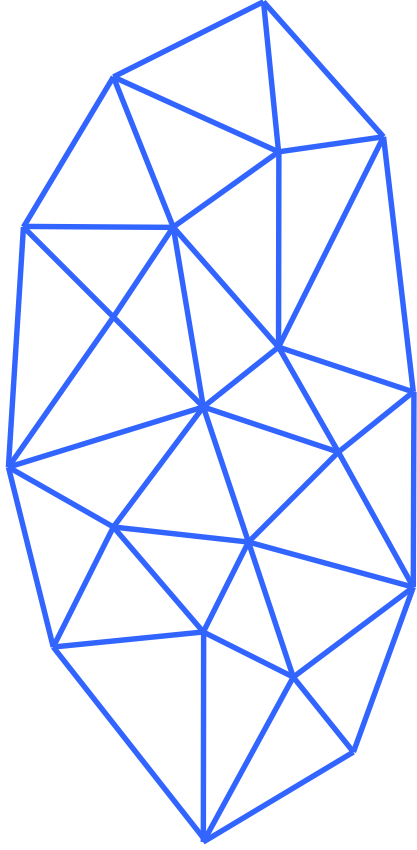
- Lipton - Tarjan
- Kirkpatrick
- Edelsbrunner - Guibas - Stolfi
- Cole
- Sarnak - Tarjan
- randomized

### Other methods:

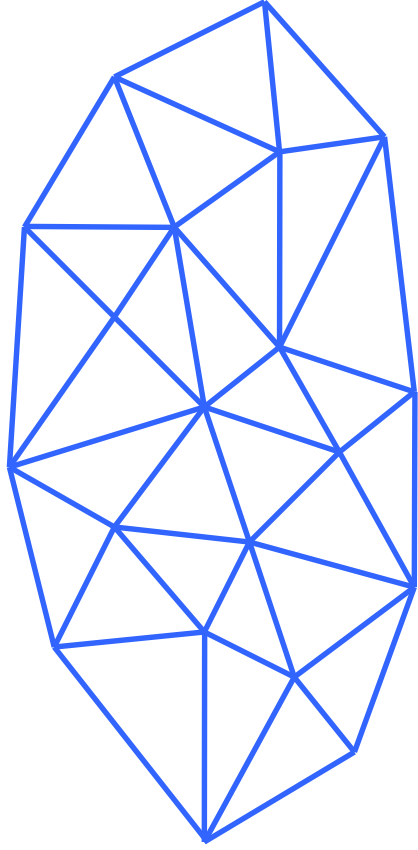
- via segment trees / via interval trees
- trapezoidal search trees
- constant optimal methods
- via cuttings
- distribution adaptive methods
- ...

# Kirkpatrick's hierarchy for straight edge, triangulated subdivisions

subdivision  $G$

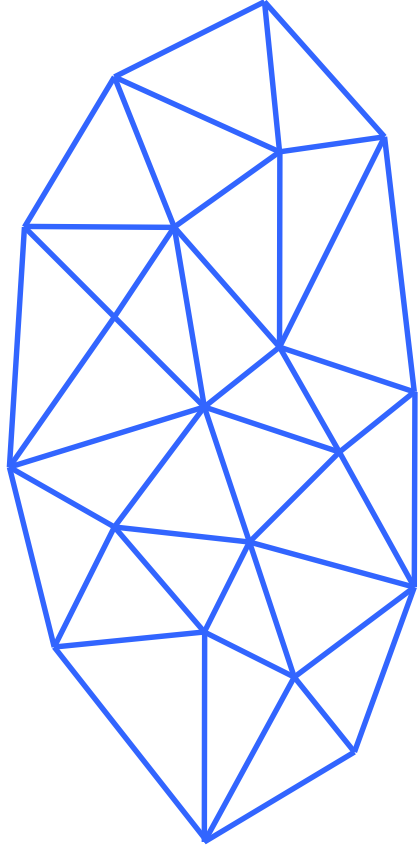


# Kirkpatrick's hierarchy for straight edge, triangulated subdivisions



subdivision  $G$   
to obtain smaller  $G'$

# Kirkpatrick's hierarchy for straight edge, triangulated subdivisions

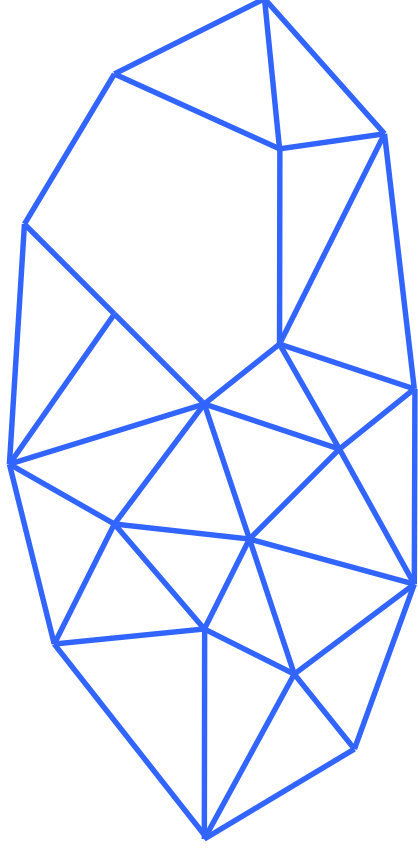


subdivision  $G$

to obtain smaller  $G'$

remove low degree vertex  
and retriangulate hole

# Kirkpatrick's hierarchy for straight edge, triangulated subdivisions

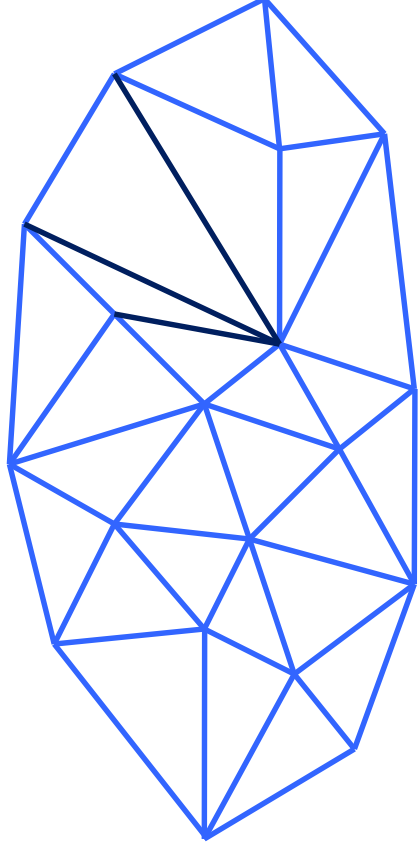


subdivision  $G$

to obtain smaller  $G'$

remove low degree vertex  
and retriangulate hole

# Kirkpatrick's hierarchy for straight edge, triangulated subdivisions



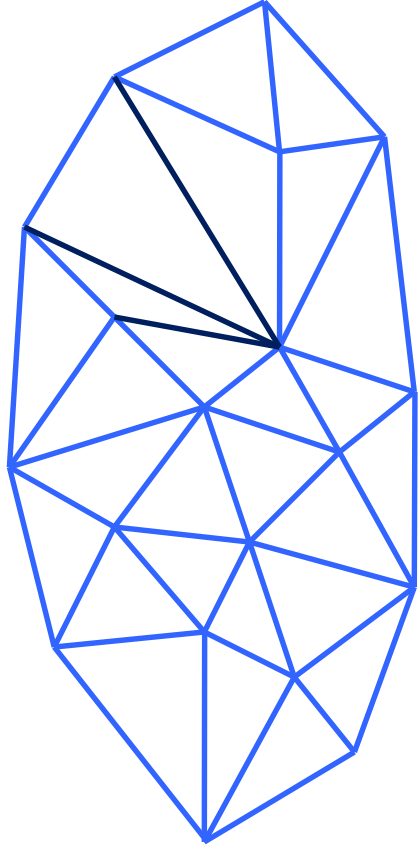
subdivision  $G$

to obtain smaller  $G'$

remove low degree vertex  
and retriangulate hole



# Kirkpatrick's hierarchy for straight edge, triangulated subdivisions



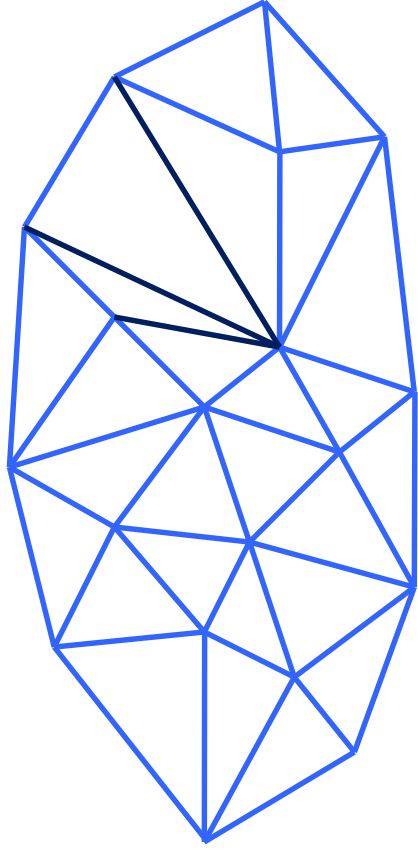
subdivision  $G$

to obtain smaller  $G'$

remove low degree vertex  
and retriangulate hole

repeat recursively

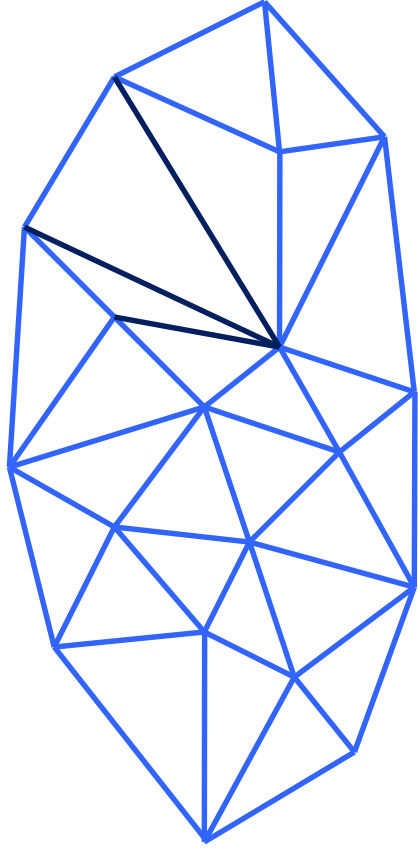
# Kirkpatrick's hierarchy for straight edge, triangulated subdivisions



Query for point  $q$  :

subdivision  $G$   
to obtain smaller  $G'$   
remove low degree vertex  
and retriangulate hole  
repeat recursively

# Kirkpatrick's hierarchy for straight edge, triangulated subdivisions



subdivision  $G$

to obtain smaller  $G'$

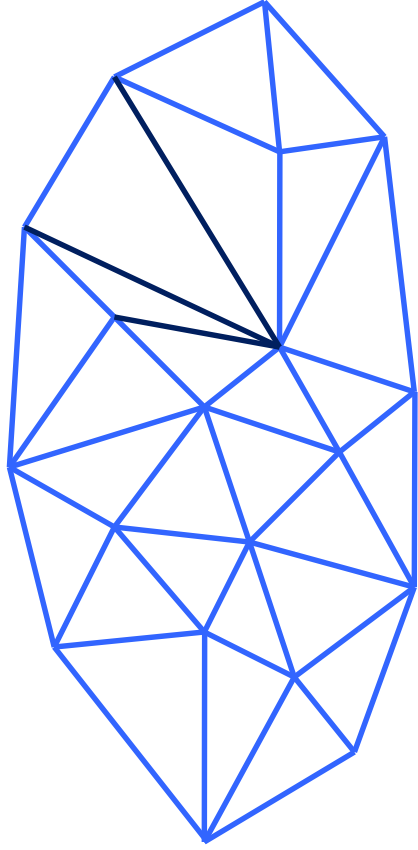
remove low degree vertex  
and retriangulate hole

repeat recursively

Query for point  $q$  :

locate  $q$  in  $G'$

# Kirkpatrick's hierarchy for straight edge, triangulated subdivisions



subdivision  $G$

to obtain smaller  $G'$

remove low degree vertex  
and retriangulate hole

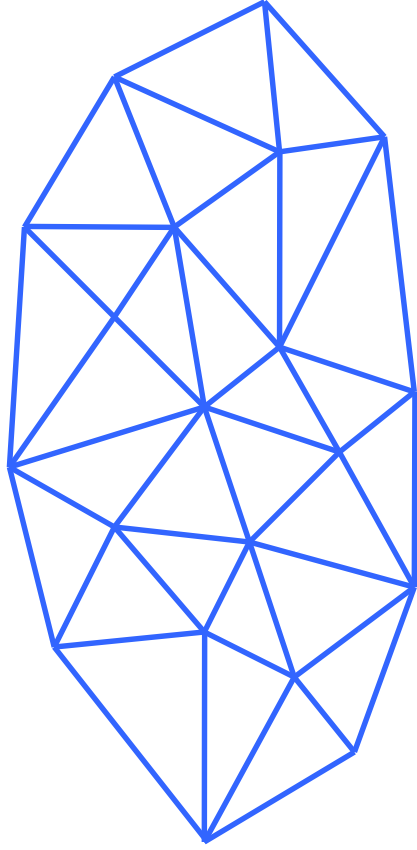
repeat recursively

Query for point  $q$  :

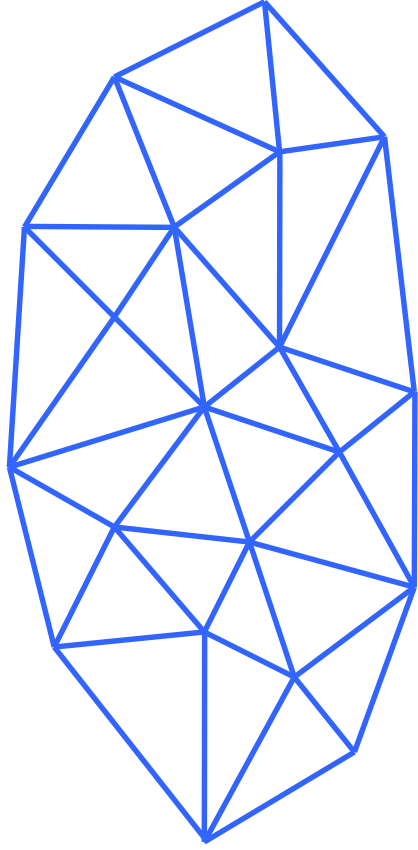
locate  $q$  in  $G'$

if  $q$  in "black" triangle **then** determine correct triangle of  $G$   
**else** triangle is correct answer already

# Kirkpatrick's hierarchy for straight edge, triangulated subdivisions



# Kirkpatrick's hierarchy for straight edge, triangulated subdivisions

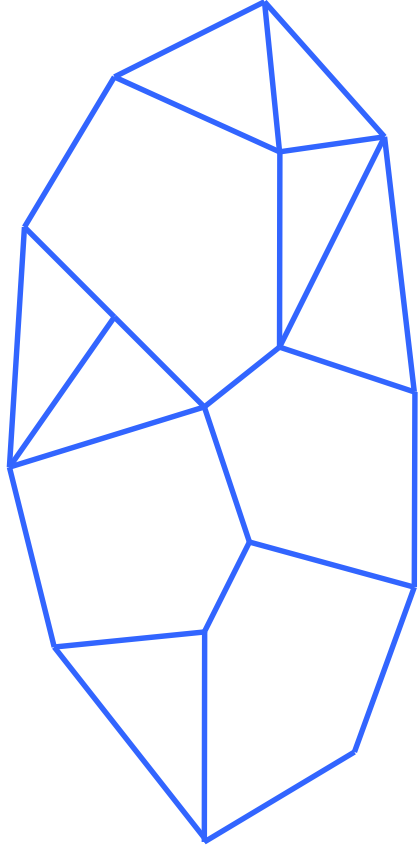


subdivision  $G$

to obtain smaller  $G'$

remove large independent  
set of low degree vertices  
and retriangulate holes

# Kirkpatrick's hierarchy for straight edge, triangulated subdivisions

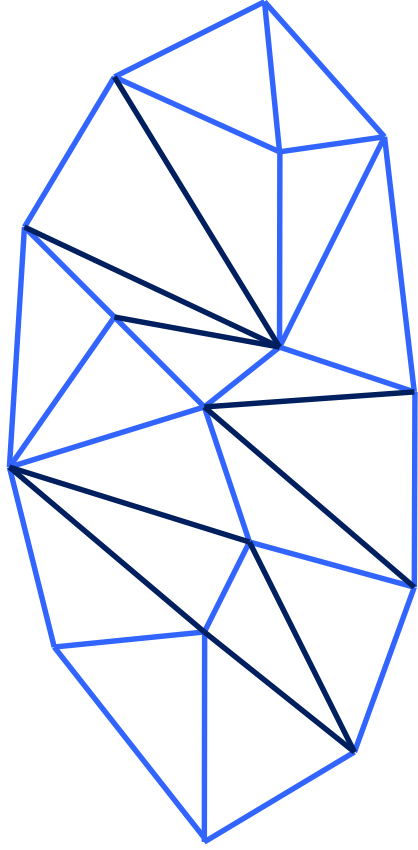


subdivision  $G$

to obtain smaller  $G'$

remove large independent  
set of low degree vertices  
and retriangulate holes

# Kirkpatrick's hierarchy for straight edge, triangulated subdivisions



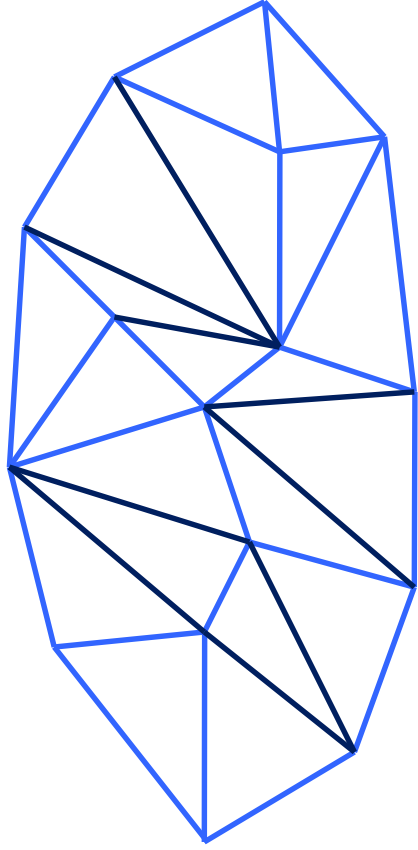
subdivision  $G$

to obtain smaller  $G'$

remove large independent  
set of low degree vertices  
and retriangulate holes



# Kirkpatrick's hierarchy for straight edge, triangulated subdivisions



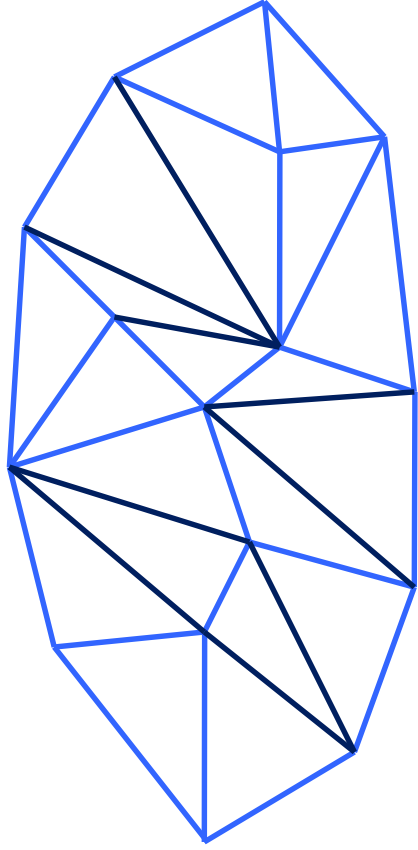
subdivision  $G$

to obtain smaller  $G'$

remove large independent  
set of low degree vertices  
and retriangulate holes

repeat recursively

# Kirkpatrick's hierarchy for straight edge, triangulated subdivisions



subdivision  $G$

to obtain smaller  $G'$

remove large independent  
set of low degree vertices  
and retriangulate holes

repeat recursively

Query for point  $q$  :

locate  $q$  in  $G'$

if  $q$  in "black" triangle then determine correct triangle of  $G$   
else triangle is correct answer already

**Lemma:** For every  $d \geq 6$  there exists an  $\alpha > 0$  such that every  $n$ -vertex planar graph has an independent set of at least  $\alpha n$  vertices of degree  $\leq d$ .

**Lemma:** For every  $d \geq 6$  there exists an  $\alpha > 0$  such that every  $n$ -vertex planar graph has an independent set of at least  $\alpha n$  vertices of degree  $\leq d$ .

$\Rightarrow$  height of hierarchy of subdivisions can be made  $O(\log n)$

**Lemma:** For every  $d \geq 6$  there exists an  $\alpha > 0$  such that every  $n$ -vertex planar graph has an independent set of at least  $\alpha n$  vertices of degree  $\leq d$ .

$\Rightarrow$  height of hierarchy of subdivisions can be made  $O(\log n)$

$\Rightarrow$  Query time  $O(\log n)$   
Space  $O(n)$   
Preprocessing  $O(n)$

**Lemma:** For every  $d \geq 6$  there exists an  $\alpha > 0$  such that every  $n$ -vertex planar graph has an independent set of at least  $\alpha n$  vertices of degree  $\leq d$ .

$\Rightarrow$  height of hierarchy of subdivisions can be made  $O(\log n)$

$\Rightarrow$  Query time  $O(\log n)$   
Space  $O(n)$   
Preprocessing  $O(n)$

### Shortcomings:

- only works for straight edge subdivisions
- constants are large
- "complicated" (needs to find independent sets)

## Shortcomings:

- only works for straight edge subdivisions
- constants are large
- "complicated" (needs to find independent sets)

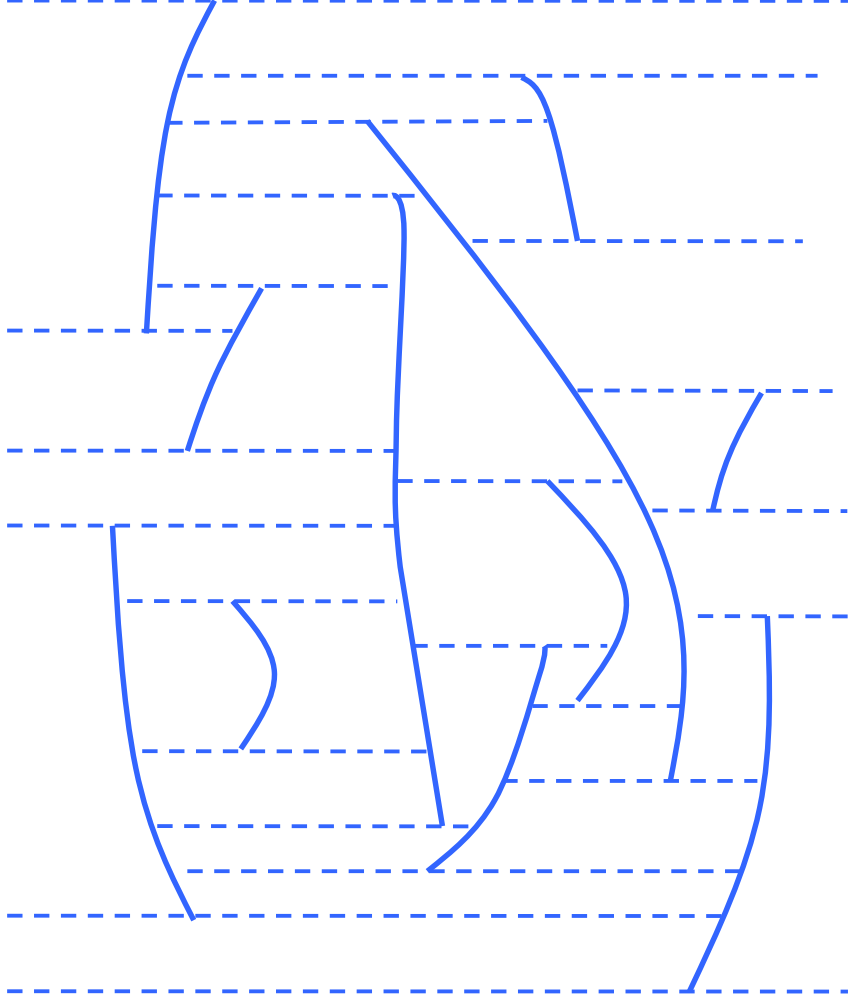
### Shortcomings:

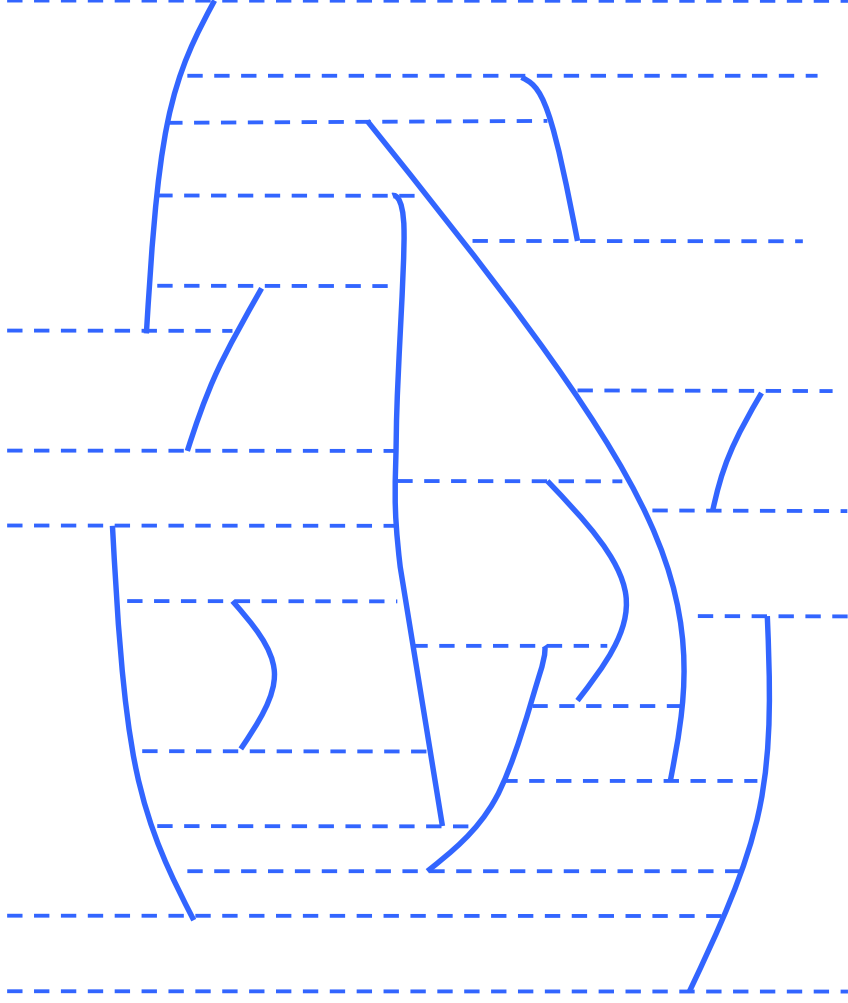
- only works for straight edge subdivisions
- constants are large
- "complicated" (needs to find independent sets)

**Idea:** apply this hierarchical approach to trapezoidations and but remove segments instead of vertices.

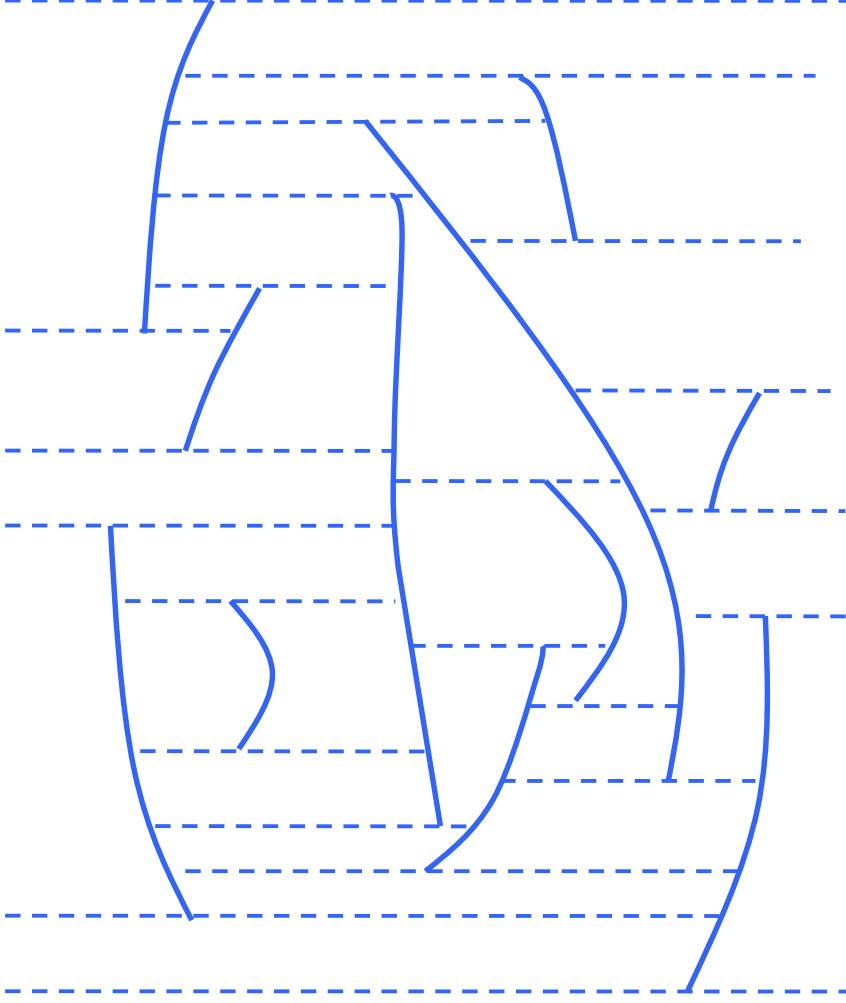


## trapezoidation G

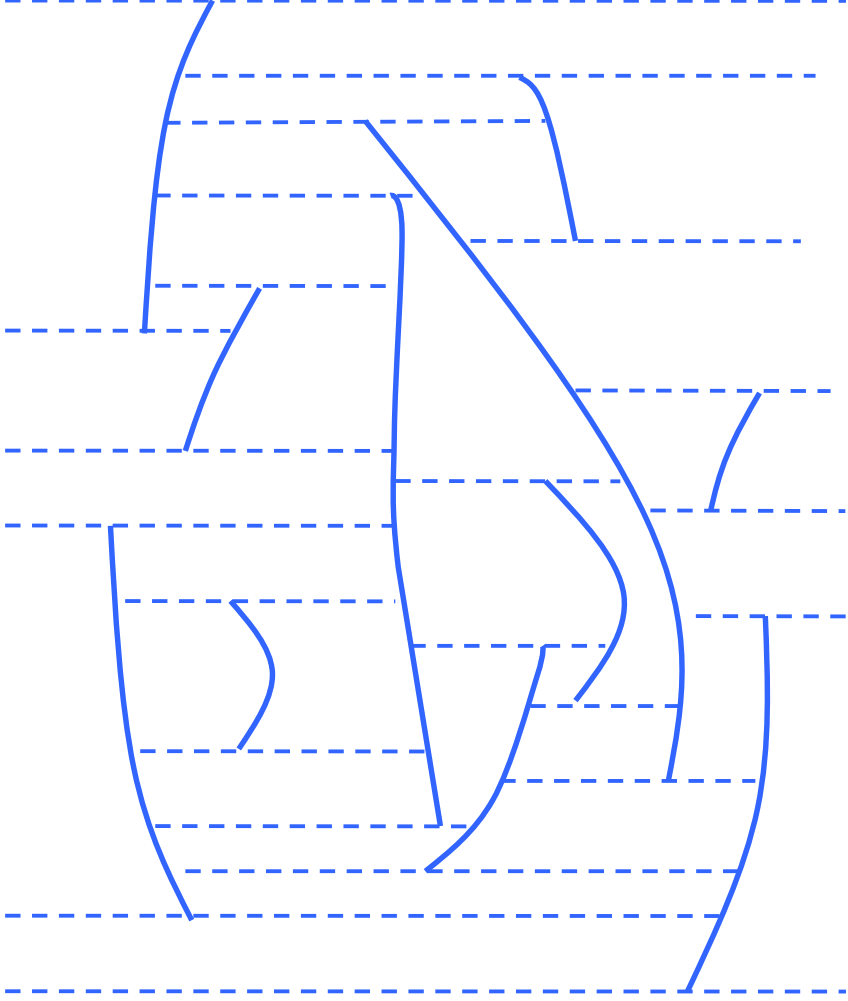




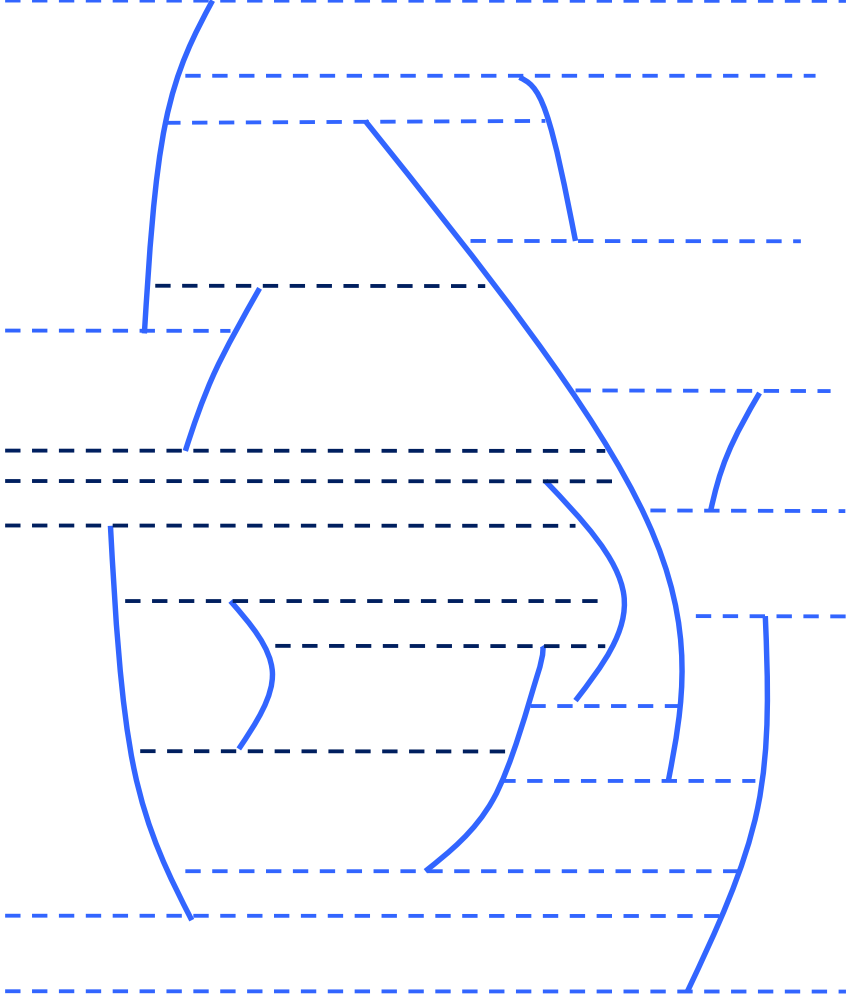
trapezoidation  $G$   
to obtain smaller  $G'$



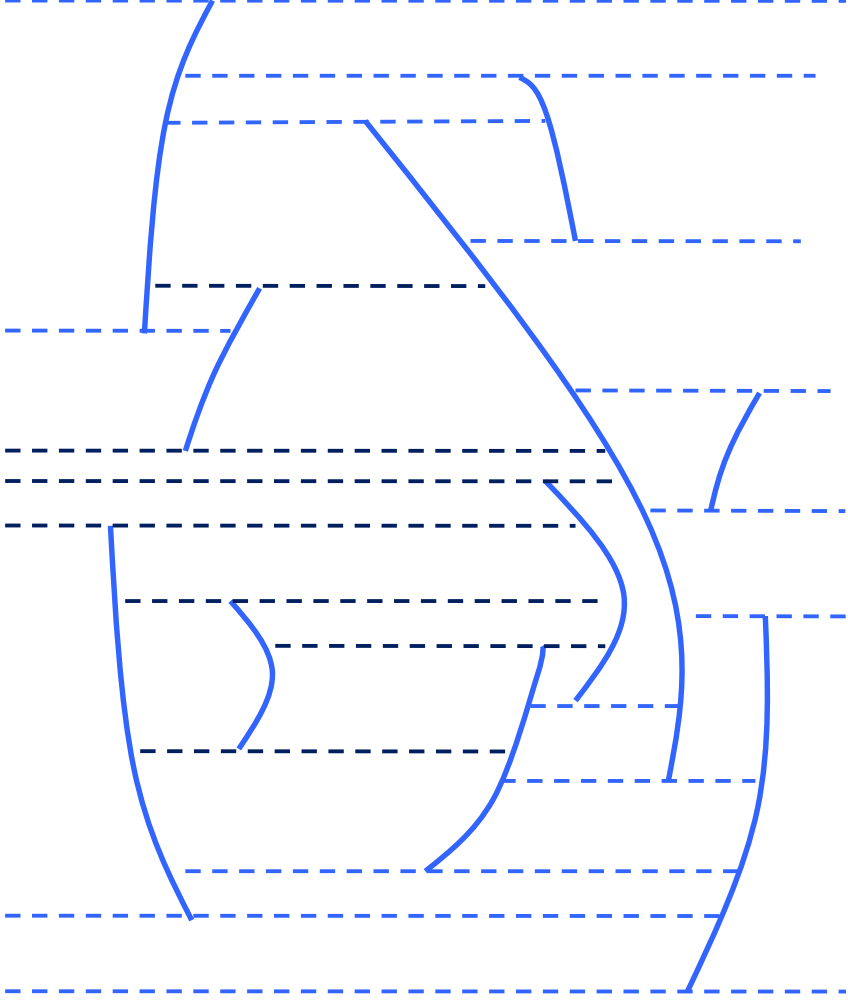
trapezoidation  $G$   
to obtain smaller  $G'$   
remove **some** segment



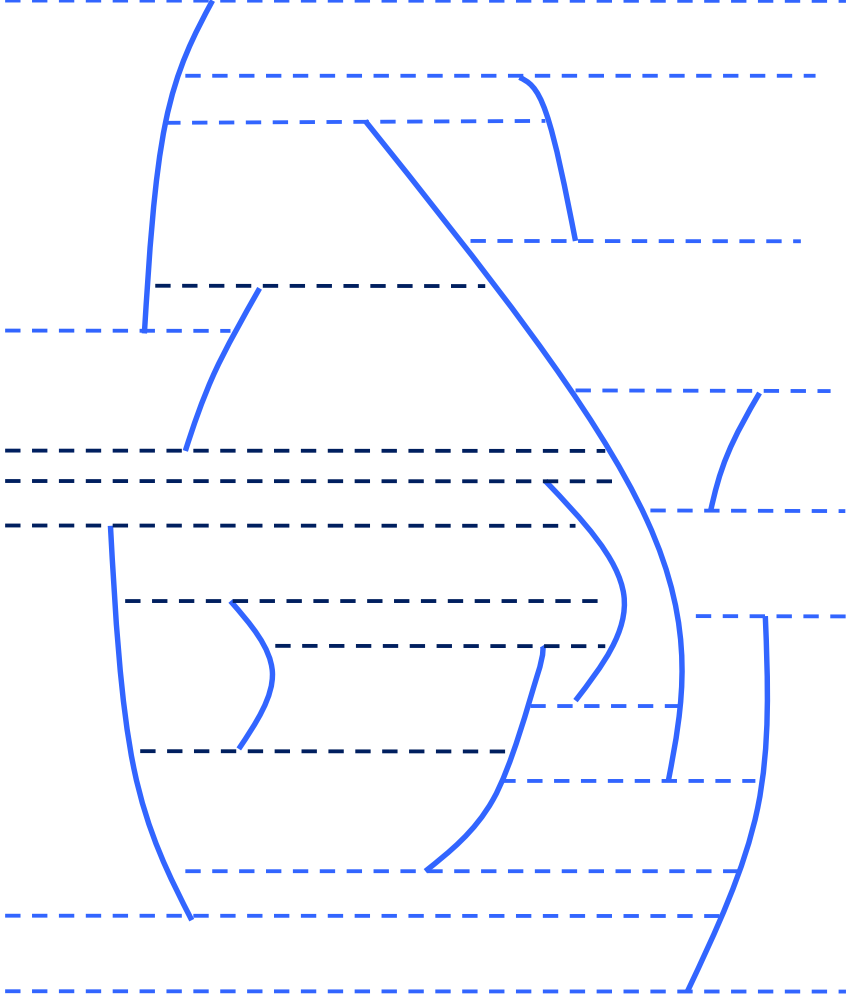
trapezoidation  $G$   
to obtain smaller  $G'$   
remove **some** segment  
and "retapezoidalize" hole



trapezoidation  $G$   
to obtain smaller  $G'$   
remove **some** segment  
and "retrapezoidalize" hole

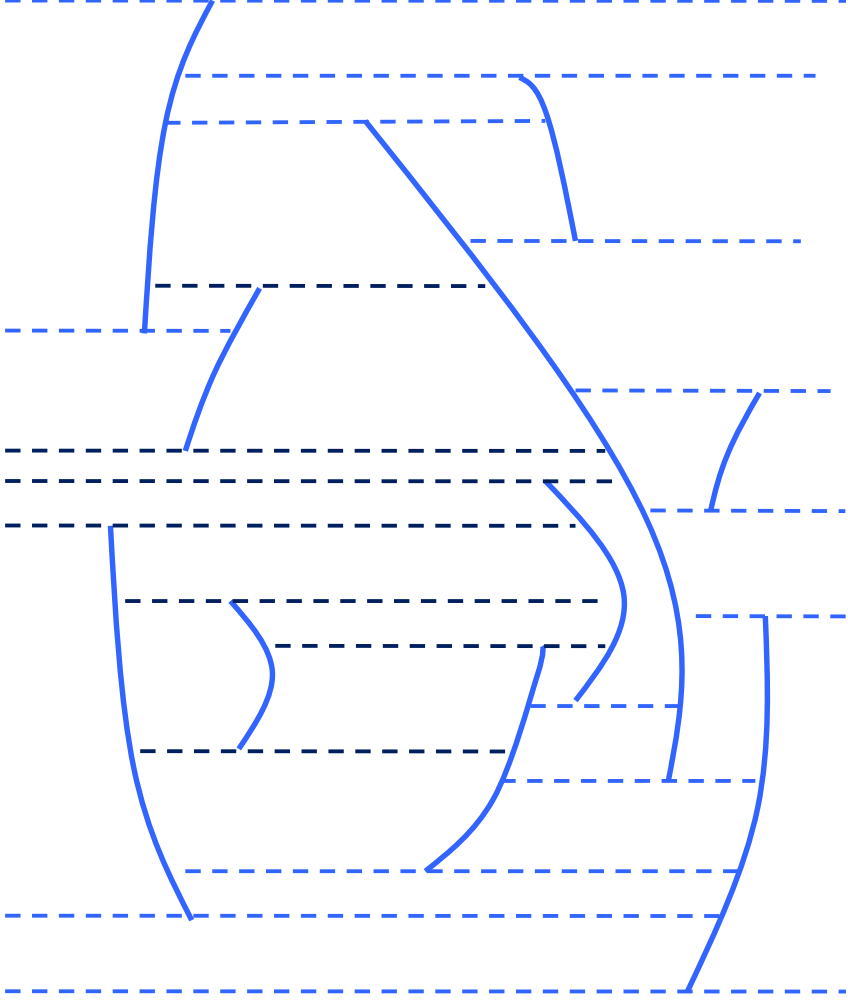


trapezoidation  $G$   
to obtain smaller  $G'$   
remove **some** segment  
and "trapezoidalize" hole  
repeat recursively



Query for point  $q$  :

trapezoidation  $G$   
to obtain smaller  $G'$   
remove **some** segment  
and "retrapezoidalize" hole  
repeat recursively

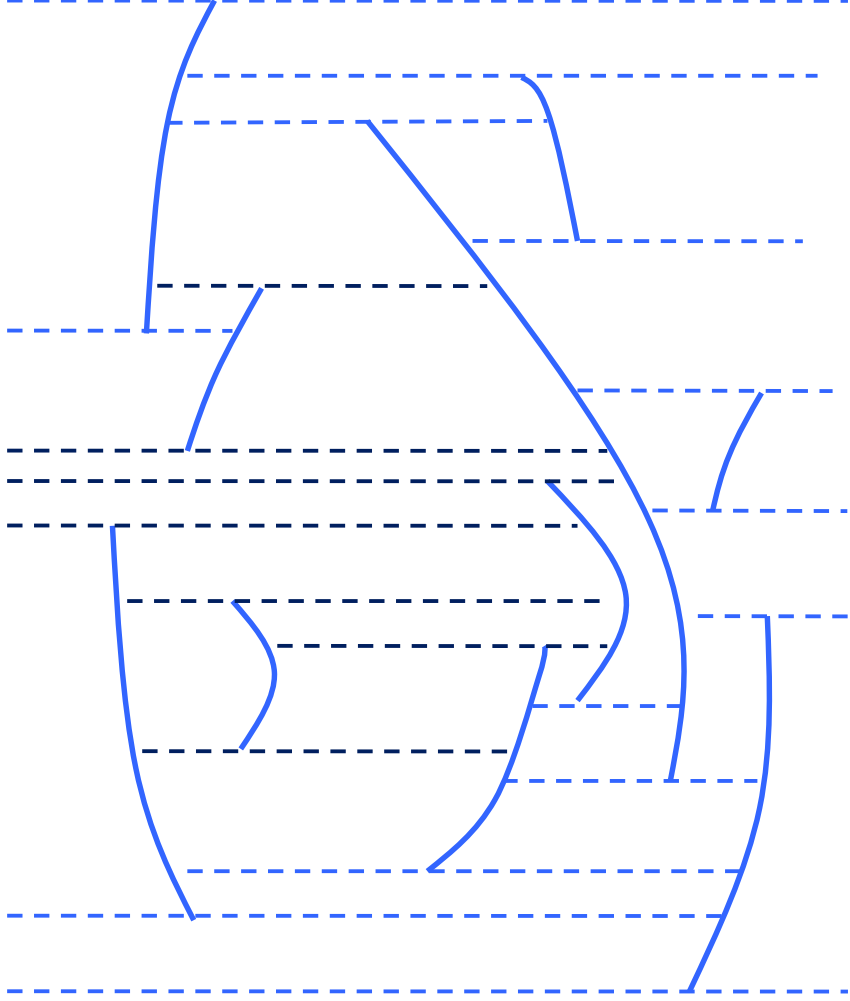


Query for point  $q$  :

locate  $q$  in  $G'$

trapezoidation  $G$   
to obtain smaller  $G'$   
remove **some** segment  
and "retrapezoidalize" hole  
repeat recursively





trapezoidation  $G$

to obtain smaller  $G'$

remove **some** segment  
and "retrapezoidalize" hole

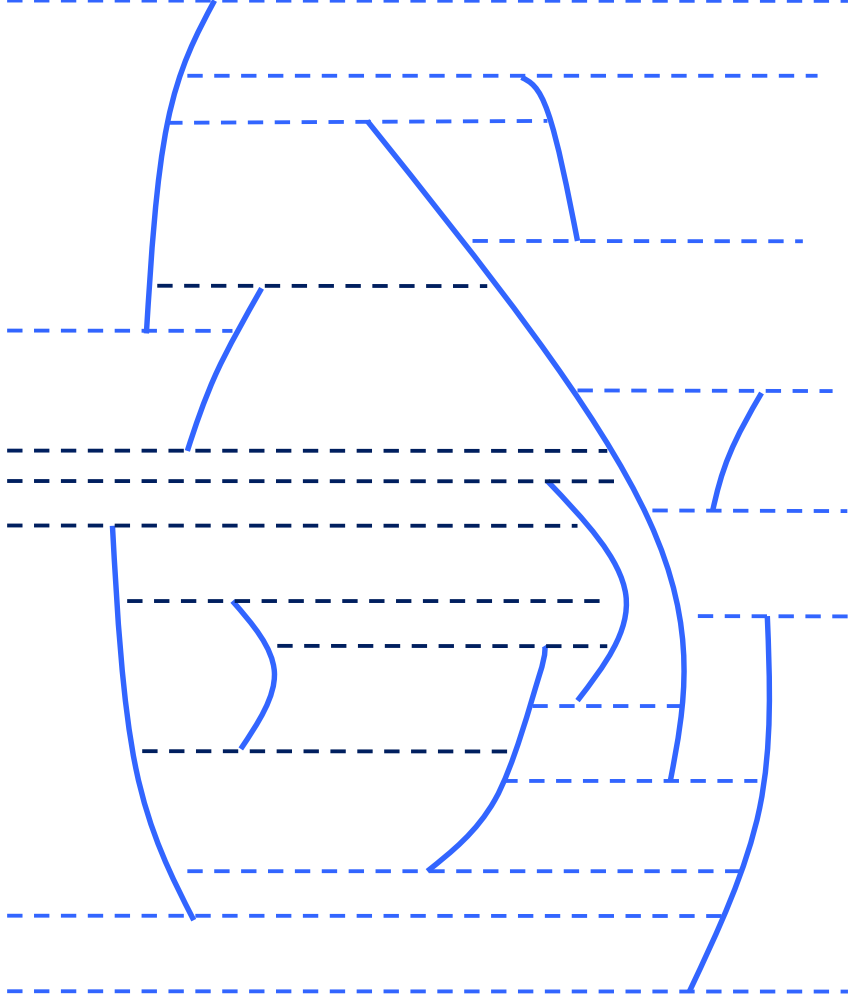
repeat recursively

Query for point  $q$  :

locate  $q$  in  $G'$

if  $q$  in "black" trapezoid **then** determine correct trapezoid of  $G$

**else** trapezoid is correct answer already



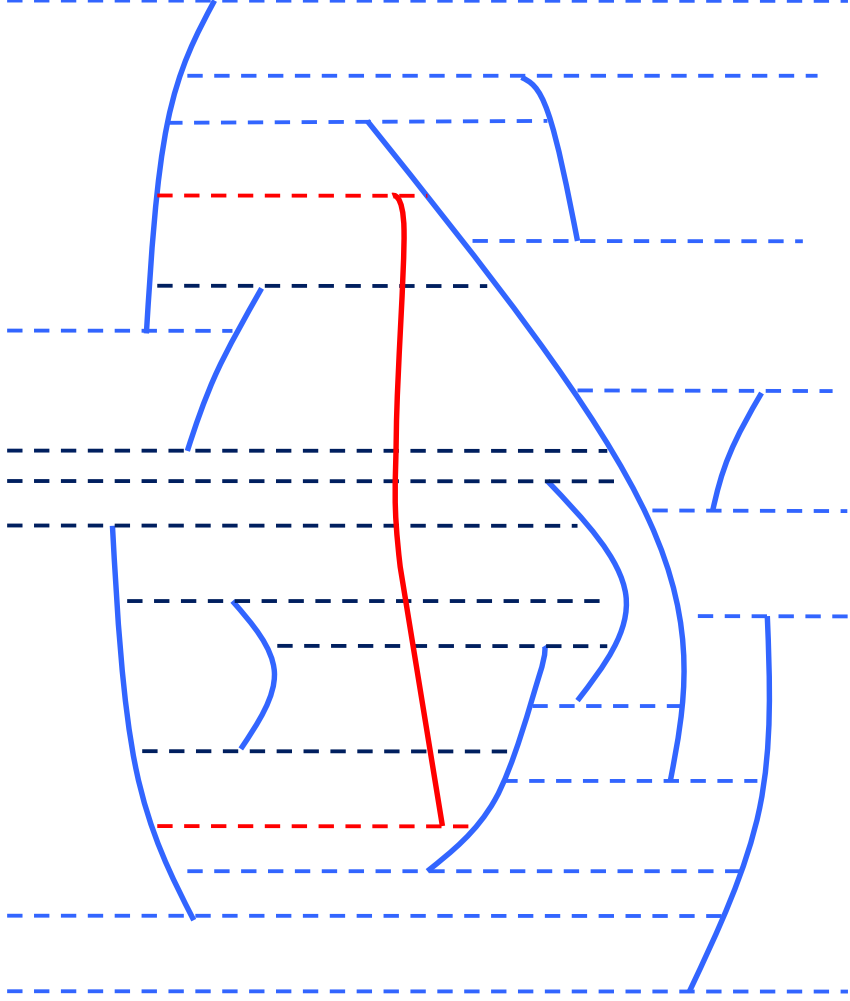
trapezoidation  $G$   
to obtain smaller  $G'$   
remove **some** segment  
and "retrapezoidalize" hole  
repeat recursively

Query for point  $q$  :

locate  $q$  in  $G'$

if  $q$  in "black" trapezoid then determine correct trapezoid of  $G$   
else trapezoid is correct answer already

1 or 2 comparisons !!



trapezoidation  $G$

to obtain smaller  $G'$

remove **some** segment  
and "retrapezoidalize" hole

repeat recursively

Query for point  $q$  :

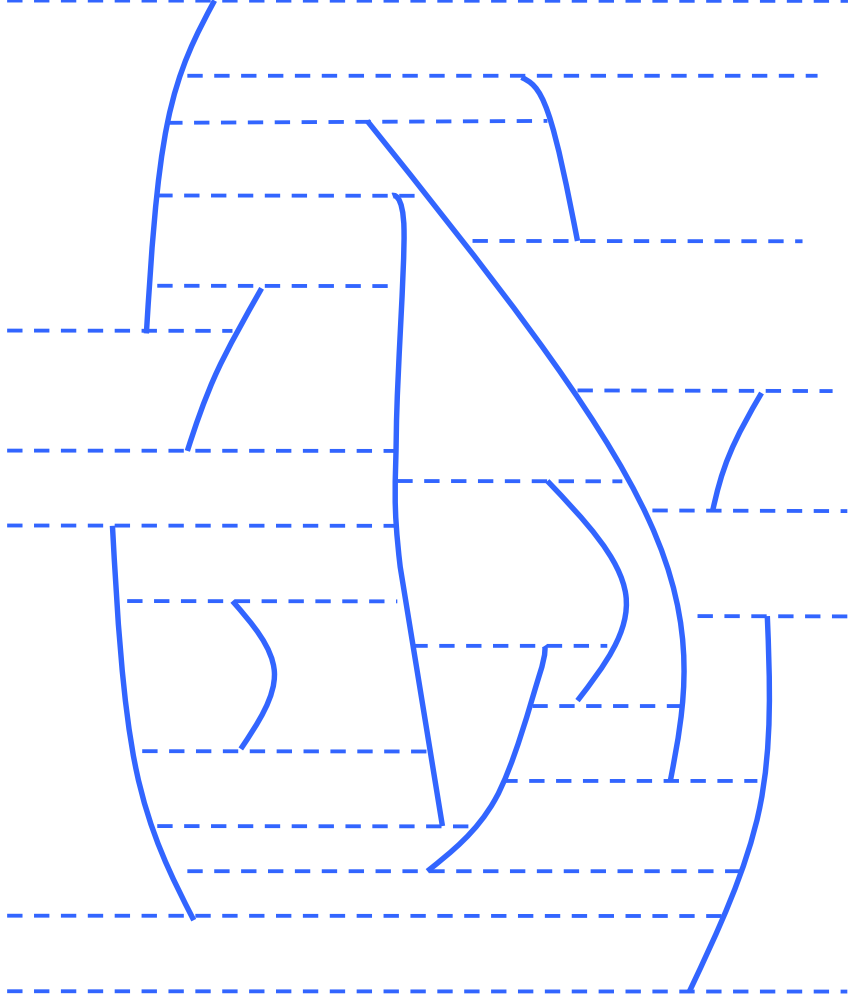
locate  $q$  in  $G'$

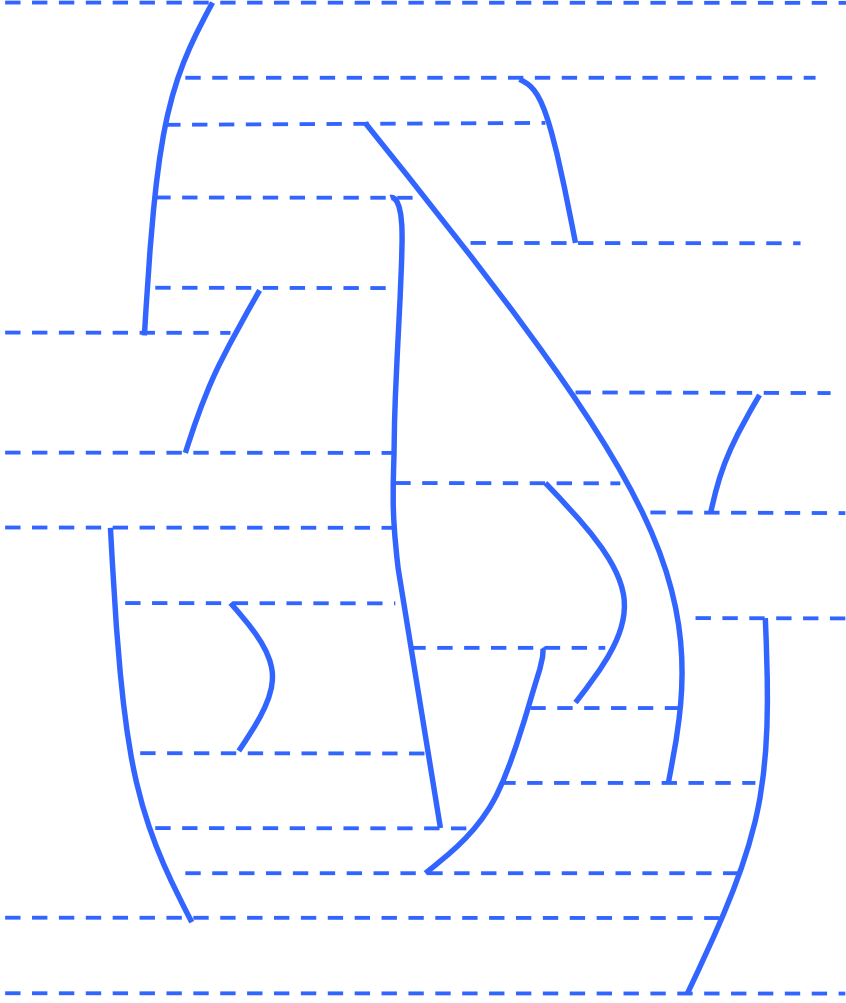
if  $q$  in "black" trapezoid **then** determine correct trapezoid of  $G$

**else** trapezoid is correct answer already

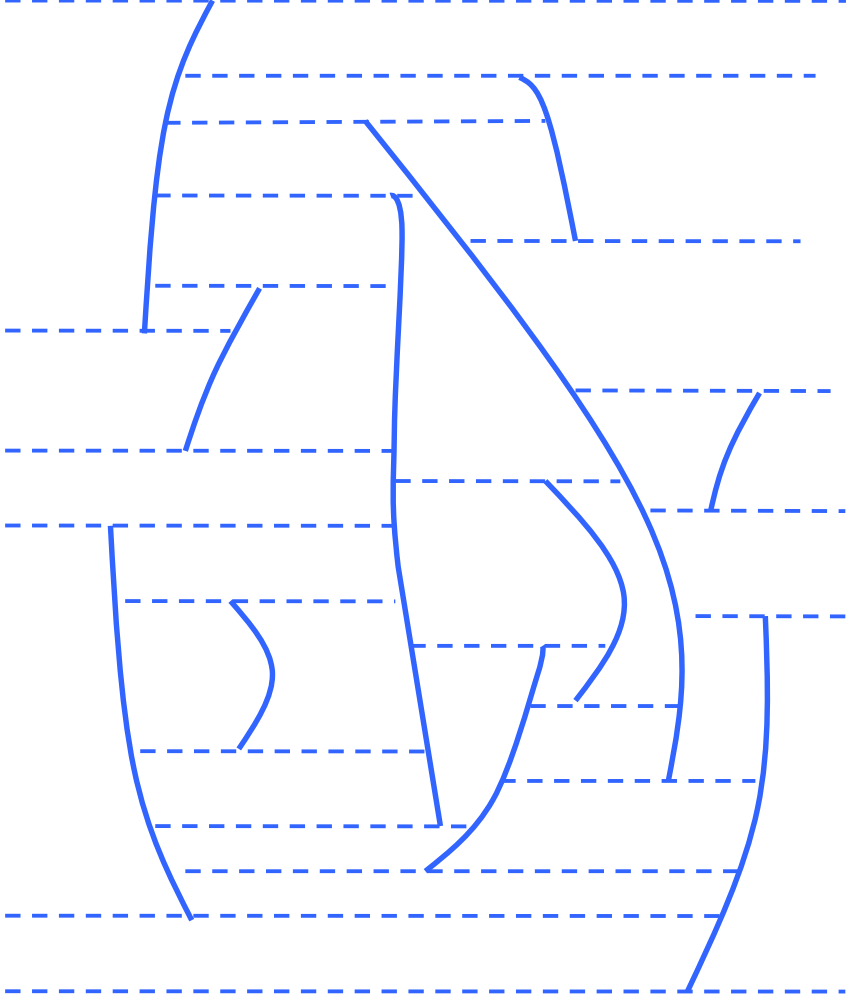
1 or 2 comparisons !!

## trapezoidation G

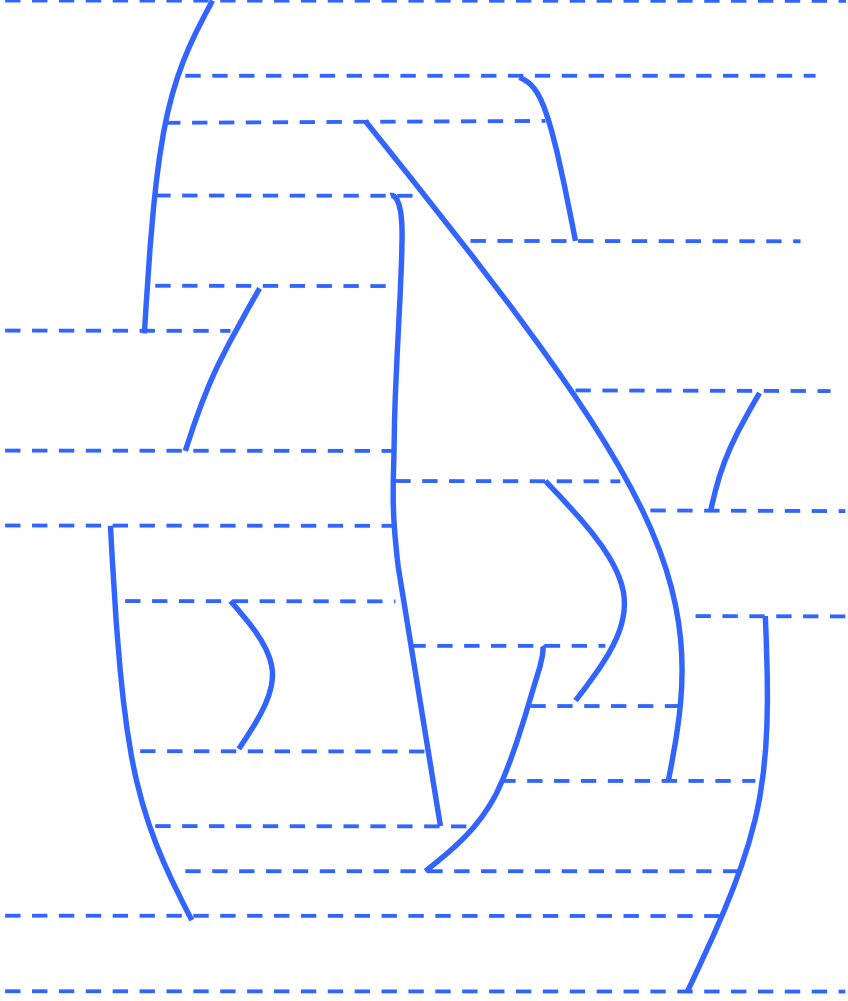




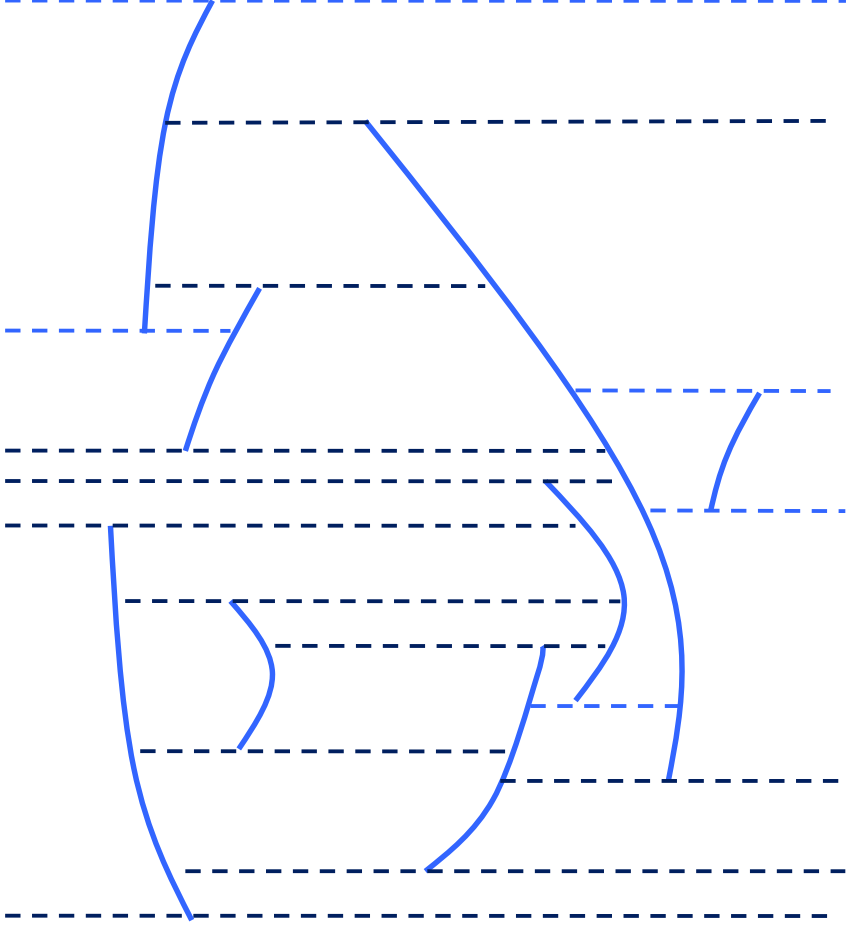
trapezoidation  $G$   
to obtain smaller  $G'$



trapezoidation  $G$   
to obtain smaller  $G'$   
remove **set of independent**  
**segments**

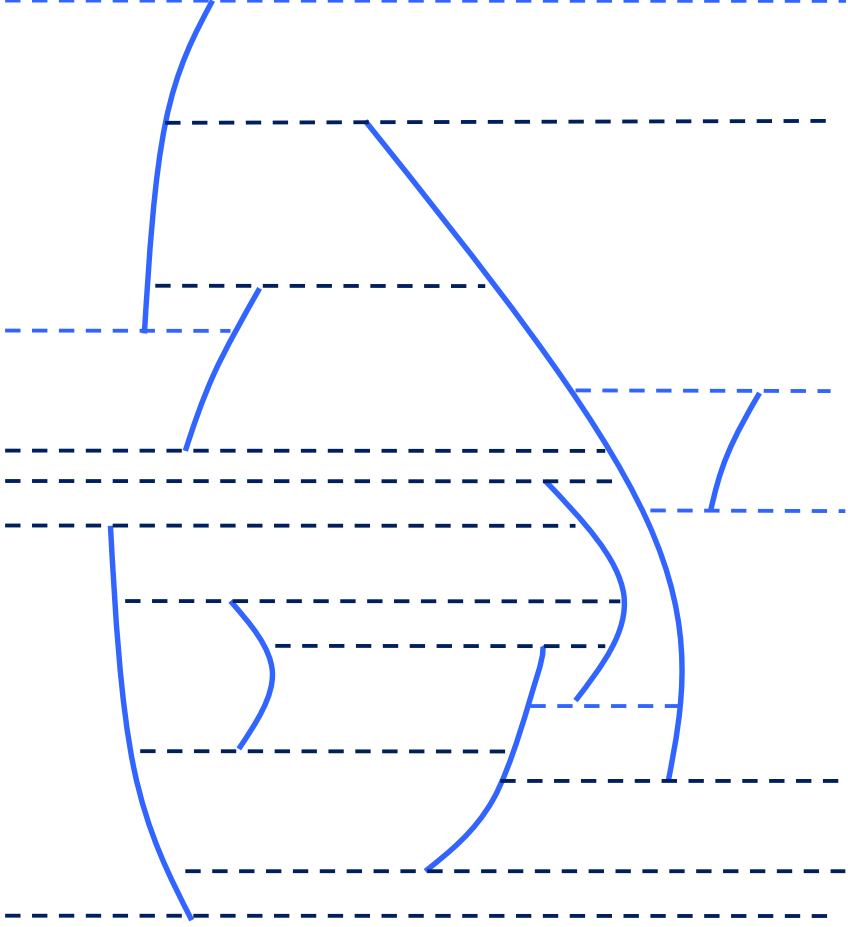


trapezoidation  $G$   
to obtain smaller  $G'$   
remove **set of independent**  
**segments**  
and “retrapezoidalize” holes



trapezoidation  $G$   
to obtain smaller  $G'$   
remove **set of independent**  
**segments**  
and “retrapezoidalize” holes





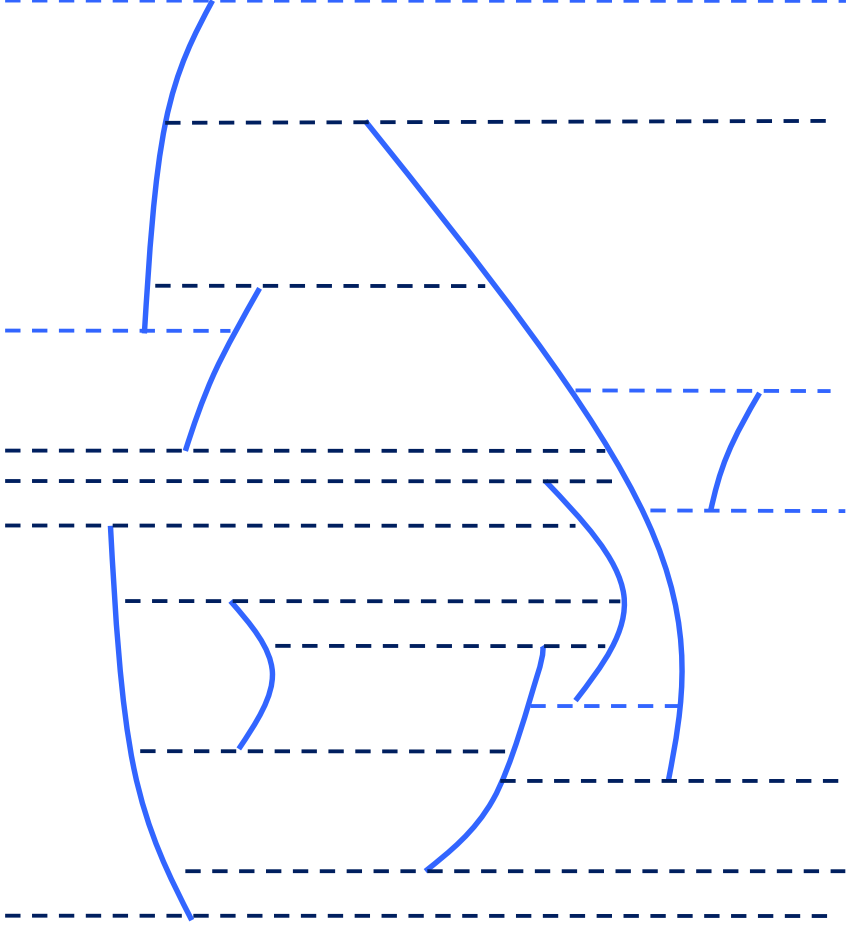
trapezoidation  $G$

to obtain smaller  $G'$

remove set of independent  
segments

and "retapezoidalize" holes

repeat recursively



trapezoidation  $G$   
 to obtain smaller  $G'$   
 remove **set of independent**  
**segments**  
 and "trapezoidalize" holes  
 repeat recursively

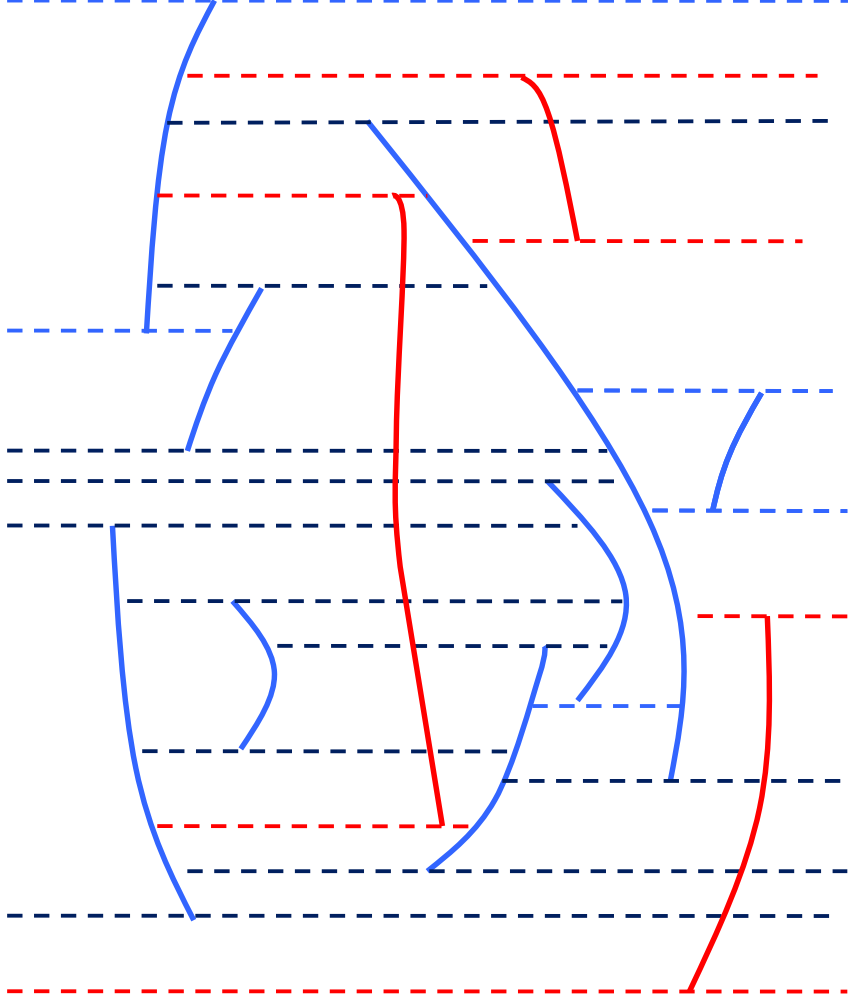
Query for point  $q$  :

locate  $q$  in  $G'$

if  $q$  in "black" trapezoid **then** determine correct trapezoid of  $G$

**else** trapezoid is correct answer already

**1 or 2 comparisons !!**



trapezoidation  $G$   
 to obtain smaller  $G'$   
 remove **set of independent**  
**segments**  
 and "trapezoidalize" holes  
 repeat recursively

Query for point  $q$  :

locate  $q$  in  $G'$

if  $q$  in "black" trapezoid **then** determine correct trapezoid of  $G$

**else** trapezoid is correct answer already

**1 or 2 comparisons !!**

**Lemma 1:** In every set of  $n \geq 4$   $x$ -monotone segments there exists an “independent” set of size at least  $n/4$ .

**Lemma 1:** In every set of  $n \geq 4$  x-monotone segments there exists an “independent” set of size at least  $n/4$ .

**Lemma 2:** In every set of  $m$  “exposed” vertical segments there exists an “independent” set of size at least  $m/2$ .

**Lemma 1:** In every set of  $n \geq 4$  x-monotone segments there exists an “independent” set of size at least  $n/4$ .

**Lemma 2:** In every set of  $m$  “exposed” vertical segments there exists an “independent” set of size at least  $m/2$ .

$\Rightarrow$  height of hierarchy of trapezoidations can be made  $O(\log n)$

**Lemma 1:** In every set of  $n \geq 4$  x-monotone segments there exists an “independent” set of size at least  $n/4$ .

**Lemma 2:** In every set of  $m$  “exposed” vertical segments there exists an “independent” set of size at least  $m/2$ .

$\Rightarrow$  height of hierarchy of trapezoidations can be made  $O(\log n)$

$\Rightarrow$  Query time  $O(\log n)$   $\leq 3.5 \log_2 n$   
Space  $O(n)$   
Preprocessing  $O(n)$   
 $O(n \log n)$

## Shortcomings of Kirkpatrick's original method:

- only works for straight edge subdivisions
- constants are large
- "complicated" (needs to find independent sets)



## Shortcomings of Kirkpatrick's original method:

- only works for straight edge subdivisions
- constants are large
- "complicated" (needs to find independent sets)

## Shortcomings of Kirkpatrick's original method:

- only works for straight edge subdivisions
- constants are large
- "complicated" (needs to find independent sets)

## Shortcomings of Kirkpatrick's original method:

- only works for straight edge subdivisions
- constants are large
- "complicated" (needs to find independent sets)

**Use randomization !!**

# 1993 Teaching Computational Geometry, I

## Topics

- Randomized algorithms as first class objects
- The importance of invariants for sweep algorithms
- Representation of planar subdivisions
- Planar point location
- Linear Programming
- 3d Convex hulls
- Perturbations
- Upper bound theorem for polytopes

- **Linear Programming**

- Linear Programming

Textbooks (e.g. "Dutch Book") typically:

only simple randomized incremental algorithm  
expected running time  $O(d! n)$

## • Linear Programming

Textbooks (e.g. "Dutch Book") typically:

only simple randomized incremental algorithm  
expected running time  $O(d! n)$

Desirable:

- Linear Programming

Textbooks (e.g. "Dutch Book") typically:

only simple randomized incremental algorithm  
expected running time  $O(d! n)$

Desirable:

- Decimation algorithms of Megiddo/Dyer



## • Linear Programming

Textbooks (e.g. "Dutch Book") typically:

only simple randomized incremental algorithm  
expected running time  $O(d! n)$

Desirable:

- Decimation algorithms of Megiddo/Dyer
- More intricate randomized incremental algorithm of Matousek-Sharir-Welzl + Clarkson

# Megiddo / Dyer deterministic Linear Programming

## Megiddo / Dyer deterministic Linear Programming

Idea: Describe algorithms with increasing level of input complexity (you can stop at any level)

## Megiddo / Dyer deterministic Linear Programming

Idea: Describe algorithms with increasing level of input complexity (you can stop at any level)

$d=2$ :

## Megiddo / Dyer deterministic Linear Programming

Idea: Describe algorithms with increasing level of input complexity (you can stop at any level)

**d=2:** only upper halfplanes (decimation)

## Megiddo / Dyer deterministic Linear Programming

Idea: Describe algorithms with increasing level of input complexity (you can stop at any level)

**d=2:** only upper halfplanes (decimation)

upper and lower halfplanes ("solution" for infeasible case)

## Megiddo / Dyer deterministic Linear Programming

Idea: Describe algorithms with increasing level of input complexity (you can stop at any level)

**d=2:** only upper halfplanes (decimation)

upper and lower halfplanes ("solution" for infeasible case)

upper, lower, and vertical halfplanes

## Megiddo / Dyer deterministic Linear Programming

Idea: Describe algorithms with increasing level of input complexity (you can stop at any level)

**d=2:** only upper halfplanes (decimation)

upper and lower halfplanes ("solution" for infeasible case)

upper, lower, and vertical halfplanes

**d=3:**



## Megiddo / Dyer deterministic Linear Programming

Idea: Describe algorithms with increasing level of input complexity (you can stop at any level)

**d=2:** only upper halfplanes (decimation)

upper and lower halfplanes ("solution" for infeasible case)

upper, lower, and vertical halfplanes

**d=3:** only upper halfspaces (more complicated decimation)

## Megiddo / Dyer deterministic Linear Programming

Idea: Describe algorithms with increasing level of input complexity (you can stop at any level)

**d=2:** only upper halfplanes (decimation)

upper and lower halfplanes ("solution" for infeasible case)

upper, lower, and vertical halfplanes

**d=3:** only upper halfspaces (more complicated decimation)

upper and lower halfspaces ("solution" for infeasible case)

## Megiddo / Dyer deterministic Linear Programming

Idea: Describe algorithms with increasing level of input complexity (you can stop at any level)

**d=2:** only upper halfplanes (decimation)

upper and lower halfplanes ("solution" for infeasible case)

upper, lower, and vertical halfplanes

**d=3:** only upper halfspaces (more complicated decimation)

upper and lower halfspaces ("solution" for infeasible case)

upper, lower, and vertical halfspaces

## Megiddo / Dyer deterministic Linear Programming

Idea: Describe algorithms with increasing level of input complexity (you can stop at any level)

**d=2:** only upper halfplanes (decimation)

upper and lower halfplanes ("solution" for infeasible case)

upper, lower, and vertical halfplanes

**d=3:** only upper halfspaces (more complicated decimation)

upper and lower halfspaces ("solution" for infeasible case)

upper, lower, and vertical halfspaces

**d>3:** ??

# MSW randomized incremental Linear Programming

# MSW randomized incremental Linear Programming

simple algorithm, with slightly intricate analysis

## MSW randomized incremental Linear Programming

simple algorithm, with slightly intricate analysis

```
function  $MSW\_LP(G, B)$ :  
     $G$ : set of  $n$  constraints in  $\mathbb{R}^d$   
     $B \subseteq G$ : candidate basis  
    returns  $v_G$  and basis of  $G$   
  
    if  $G = B$  then return  $(v_B, B)$   
    else choose random  $h \in G - B$   
         $(v, B') := MSW\_LP(G - \{h\}, B)$   
        if  $h$  violates  $v$  then return  $MSW\_LP(G, \text{basis}(B', h))$   
        else return  $(v, B')$ 
```

## MSW randomized incremental Linear Programming

simple algorithm, with slightly intricate analysis

```
function MSW_LP( $G, B$ ):  
     $G$ : set of  $n$  constraints in  $\mathbb{R}^d$   
     $B \subseteq G$ : candidate basis  
    returns  $v_G$  and basis of  $G$   
  
    if  $G = B$  then return ( $v_B, B$ )  
    else choose random  $h \in G - B$   
        ( $v, B'$ ) := MSW_LP( $G - \{h\}, B$ )  
        if  $h$  violates  $v$  then return MSW_LP( $G$ , basis( $B', h$ ))  
        else return ( $v, B'$ )
```

expected running time of  $O((n-d)2^d)$  easy to prove



## MSW randomized incremental Linear Programming

simple algorithm, with slightly intricate analysis

```
function MSW_LP( $G, B$ ):  
     $G$ : set of  $n$  constraints in  $\mathbb{R}^d$   
     $B \subseteq G$ : candidate basis  
    returns  $v_G$  and basis of  $G$   
  
    if  $G = B$  then return ( $v_B, B$ )  
    else choose random  $h \in G - B$   
        ( $v, B'$ ) := MSW_LP( $G - \{h\}, B$ )  
        if  $h$  violates  $v$  then return MSW_LP( $G$ , basis( $B', h$ ))  
        else return ( $v, B'$ )
```

expected running time of  $O((n-d)2^d)$  easy to prove

proof of subexponential running time beyond introductory course

# MSW randomized incremental Linear Programming plus Clarkson's random sampling algorithms

## MSW randomized incremental Linear Programming plus Clarkson's random sampling algorithms

yields **best** currently known bound for Linear Programming:

$$O(d^2n + e^{(d \ln d)^{1/2}})$$

# MSW randomized incremental Linear Programming plus Clarkson's random sampling algorithms

yields **best** currently known bound for Linear Programming:

$$O(d^2n + e^{(d \ln d)^{1/2}})$$

teachable in 1 to 1.5 lectures

# MSW randomized incremental Linear Programming plus Clarkson's random sampling algorithms

yields **best** currently known bound for **Linear Programming**:

$$O(d^2n + e^{(d \ln d)^{1/2}})$$

teachable in 1 to 1.5 lectures  
following about 5 pages from

## Linear Programming – Randomization and Abstract Frameworks

by B. Gärtner and E. Welzl

Proc. 13th STACS, Springer LNCS 1046 (1996) 669-687.

[http://www.inf.ethz.ch/personal/emo/PubFiles/LinProgRandAbstr\\_LNCS1046\\_96.pdf](http://www.inf.ethz.ch/personal/emo/PubFiles/LinProgRandAbstr_LNCS1046_96.pdf)

# 1993 Teaching Computational Geometry, I

## Topics

- Randomized algorithms as first class objects
- The importance of invariants for sweep algorithms
- Representation of planar subdivisions
- Planar point location
- Linear Programming
- 3d Convex hulls
- Perturbations
- Upper bound theorem for polytopes

- 3d Convex Hulls

- 3d Convex Hulls

Very basic computational geometry problem



## • 3d Convex Hulls

Very basic computational geometry problem

Many other problems can be reduced to it

(Voronoi diagrams, Delaunay triangulations,  
intersection of halfspaces, union/intersection of disks, ...)

## • 3d Convex Hulls

Very basic computational geometry problem

Many other problems can be reduced to it

(Voronoi diagrams, Delaunay triangulations,  
intersection of halfspaces, union/intersection of disks, ...)

No "easy" optimal algorithms are known (?)

## • 3d Convex Hulls

Very basic computational geometry problem

Many other problems can be reduced to it

(Voronoi diagrams, Delaunay triangulations,  
intersection of halfspaces, union/intersection of disks, ...)

No "easy" optimal algorithms are known (?)

Possibly the most complicated topic in an  
introductory computational geometry course

## Difficulties:

## Difficulties:

- **geometric:**
- **data structures:**
- **analysis:**

## Difficulties:

- **geometric:** 3-dimensional;
- **data structures:**
- **analysis:**

## Difficulties:

- **geometric:** 3-dimensional;  
sleeve structure in divide & conquer algorithm;
- **data structures:**
- **analysis:**

## Difficulties:

- **geometric:** 3-dimensional;  
sleeve structure in divide & conquer algorithm;
- **data structures:** triangulation that needs to be navigatable  
and to be updated;
- **analysis:**



## Difficulties:

- **geometric:** 3-dimensional;  
sleeve structure in divide & conquer algorithm;
- **data structures:** triangulation that needs to be navigatable  
and to be updated;  
conflict structure for RIC;
- **analysis:**

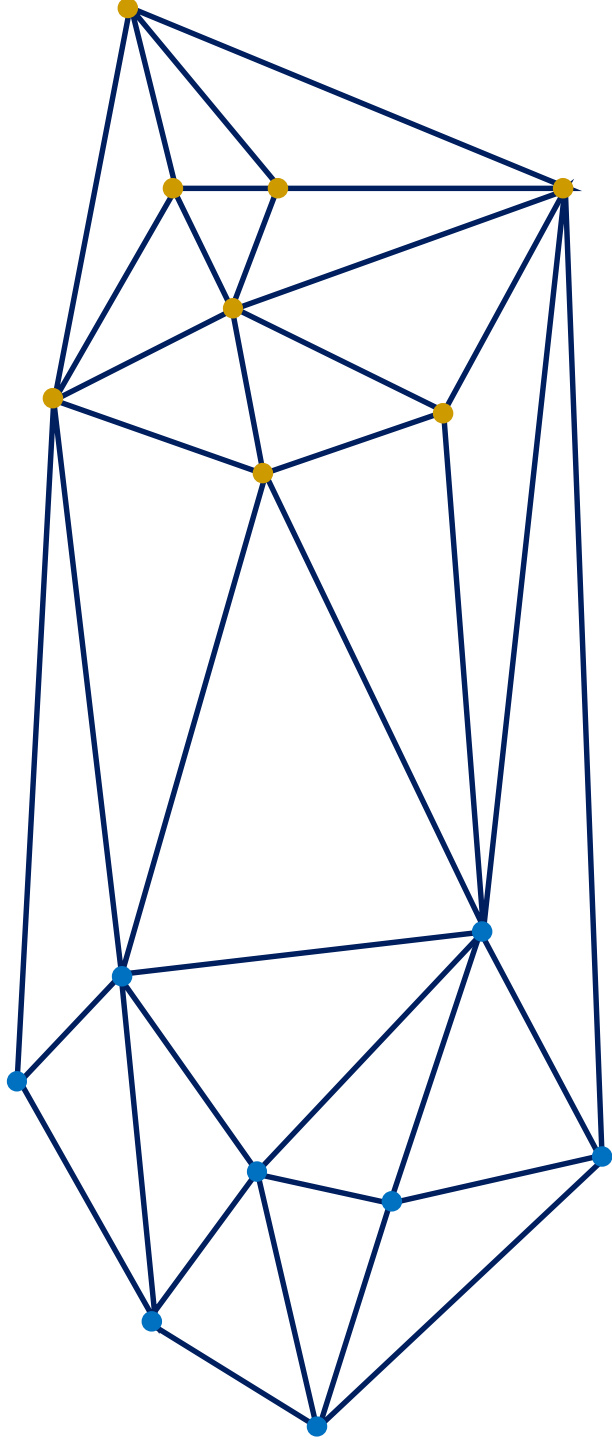
## Difficulties:

- **geometric:** 3-dimensional;  
sleeve structure in divide & conquer algorithm;
- **data structures:** triangulation that needs to be navigatable  
and to be updated;  
conflict structure for RIC;
- **analysis:** for RIC quite involved  
(e.g. in "Dutch Book" not self-contained);

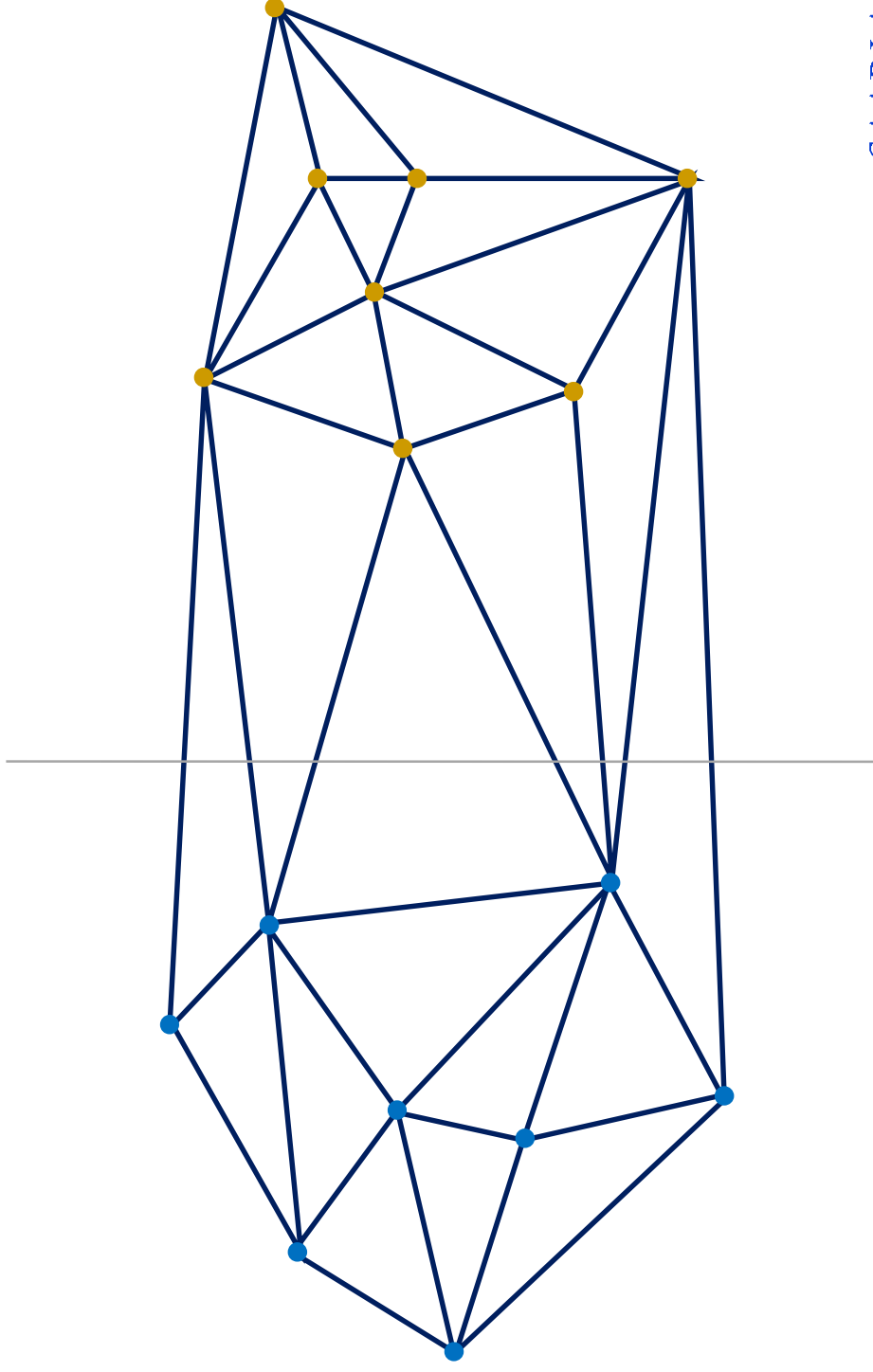
## Difficulties:

- **geometric:** 3-dimensional;  
sleeve structure in divide & conquer algorithm;
- **data structures:** triangulation that needs to be navigatable  
and to be updated;  
conflict structure for RIC;
- **analysis:** for RIC quite involved  
(e.g. in "Dutch Book" not self-contained);  
for D&C geometrically challenging

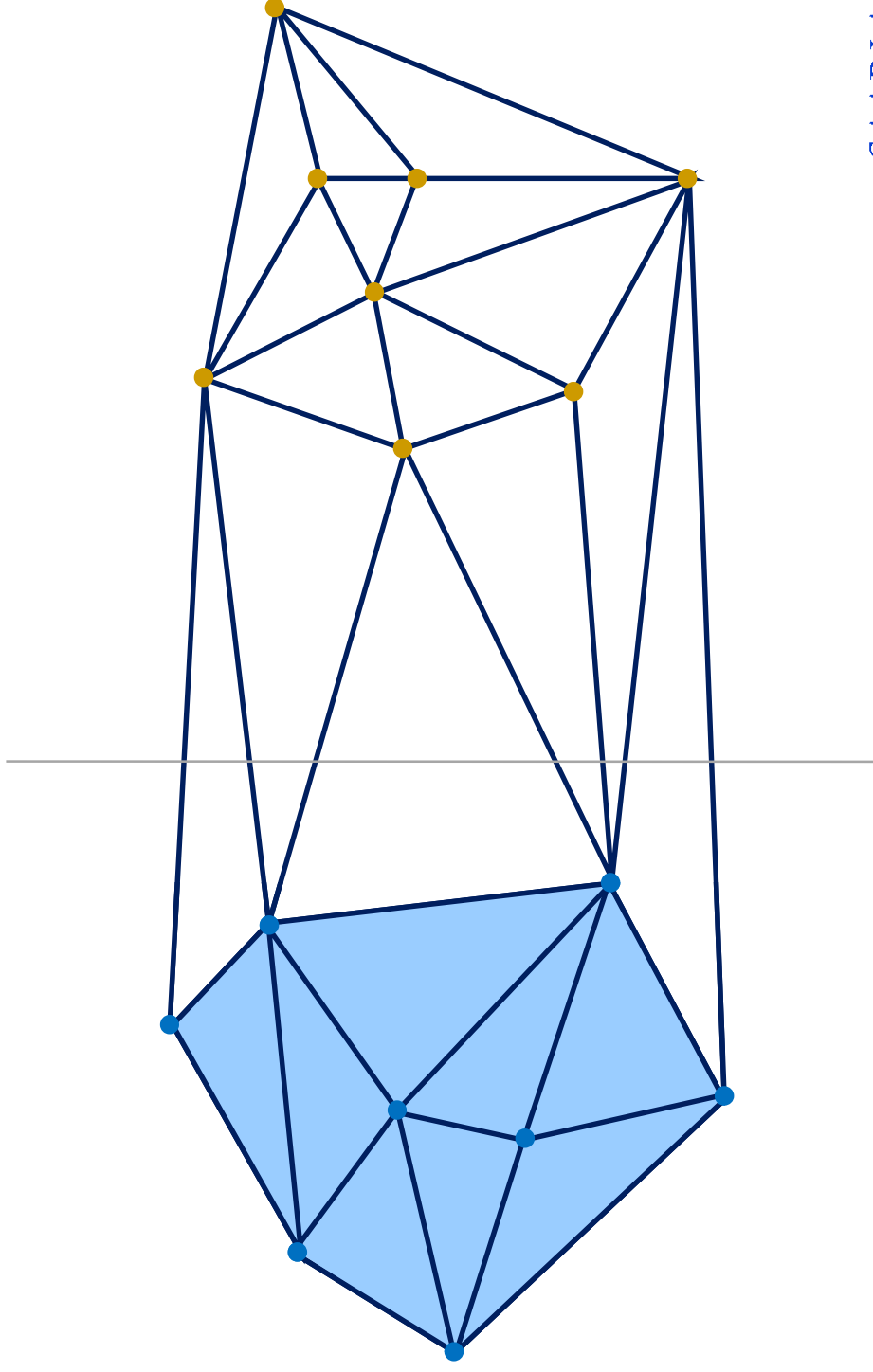
**Difficulty:** The possibly complicated sleeve boundary  
in the Preparata-Hong D & C algorithm



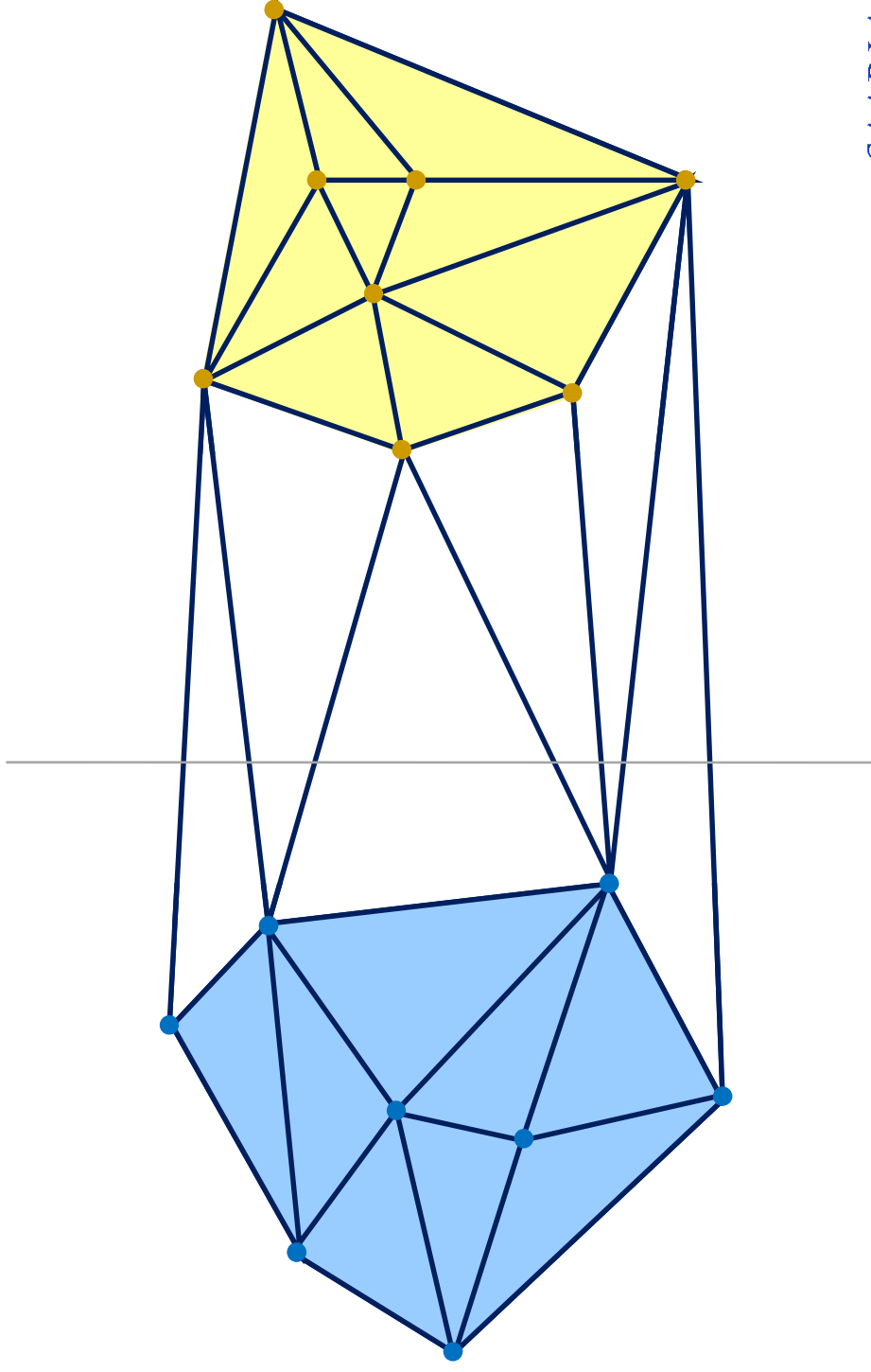
**Difficulty:** The possibly complicated sleeve boundary  
in the Preparata-Hong D & C algorithm



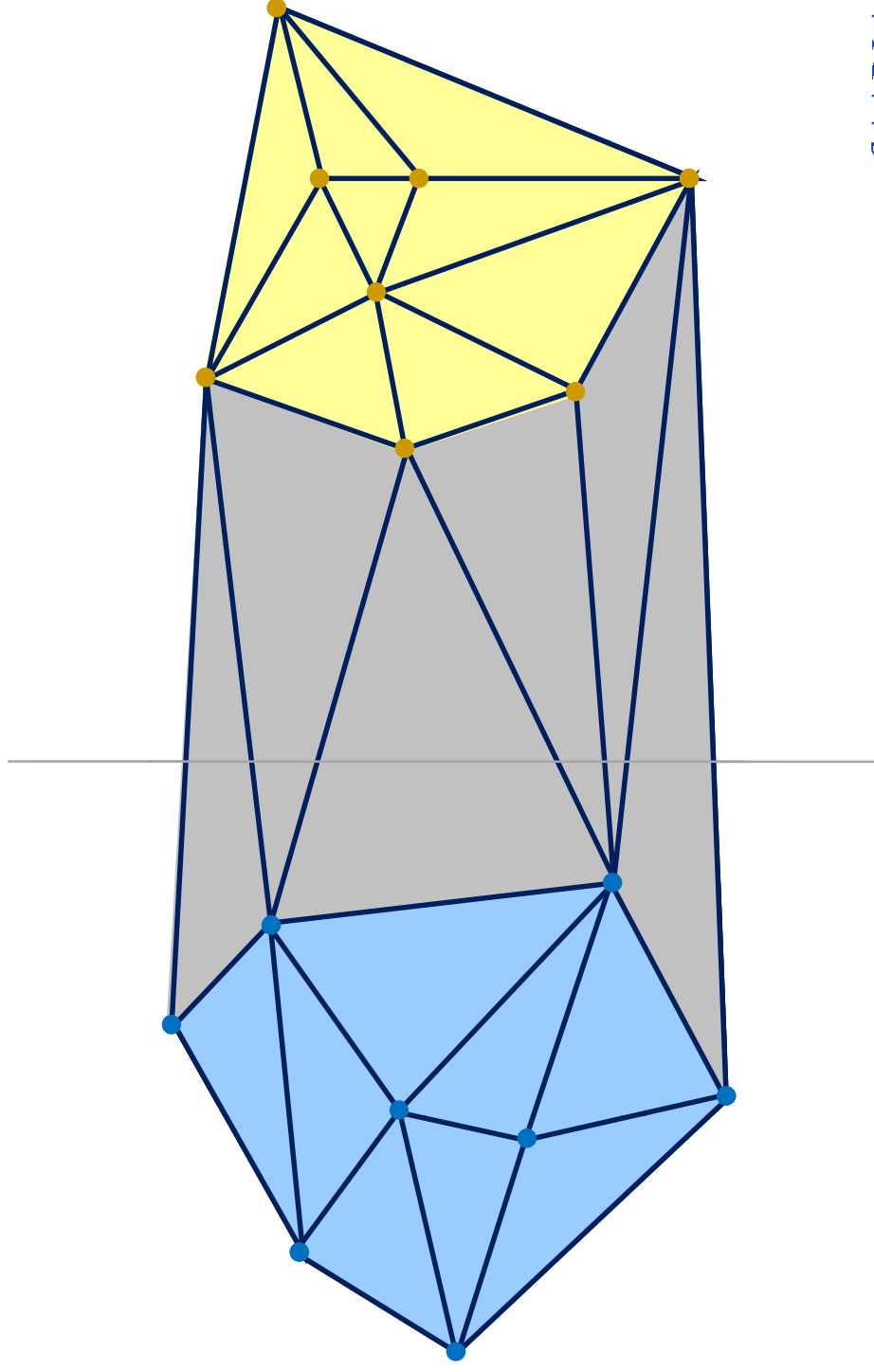
**Difficulty:** The possibly complicated sleeve boundary  
in the Preparata-Hong D & C algorithm



**Difficulty:** The possibly complicated sleeve boundary  
in the Preparata-Hong D & C algorithm

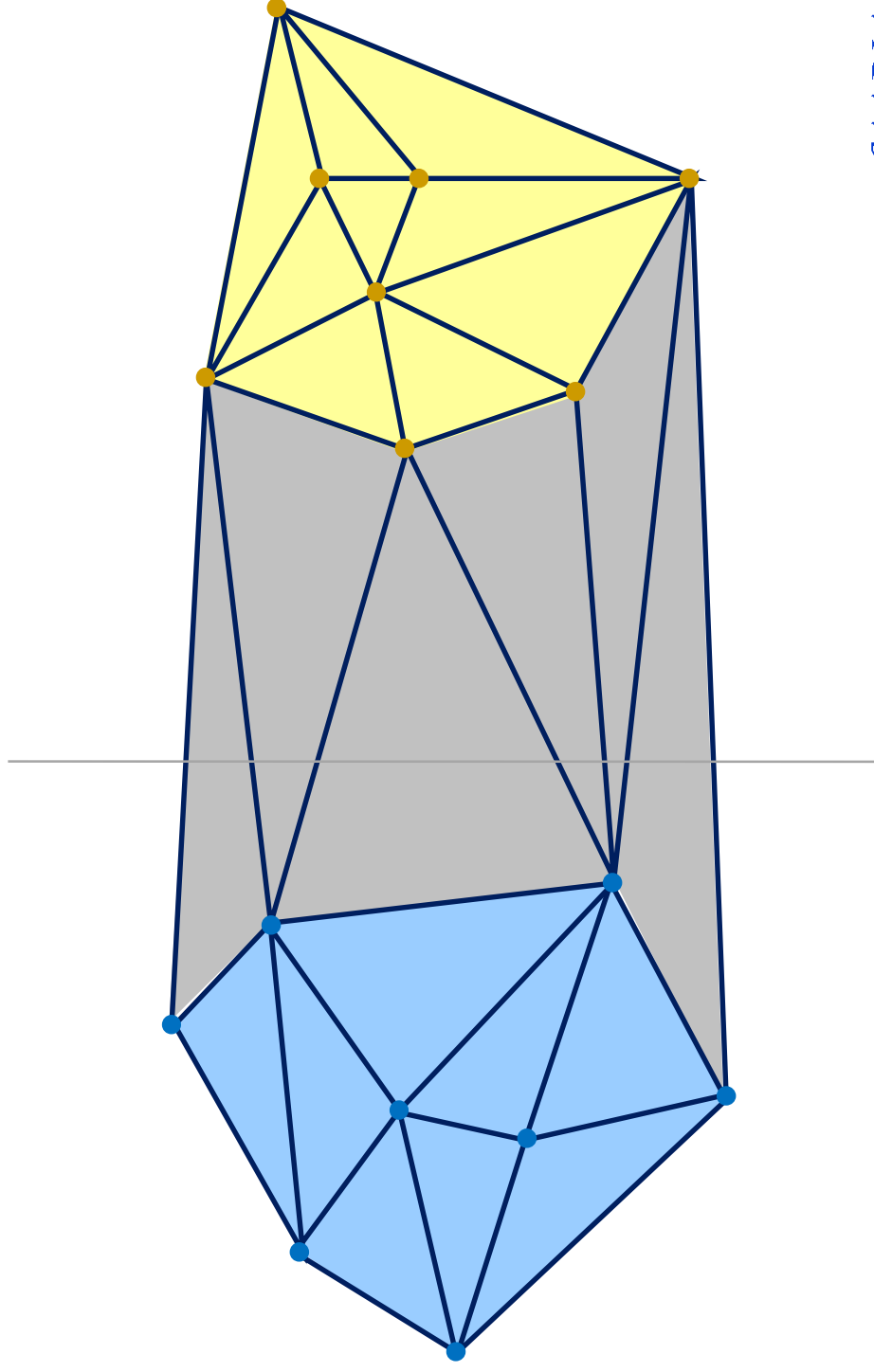


**Difficulty:** The possibly complicated sleeve boundary  
in the Preparata-Hong D & C algorithm

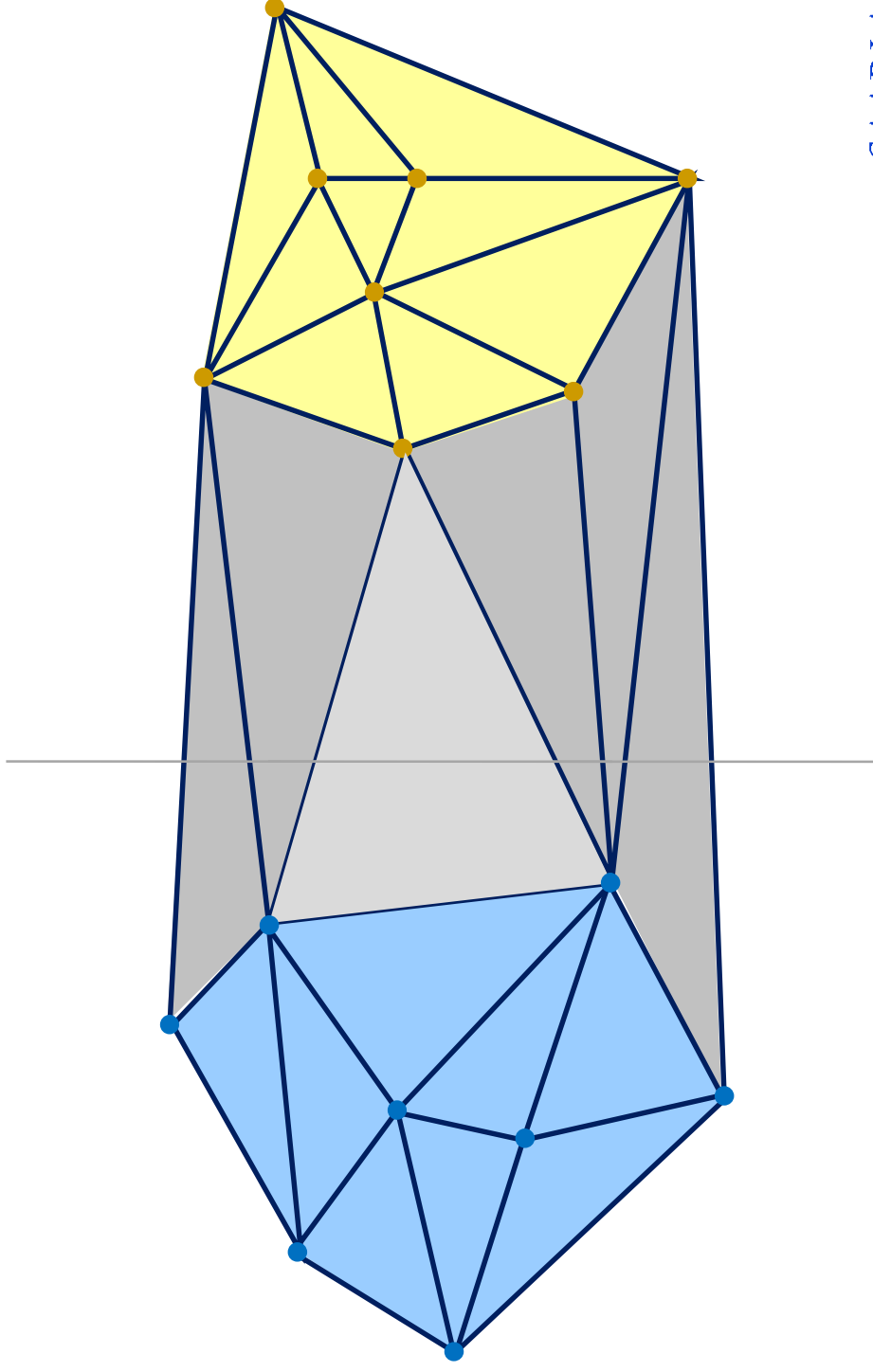




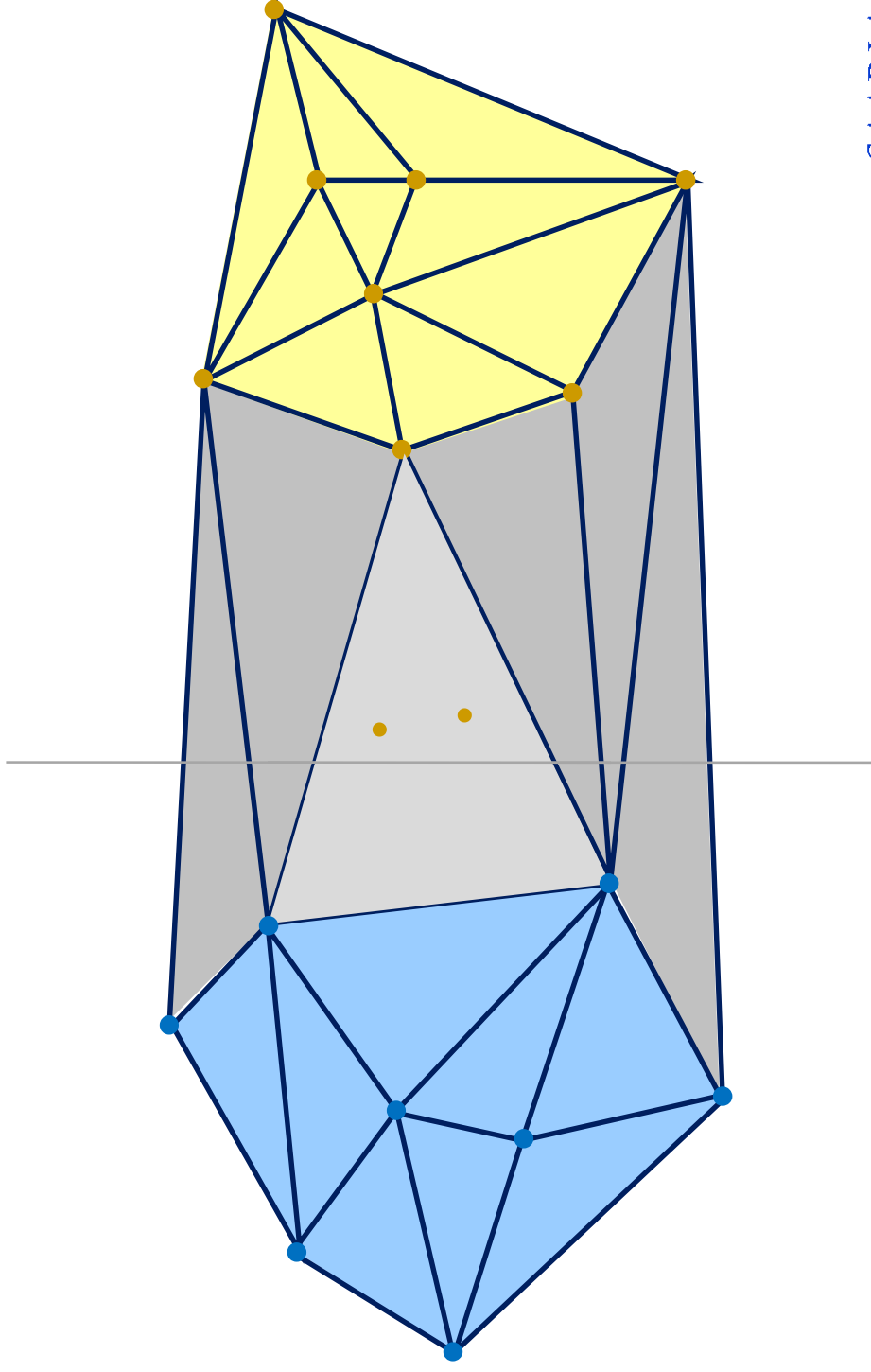
**Difficulty:** The possibly complicated sleeve boundary  
in the Preparata-Hong D & C algorithm



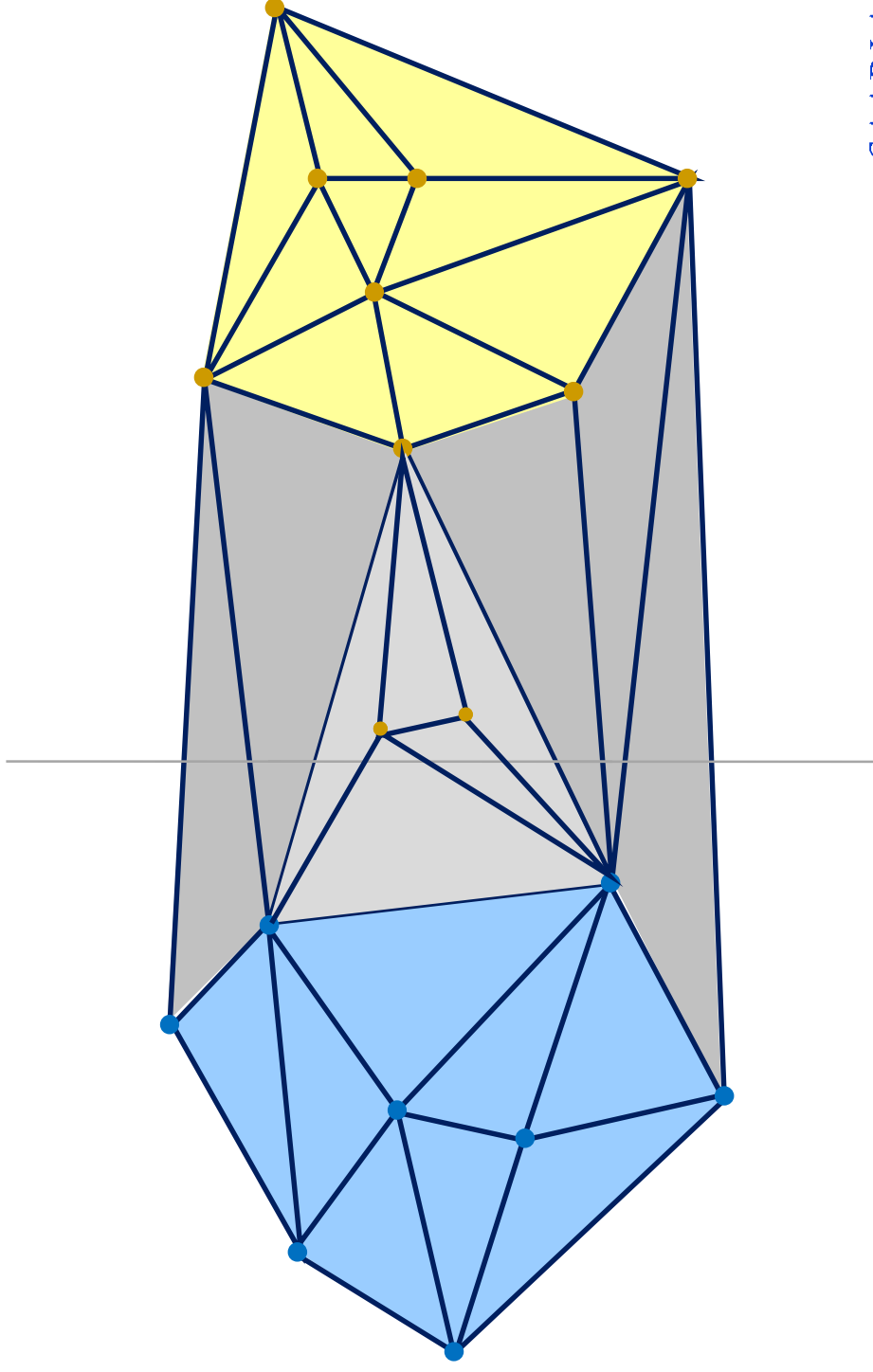
**Difficulty:** The possibly complicated sleeve boundary  
in the Preparata-Hong D & C algorithm



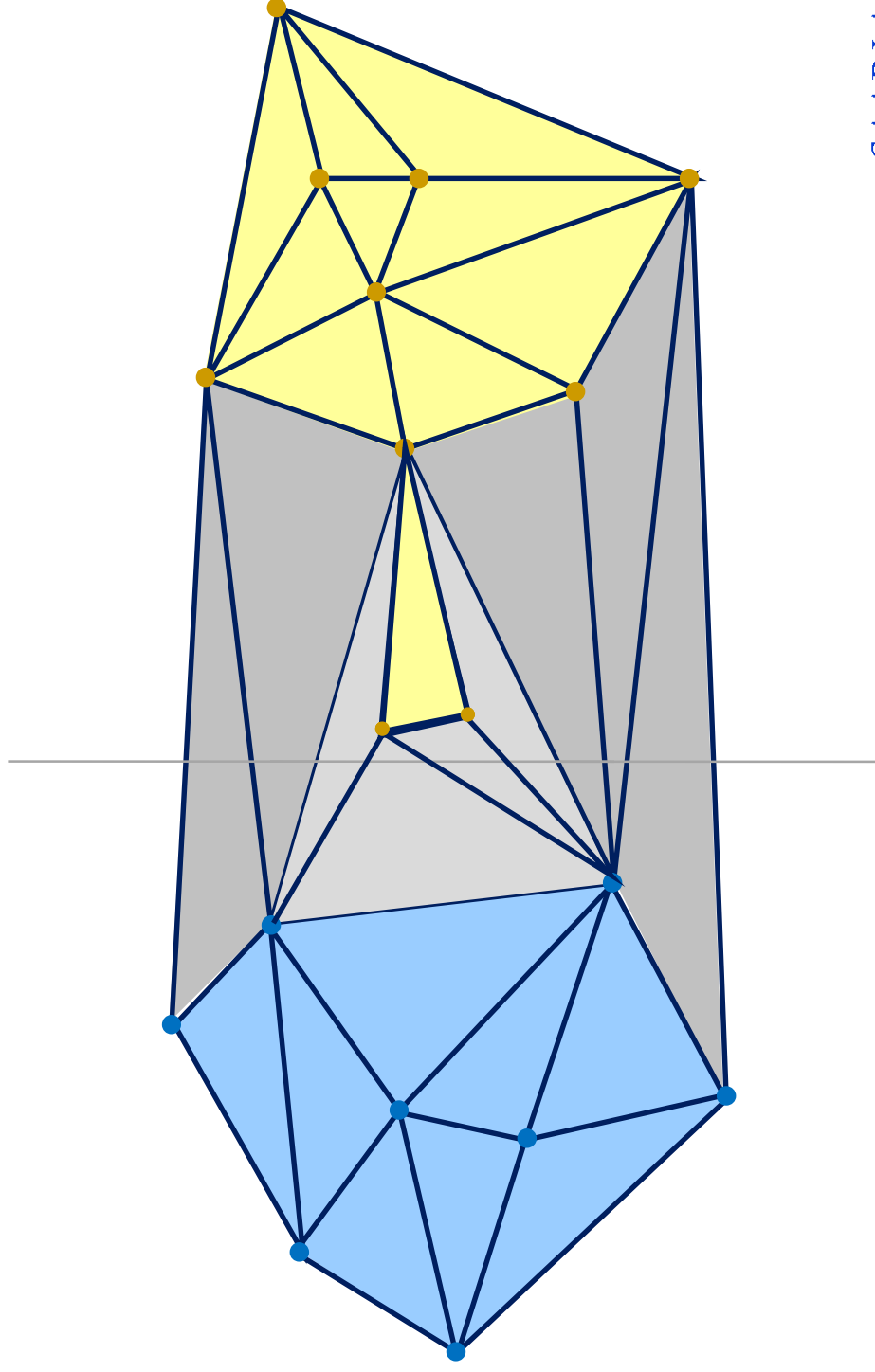
**Difficulty:** The possibly complicated sleeve boundary  
in the Preparata-Hong D & C algorithm



**Difficulty:** The possibly complicated sleeve boundary  
in the Preparata-Hong D & C algorithm



**Difficulty:** The possibly complicated sleeve boundary  
in the Preparata-Hong D & C algorithm



## Difficulties:

- **geometric:** 3-dimensional;  
sleeve structure in divide & conquer algorithm;
- **data structures:** triangulation that needs to be navigatable  
and to be updated;  
conflict structure for RIC;
- **analysis:** for RIC quite involved  
(e.g. in "Dutch Book" not self-contained);  
for D&C geometrically challenging

**Why the merge-step of the Preparata-Hong algorithm can be made to work in linear time.**

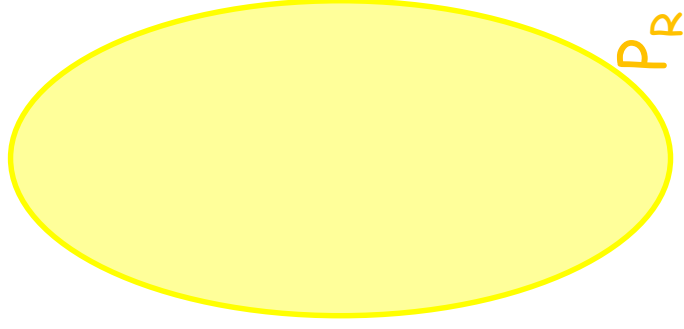
**Why the merge-step of the Preparata-Hong algorithm can be made to work in linear time.**

Merge phase finds sleeve between  $P_L$  and  $P_R$  via successive giftwrapping steps.



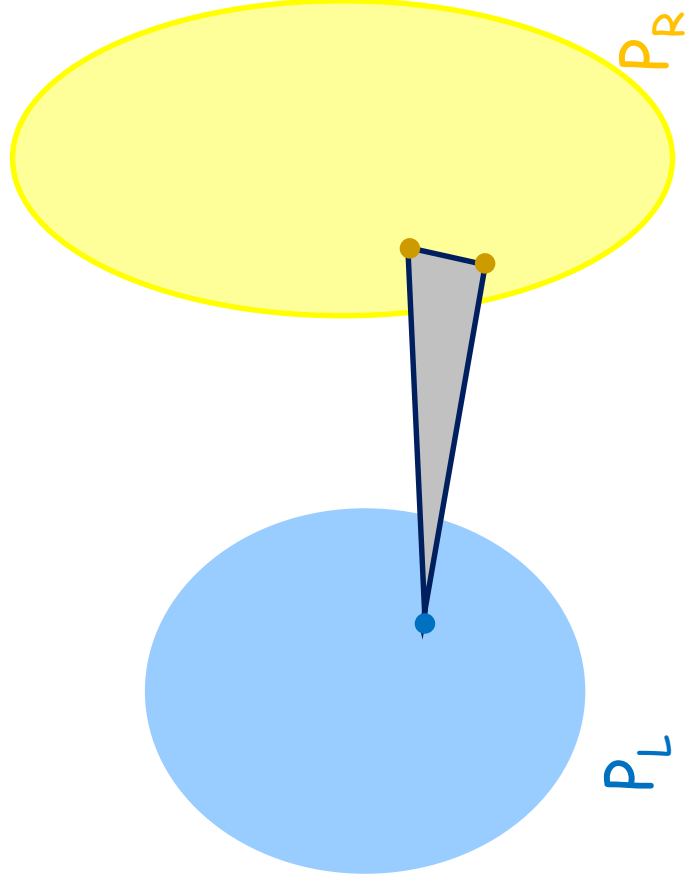
**Why the merge-step of the Preparata-Hong algorithm can be made to work in linear time.**

Merge phase finds sleeve between  $P_L$  and  $P_R$  via successive giftwrapping steps.



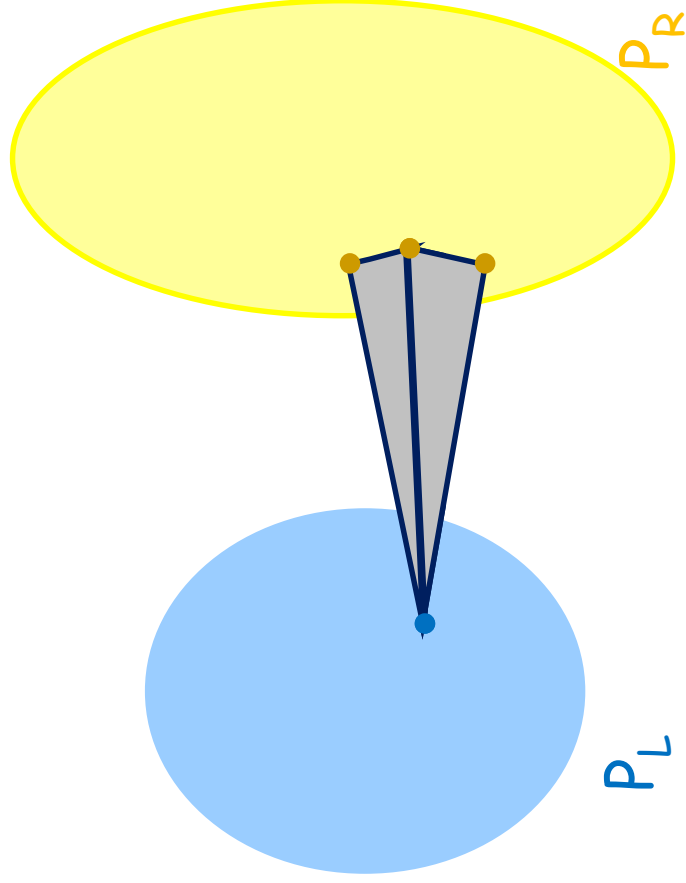
**Why the merge-step of the Preparata-Hong algorithm can be made to work in linear time.**

Merge phase finds sleeve between  $P_L$  and  $P_R$  via successive giftwrapping steps.



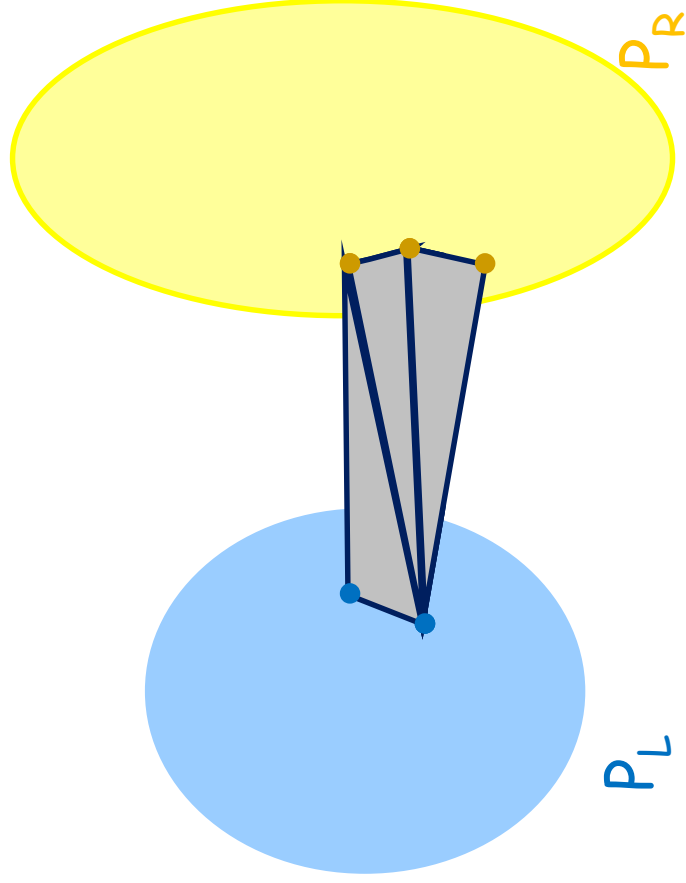
**Why the merge-step of the Preparata-Hong algorithm can be made to work in linear time.**

Merge phase finds sleeve between  $P_L$  and  $P_R$  via successive giftwrapping steps.



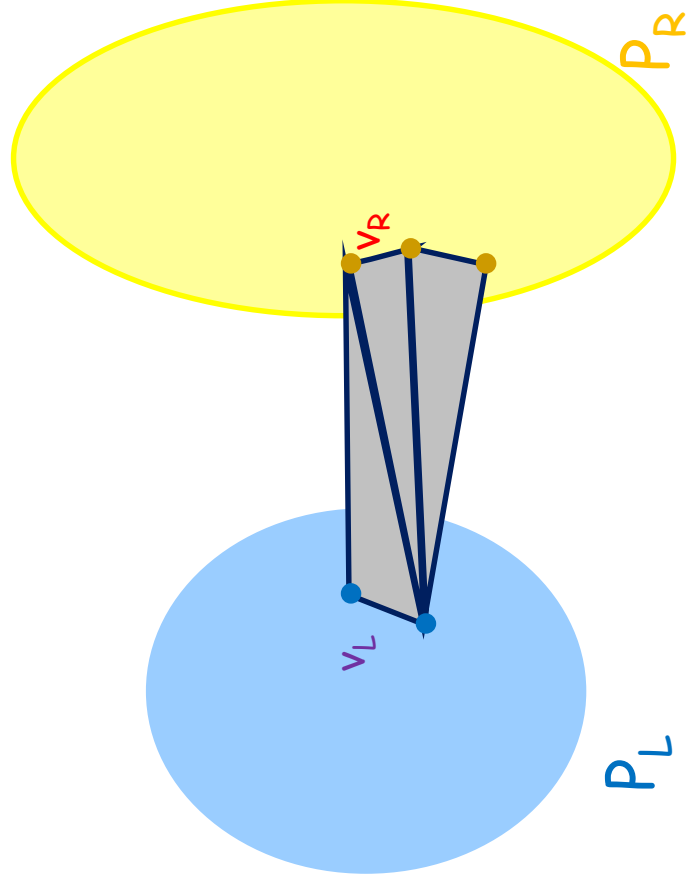
**Why the merge-step of the Preparata-Hong algorithm can be made to work in linear time.**

Merge phase finds sleeve between  $P_L$  and  $P_R$  via successive giftwrapping steps.



**Why the merge-step of the Preparata-Hong algorithm can be made to work in linear time.**

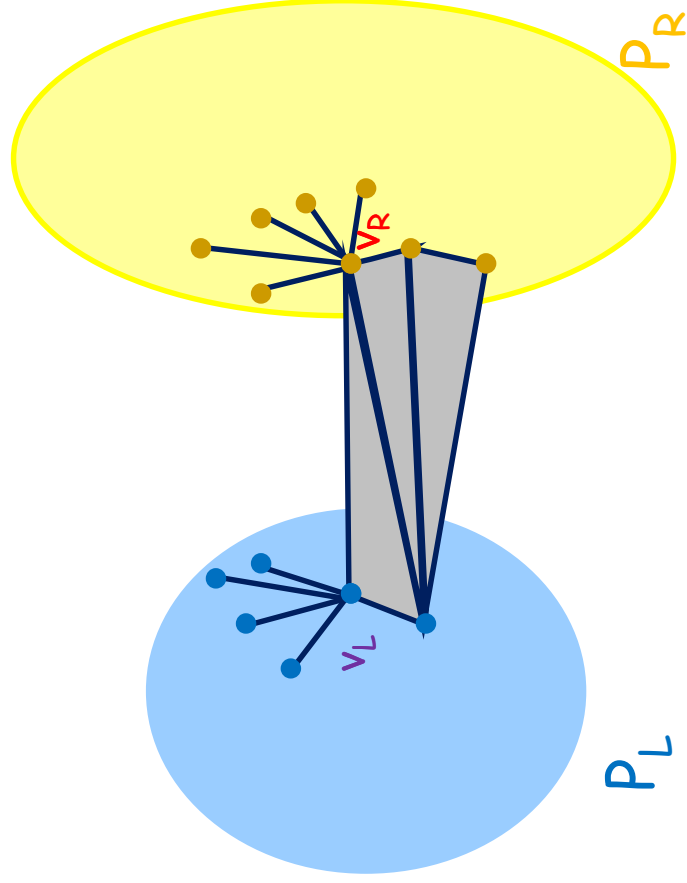
Merge phase finds sleeve between  $P_L$  and  $P_R$  via successive giftwrapping steps.



giftwrapping over  $v_L$   $v_R$  :  
the only candidates for  $v$  :  
neighbors of  $v_L$  in  $P_L$  or  
neighbors of  $v_R$  in  $P_R$

**Why the merge-step of the Preparata-Hong algorithm can be made to work in linear time.**

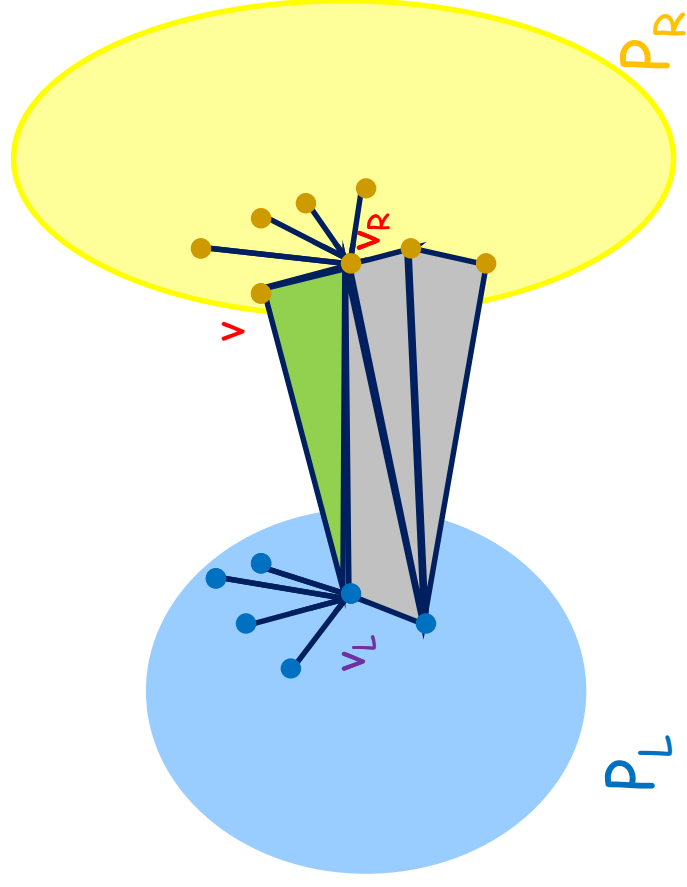
Merge phase finds sleeve between  $P_L$  and  $P_R$  via successive giftwrapping steps.



giftwrapping over  $v_L$   $v_R$  :  
the only candidates for  $v$  :  
neighbors of  $v_L$  in  $P_L$  or  
neighbors of  $v_R$  in  $P_R$

**Why the merge-step of the Preparata-Hong algorithm can be made to work in linear time.**

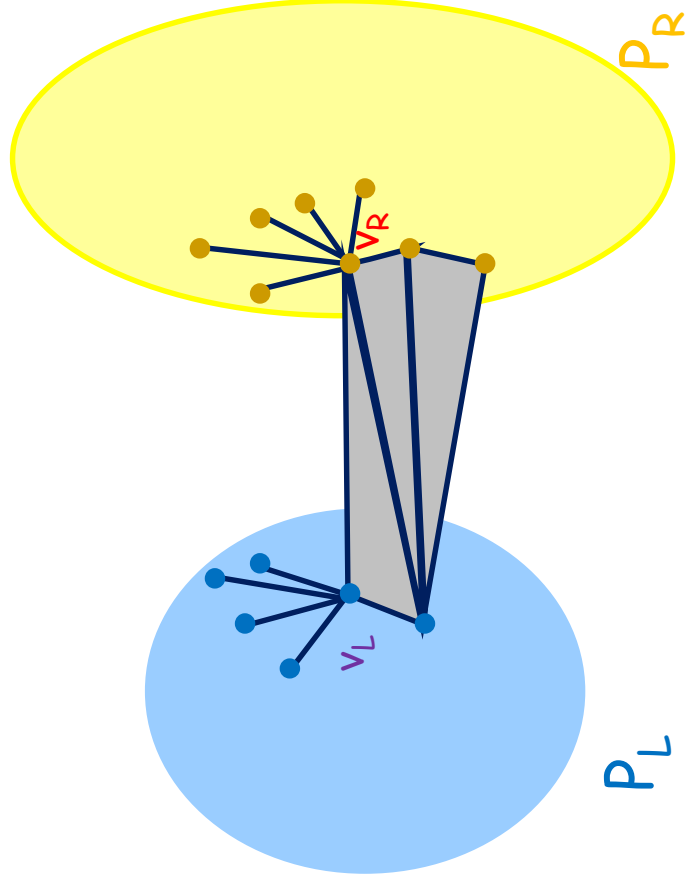
Merge phase finds sleeve between  $P_L$  and  $P_R$  via successive giftwrapping steps.



giftwrapping over  $v_L$   $v_R$  :  
the only candidates for  $v$  :  
neighbors of  $v_L$  in  $P_L$  or  
neighbors of  $v_R$  in  $P_R$

**Why the merge-step of the Preparata-Hong algorithm can be made to work in linear time.**

Merge phase finds sleeve between  $P_L$  and  $P_R$  via successive giftwrapping steps.

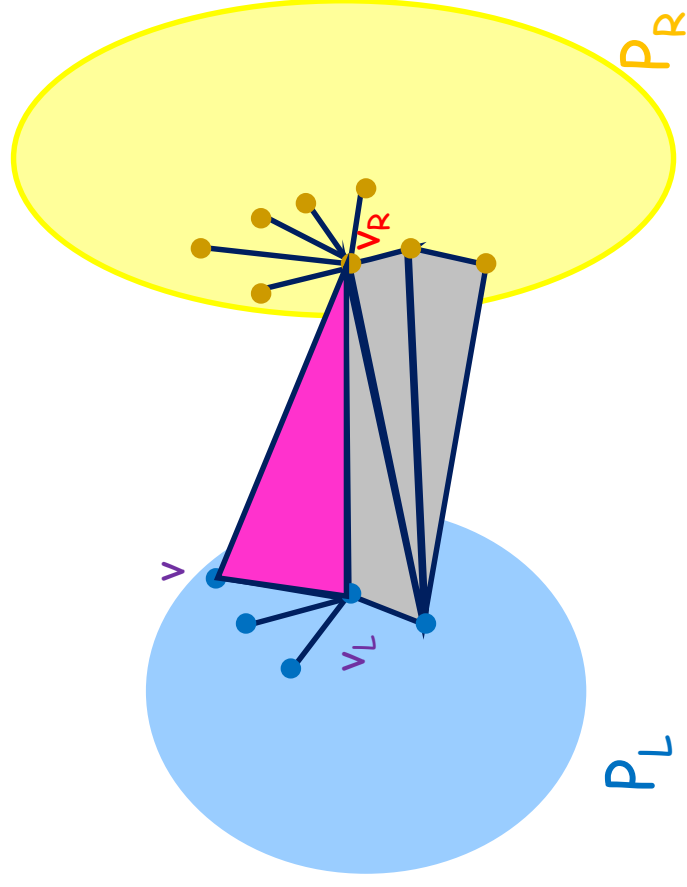


giftwrapping over  $v_L$   $v_R$  :  
the only candidates for  $v$  :  
neighbors of  $v_L$  in  $P_L$  or  
neighbors of  $v_R$  in  $P_R$



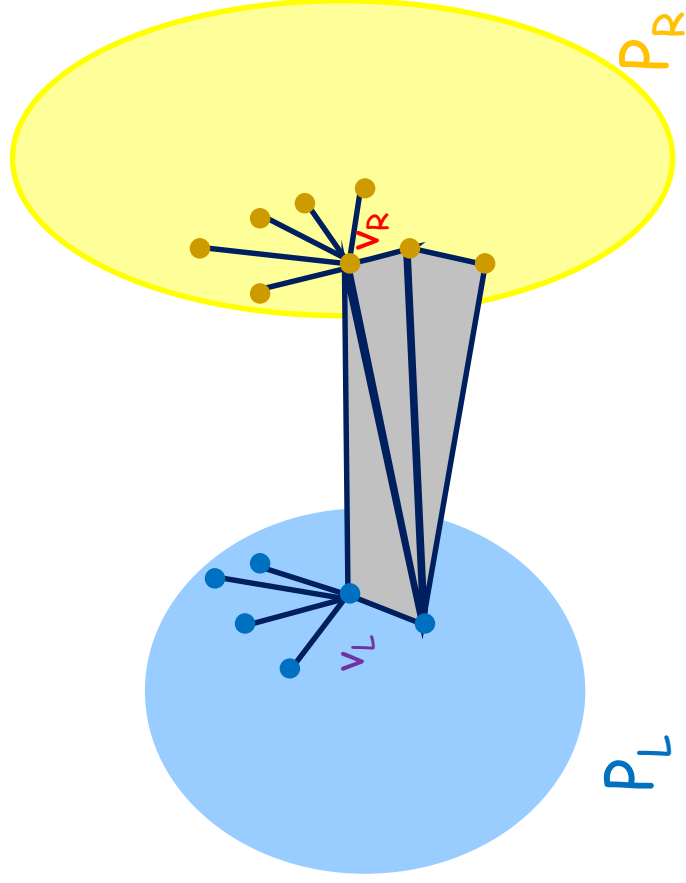
**Why the merge-step of the Preparata-Hong algorithm can be made to work in linear time.**

Merge phase finds sleeve between  $P_L$  and  $P_R$  via successive giftwrapping steps.



giftwrapping over  $v_L$   $v_R$  :  
the only candidates for  $v$  :  
neighbors of  $v_L$  in  $P_L$  or  
neighbors of  $v_R$  in  $P_R$

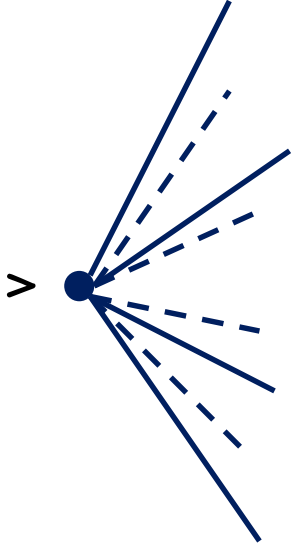
**Constant time giftwrapping step if every vertex in  $P_L$  and  $P_R$  has constant degree !!**



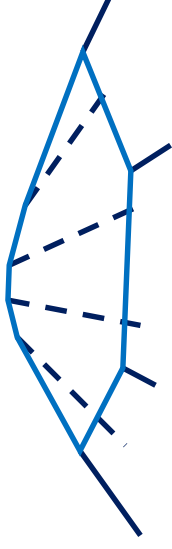
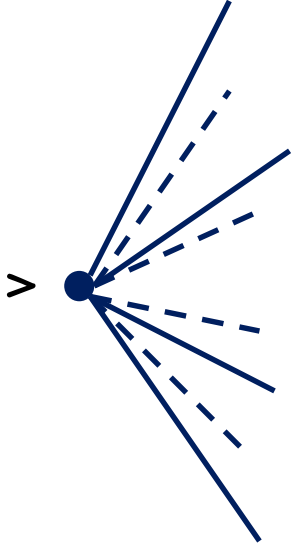
giftwrapping over  $v_L$   $v_R$  :  
the only candidates for  $v$  :  
neighbors of  $v_L$  in  $P_L$  or  
neighbors of  $v_R$  in  $P_R$

**Idea:** Make every vertex have constant degree by  
(infinitesimal) truncation

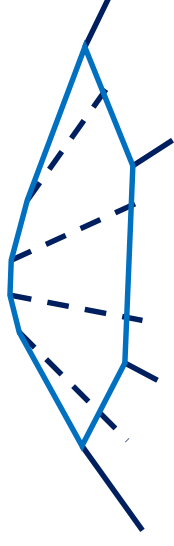
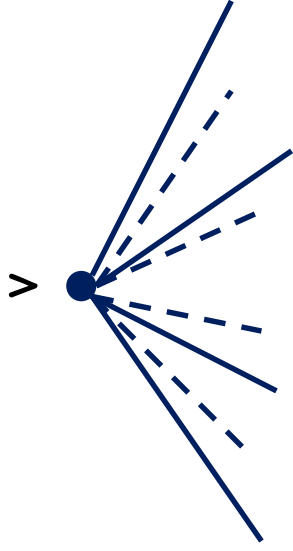
**Idea:** Make every vertex have constant degree by  
(infinitesimal) truncation



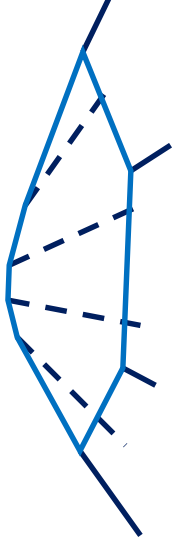
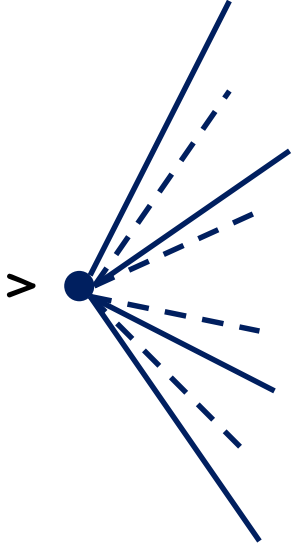
**Idea:** Make every vertex have constant degree by  
(infinitesimal) truncation



**Idea:** Make every vertex have constant degree by (infinitesimal) truncation (arbitrarily close to  $v$ )

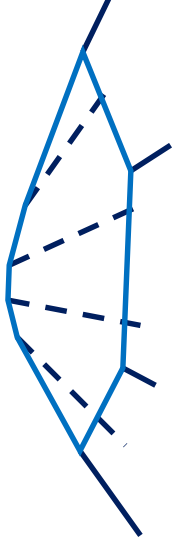
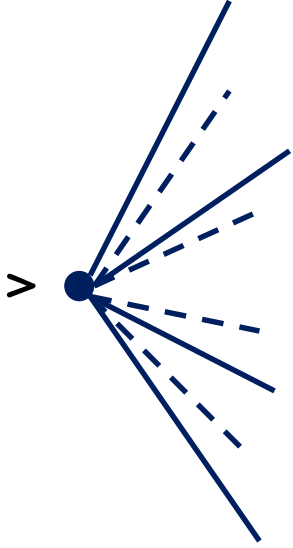


**Idea:** Make every vertex have constant degree by (infinitesimal) truncation (arbitrarily close to  $v$ )



# resulting vertices =  $2 \cdot \# \text{ edges of } P_L \text{ and } P_R = O(n)$

**Idea:** Make every vertex have constant degree by (infinitesimal) truncation (arbitrarily close to  $v$ )

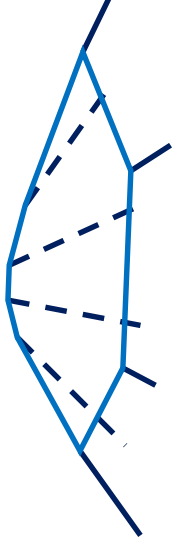
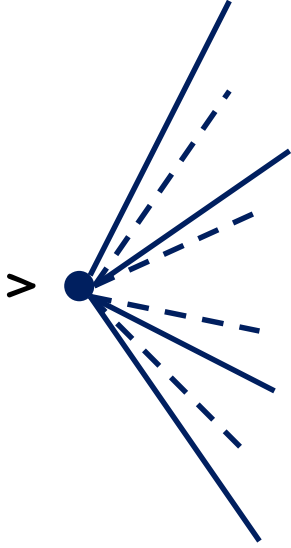


# resulting vertices =  $2 \cdot \# \text{ edges of } P_L \text{ and } P_R = O(n)$

↑  
"perturbed"  $P$  has size  $O(n)$   
"perturbed" sleeve has size  $O(n)$   
and takes  $O(n)$  time to construct



**Idea:** Make every vertex have constant degree by (infinitesimal) truncation (arbitrarily close to  $v$ )



# resulting vertices =  $2 \cdot \# \text{ edges of } P_L \text{ and } P_R = O(n)$

$\Rightarrow$  "perturbed"  $P$  has size  $O(n)$   
"perturbed" sleeve has size  $O(n)$   
and takes  $O(n)$  time to construct

Afterwards "contract" zero-length edges to recover true  $P$ .

## Difficulties:

- **geometric:** 3-dimensional;  
sleeve structure in divide & conquer algorithm;
- **data structures:** triangulation that needs to be navigatable  
and to be updated;  
conflict structure for RIC;
- **analysis:** for RIC quite involved  
(e.g. in "Dutch Book" not self-contained);  
for D&C geometrically challenging

## Difficulties:

- **geometric:** 3 dimensional;  
sleeve structure in divide & conquer algorithm;
- **data structures:** triangulation that needs to be navigatable  
and to be updated;  
conflict structure for RIC;
- **analysis:** for RIC quite involved  
(e.g. in "Dutch Book" not self-contained);  
for D&C geometrically challenging

## Difficulties:

- **geometric:** 3 dimensional;  
sleeve structure in divide & conquer algorithm;
- **data structures:** triangulation that needs to be navigatable  
and to be updated;  
conflict structure for RIC;
- **analysis:** for RIC quite involved  
(e.g. in "Dutch Book" not self-contained);  
for D&C geometrically challenging

Timothy Chan   3-dimensional convex hulls via  
kinetic 2-dimensional convex hulls

## Difficulties:

- **geometric:** 3 dimensional;  
sleeve structure in divide & conquer algorithm;
- **data structures:** triangulation that needs to be navigatable  
and to be updated;  
conflict structure for RIC;
- **analysis:** for RIC quite involved  
(e.g. in "Dutch Book" not self-contained);  
for D&C geometrically challenging

Timothy Chan 3-dimensional convex hulls via  
kinetic 2-dimensional convex hulls

- **kinetic algorithms**

Timothy Chan

3-dimensional convex hulls via  
kinetic 2-dimensional convex hulls

Timothy Chan

3-dimensional convex hulls via  
kinetic 2-dimensional convex hulls



Timothy Chan

3-dimensional convex hulls via  
kinetic 2-dimensional convex hulls

Want upper hull of  $\{p_1, \dots, p_n\}$



Timothy Chan

3-dimensional convex hulls via  
kinetic 2-dimensional convex hulls

Want upper hull of  $\{p_1, \dots, p_n\}$

Compute kinetic upper hull  
of  $\{p_1(t), \dots, p_n(t)\}$  in  $\mathbb{R}^2$

Timothy Chan

3-dimensional convex hulls via  
kinetic 2-dimensional convex hulls

Want upper hull of  $\{p_1, \dots, p_n\}$

$$p_i = (x_i, y_i, z_i)$$

Compute kinetic upper hull  
of  $\{p_1(t), \dots, p_n(t)\}$  in  $\mathbb{R}^2$

Want upper hull of  $\{p_1, \dots, p_n\}$

$$p_i = (x_i, y_i, z_i)$$

Compute kinetic upper hull  
of  $\{p_1(t), \dots, p_n(t)\}$  in  $\mathbb{R}^2$

$$p_i(t) = (x_i, z_i - ty_i)$$

Want upper hull of  $\{p_1, \dots, p_n\}$

$$p_i = (x_i, y_i, z_i)$$

$p_i$       above  
          on    plane  $z = ax + by + c$   
          below

Compute kinetic upper hull  
of  $\{p_1(t), \dots, p_n(t)\}$  in  $\mathbb{R}^2$

$$p_i(t) = (x_i, z_i - ty_i)$$

Want upper hull of  $\{p_1, \dots, p_n\}$

$$p_i = (x_i, y_i, z_i)$$

$p_i$  above  
on  
below

$$z = ax + by + c$$

Compute kinetic upper hull  
of  $\{p_1(t), \dots, p_n(t)\}$  in  $\mathbb{R}^2$

$$p_i(t) = (x_i, z_i - ty_i)$$

$p_i(t)$  above  
on  
below

$$\text{line } \eta = a\xi + c$$

for some  $t$  (i.e.  $t=b$ )

Want upper hull of  $\{p_1, \dots, p_n\}$

$$p_i = (x_i, y_i, z_i)$$

above  
on plane  $z = ax + by + c$   
below

$$z_i \stackrel{\geq}{=} ax_i + by_i + c \stackrel{\leq}{=}$$

Compute kinetic upper hull  
of  $\{p_1(t), \dots, p_n(t)\}$  in  $\mathbb{R}^2$

$$p_i(t) = (x_i, y_i - ty_i)$$

above  
on line  $\eta = a\xi + c$   
below  
for some  $t$  (i.e.  $t=b$ )

Want upper hull of  $\{p_1, \dots, p_n\}$

$$p_i = (x_i, y_i, z_i)$$

above  
on plane  $z = ax + by + c$   
below

$$z_i \begin{matrix} \geq \\ = \\ \leq \end{matrix} ax_i + by_i + c$$

Compute kinetic upper hull  
of  $\{p_1(t), \dots, p_n(t)\}$  in  $\mathbb{R}^2$

$$p_i(t) = (x_i, z_i - ty_i)$$

above  
on line  $\eta = a\xi + c$   
below  
for some  $t$  (i.e.  $t=b$ )

$$z_i - by_i \begin{matrix} \geq \\ = \\ \leq \end{matrix} ax_i + c$$

Timothy Chan

3-dimensional convex hulls via  
kinetic 2-dimensional convex hulls



Timothy Chan

3-dimensional convex hulls via  
kinetic 2-dimensional convex hulls

Compute kinetic upper hull of  $\{ p_1(\mathbf{t}), \dots, p_n(\mathbf{t}) \}$  in  $\mathbb{R}^2$

Timothy Chan

3-dimensional convex hulls via  
kinetic 2-dimensional convex hulls

Compute kinetic upper hull of  $\{p_1(\mathbf{t}), \dots, p_n(\mathbf{t})\}$  in  $\mathbb{R}^2$

straightforward divide and conquer algorithm:

merge “movie” of upper hull of left point set half and  
“movie” of upper hull of right point set half

Timothy Chan

3-dimensional convex hulls via  
kinetic 2-dimensional convex hulls

Compute kinetic upper hull of  $\{p_1(\mathbf{t}), \dots, p_n(\mathbf{t})\}$  in  $\mathbb{R}^2$

straightforward divide and conquer algorithm:

merge “movie” of upper hull of left point set half and  
“movie” of upper hull of right point set half

colinearity events (i.e. changes on the kinetic upper hull)  
correspond to facets of the 3d upper hull

**Timothy Chan**

3-dimensional convex hulls via  
kinetic 2-dimensional convex hulls

Compute kinetic upper hull of  $\{p_1(\dagger), \dots, p_n(\dagger)\}$  in  $\mathbb{R}^2$

straightforward divide and conquer algorithm:

merge “movie” of upper hull of left point set half and  
“movie” of upper hull of right point set half

colinearity events (i.e. changes on the kinetic upper hull)  
correspond to facets of the 3d upper hull

**see** [www.cs.uwaterloo.ca/~tmchan/ch3d/ch3d.pdf](http://www.cs.uwaterloo.ca/~tmchan/ch3d/ch3d.pdf)

```

1 // Timothy Chan "ch3d.cc" 12/02 3-d lower hull (in C++)
2 // a simple implementation of the  $O(n \log n)$  divide-and-conquer algorithm
3
4 // input: coordinates of points
5 // n x_0 y_0 z_0 ... x_{n-1} y_{n-1} z_{n-1}
6
7 // output: indices of facets
8 // i_1 j_1 k_1 i_2 j_2 k_2 ...
9
10 // warning: ignores degeneracies and robustness
11 // space: uses 8n pointers
12
13
14
15 #include <istream.h>
16
17 struct Point {
18     double x, y, z;
19     Point *prev, *next;
20     void act() {
21         if (prev->next != this) prev->next = next->prev = this; // insert
22     }
23     else { prev->next = next; next->prev = prev; } // delete
24 };
25
26 const double INF = 1e99;
27 static Point nil = {INF, INF, INF, 0, 0};
28 Point *NIL = &nil;
29
30 inline double turn(Point vp, Point wq, Point vr) { // <0 iff cw
31     if (p == NIL || q == NIL || r == NIL) return 1.0;
32     return (q->x-p->x)*(r->y-p->y) - (r->x-p->x)*(q->y-p->y);
33 }
34
35 inline double time(Point vp, Point wq, Point vr) { // when turn changes
36     if (p == NIL || q == NIL || r == NIL) return INF;
37     return ((q->x-p->x)*(r->z-p->z) - (r->x-p->x)*(q->z-p->z)) / turn(p,q,r);
38 }
39
40 Point *sort(Point P[], int n) { // mergesort
41     Point *a, *b, *c, head;
42
43     if (n == 1) { p[0].next = NIL; return p; }
44     a = sort(p, n/2);
45     b = sort(p+n/2, n-n/2);
46     c = &head;
47     do
48     if (a->x < b->x) { c = c->next = a; a = a->next; }
49     else { c = c->next = b; b = b->next; }
50     while (c != NIL);
51     return head.next;
52 }
53
54 void hull(Point *list, int n, Point **a, Point **b) { // the algorithm
55
56     Point *u, *v, *w, *old;
57     double t(s), oldt, nextt;
58     int i, j, k, l, m, n1;
59
60

```


 SAARLAND  
UNIVERSITY  
COMPUTER SCIENCE

# 1993 Teaching Computational Geometry, I

## Topics

- Randomized algorithms as first class objects
- The importance of invariants for sweep algorithms
- Representation of planar subdivisions
- Planar point location
- Linear Programming
- 3d Convex hulls
- Perturbations
- Upper bound theorem for polytopes

## • Perturbations

- basic method to “deal” with degeneracies in geometric computations

## • Perturbations

- basic method to “deal” with degeneracies in geometric computations
- sufficiently well understood and enough theory existent to be taught in introductory computational geometry course



## • Perturbations

- basic method to “deal” with degeneracies in geometric computations
- sufficiently well understood and enough theory existent to be taught in introductory computational geometry course

**BUT:**

## • Perturbations

- basic method to “deal” with degeneracies in geometric computations
- sufficiently well understood and enough theory existent to be taught in introductory computational geometry course

**BUT:** theoretical solution for a practical problem  
that is not viable in practice in most cases

## • Perturbations

- basic method to “deal” with degeneracies in geometric computations
- sufficiently well understood and enough theory existent to be taught in introductory computational geometry course

**BUT:** theoretical solution for a practical problem that is not viable in practice in most cases

contrary to some expectations not that useful in the **classification** and **enumeration** of the possible degenerate cases

# 1993 Teaching Computational Geometry, I

## Topics

- Randomized algorithms as first class objects
- The importance of invariants for sweep algorithms
- Representation of planar subdivisions
- Planar point location
- Linear Programming
- 3d Convex hulls
- Perturbations
- Upper bound theorem for polytopes

# 1993 Teaching Computational Geometry, I

## Topics

- Randomized algorithms as first class objects
- The importance of invariants for sweep algorithms
- Representation of planar subdivisions
- Planar point location
- Linear Programming
- 3d Convex hulls
- Perturbations
- ~~• Upper bound theorem for polytopes~~

# 1993 Teaching Computational Geometry, I

## Topics

- Randomized algorithms as first class objects
- The importance of invariants for sweep algorithms
- Representation of planar subdivisions
- Planar point location
- Linear Programming
- 3d Convex hulls
- Perturbations
- ~~• Upper bound theorem for polytopes~~
- Duality viewed geometrically

- Duality viewed geometrically

- **Duality viewed geometrically**

point — line duality in the plane



- Duality viewed geometrically

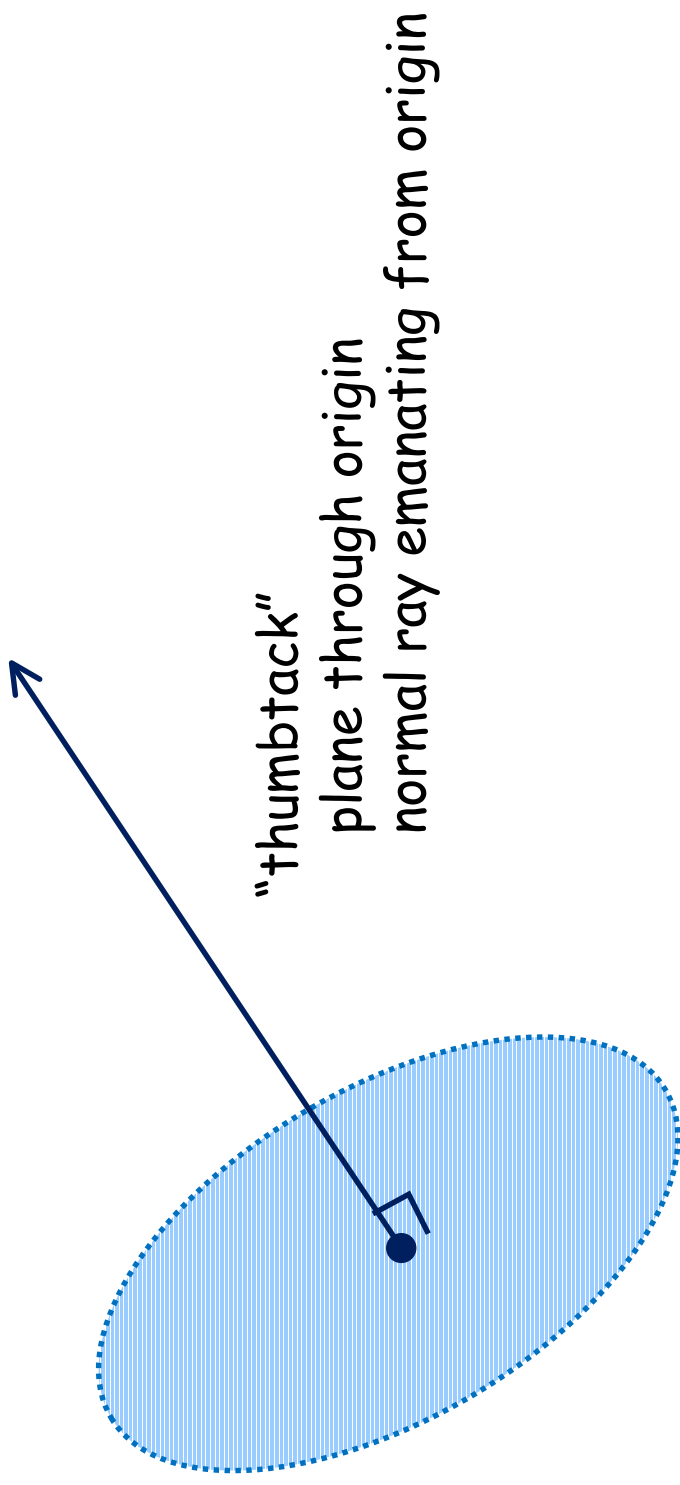
point — line duality in the plane



origin in  $\mathbb{R}^3$

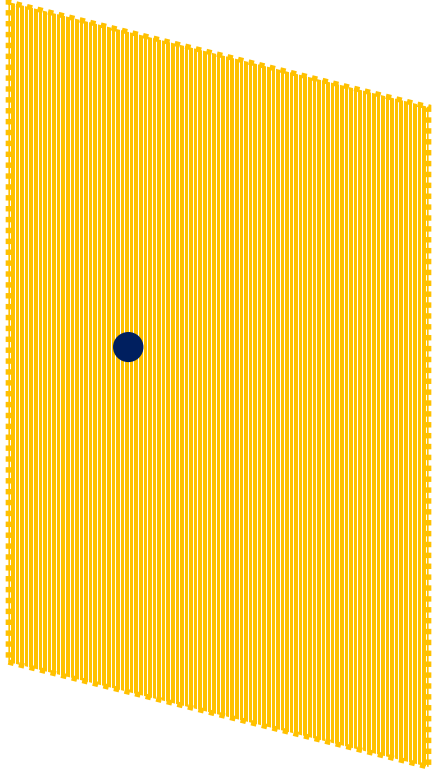
- Duality viewed geometrically

point — line duality in the plane



- Duality viewed geometrically

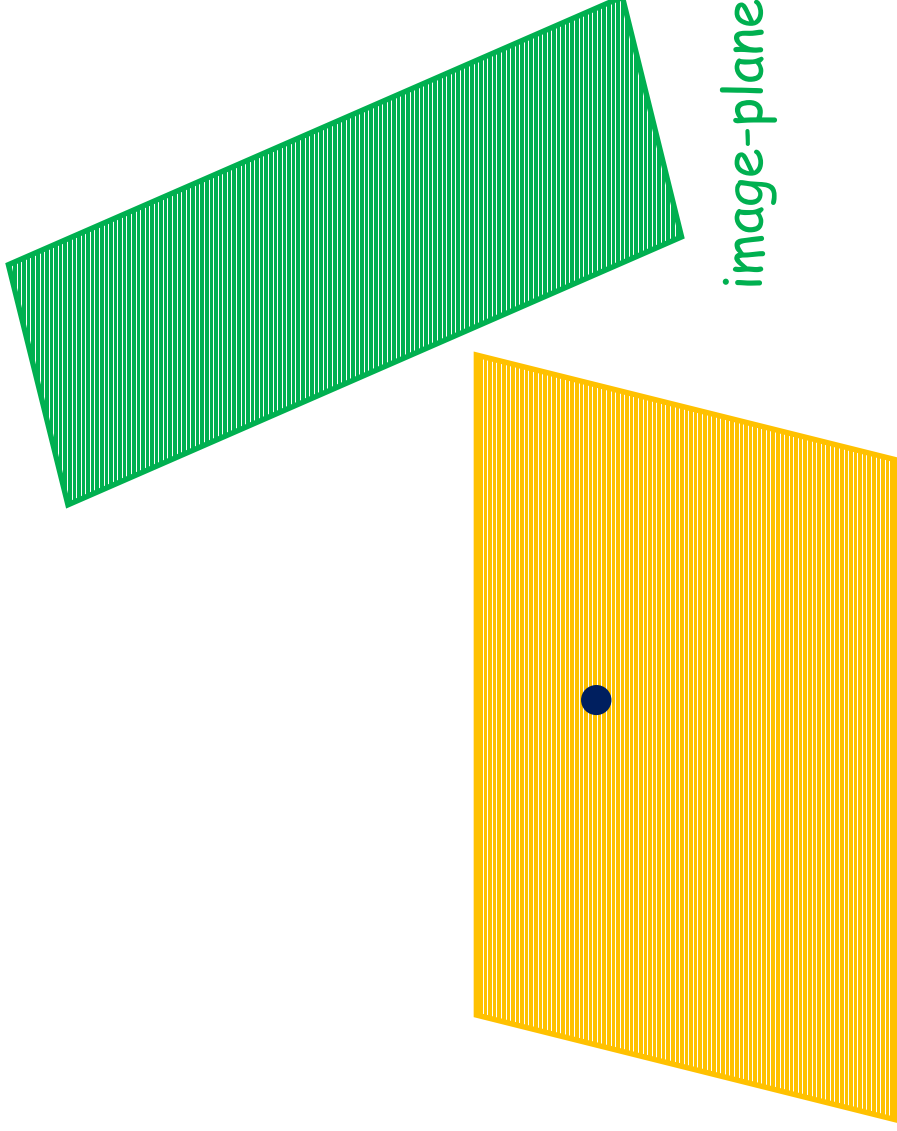
point — line duality in the plane



source-plane (not containing origin)

- Duality viewed geometrically

point — line duality in the plane

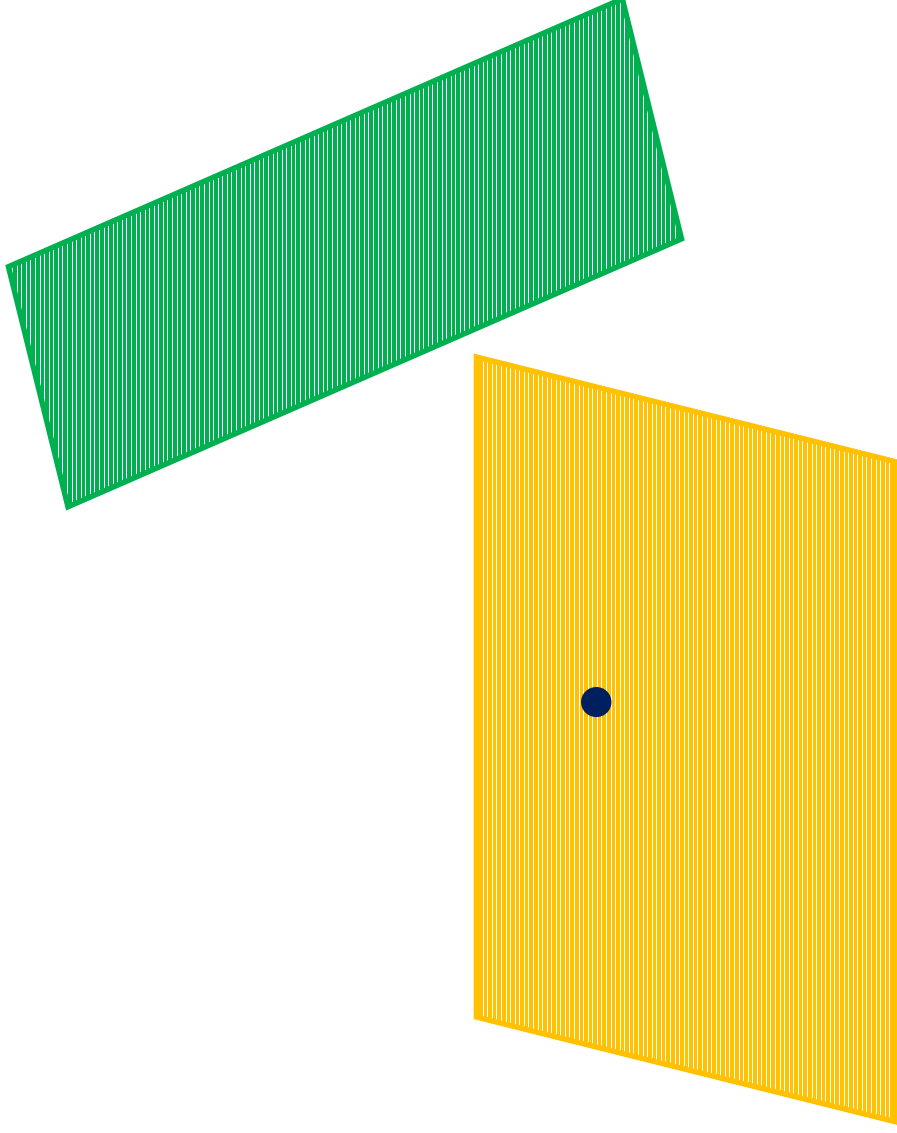


source-plane (not containing origin)

image-plane (not containing origin)

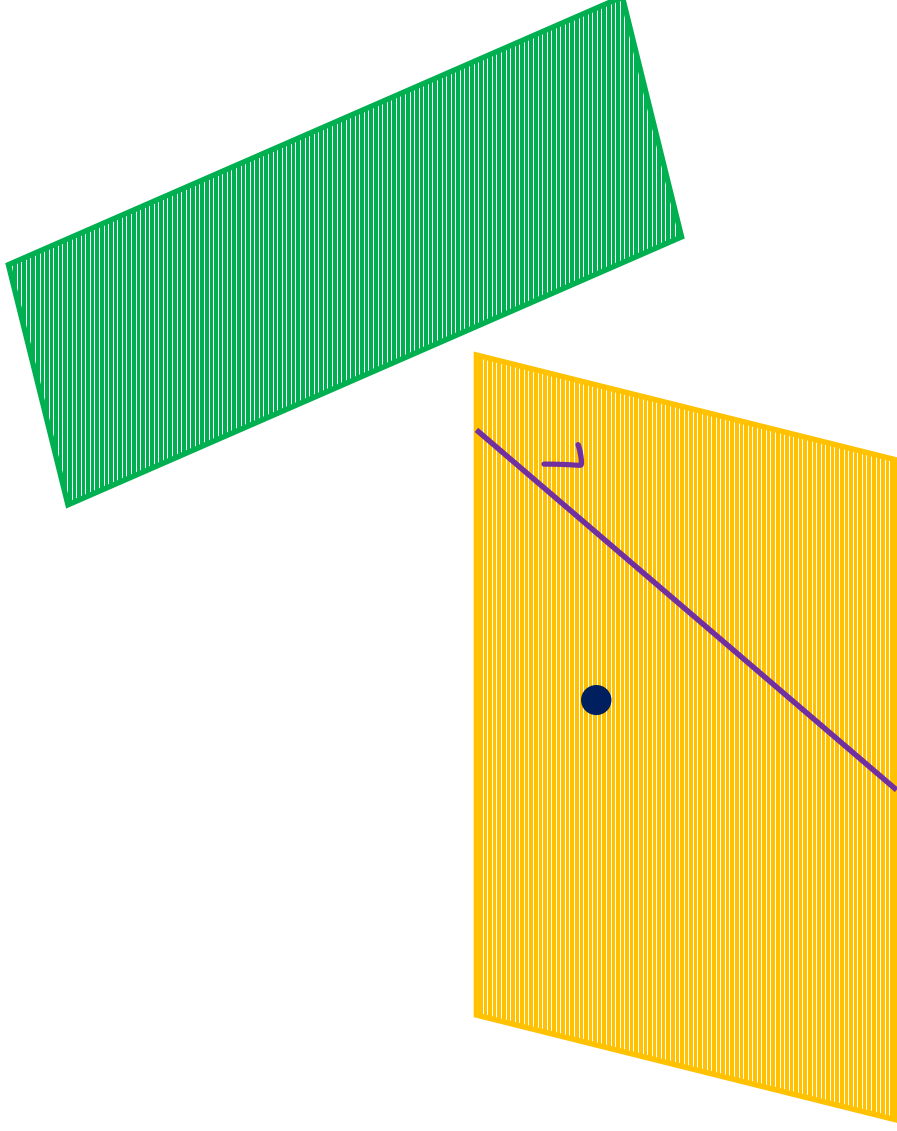
- Duality viewed geometrically

point — line duality in the plane



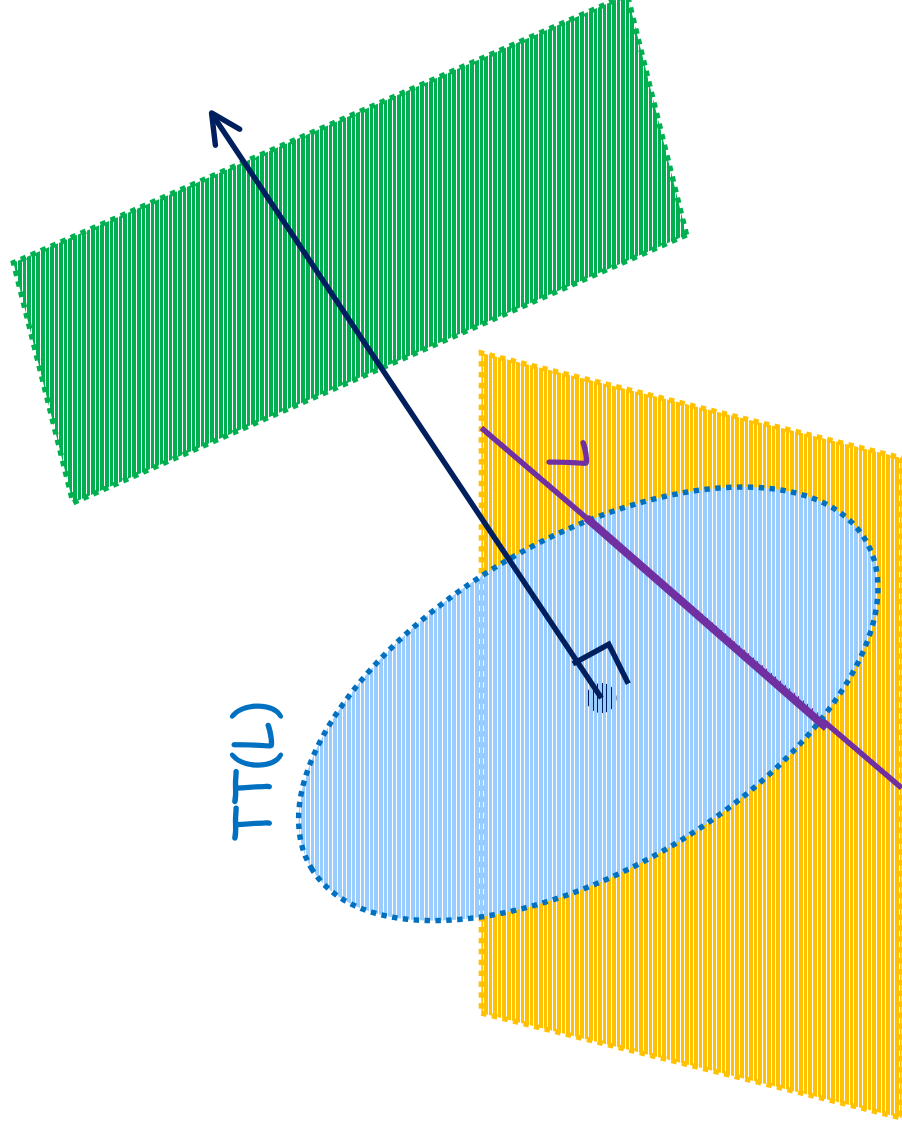
- Duality viewed geometrically

point — line duality in the plane



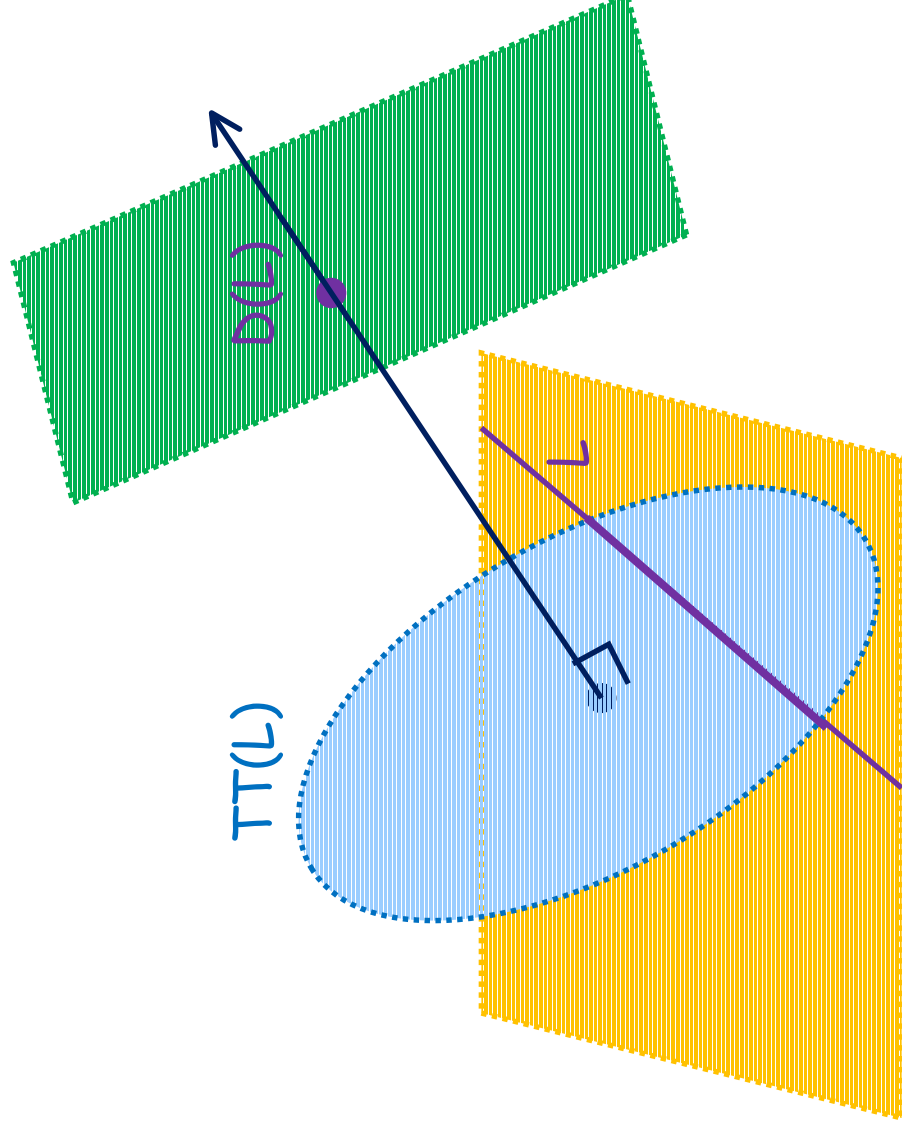
- Duality viewed geometrically

point — line duality in the plane



# • Duality viewed geometrically

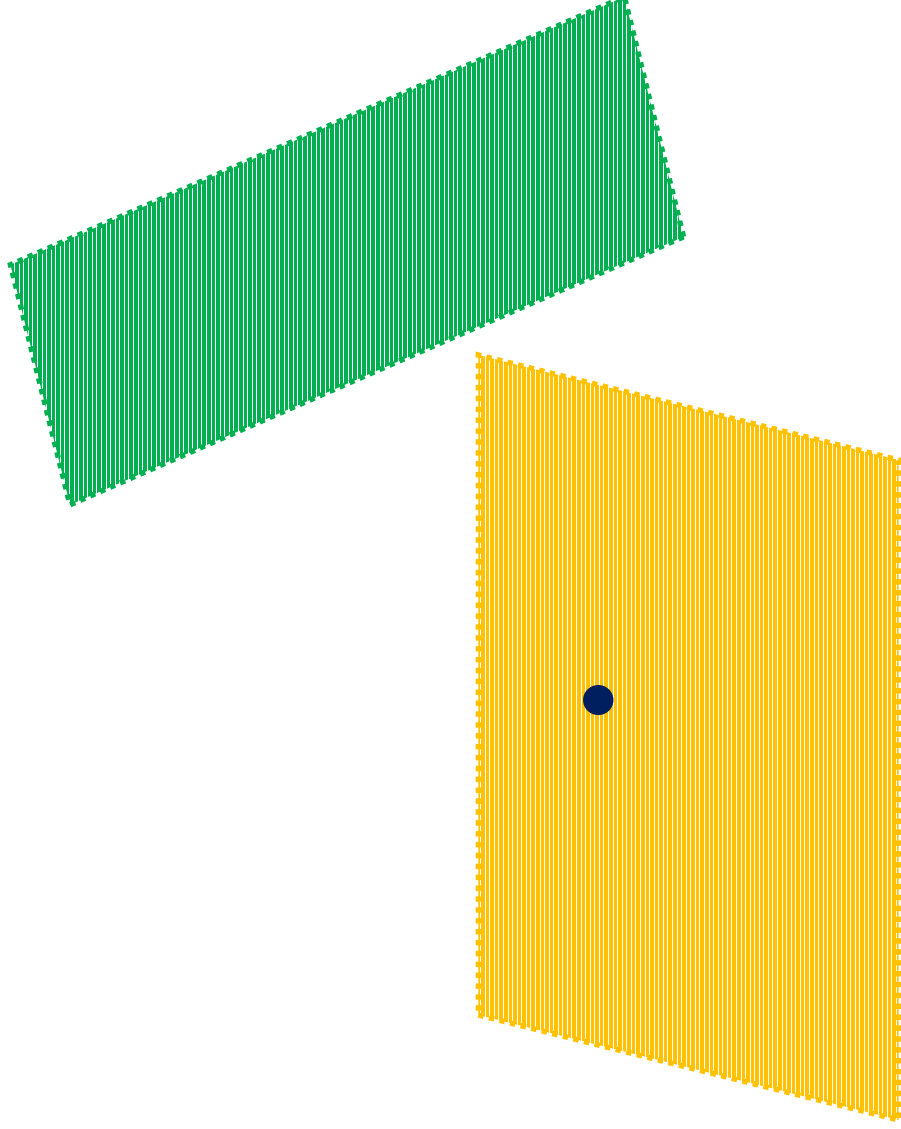
point — line duality in the plane





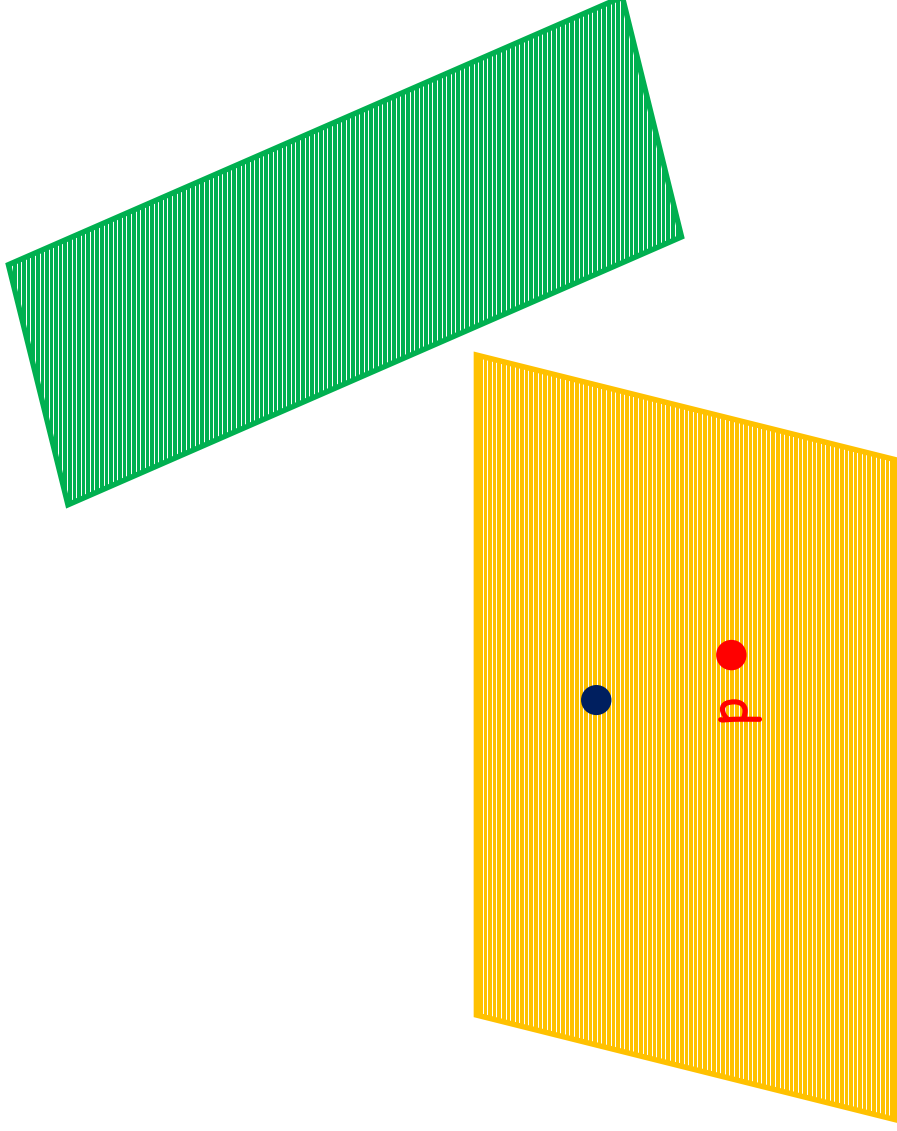
# • Duality viewed geometrically

point — line duality in the plane



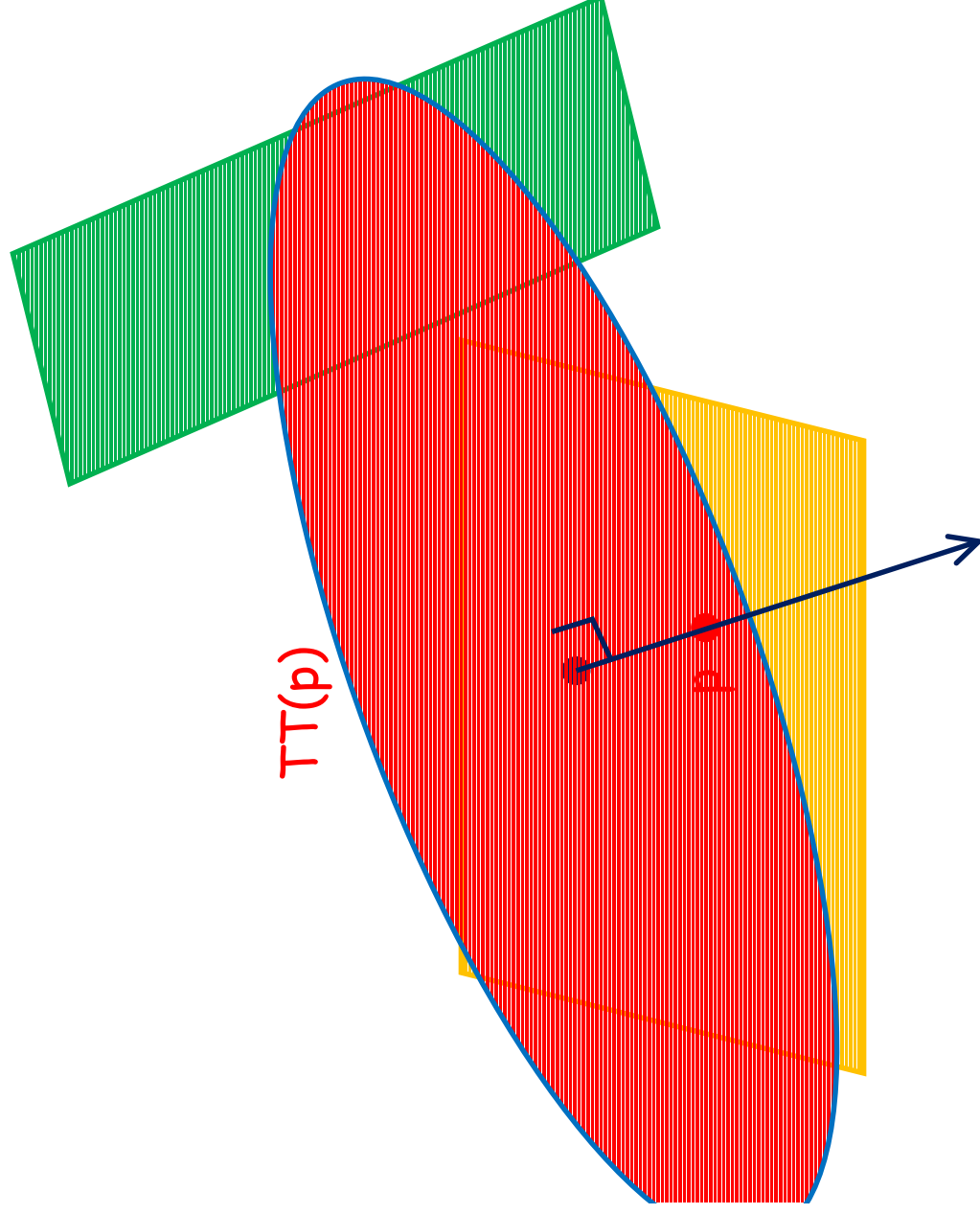
- Duality viewed geometrically

point — line duality in the plane



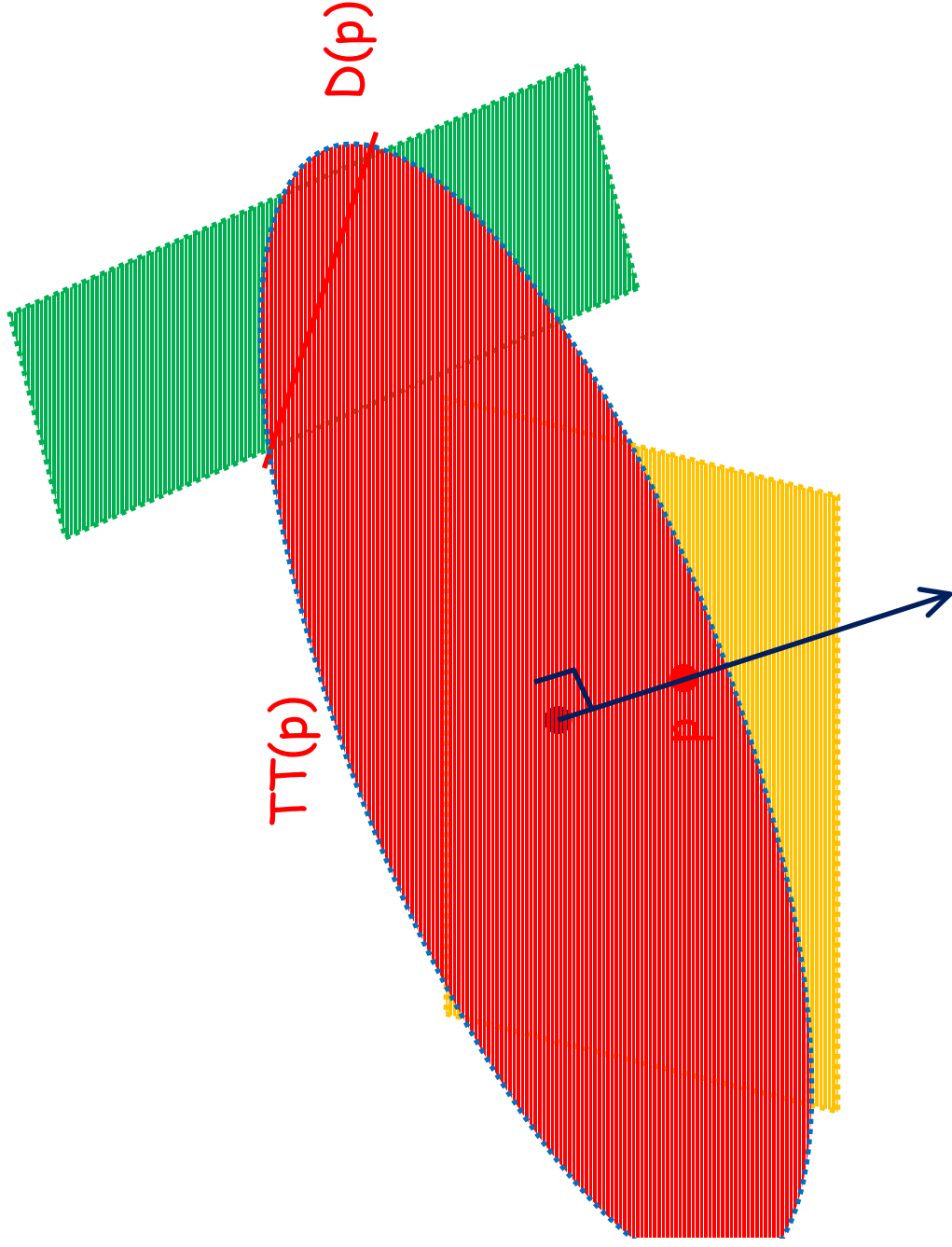
- Duality viewed geometrically

point — line duality in the plane



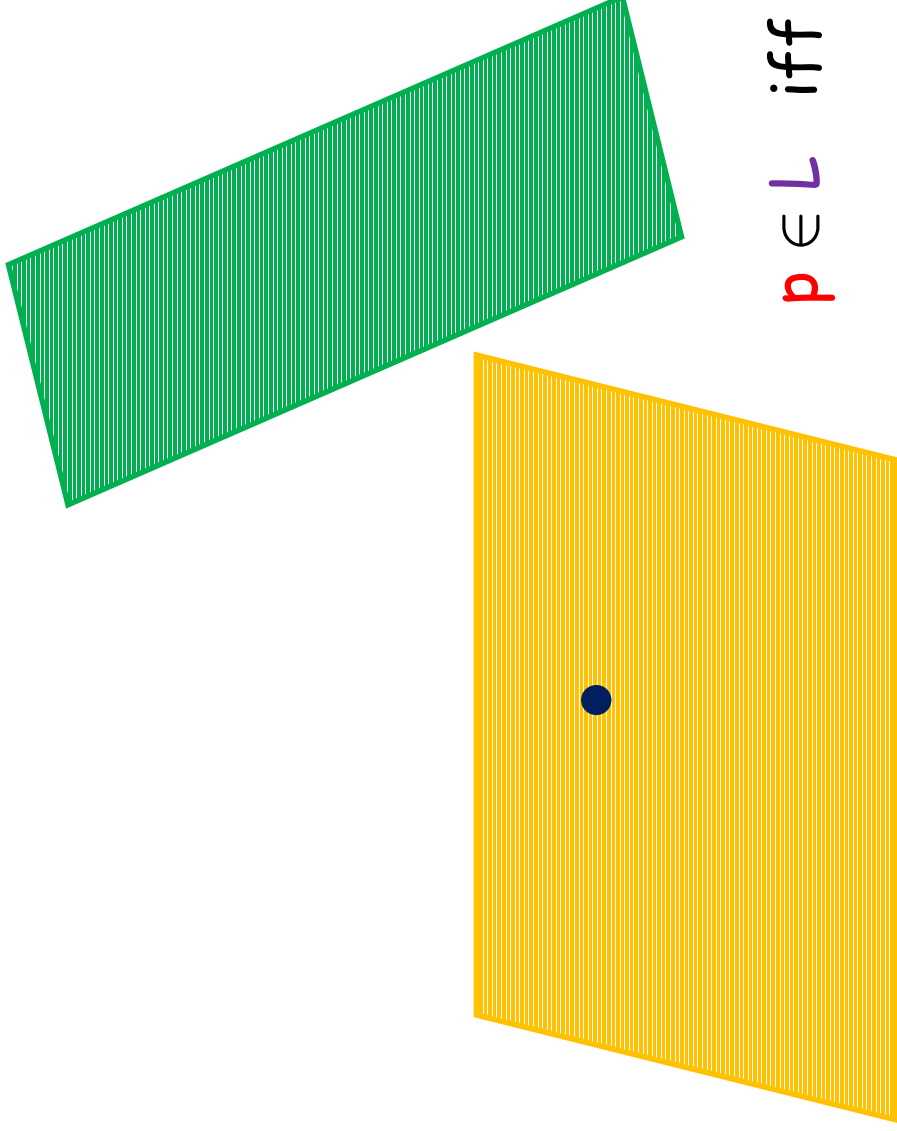
- Duality viewed geometrically

point — line duality in the plane



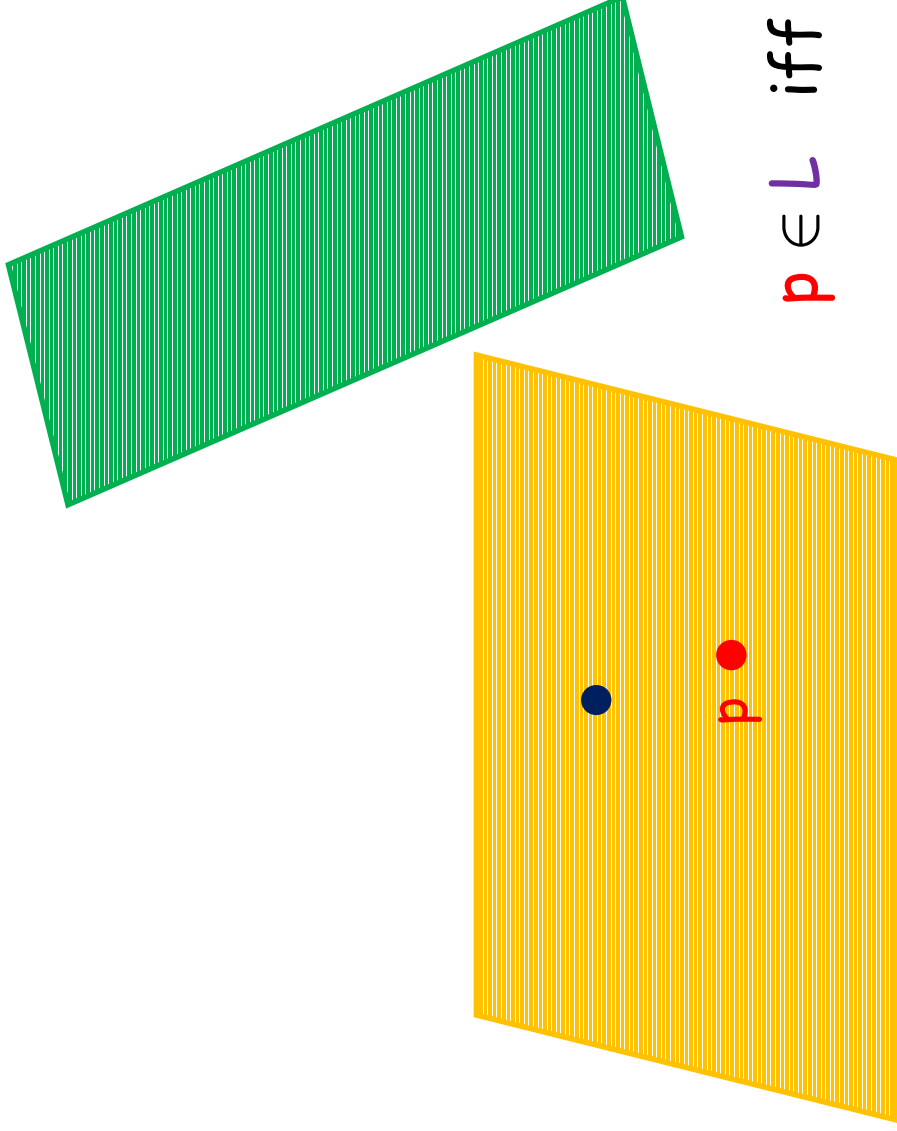
- Duality viewed geometrically

point — line duality in the plane



- Duality viewed geometrically

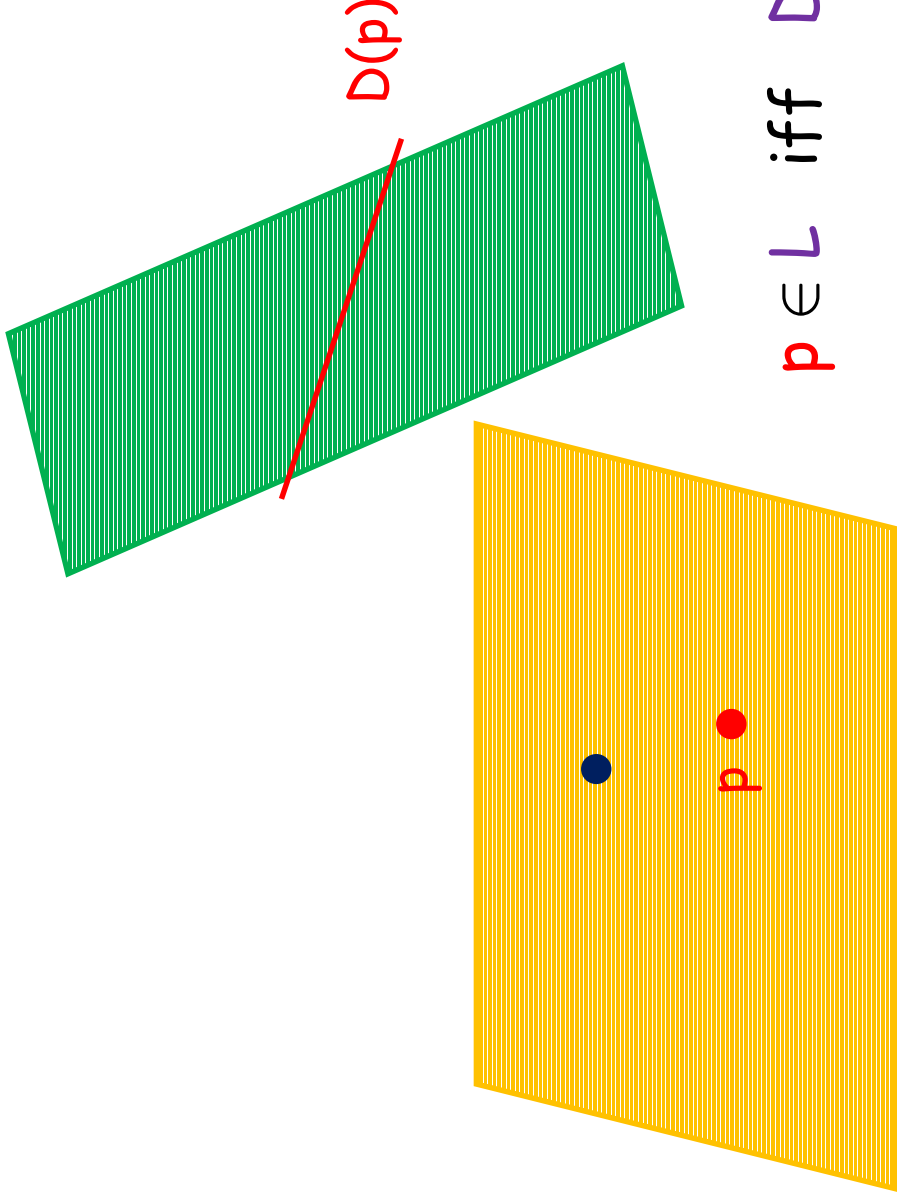
point — line duality in the plane



$$p \in L \text{ iff } D(L) \in D(p)$$

- Duality viewed geometrically

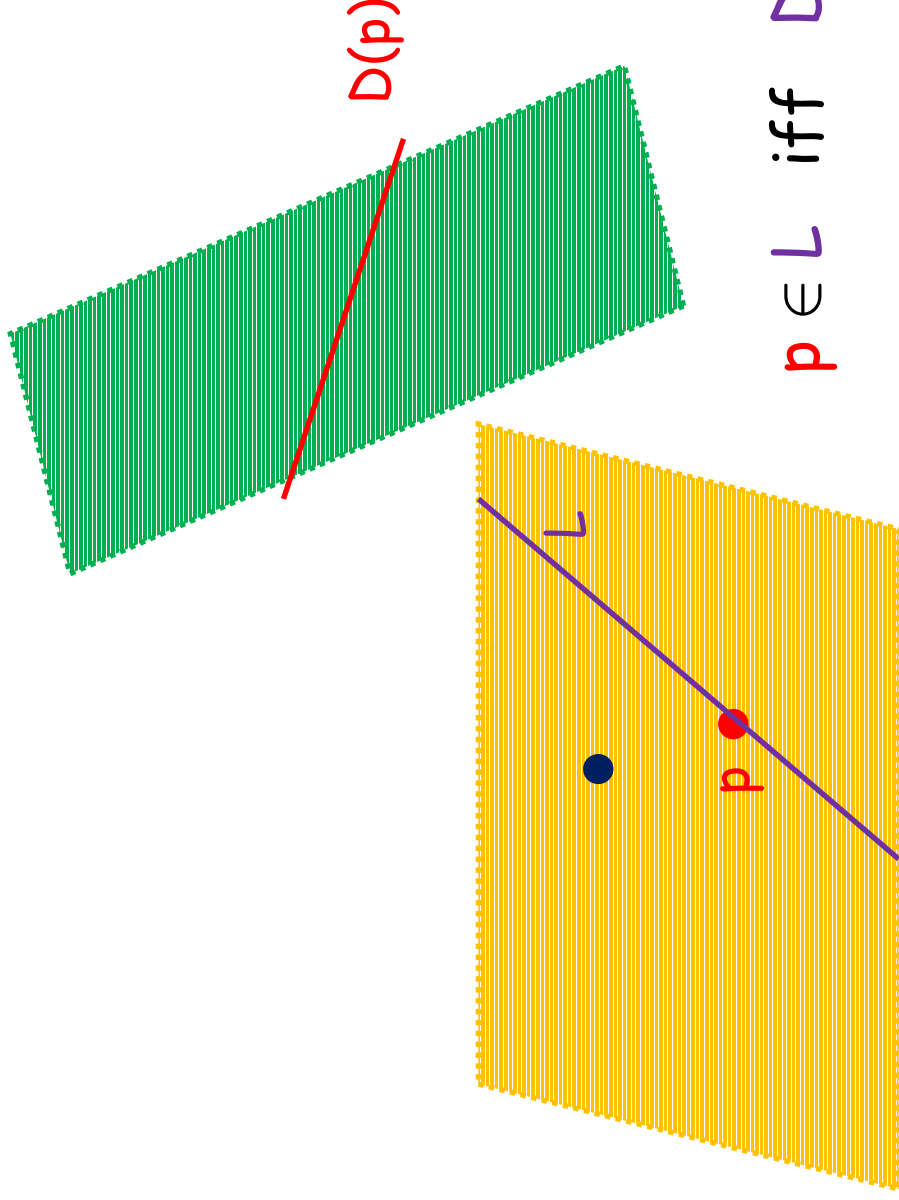
point — line duality in the plane



$$p \in L \text{ iff } D(L) \in D(p)$$

- Duality viewed geometrically

point — line duality in the plane

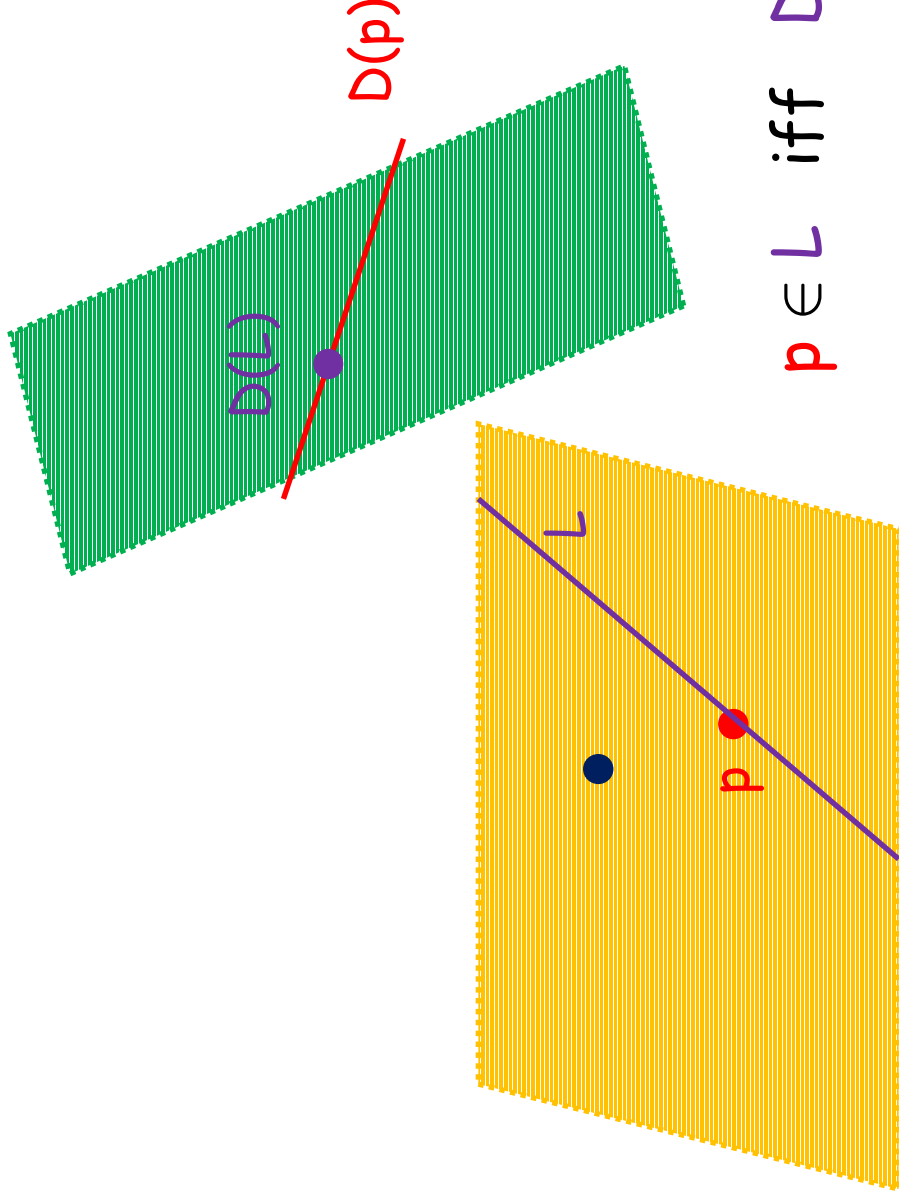


$$p \in L \text{ iff } D(L) \in D(p)$$



- Duality viewed geometrically

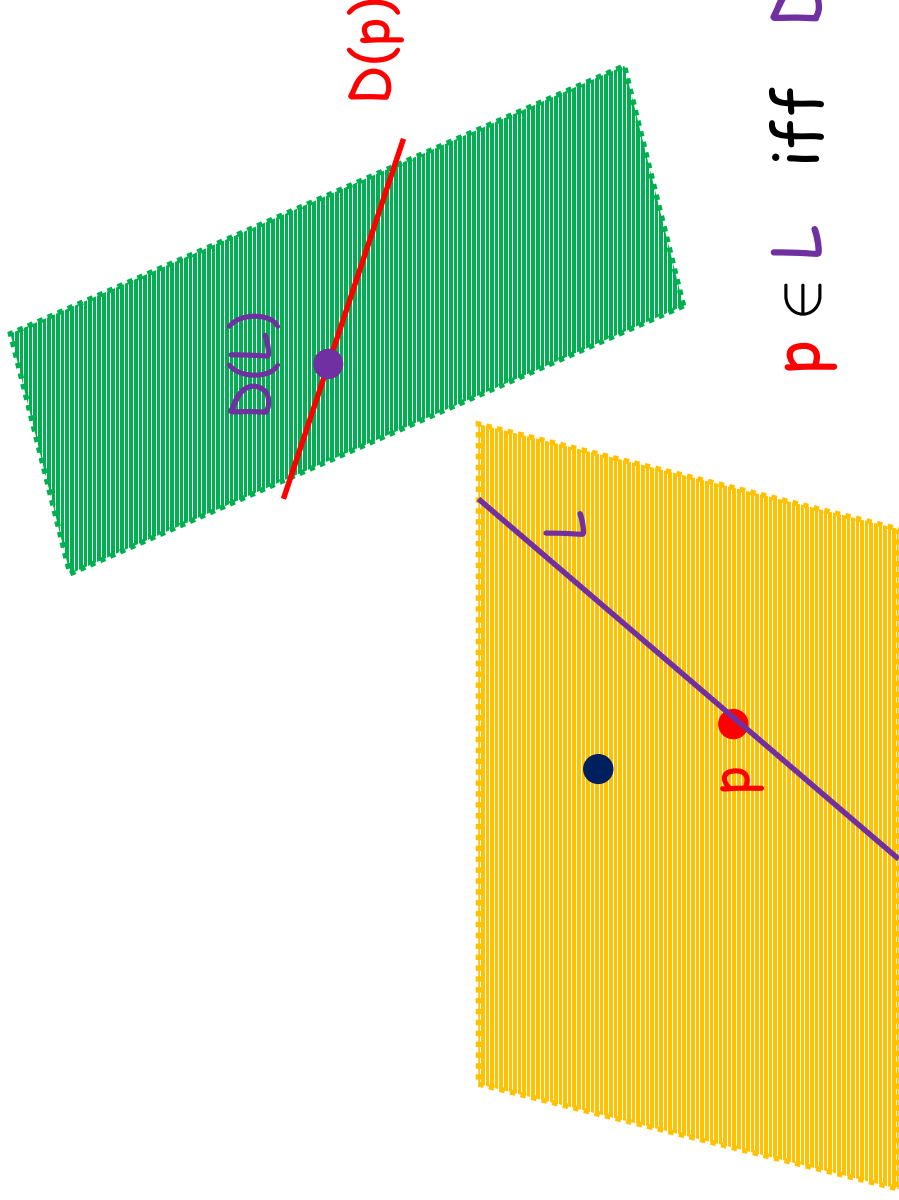
point — line duality in the plane



$$p \in L \text{ iff } D(L) \in D(p)$$

- Duality viewed geometrically

point — line duality in the plane

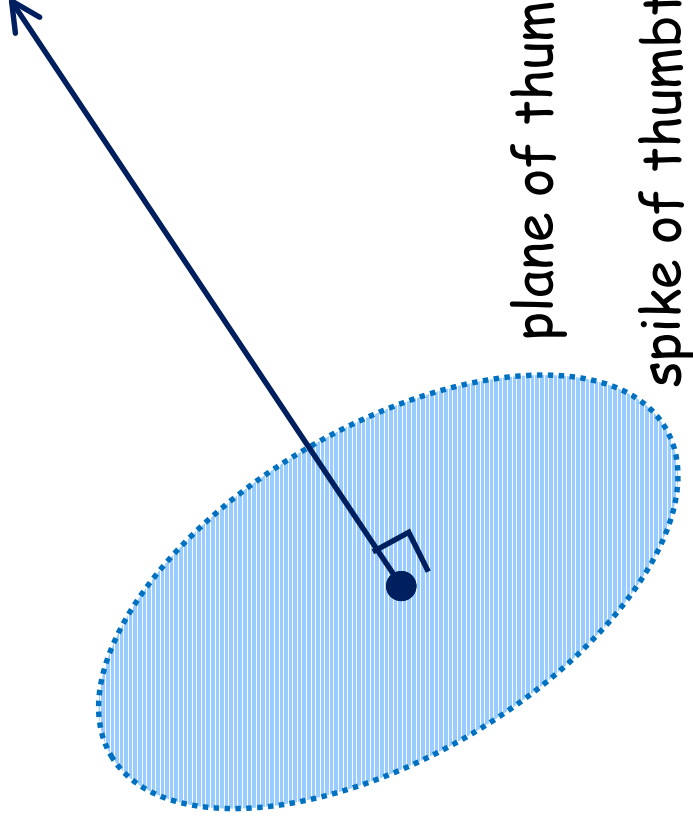


$p \in L$  iff  $D(L) \in D(p)$

since

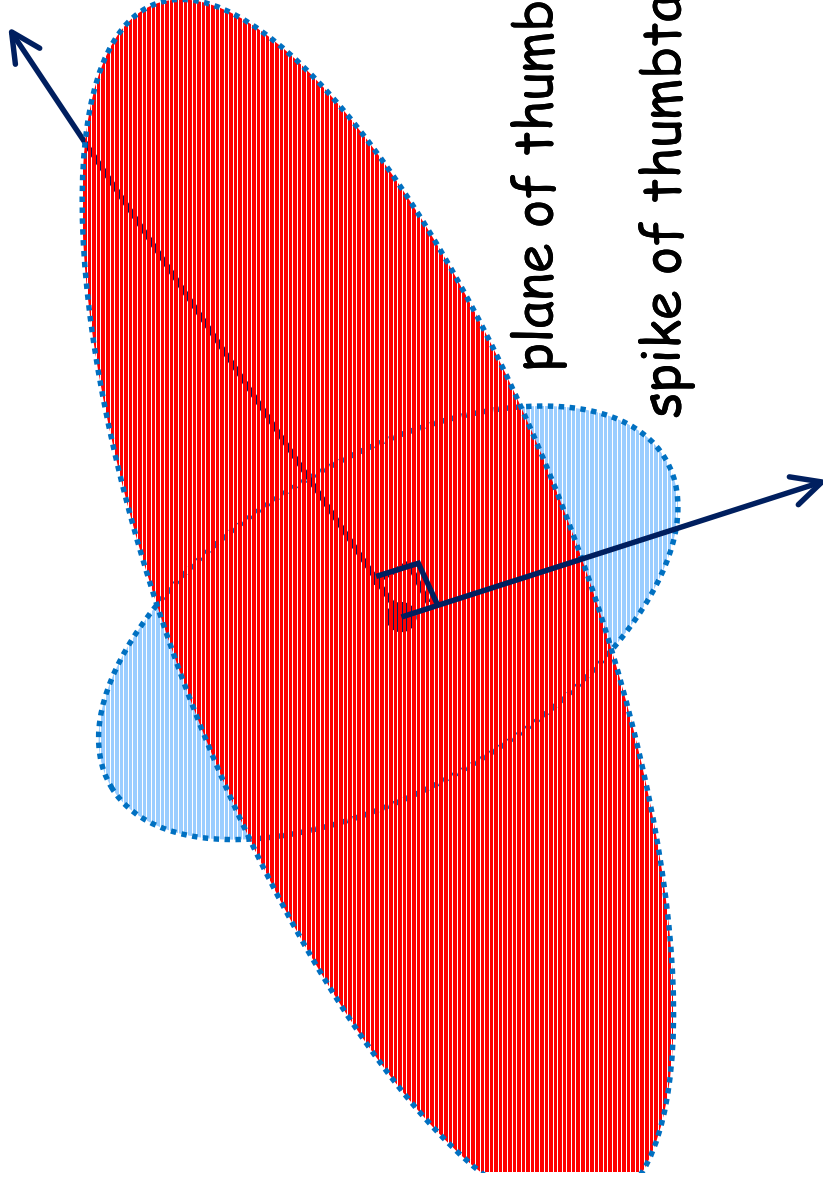
- Duality viewed geometrically

point — line duality in the plane



- Duality viewed geometrically

point — line duality in the plane



- **Duality viewed geometrically**

point — line duality in the plane

**Coordinate representations:**

- Duality viewed geometrically

point — line duality in the plane

Coordinate representations:

$$ax + by = 1 \quad \longleftrightarrow \quad (a, b)$$

- Duality viewed geometrically

point — line duality in the plane

Coordinate representations:

$$ax + by = 1 \quad \longleftrightarrow \quad (a, b)$$

source plane:  $z = 1$

image plane:  $z = -1$

- Duality viewed geometrically

point — line duality in the plane

Coordinate representations:

$$ax + by = 1 \quad \xleftrightarrow{D} \quad (a, b)$$

source plane:  $z = 1$       image plane:  $z = -1$

$$y + d = kx \quad \xleftrightarrow{D} \quad (k, d)$$



- Duality viewed geometrically

point — line duality in the plane

Coordinate representations:

$$ax + by = 1 \quad \xleftrightarrow{D} \quad (a, b)$$

source plane:  $z = 1$       image plane:  $z = -1$

$$y + d = kx \quad \xleftrightarrow{D} \quad (k, d)$$

source plane:  $z = 1$       image plane:  $y = 1$

# Issues: data structures

## Issues: data structures

segment trees, interval trees, kd-trees, etc.

## Issues: data structures

segment trees, interval trees, kd-trees, etc.

students find them uninteresting/boring;  
yet, they are important tools

## Issues: data structures

segment trees, interval trees, kd-trees, etc.

students find them uninteresting/boring;  
yet, they are important tools

Any tricks for teaching them ?

## Issues: data structures

segment trees, interval trees, kd-trees, etc.

students find them uninteresting/boring;  
yet, they are important tools

Any tricks for teaching them  
or for rendering them unnecessary ?

**Issues:** the assumption of perfect real arithmetic

## **Issues:** the assumption of perfect real arithmetic

- good for teaching geometric algorithms
- completely unrealistic in practice



## **Issues:** the assumption of perfect real arithmetic

- good for teaching geometric algorithms
- completely unrealistic in practice

We need an economical way of introducing these issues and dealing with them.

# Issues: implementing

## Issues: implementing

- students should implement some basic geometric algorithms and toy with such implementations

## Issues: implementing

- students should implement some basic geometric algorithms and toy with such implementations
- inspite of LEDA, CGAL, etc. the necessary effort still seems incommensurate to the expected gain

## Issues: implementing

- students should implement some basic geometric algorithms and toy with such implementations
- inspite of LEDA, CGAL, etc. the necessary effort still seems incommensurate to the expected gain

Needed: interactive "geometry algorithm sandbox"

# How will we teach Computational Geometry in 2025 ?

