



Delaunay triangulation,

Theory vs practice

Olivier Devillers

*Inria*  
informatiques mathématiques



# Delaunay triangulation,

## Theory vs practice

<http://www.inria.fr/sophia/members/Olivier.Devillers/EuroCG2012/>

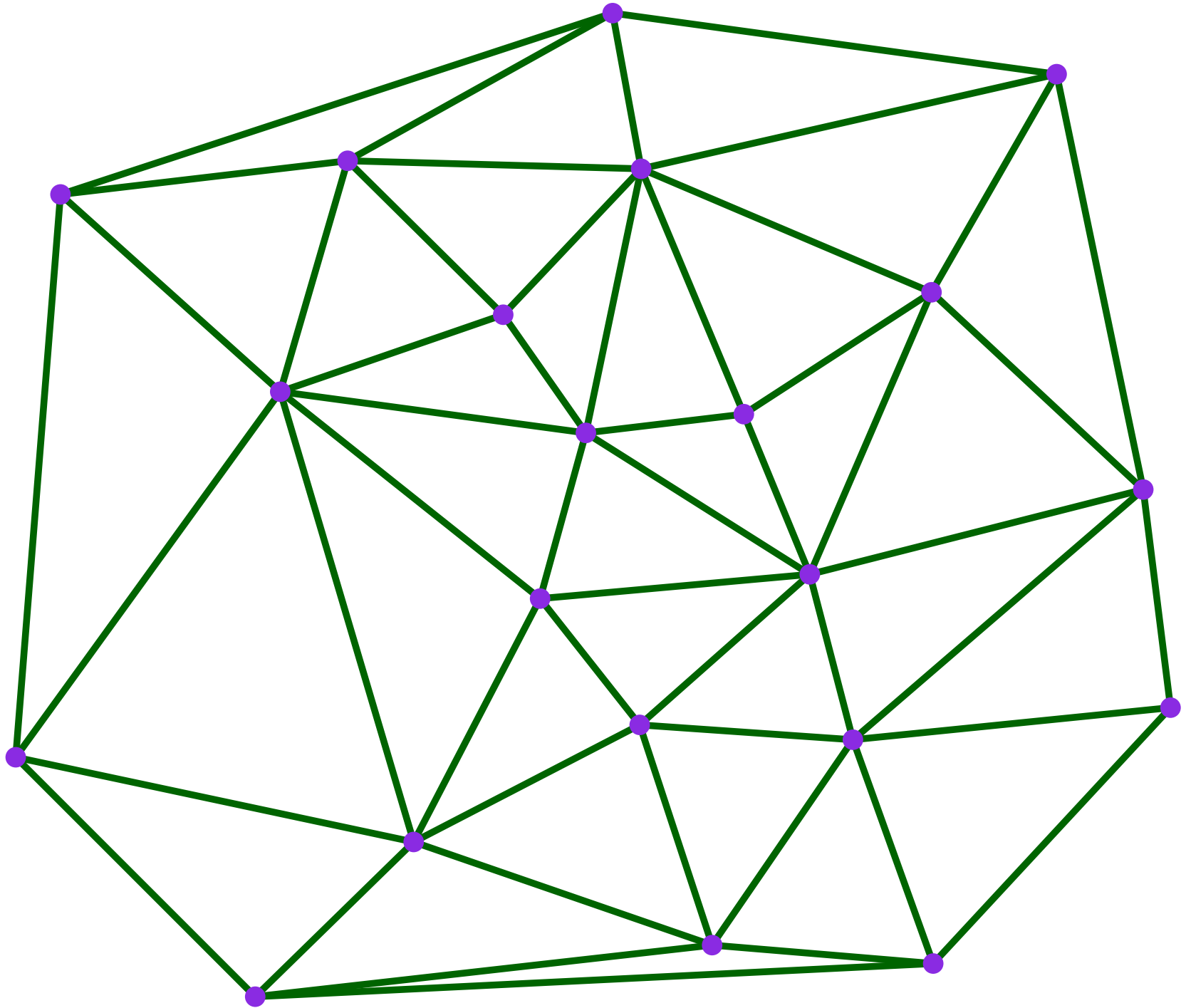
Bibliographical notes

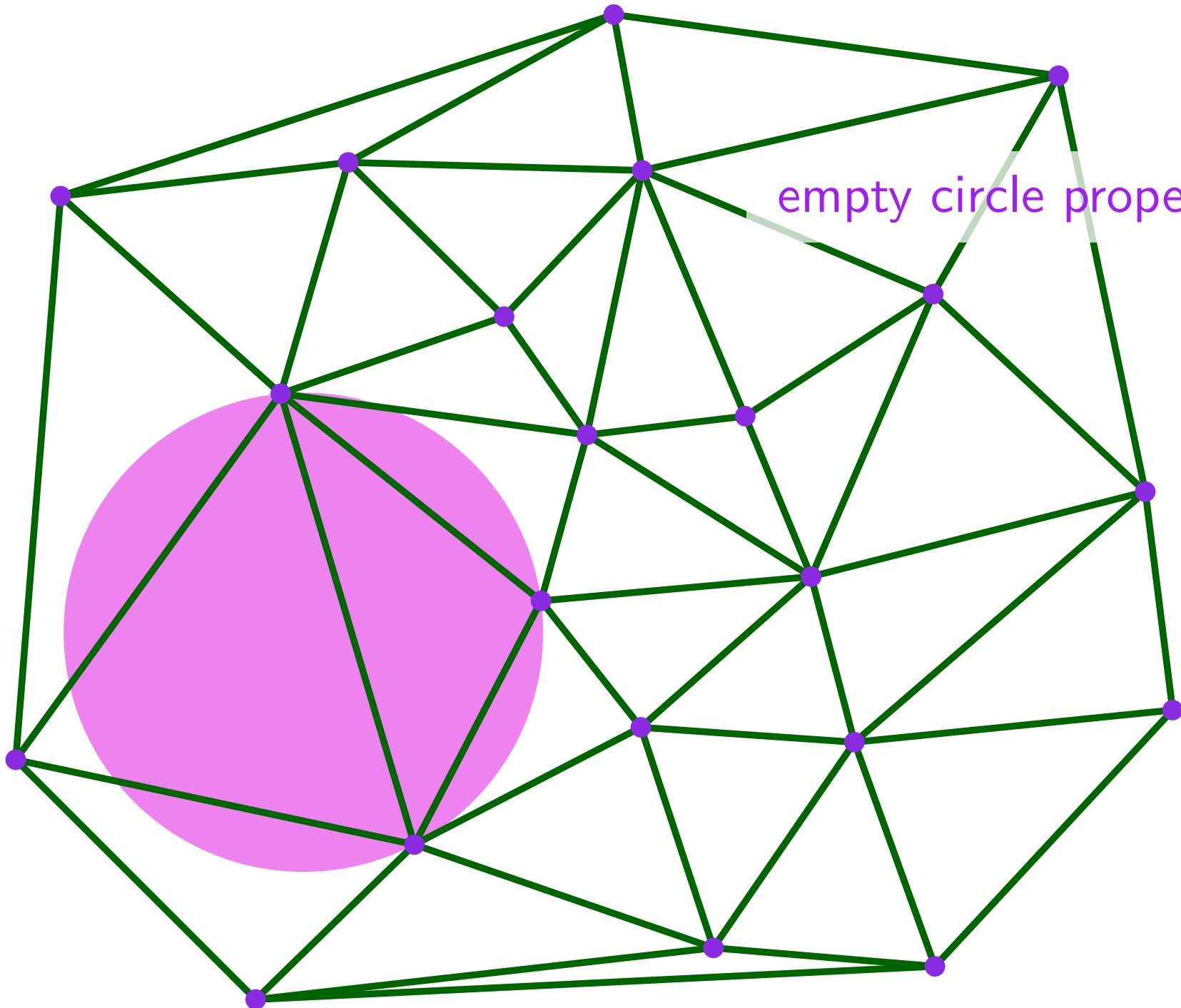
[References, given in the abstract]

Benchmarks (src code)

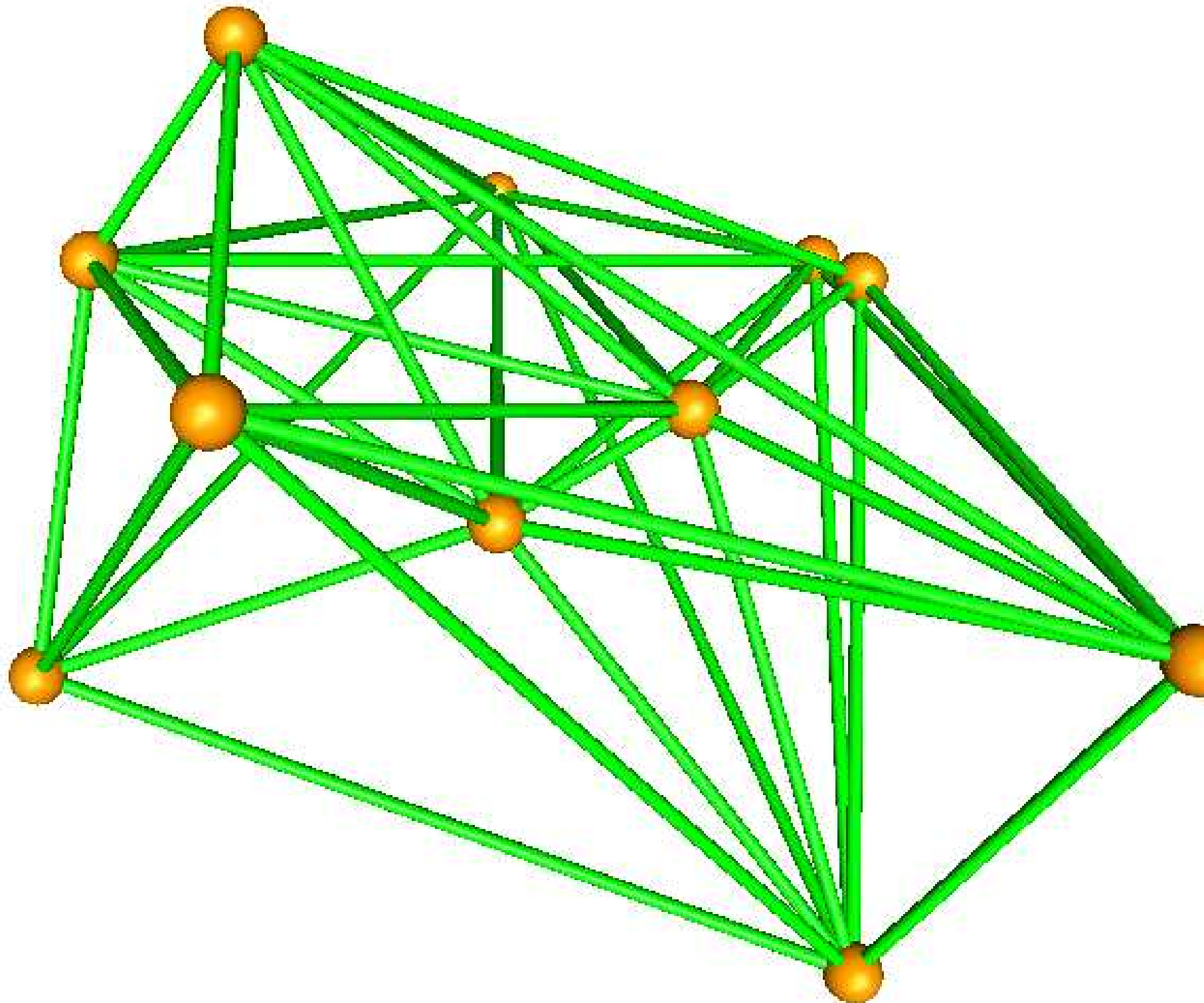


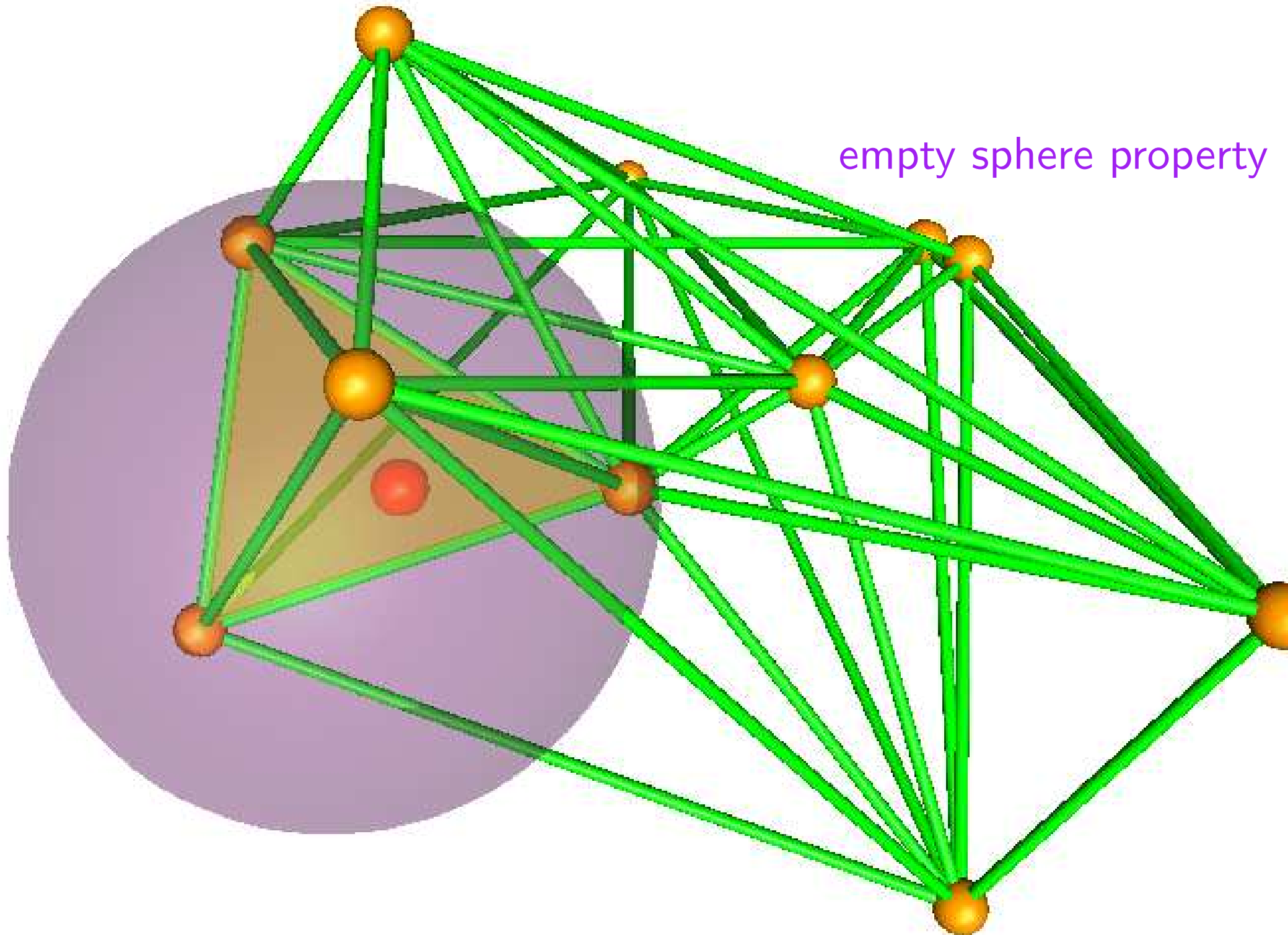
Everybody in this room know what Delaunay is!





empty circle property





# A (partial) history of Delaunay algorithms (and of computational geometry)



# A (partial) history of Delaunay algorithms (and of computational geometry)

Earlier algorithms [1970...]

# A (partial) history of Delaunay algorithms (and of computational geometry)

Earlier algorithms [1970...]

Incremental 2D/3D

Gift wrapping 2D

# A (partial) history of Delaunay algorithms (and of computational geometry)

Earlier algorithms [1970...]

Incremental 2D/3D

Gift wrapping 2D

Non optimal

"simple"

actually coded

linked to applications: meshes, reconstruction

# A (partial) history of Delaunay algorithms (and of computational geometry)

Earlier algorithms [1970...]

Worst case algorithms [1980...]

# A (partial) history of Delaunay algorithms (and of computational geometry)

Earlier algorithms [1970...]

Worst case algorithms [1980...]

Divide & conquer 2D

Plane sweep 2D

# A (partial) history of Delaunay algorithms (and of computational geometry)

Earlier algorithms [1970...]

Worst case algorithms [1980...]

Divide & conquer 2D

Plane sweep 2D

~~Non optimal~~

'simple'

actually coded

linked to applications: meshes, reconstruction

# A (partial) history of Delaunay algorithms (and of computational geometry)

Earlier algorithms [1970...]

Worst case algorithms [1980...]

Randomized algorithms [1990...]

# A (partial) history of Delaunay algorithms (and of computational geometry)

Earlier algorithms [1970...]

Worst case algorithms [1980...]

Randomized algorithms [1990...]

Delaunay tree

Clarkson & Shor

Dynamic updates (history graph)

Delaunay hierarchy

Spatial sorting (BRIO)



# A (partial) history of Delaunay algorithms (and of computational geometry)

Earlier algorithms [1970...]

Worst case algorithms [1980...]

Randomized algorithms [1990...]

Robustness issues [1995...]

# A (partial) history of Delaunay algorithms (and of computational geometry)

Earlier algorithms [1970...]

Worst case algorithms [1980...]

Randomized algorithms [1990...]

Robustness issues [1995...]

Properties checking

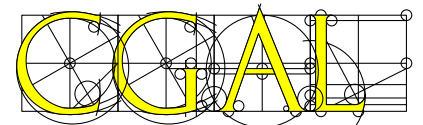
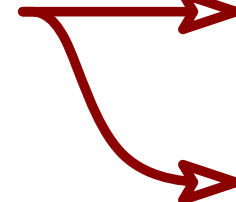


VRONI

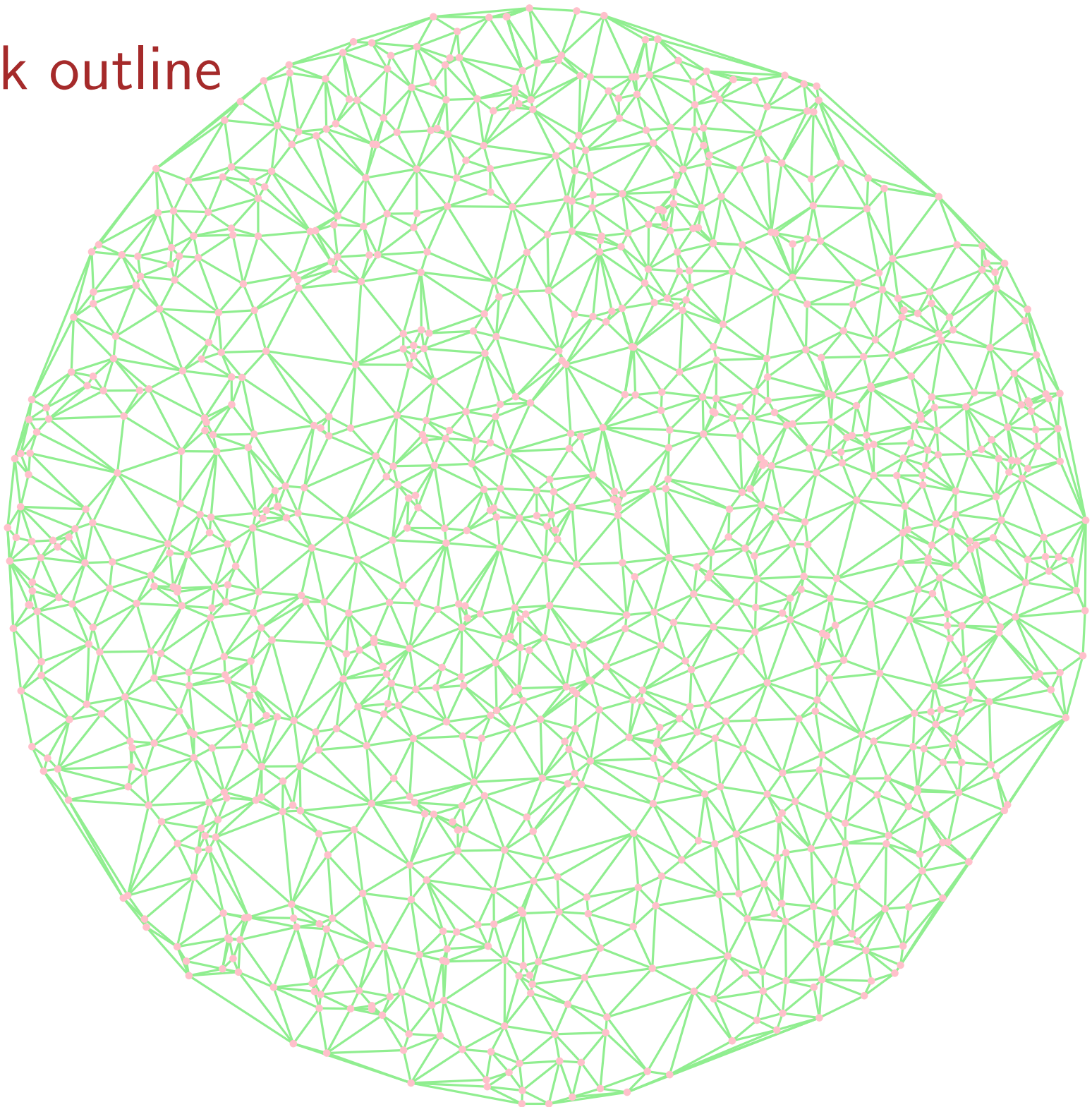
Exact computation paradigm



TRIANGLE



# Talk outline



# Talk outline

One word on robustness issues

Basic incremental algorithm

Locate by walk

Locate using randomized data structures

Vertex removal in 2D

Remarks on CGAL programming

Conclusion



## One word on robustness issues

Basic incremental algorithm

Locate by walk

Locate using randomized data structures

Vertex removal in 2D

Remarks on CGAL programming

Conclusion

# One word on robustness issues



# One word on robustness issues

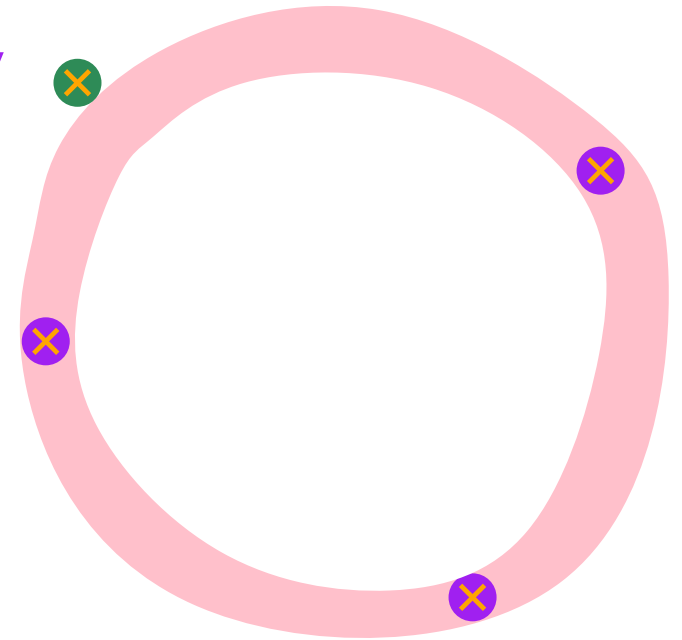


The exact computation paradigm



# One word on robustness issues

Compute predicates approximately

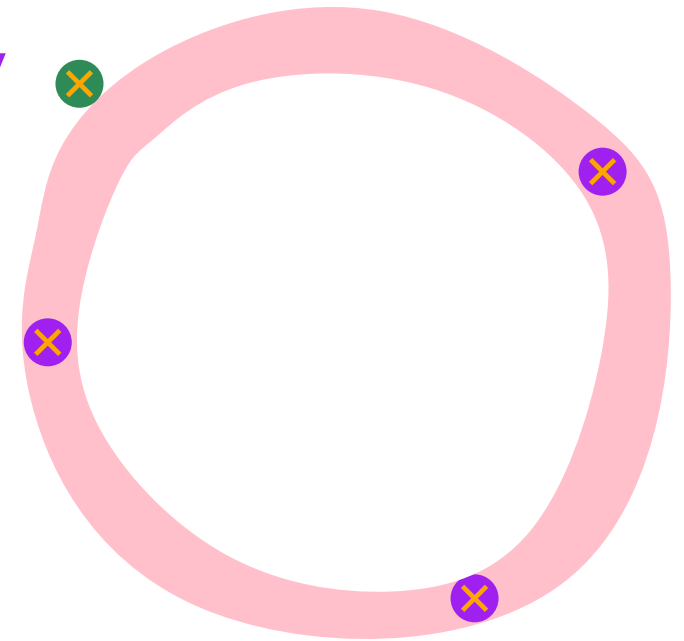




# One word on robustness issues

Compute predicates approximately

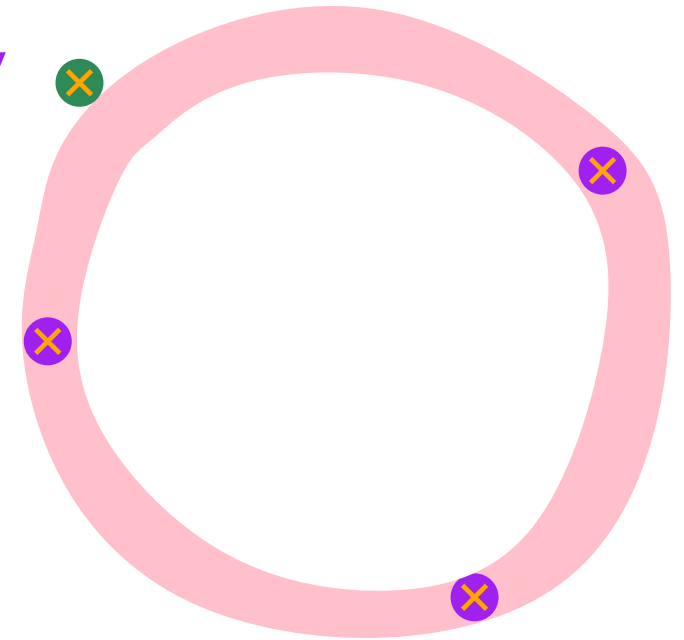
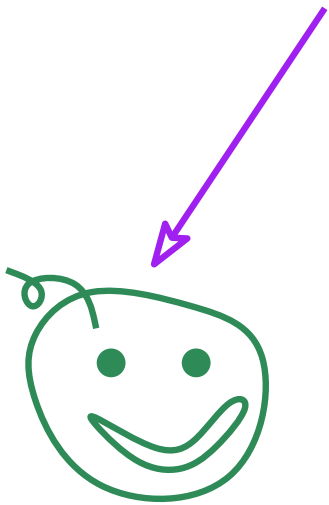
Certify result



# One word on robustness issues

Compute predicates approximately

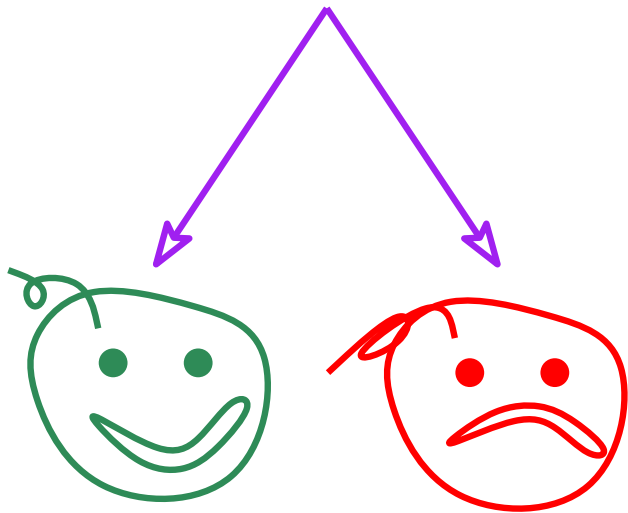
Certify result



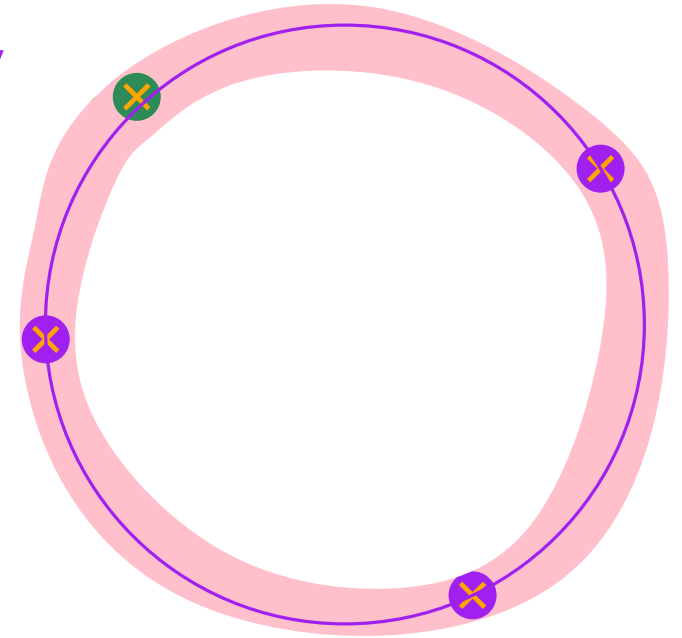
# One word on robustness issues

Compute predicates approximately

Certify result



Run exact computation

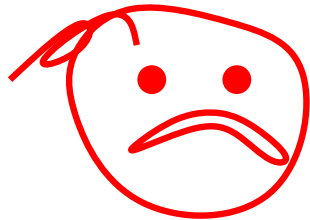


# One word on robustness issues

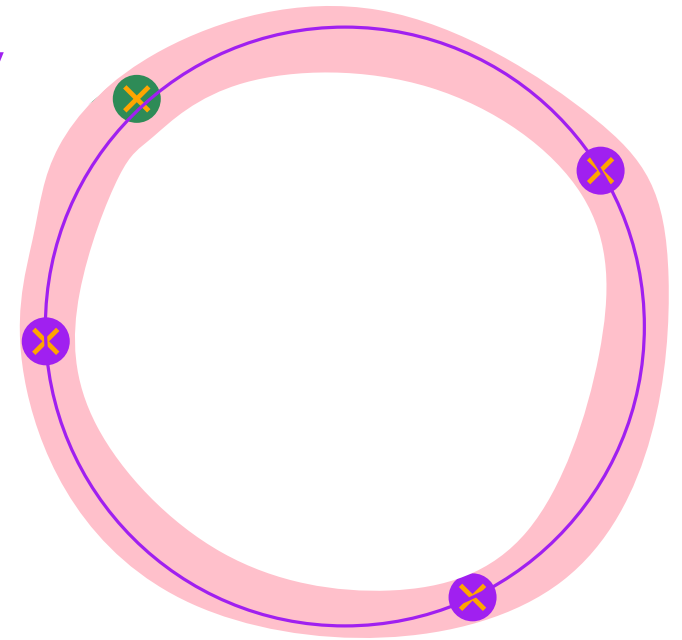
Compute predicates approximately

Certify result

Extra cost



Run exact computation

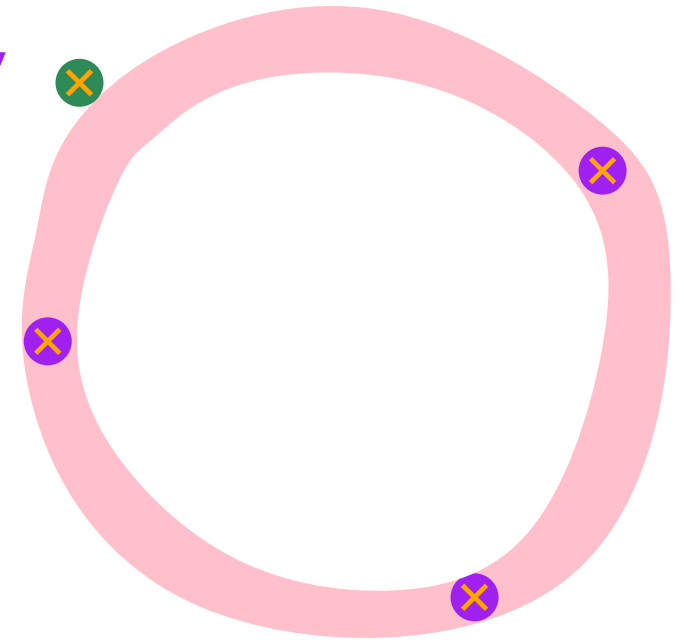


# One word on robustness issues

Compute predicates approximately

Certify result

Extra cost



Delaunay 2D 10M points

`Exact_predicates_inexact_constructions_kernel`

10.6 seconds

`Cartesian<double>`

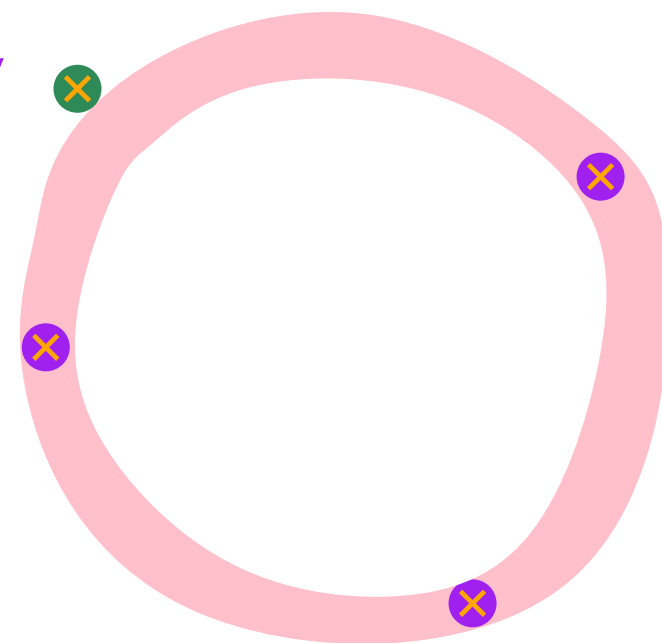
9.7 seconds

# One word on robustness issues

Compute predicates approximately

Certify result

Extra cost



Delaunay 2D 10M points

3D

`Exact_predicates_inexact_constructions_kernel`

10.6 seconds

82 seconds

`Cartesian<double>`

9.7 seconds

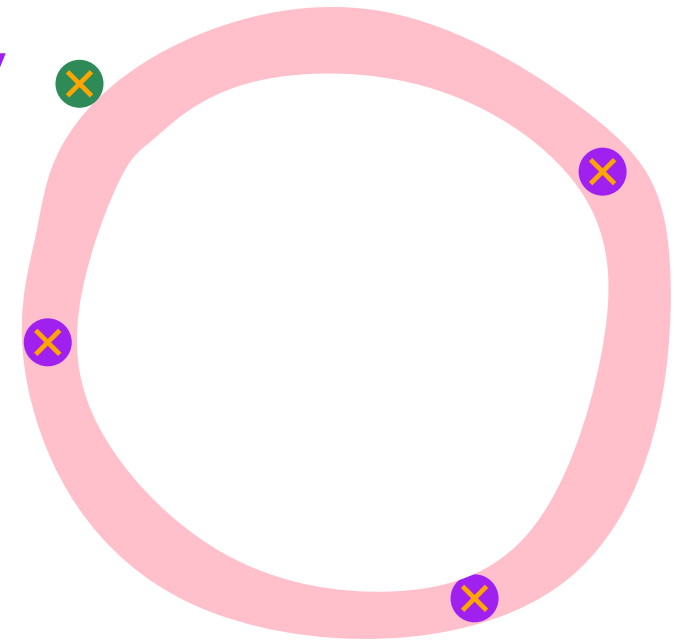
75 seconds

# One word on robustness issues

Compute predicates approximately

Certify result

Extra cost



Delaunay 2D 10M points

3D

`Exact_predicates_inexact_constructions_kernel`

10.6 seconds


82 seconds

`Cartesian<double>`

may loop (or crash)

## All benchmarks

2.3 GHz, 16 GByte workstation

 3.9 (Release mode)

Exact\_predicates\_inexact\_constructions\_kernel

Exact\_predicates\_inexact\_constructions\_kernel

10.6 seconds      82 seconds

Cartesian<double>

9.7 seconds      75 seconds





One word on robustness issues

**Basic incremental algorithm**

Locate by walk

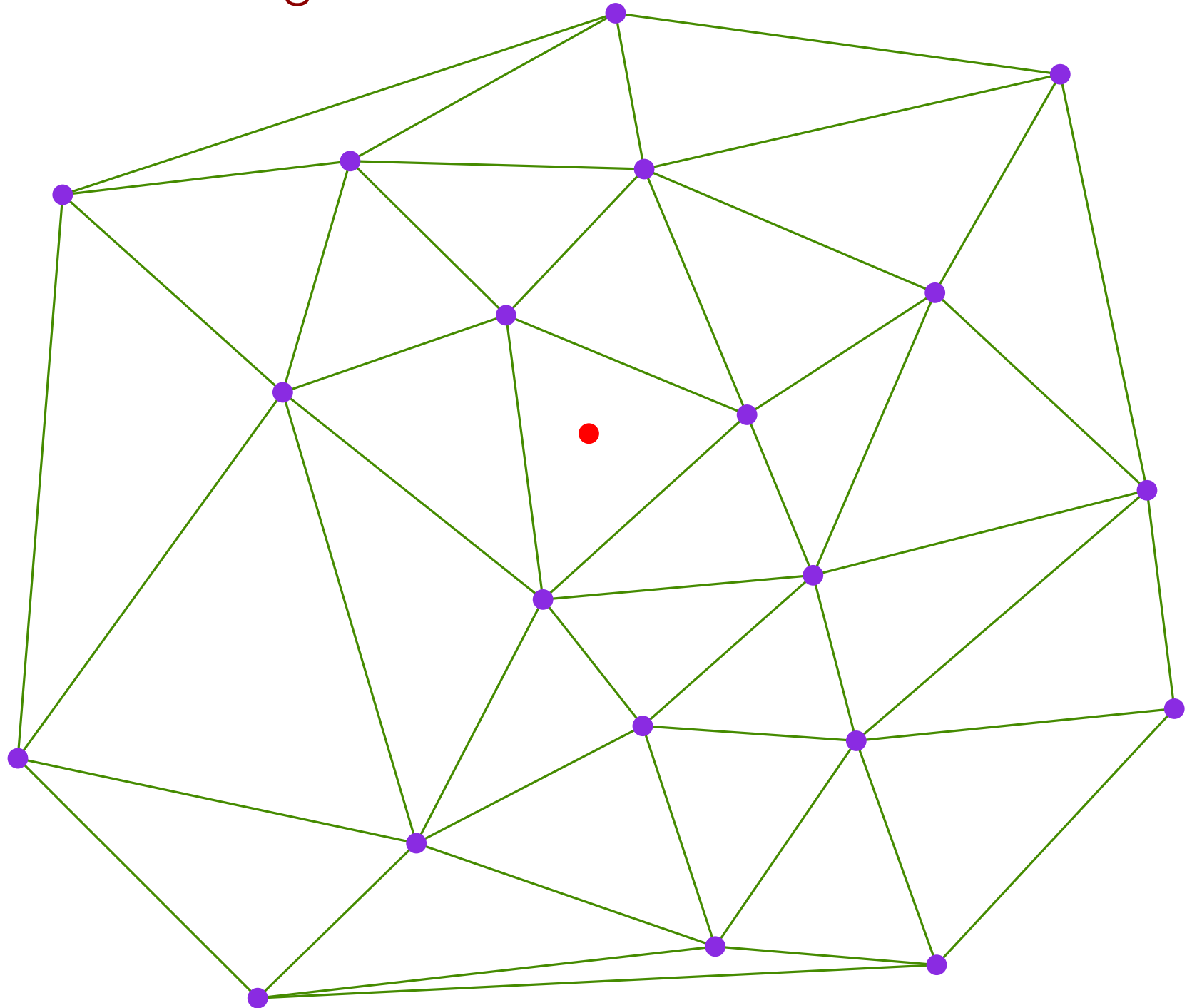
Locate using randomized data structures

Vertex removal in 2D

Remarks on CGAL programming

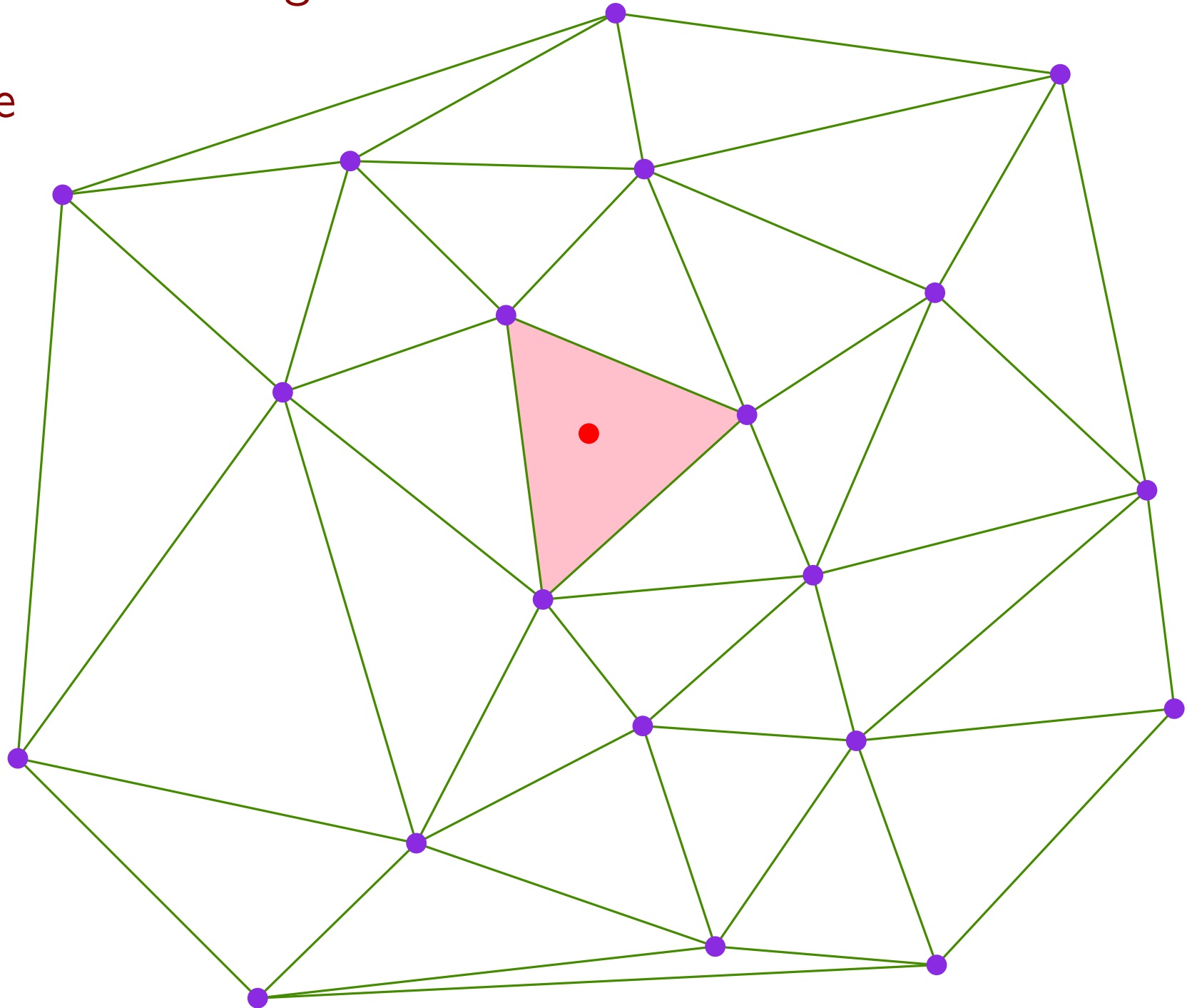
Conclusion

# Basic incremental algorithm



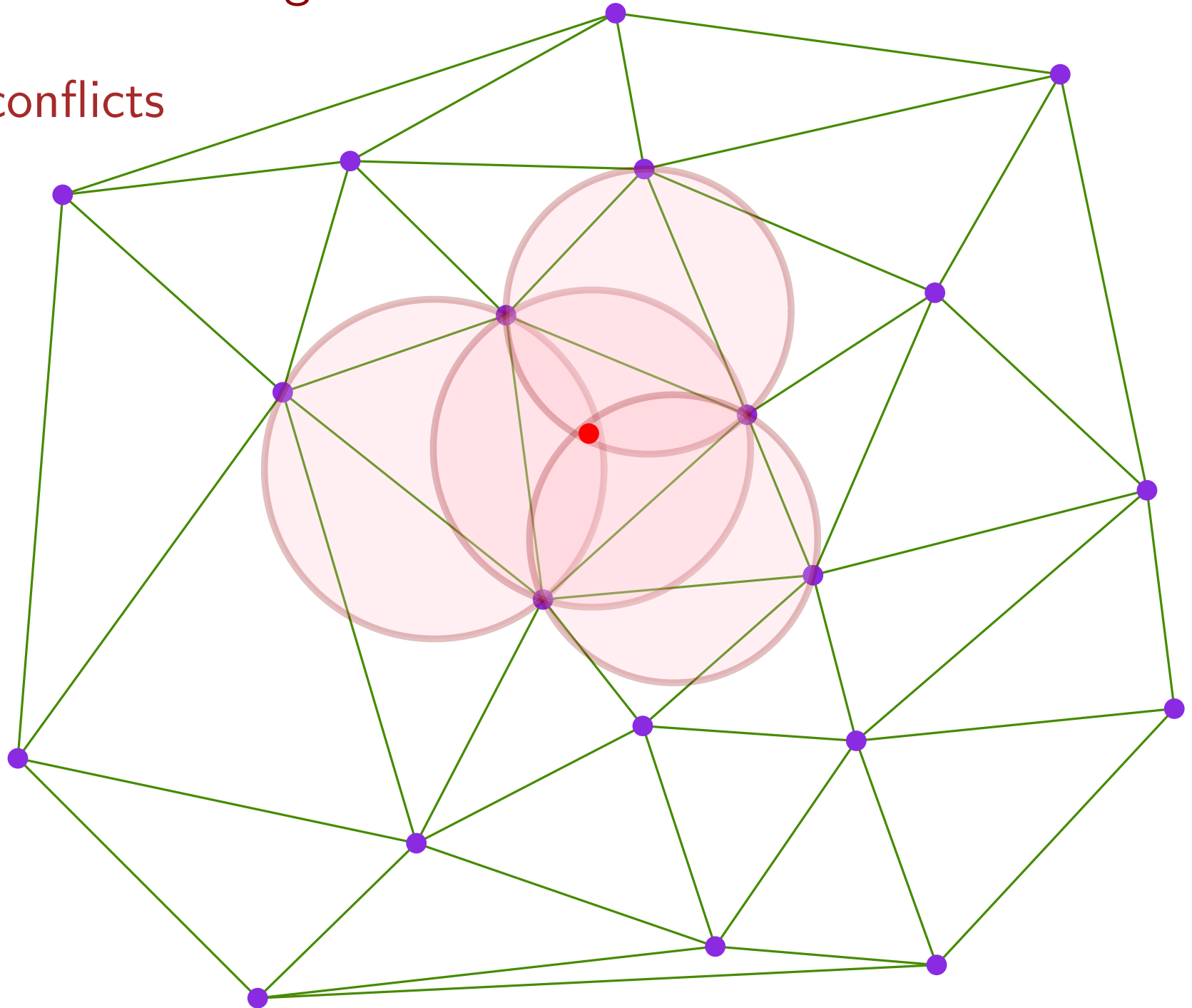
# Basic incremental algorithm

Locate



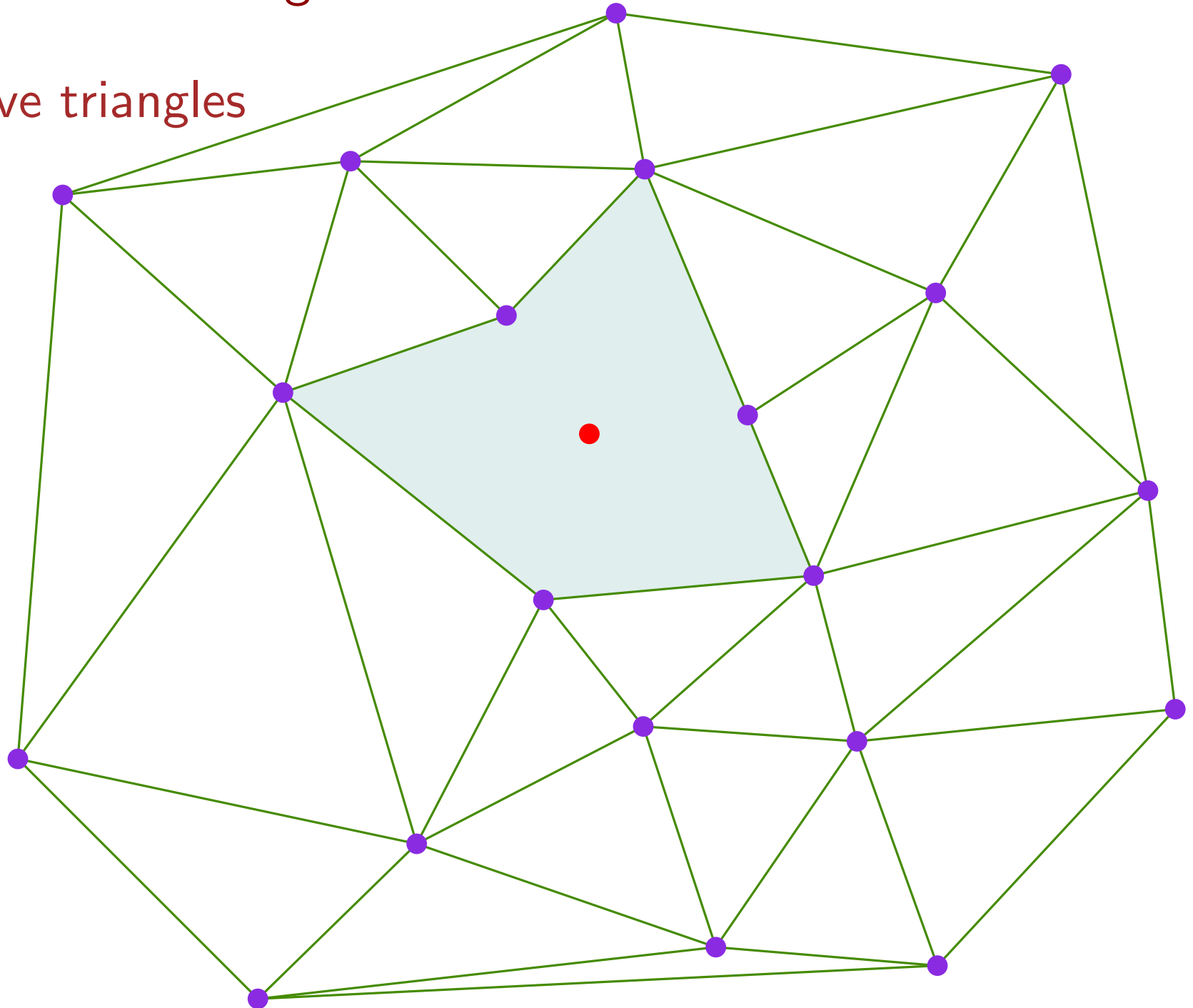
# Basic incremental algorithm

Find conflicts



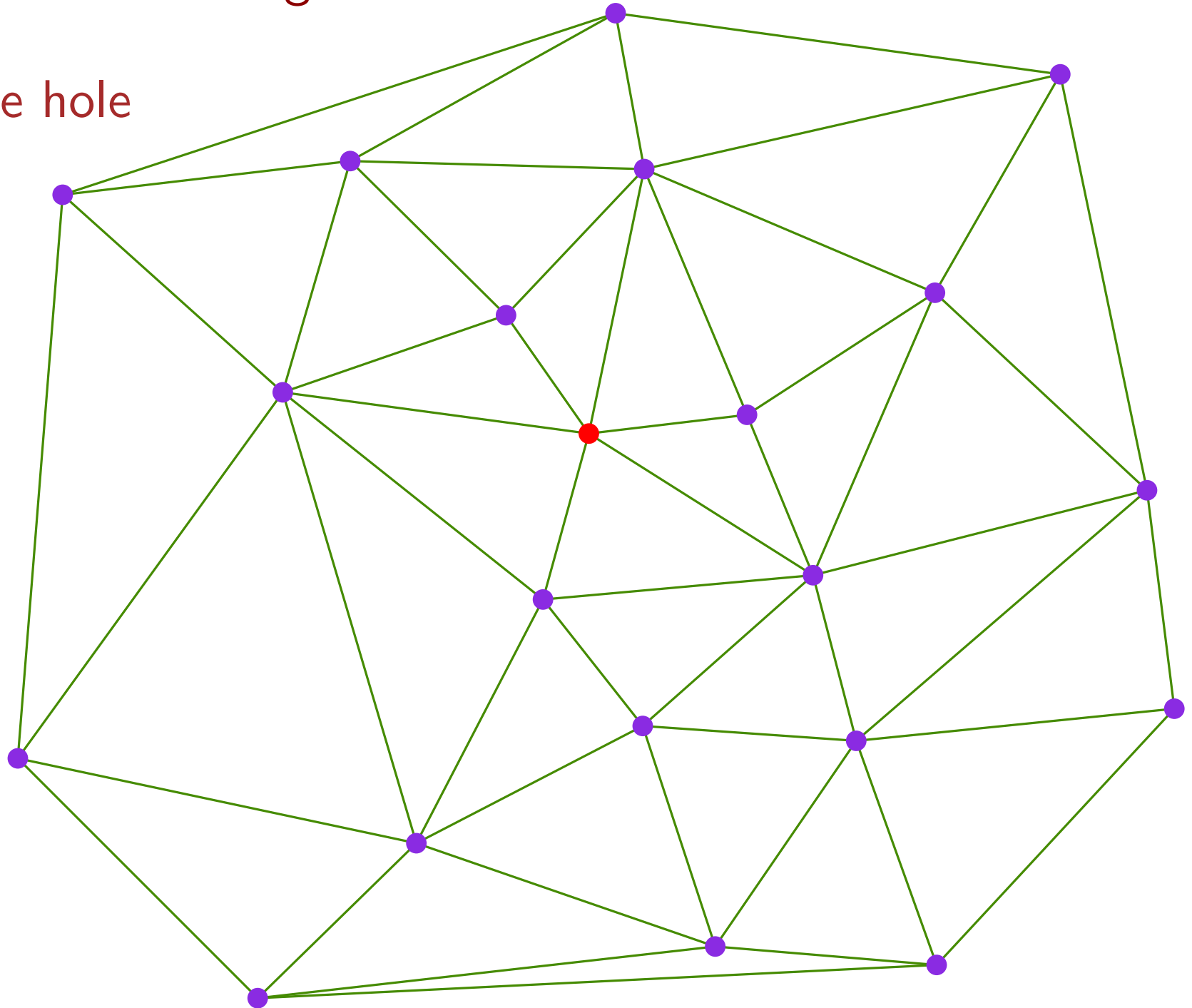
# Basic incremental algorithm

Remove triangles



# Basic incremental algorithm

Fill the hole





One word on robustness issues  
Basic incremental algorithm

**Locate by walk**

Straight walk

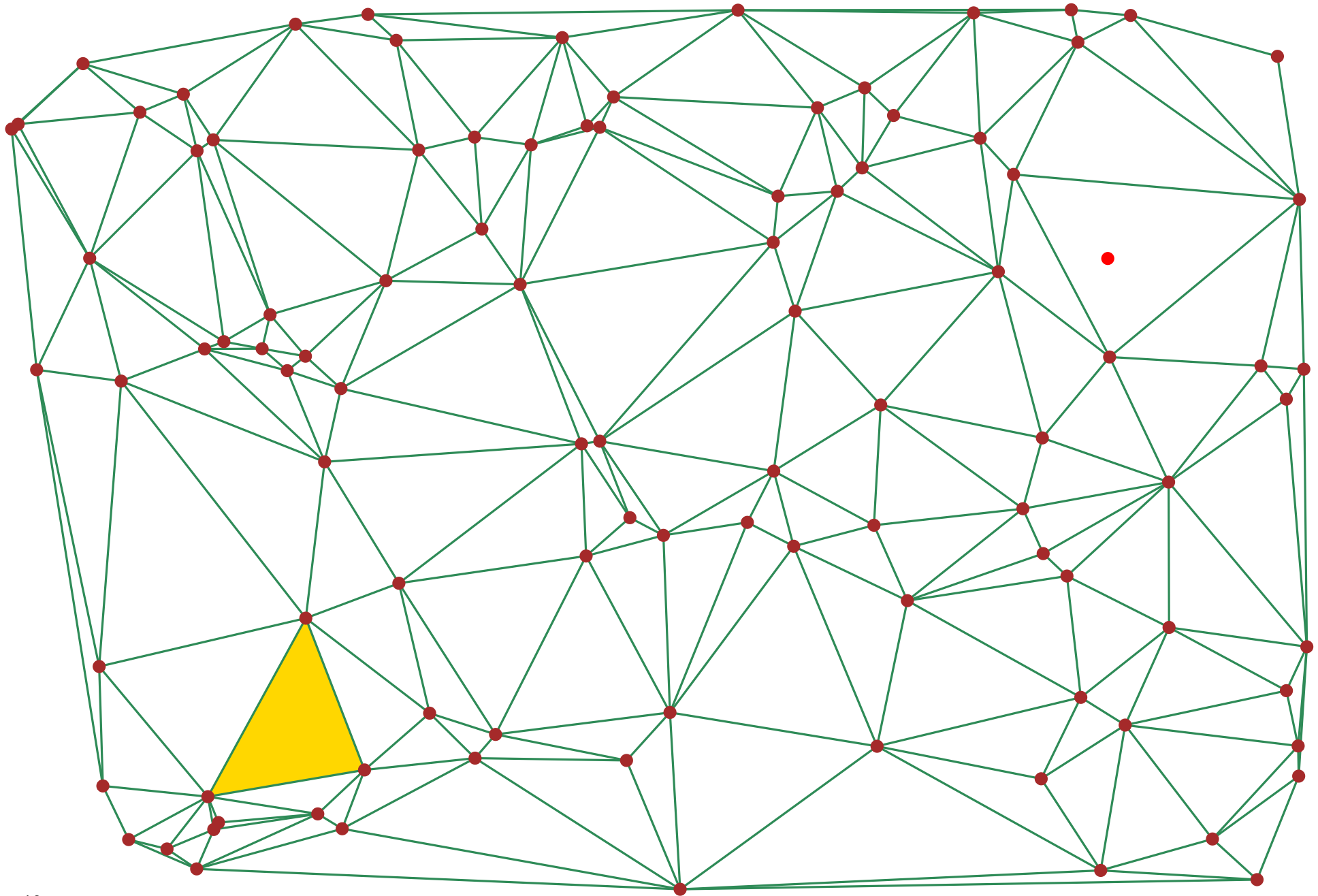
Locate using randomized data structures

Vertex removal in 2D

Remarks on CGAL programming

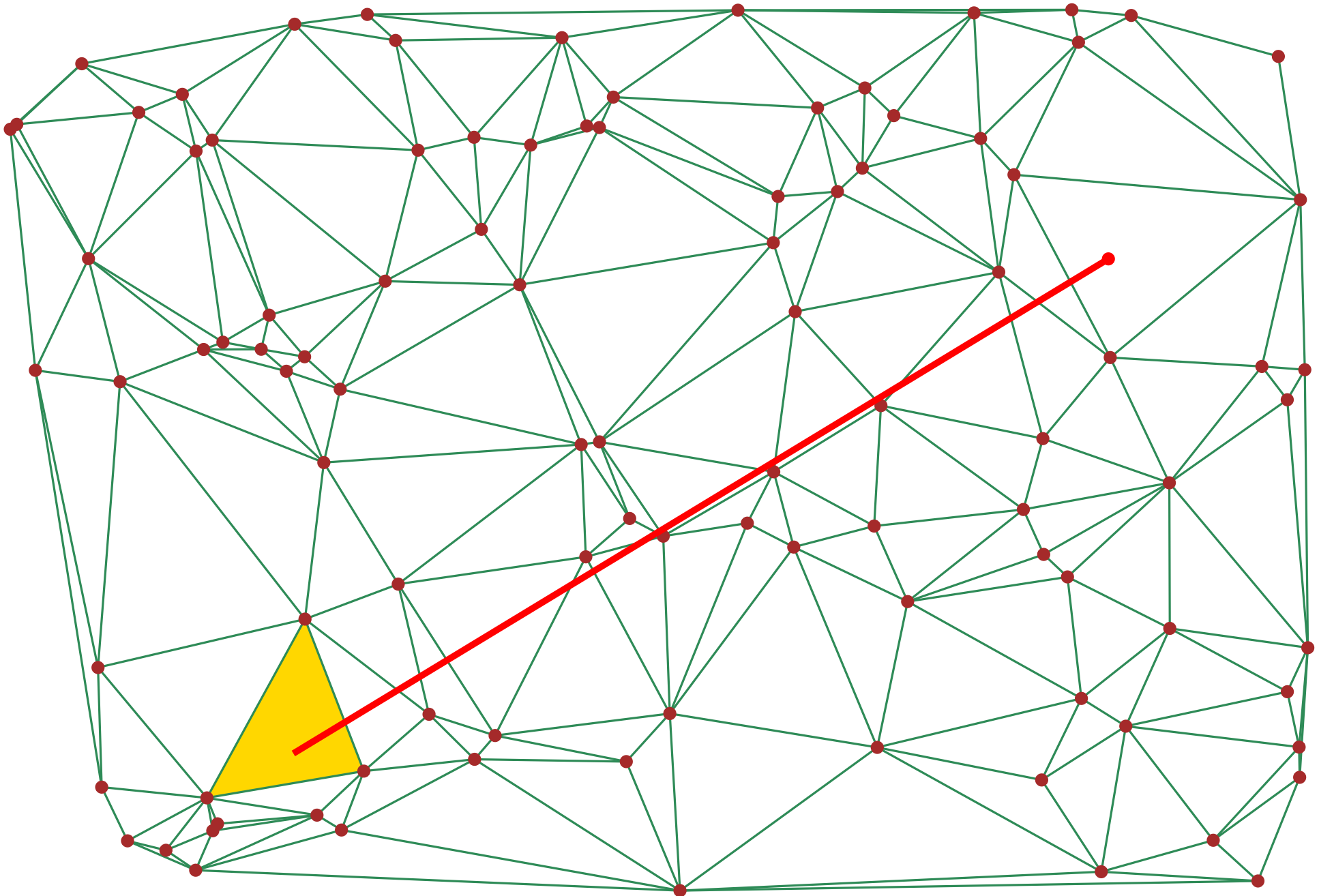
Conclusion

# Locate by walk - straight walk

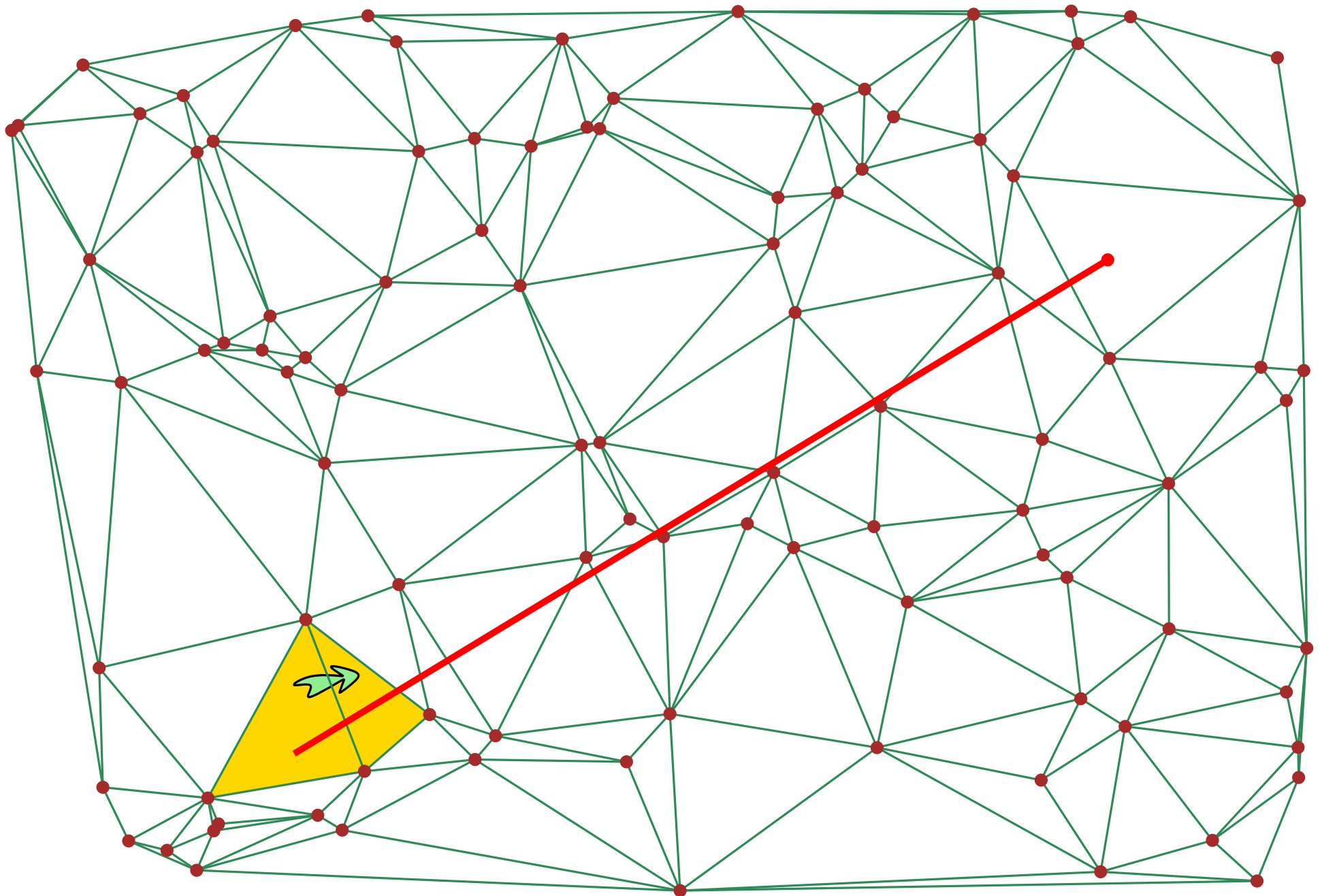




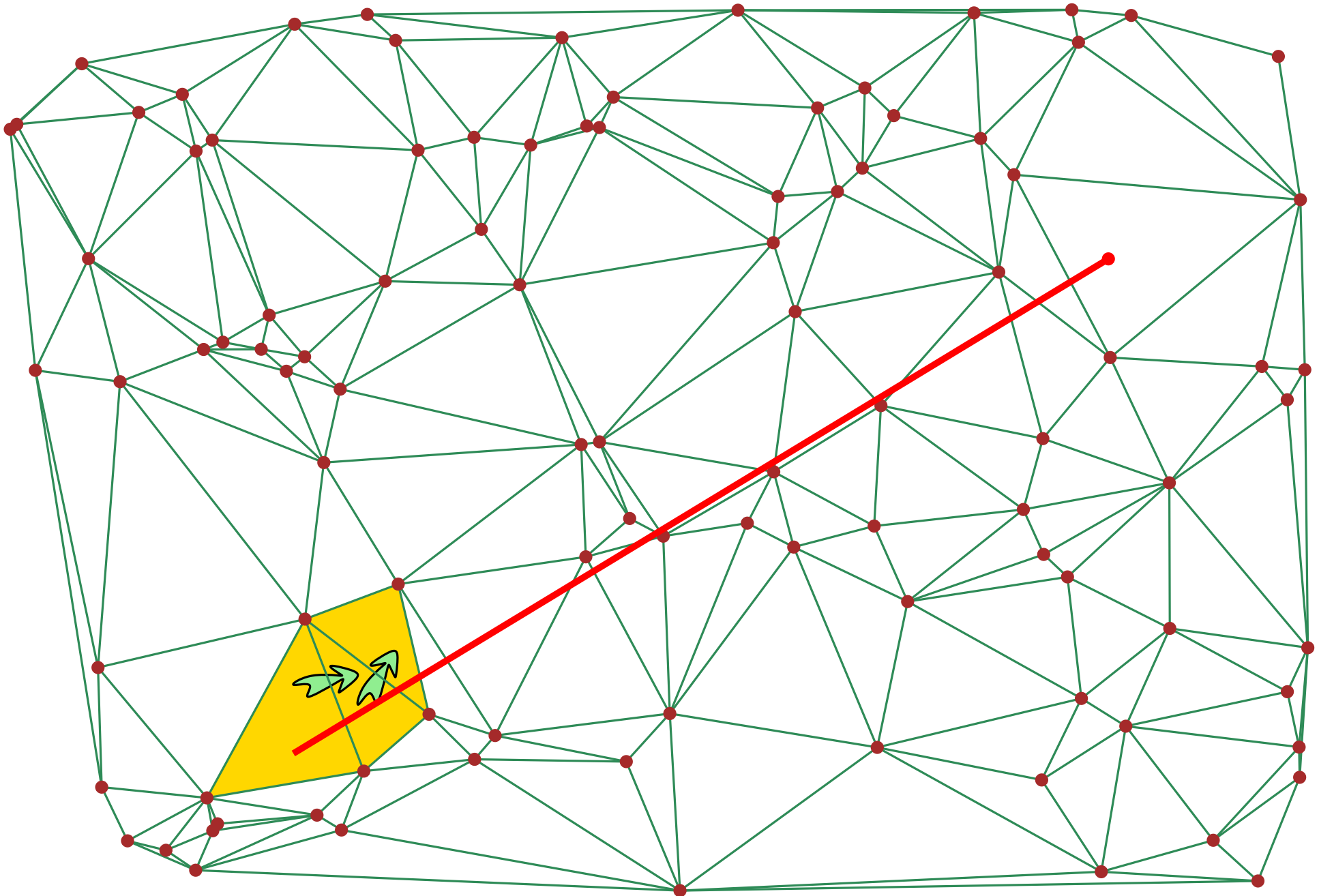
# Locate by walk - straight walk



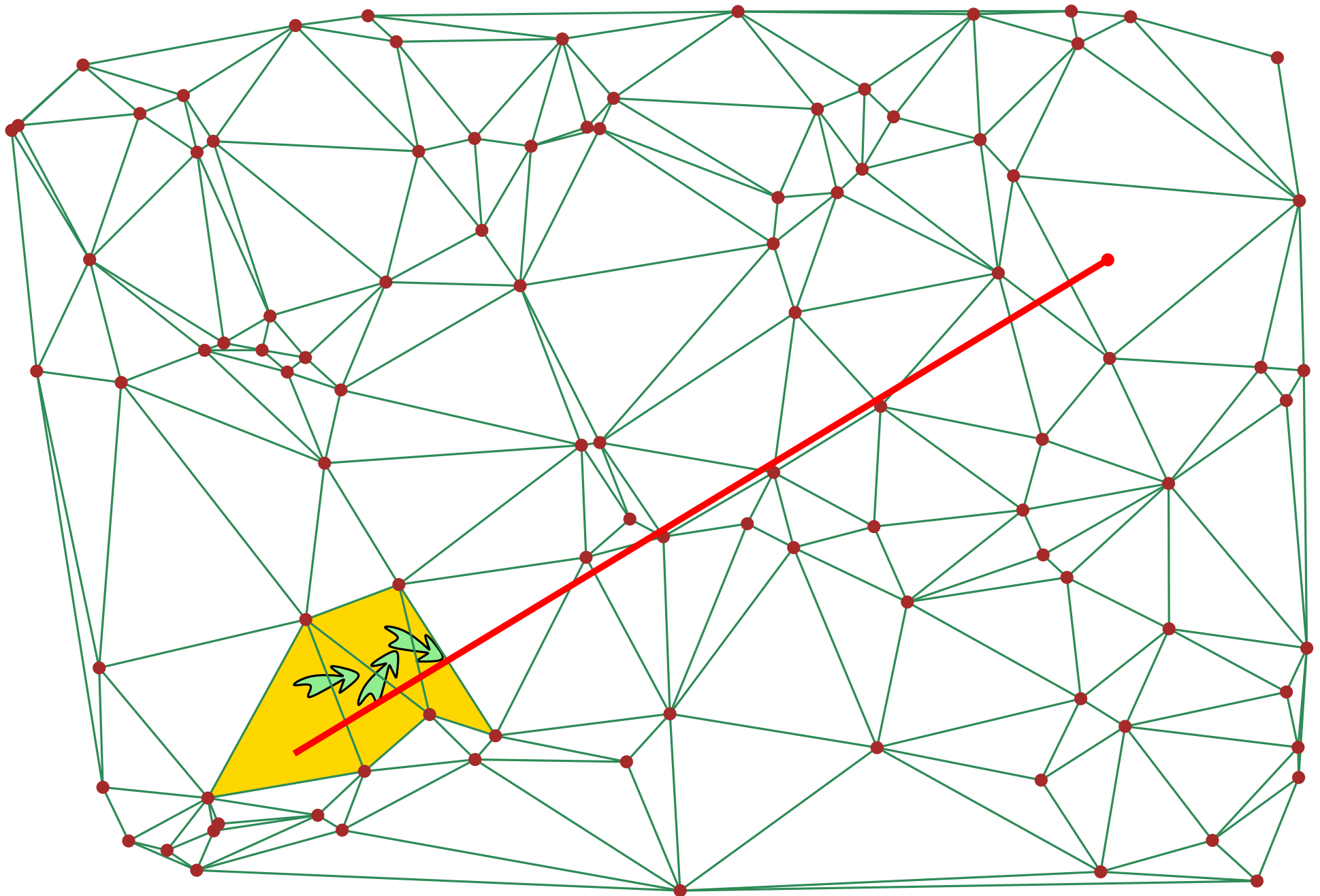
# Locate by walk - straight walk



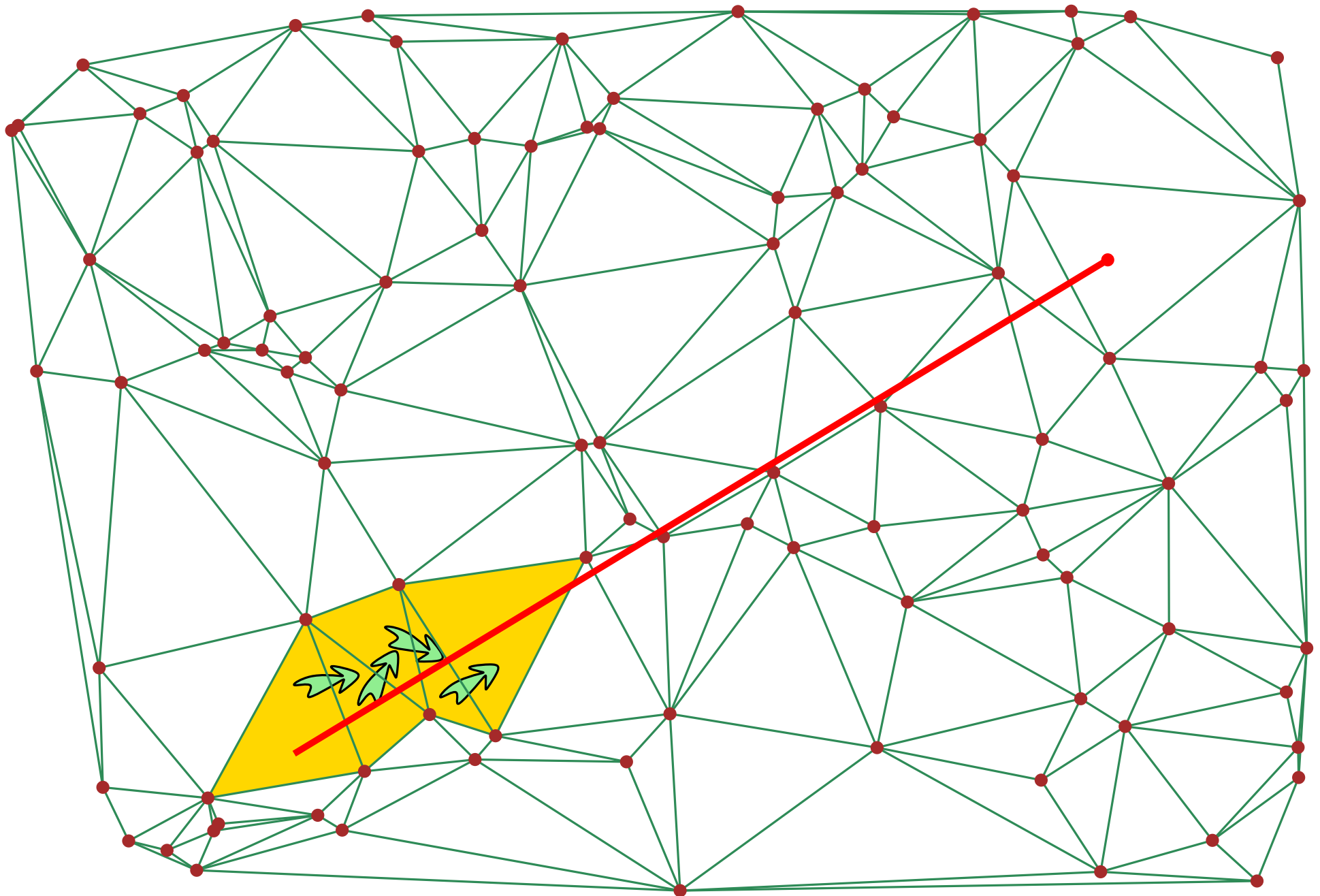
# Locate by walk - straight walk



# Locate by walk - straight walk

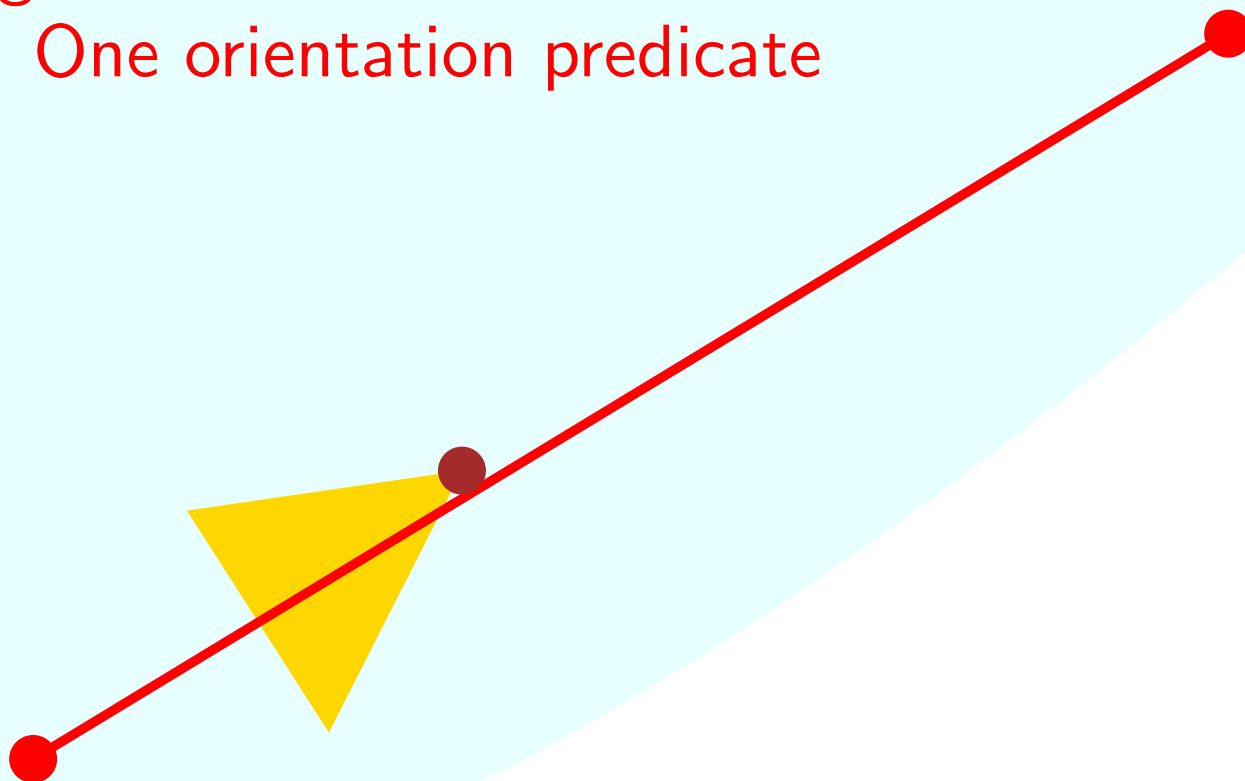


# Locate by walk - straight walk

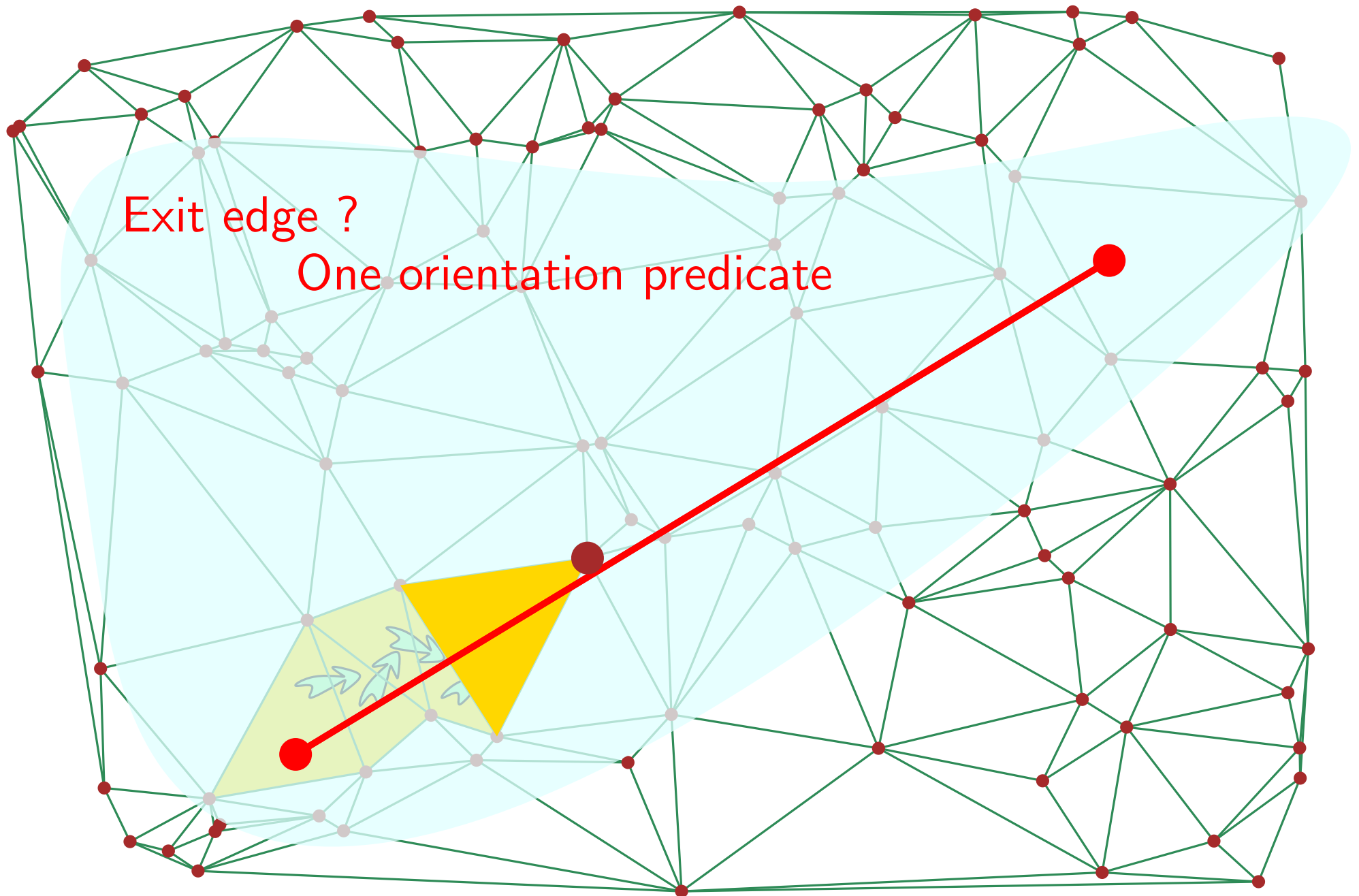


Exit edge ?

One orientation predicate

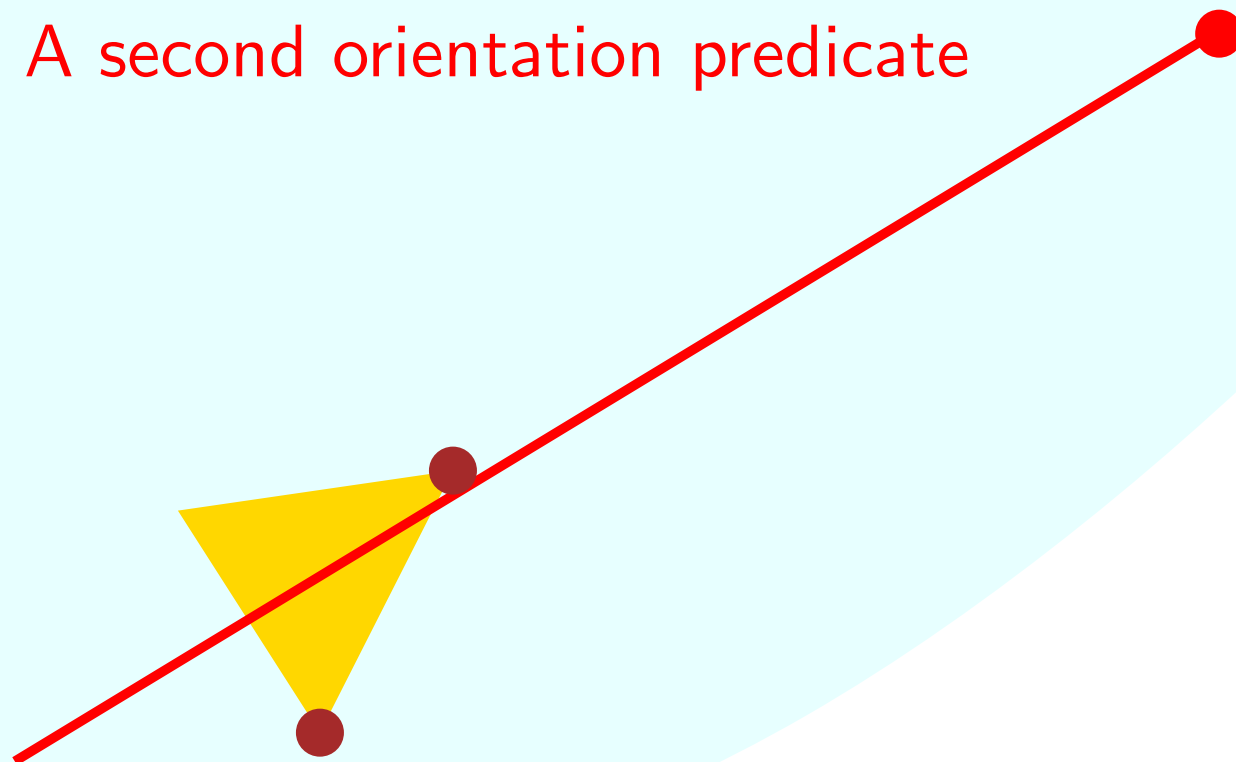


# Locate by walk - straight walk



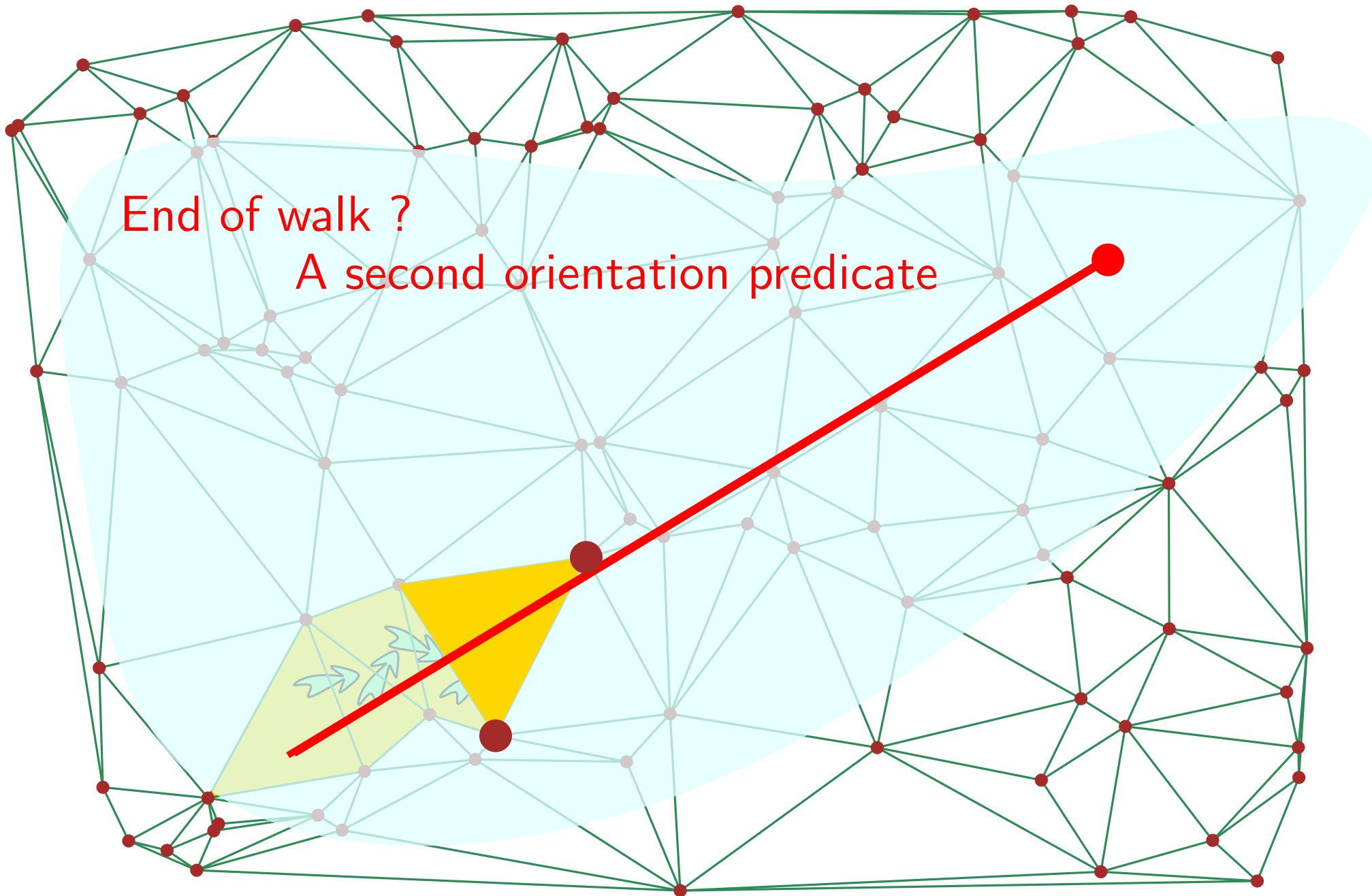
End of walk ?

A second orientation predicate





# Locate by walk - straight walk





One word on robustness issues  
Basic incremental algorithm

**Locate by walk**

Straight walk  
Visibility walk

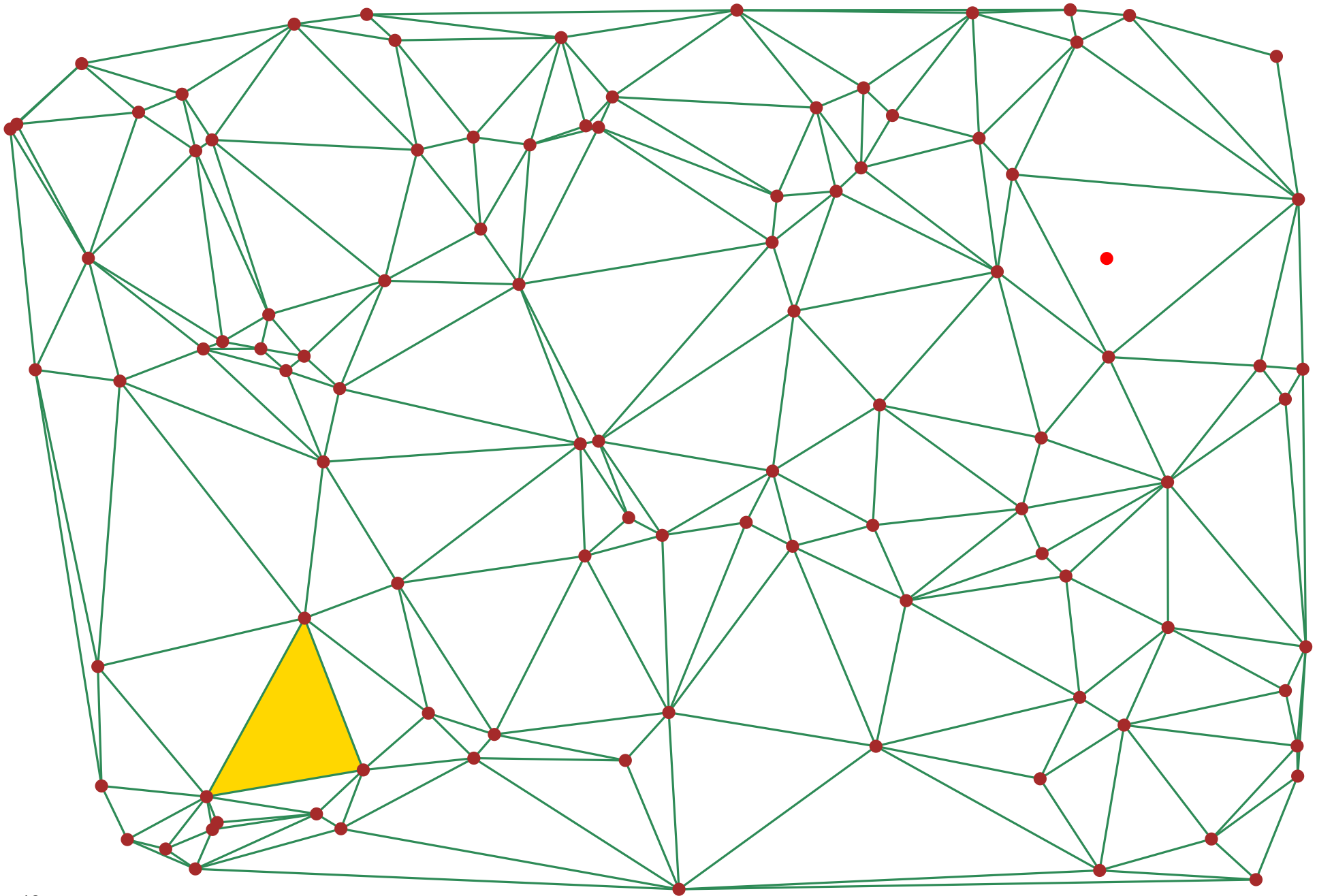
Locate using randomized data structures

Vertex removal in 2D

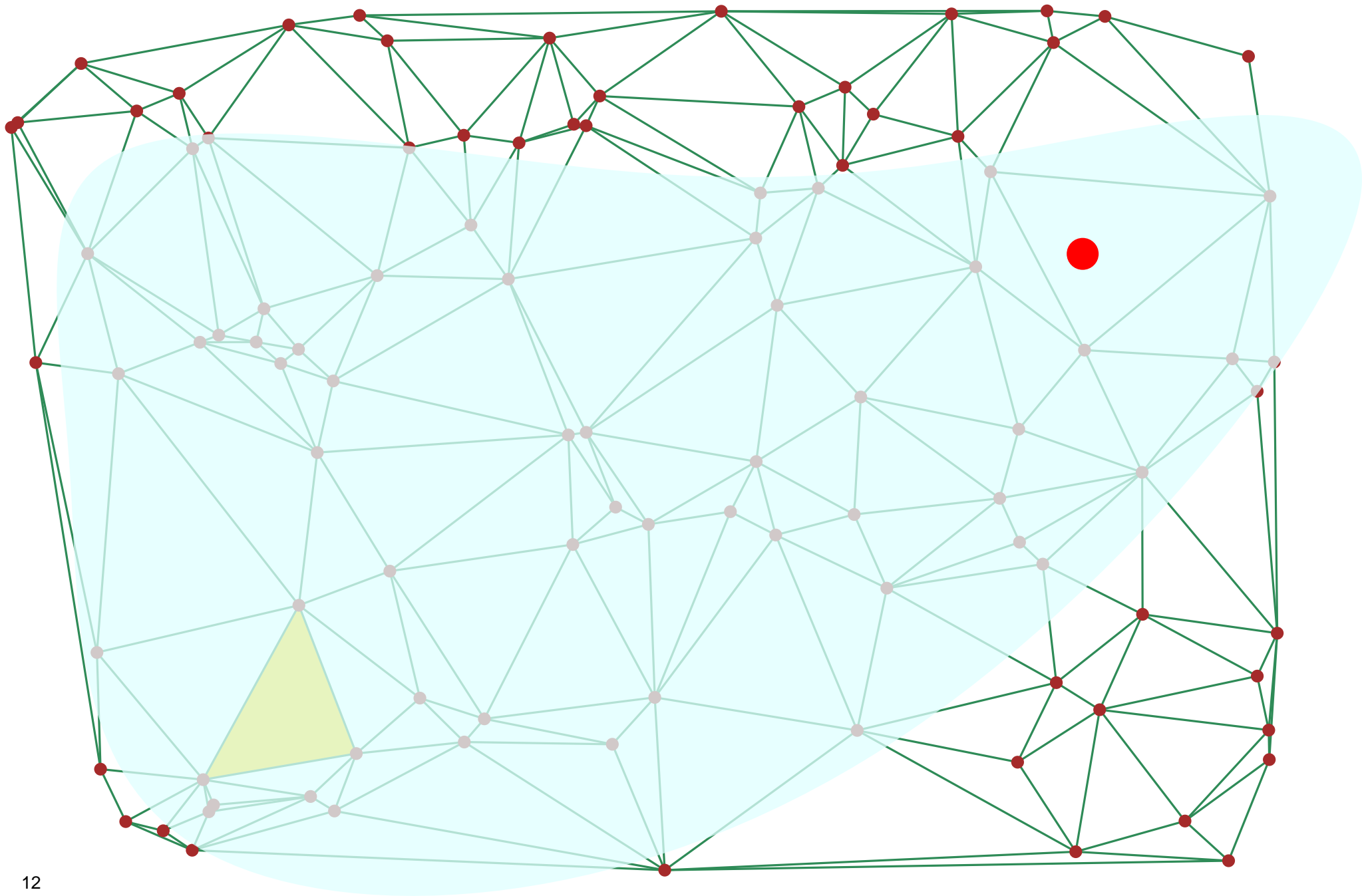
Remarks on CGAL programming

Conclusion

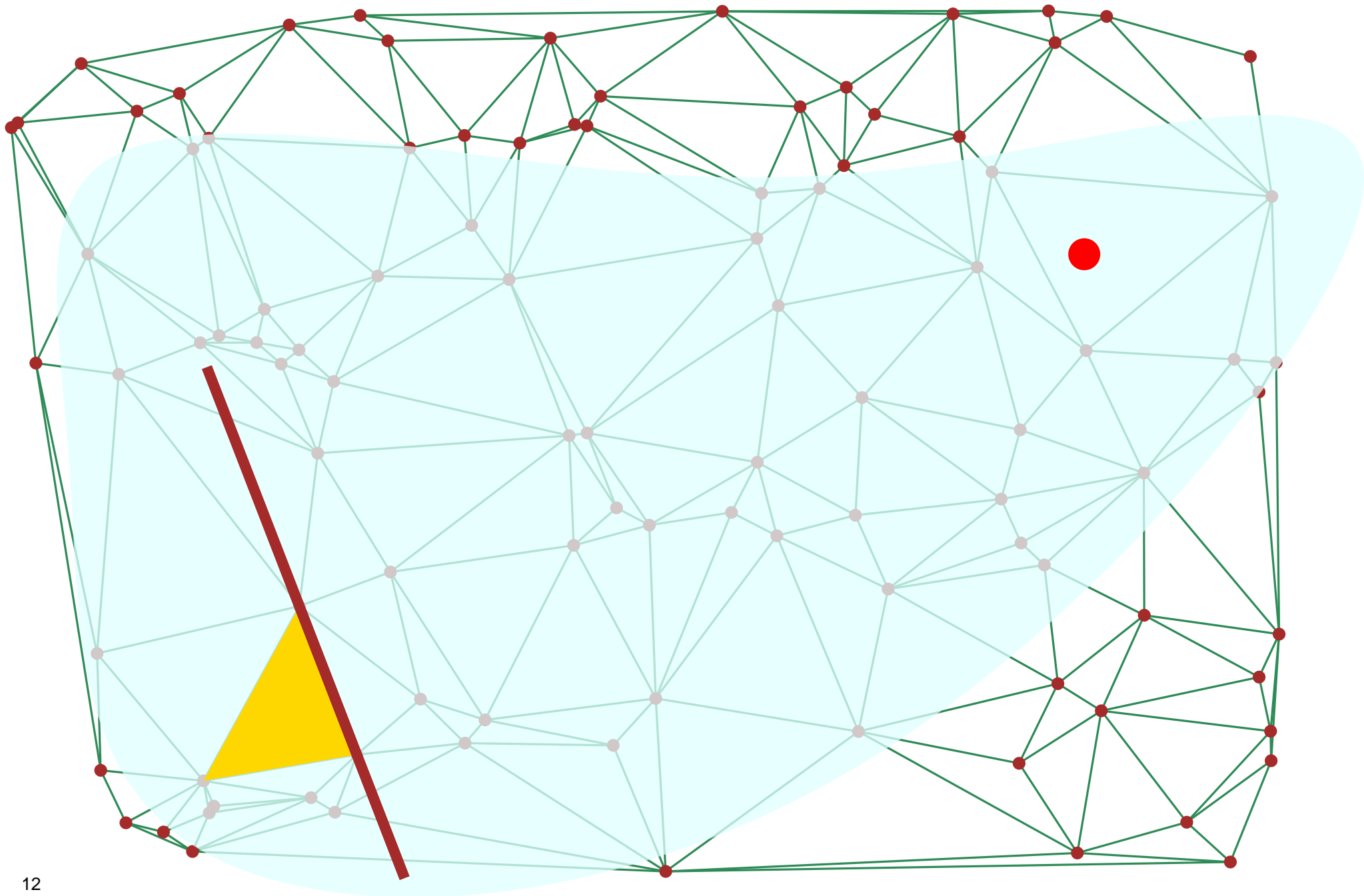
# Locate by walk - visibility walk



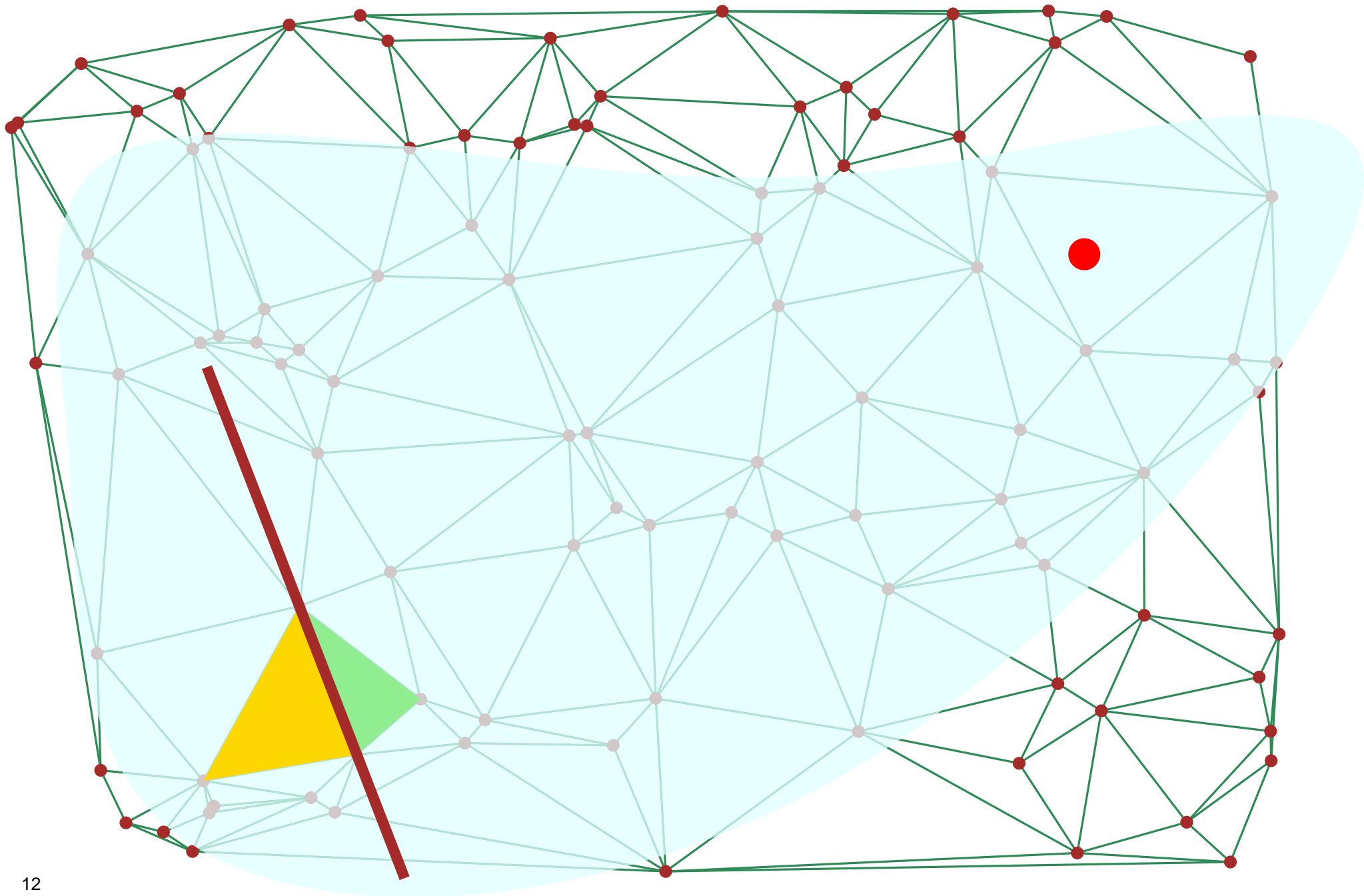
# Locate by walk - visibility walk



# Locate by walk - visibility walk

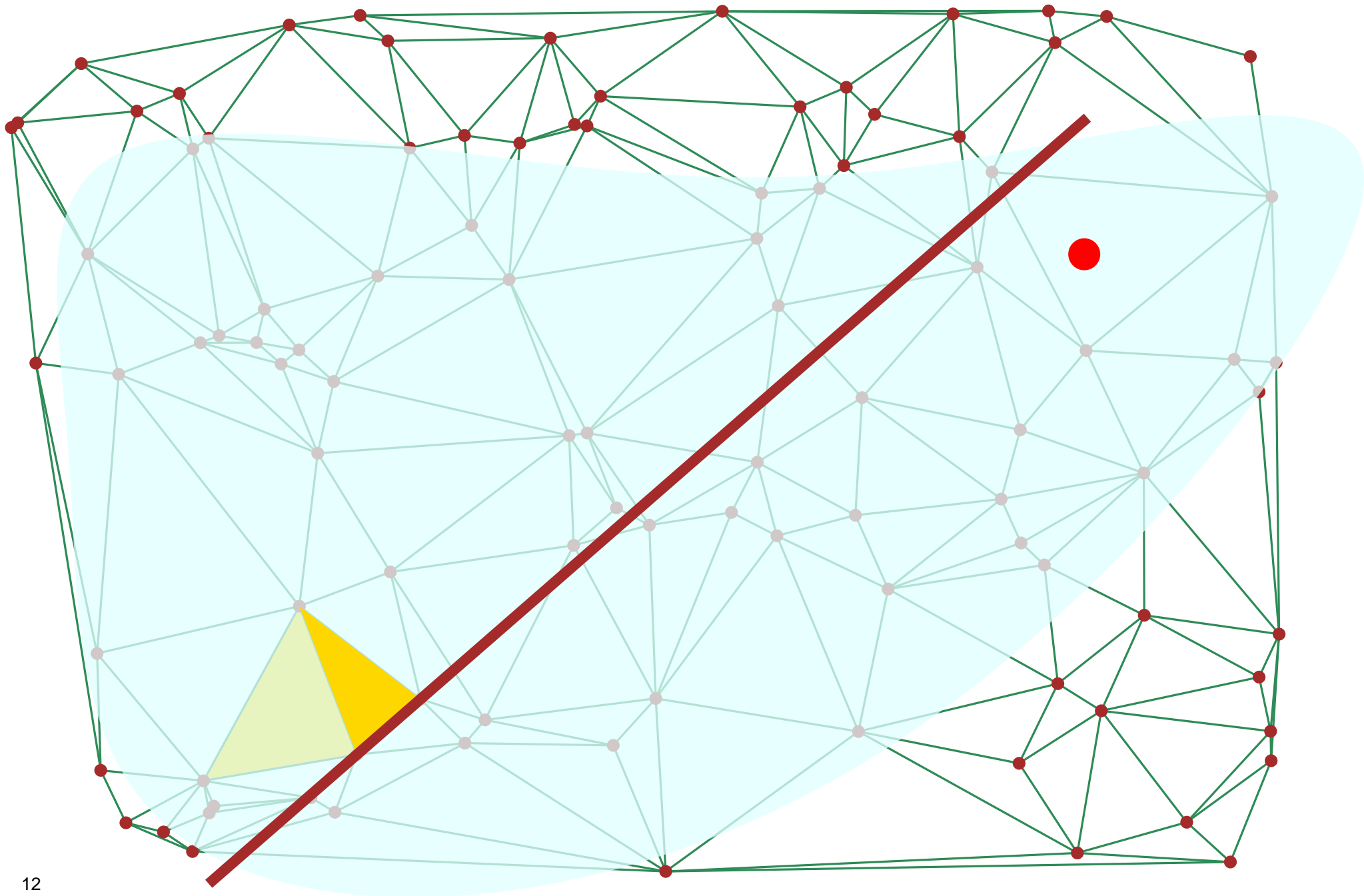


# Locate by walk - visibility walk

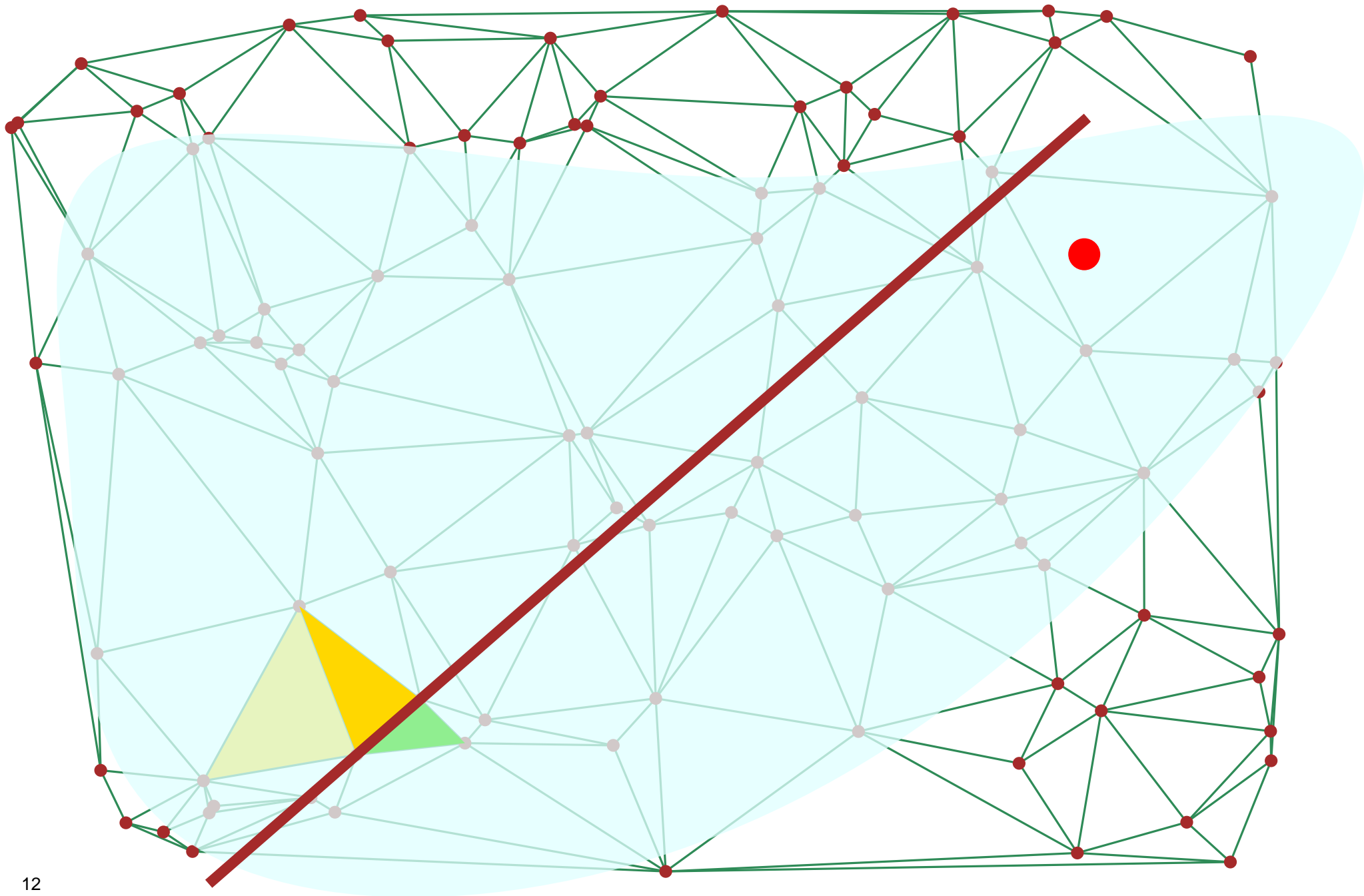




# Locate by walk - visibility walk

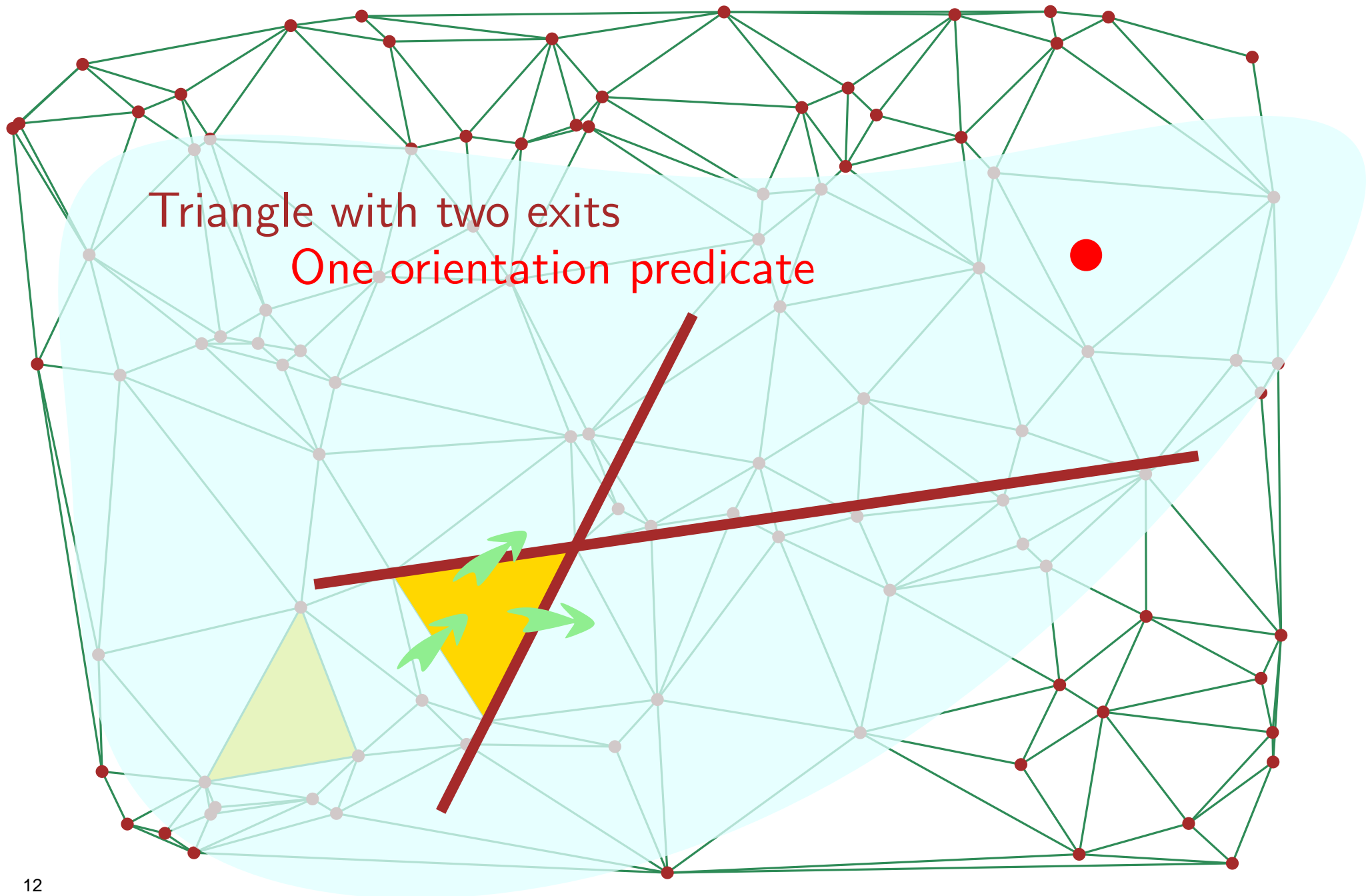


# Locate by walk - visibility walk

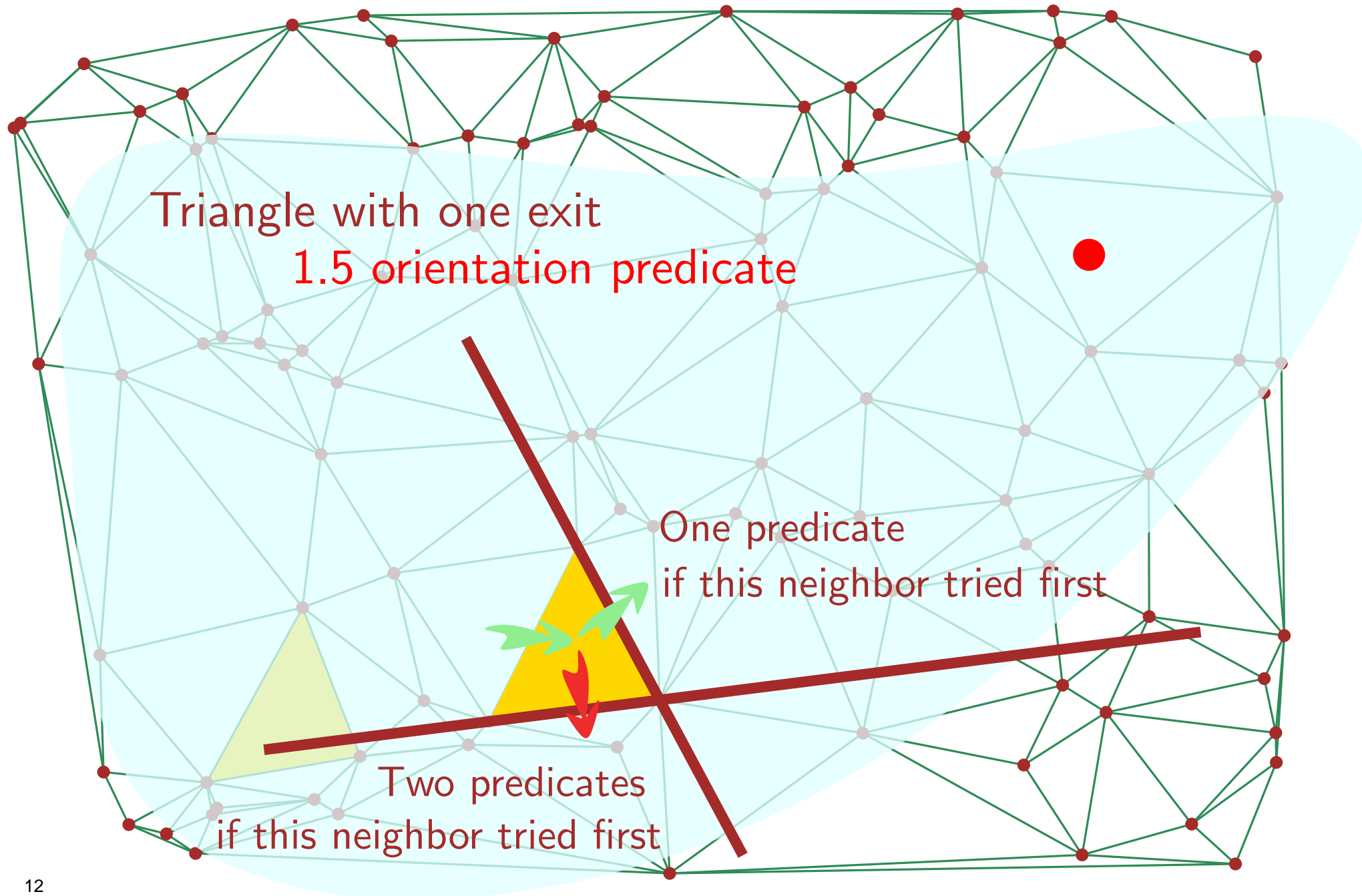




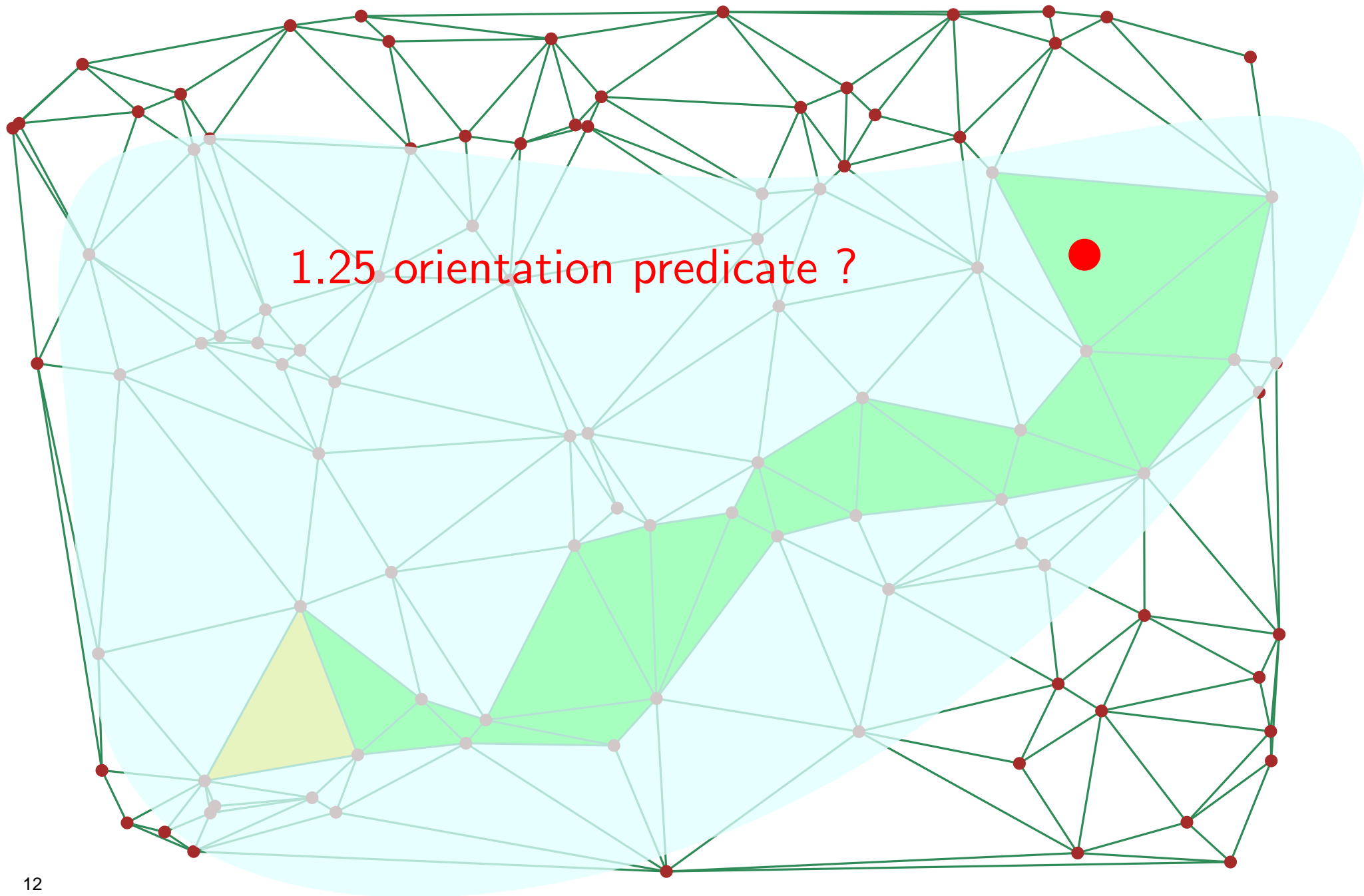
# Locate by walk - visibility walk



# Locate by walk - visibility walk



# Locate by walk - visibility walk



Locate by walk

Visibility vs straight walk

# Locate by walk

Visibility vs straight walk

2D and 3D

less predicates per crossed edge

similar number of crossed edges

experimental / theoretical

# Locate by walk

Visibility vs straight walk

Speed improvement ?

# Locate by walk

Visibility vs straight walk

Speed improvement?

Walk in Delaunay 1 Mpoints

Straight: 324  $\mu s$

Visibility: 285  $\mu s$

3D: 97  $\mu s$

# Locate by walk

Visibility vs straight walk

Speed improvement?

Walk in Delaunay 1 Mpoints

Straight: 324  $\mu$ s

Visibility: 285  $\mu$ s

3D: 97  $\mu$ s

**Much easier to code**





One word on robustness issues  
Basic incremental algorithm

## Locate by walk

Straight walk  
Visibility walk  
Structural filtering

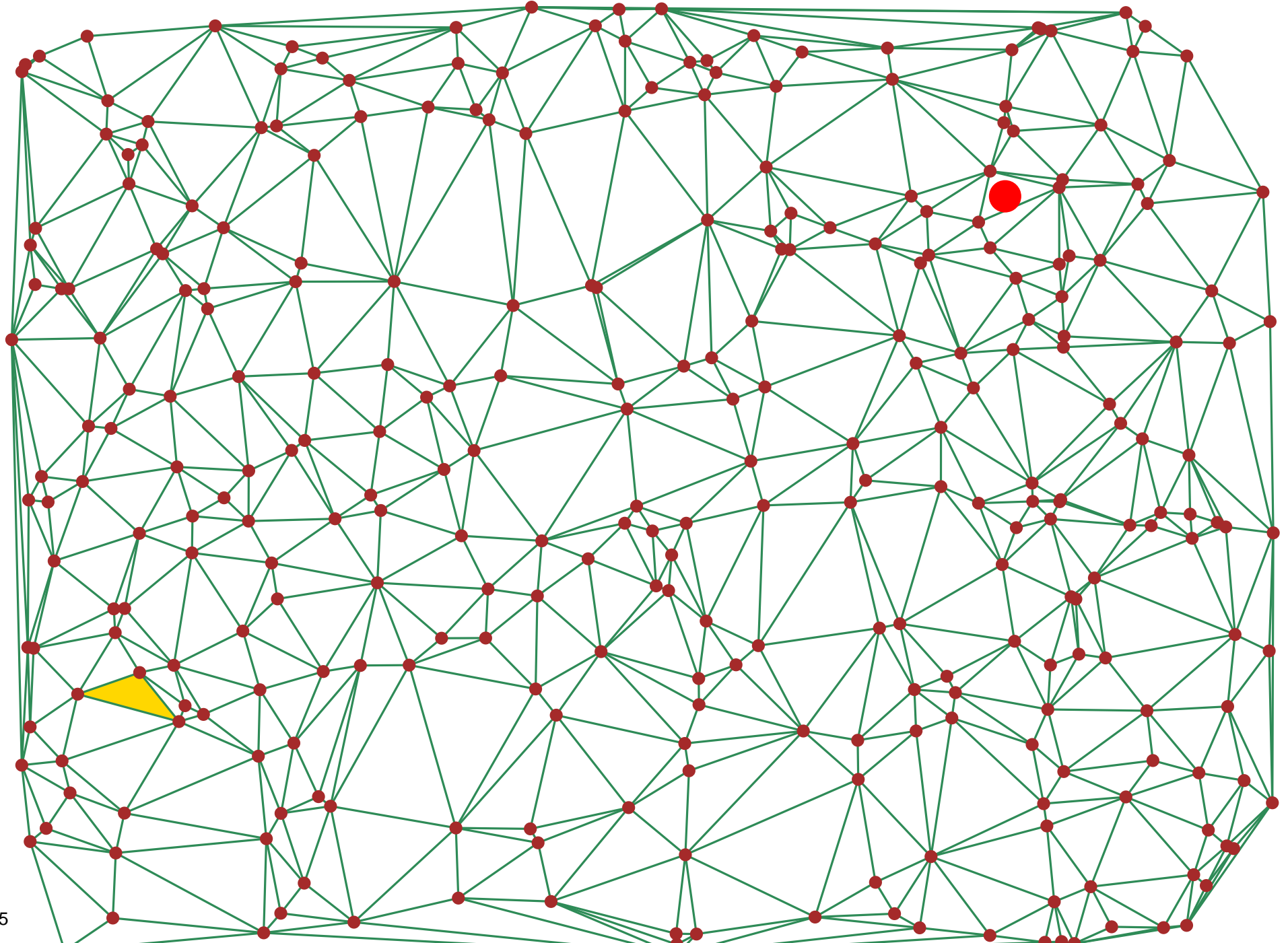
Locate using randomized data structures

Vertex removal in 2D

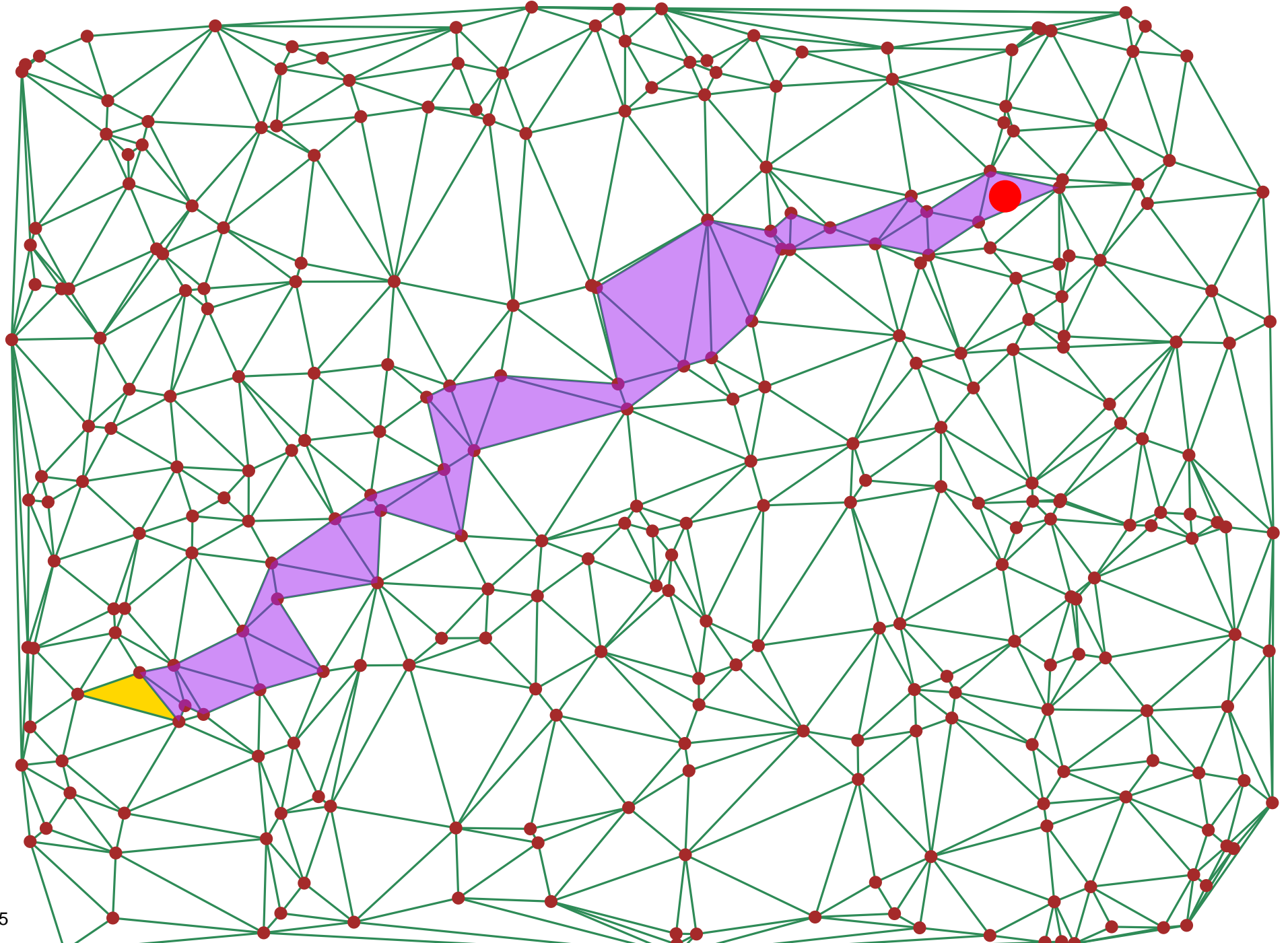
Remarks on CGAL programming

Conclusion

# Locate by visibility walk - structural filtering

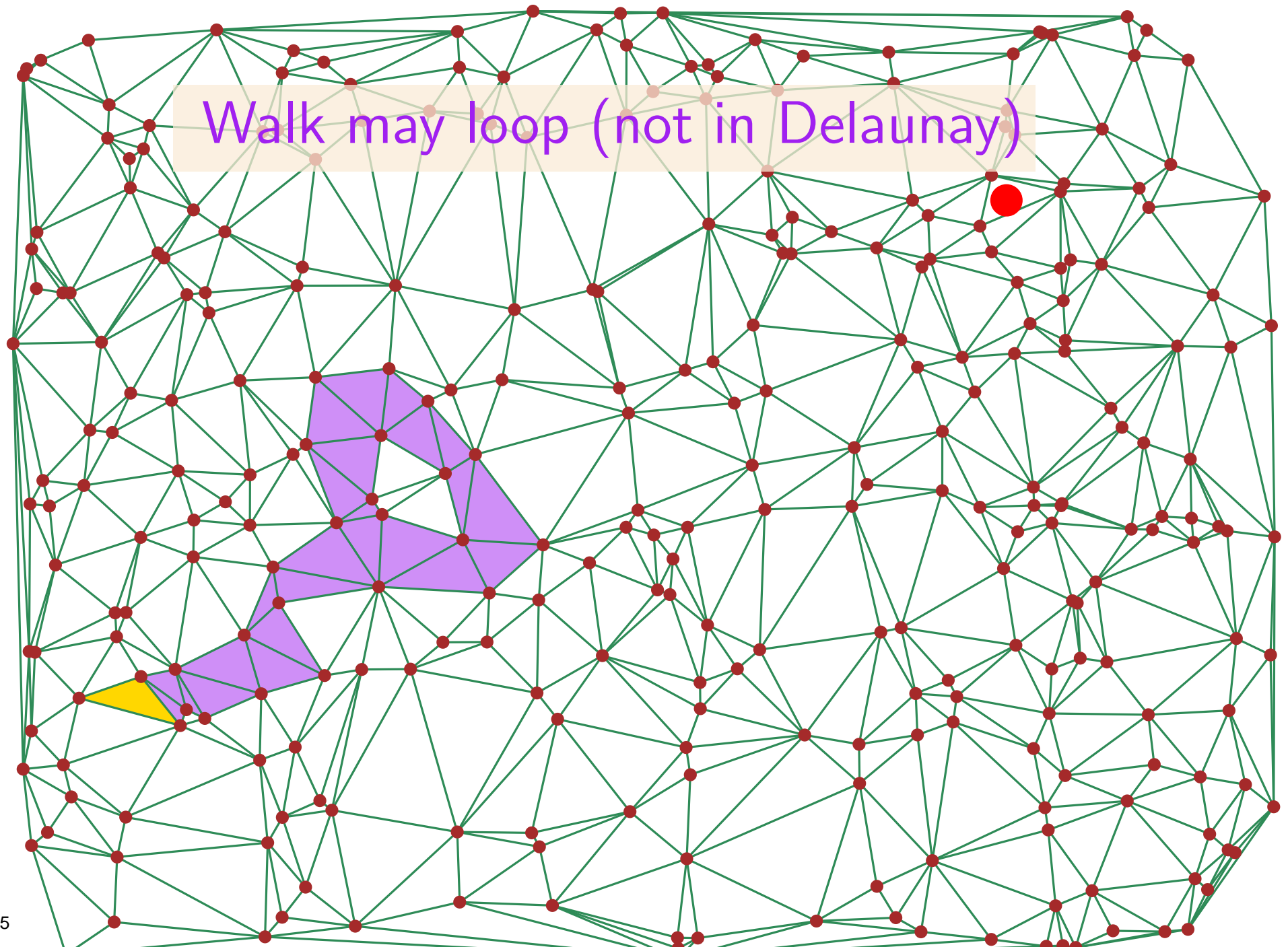


# Locate by visibility walk - structural filtering

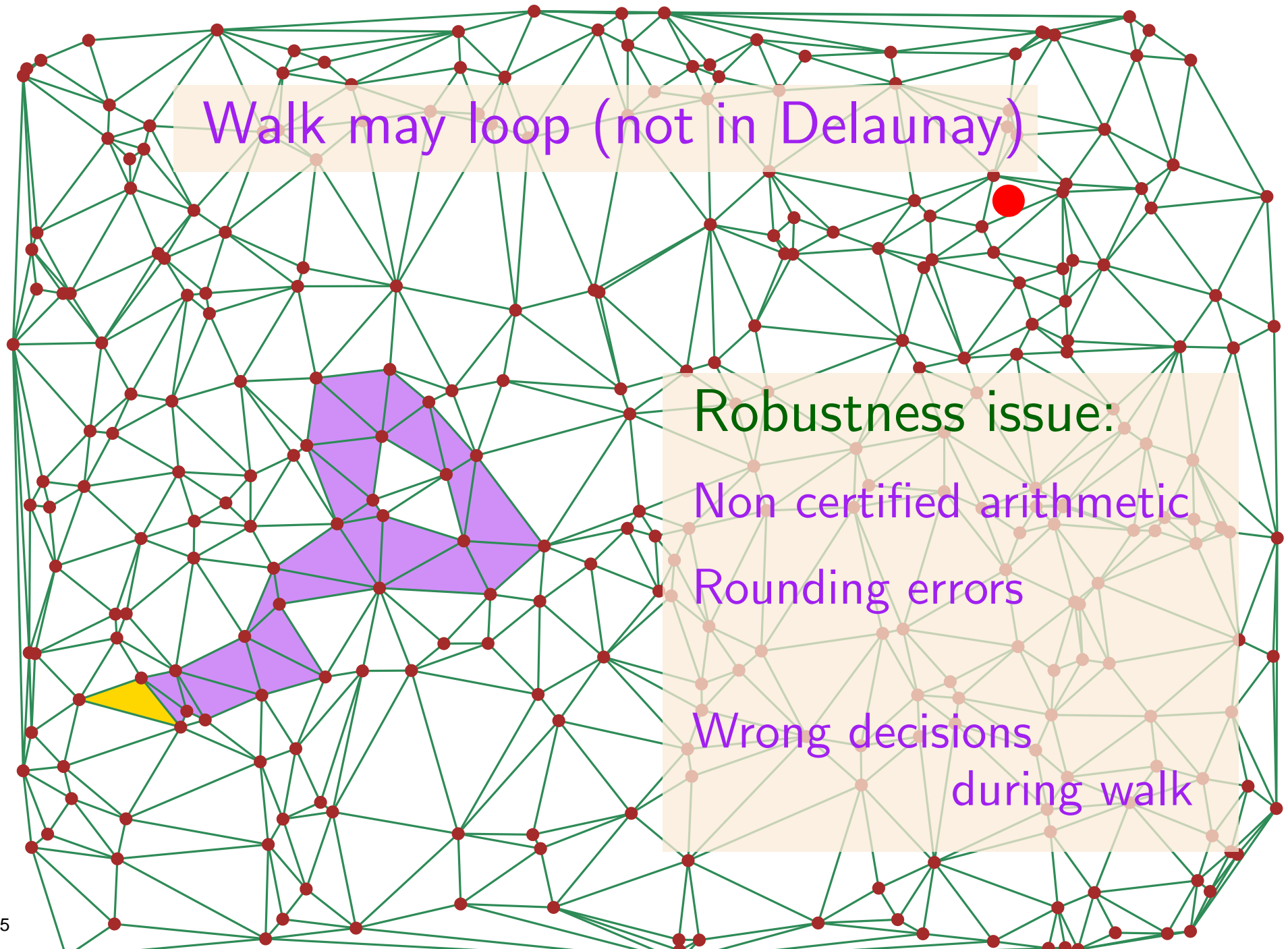




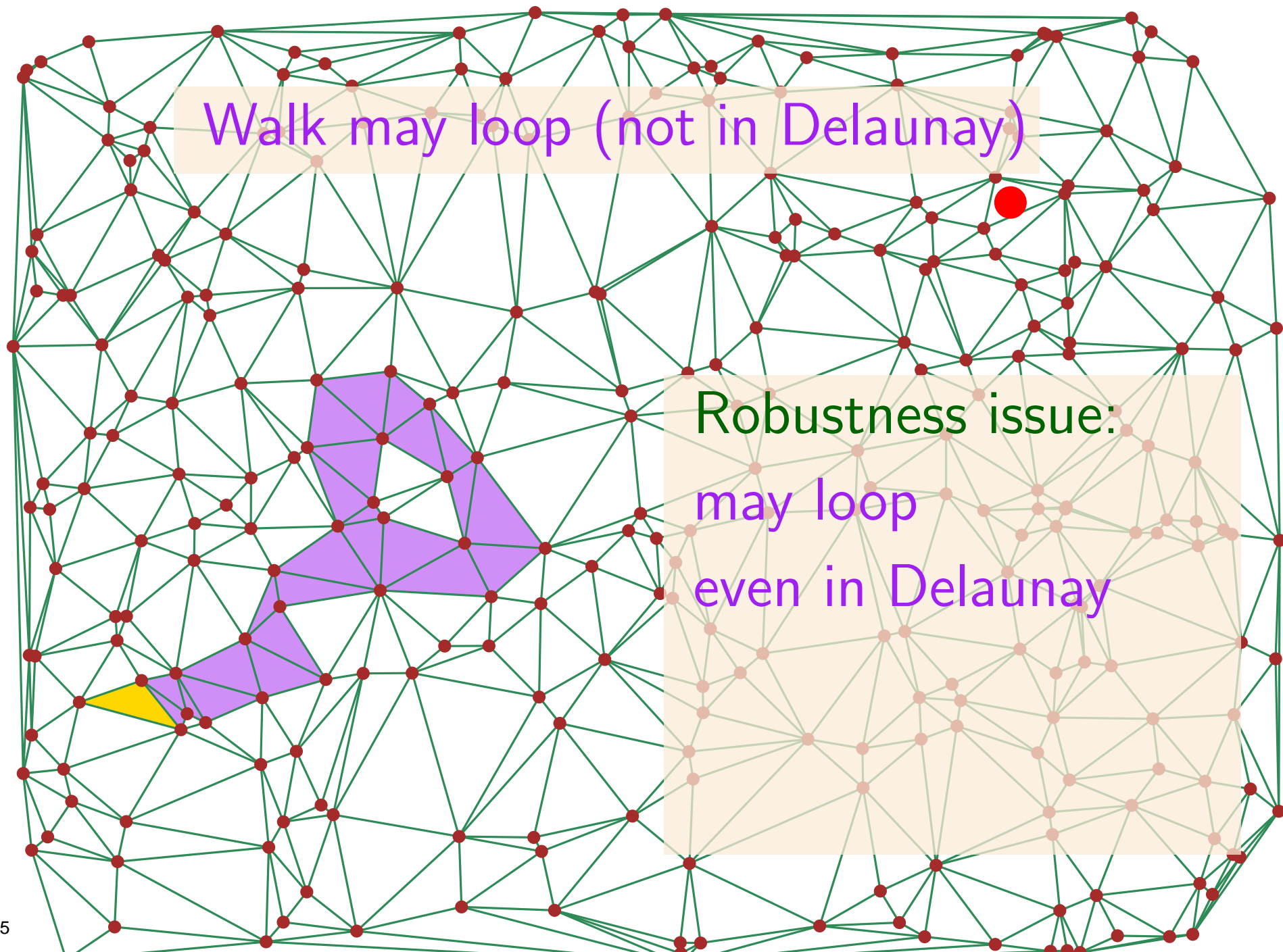
# Locate by visibility walk - structural filtering



# Locate by visibility walk - structural filtering

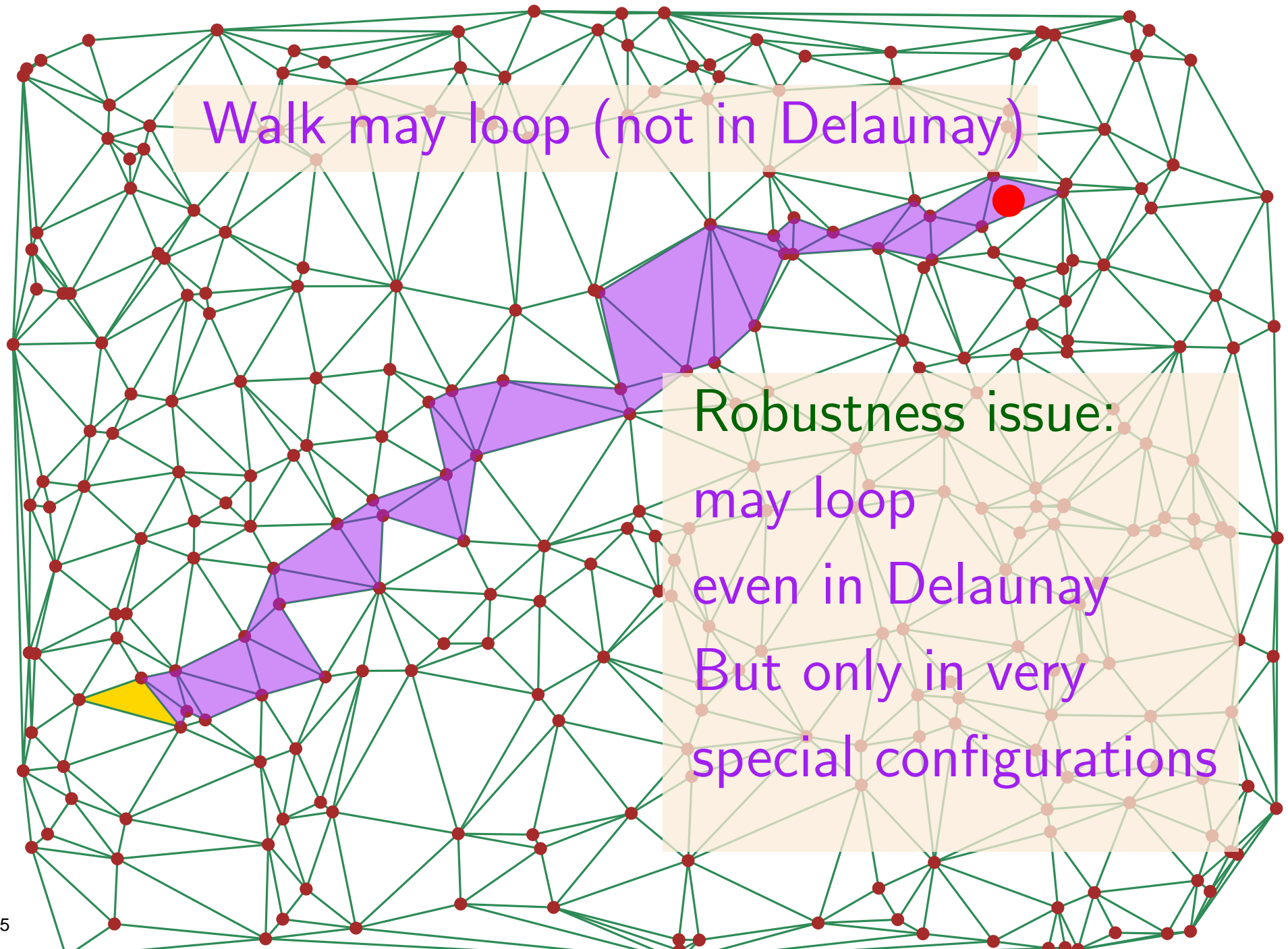


# Locate by visibility walk - structural filtering

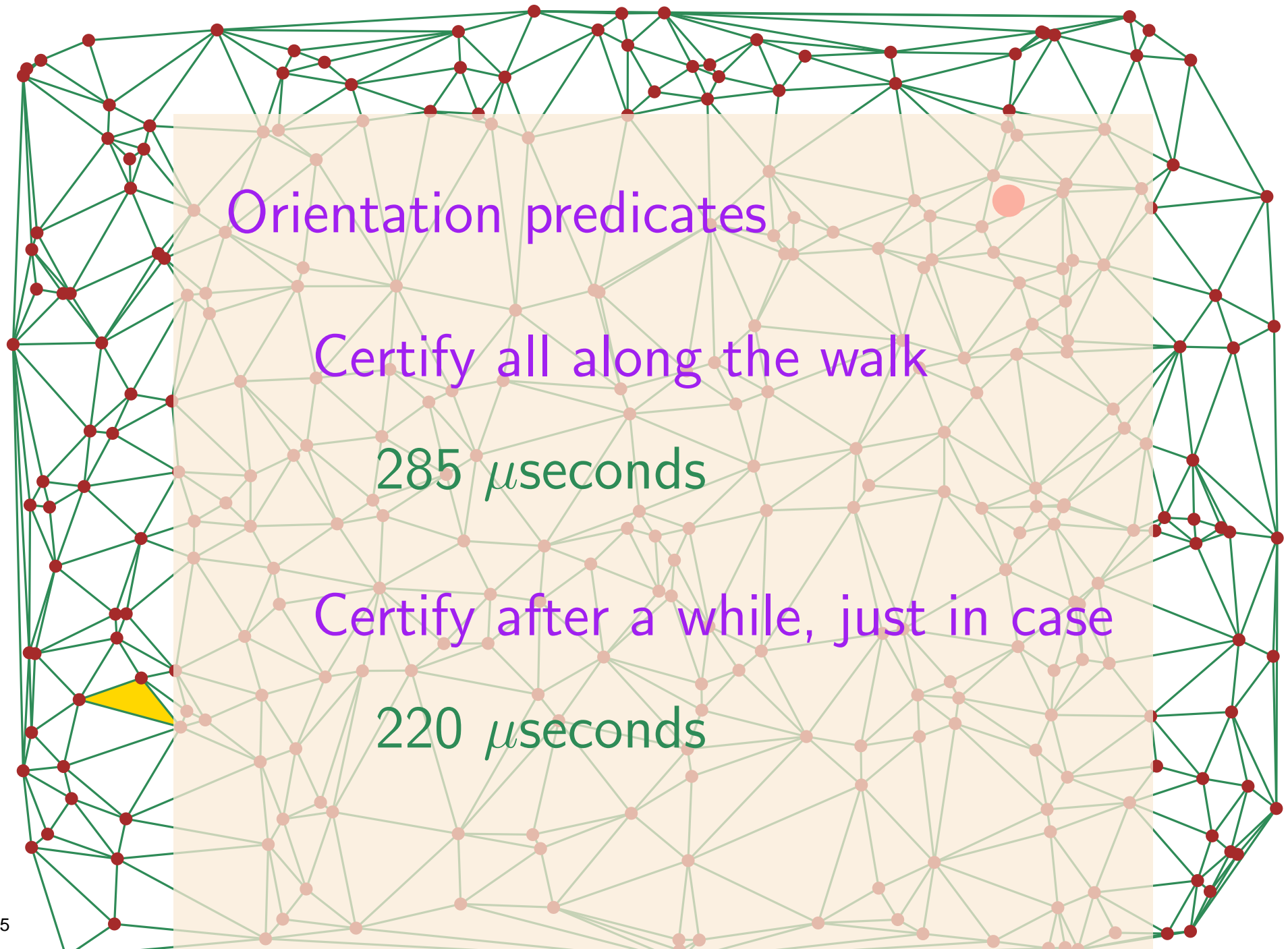




# Locate by visibility walk - structural filtering

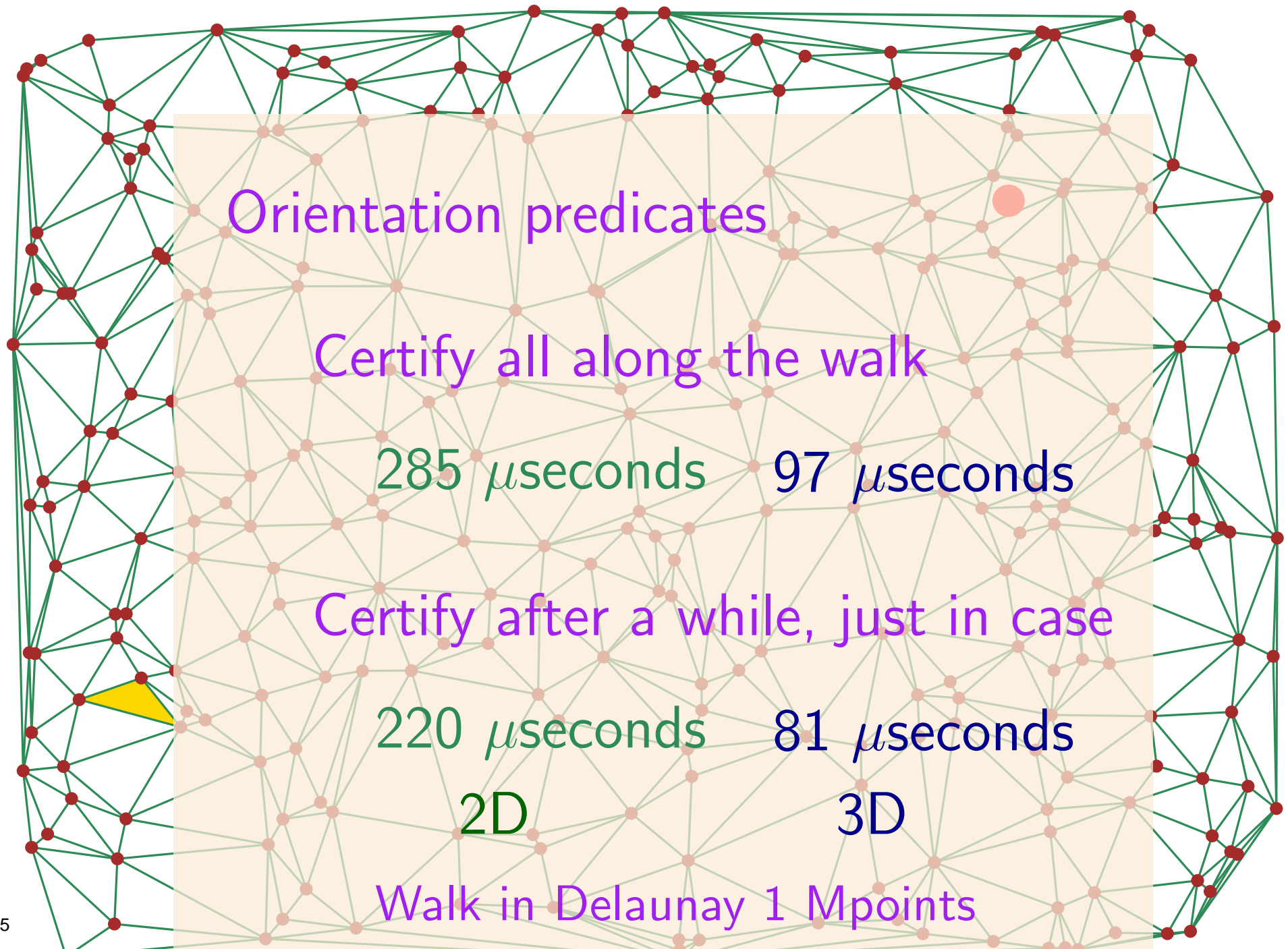


# Locate by visibility walk - structural filtering





# Locate by visibility walk - structural filtering





One word on robustness issues  
Basic incremental algorithm

## Locate by walk

Straight walk  
Visibility walk  
Structural filtering  
Walk shape

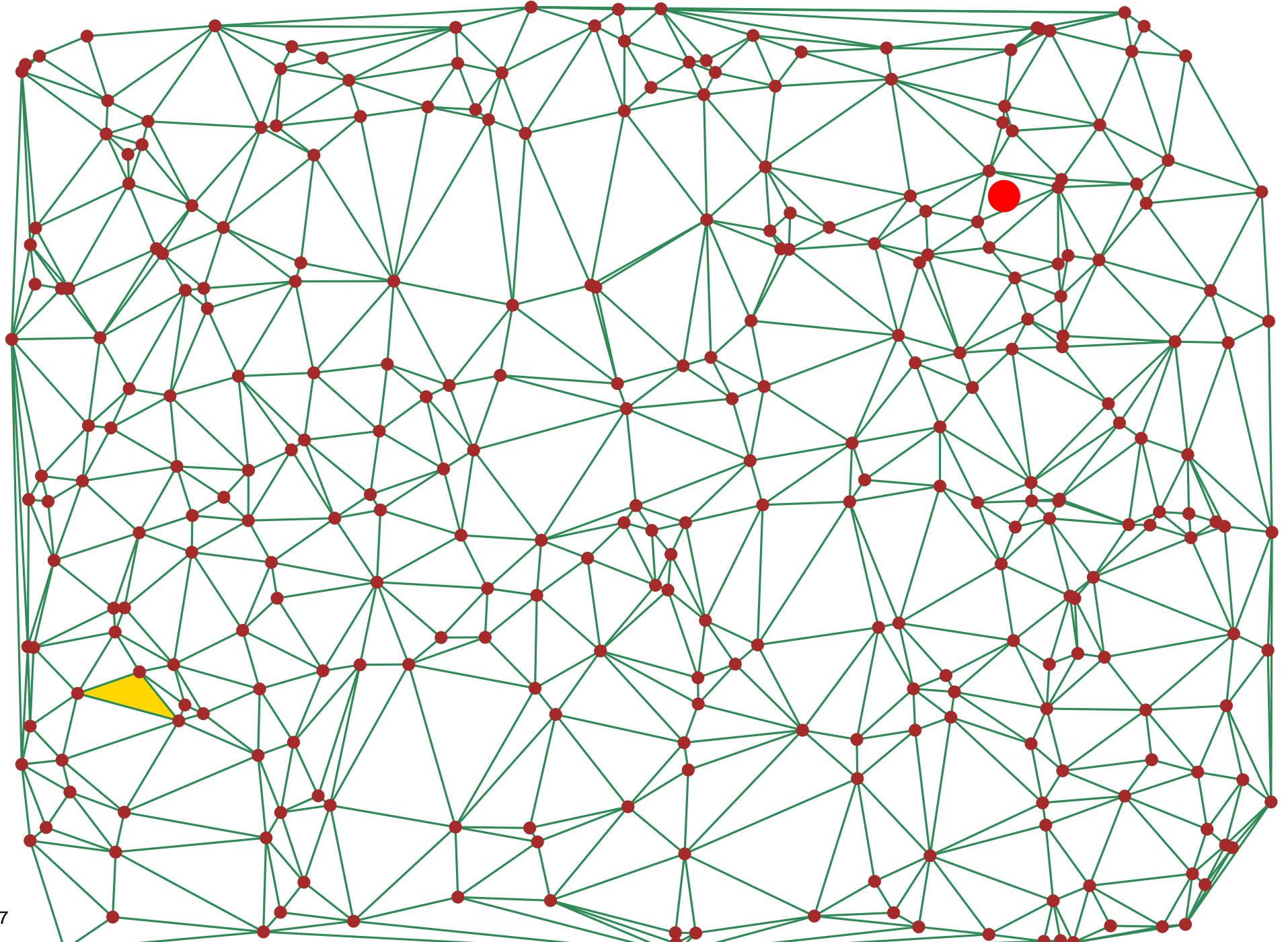
Locate using randomized data structures

Vertex removal in 2D

Remarks on CGAL programming

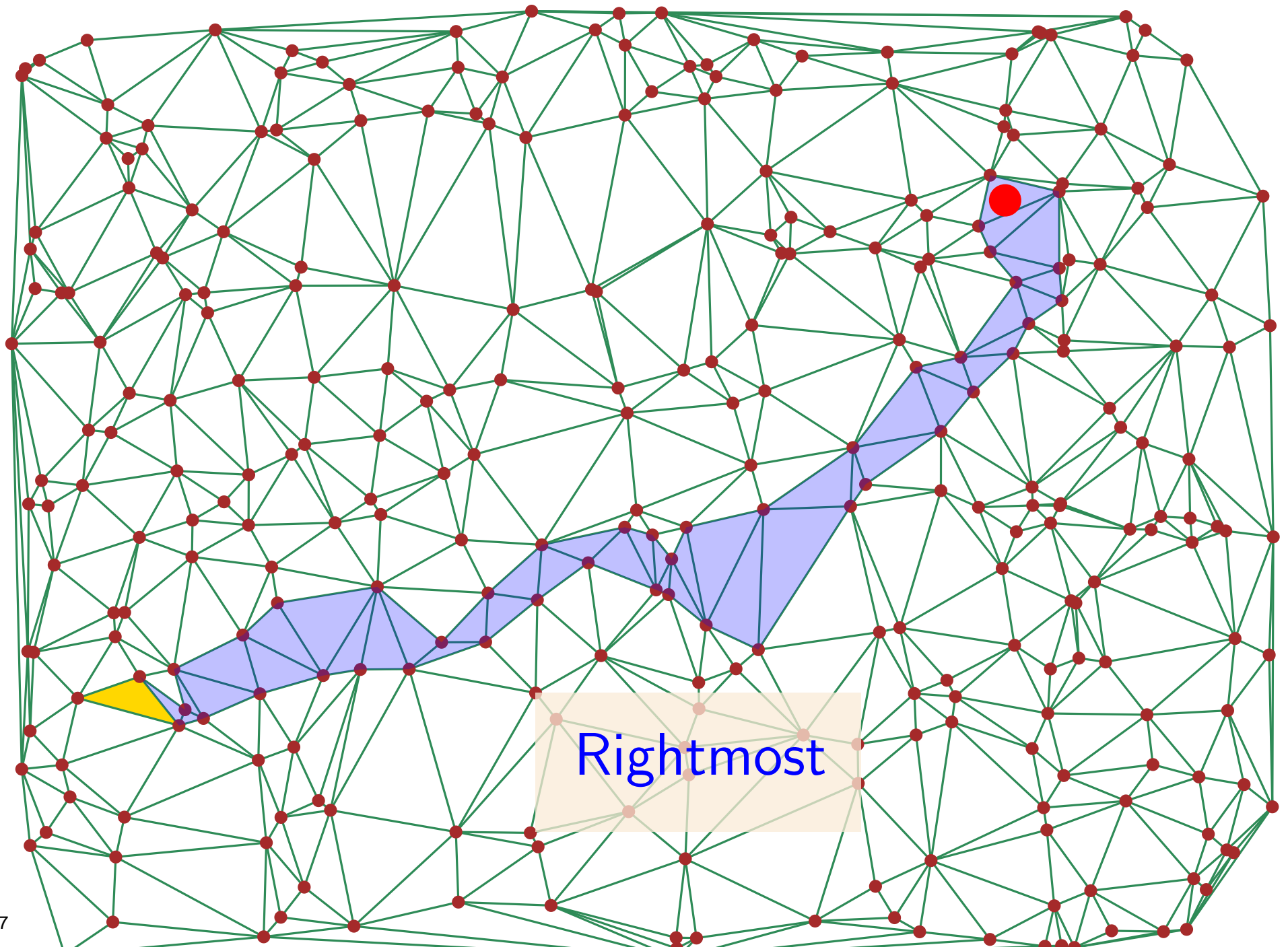
Conclusion

# Locate by visibility walk - walk shape

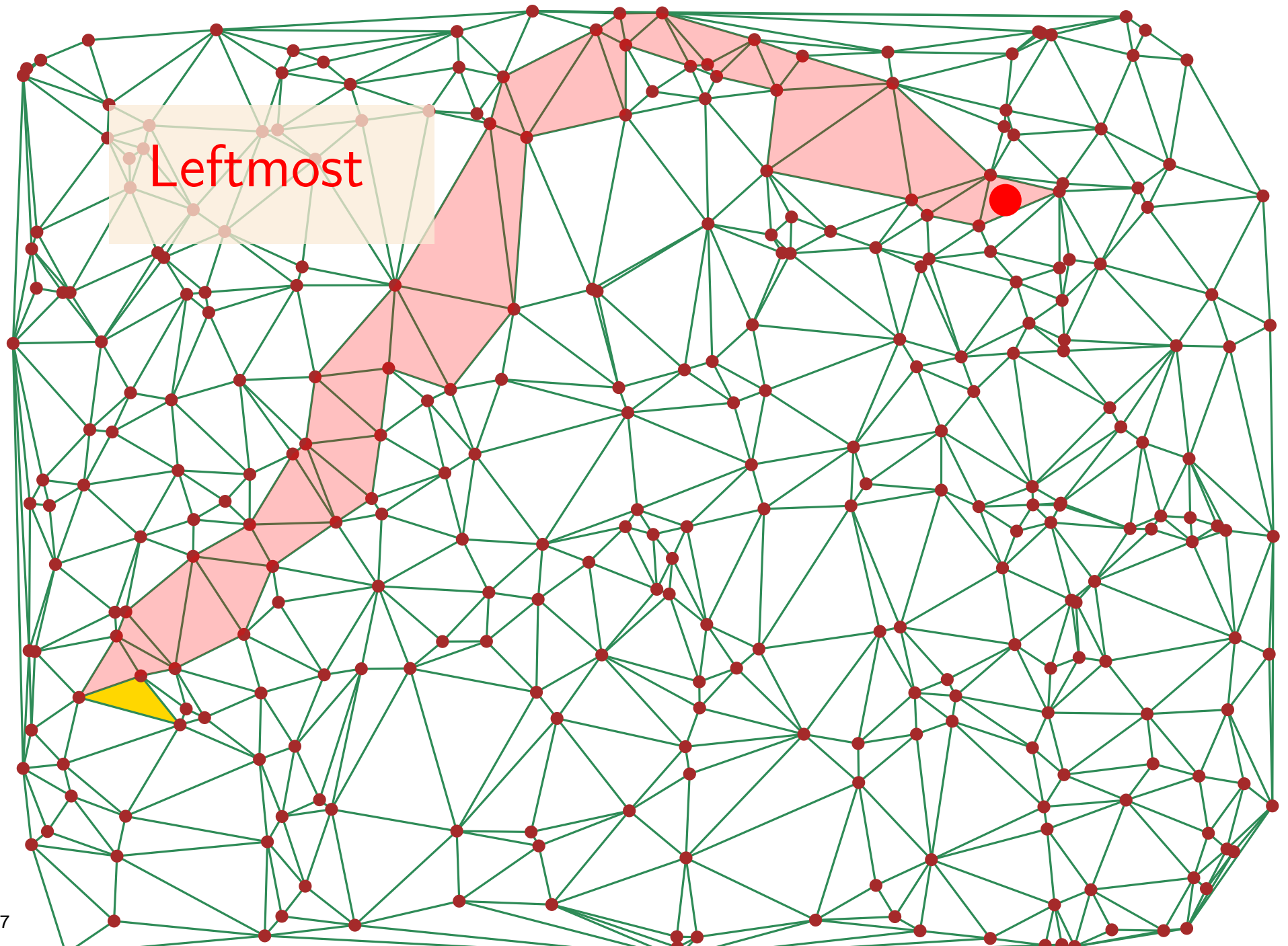




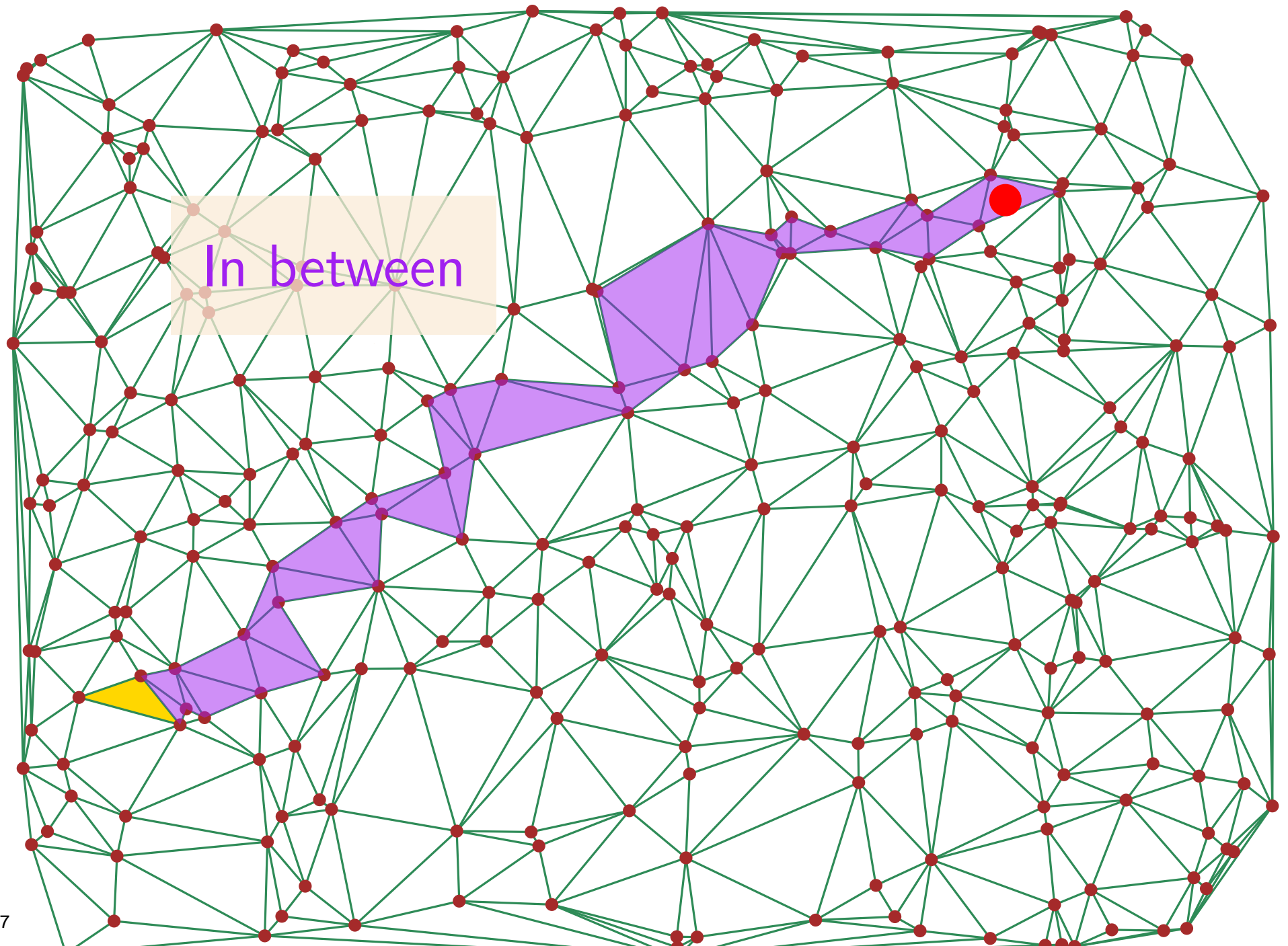
# Locate by visibility walk - walk shape



# Locate by visibility walk - walk shape

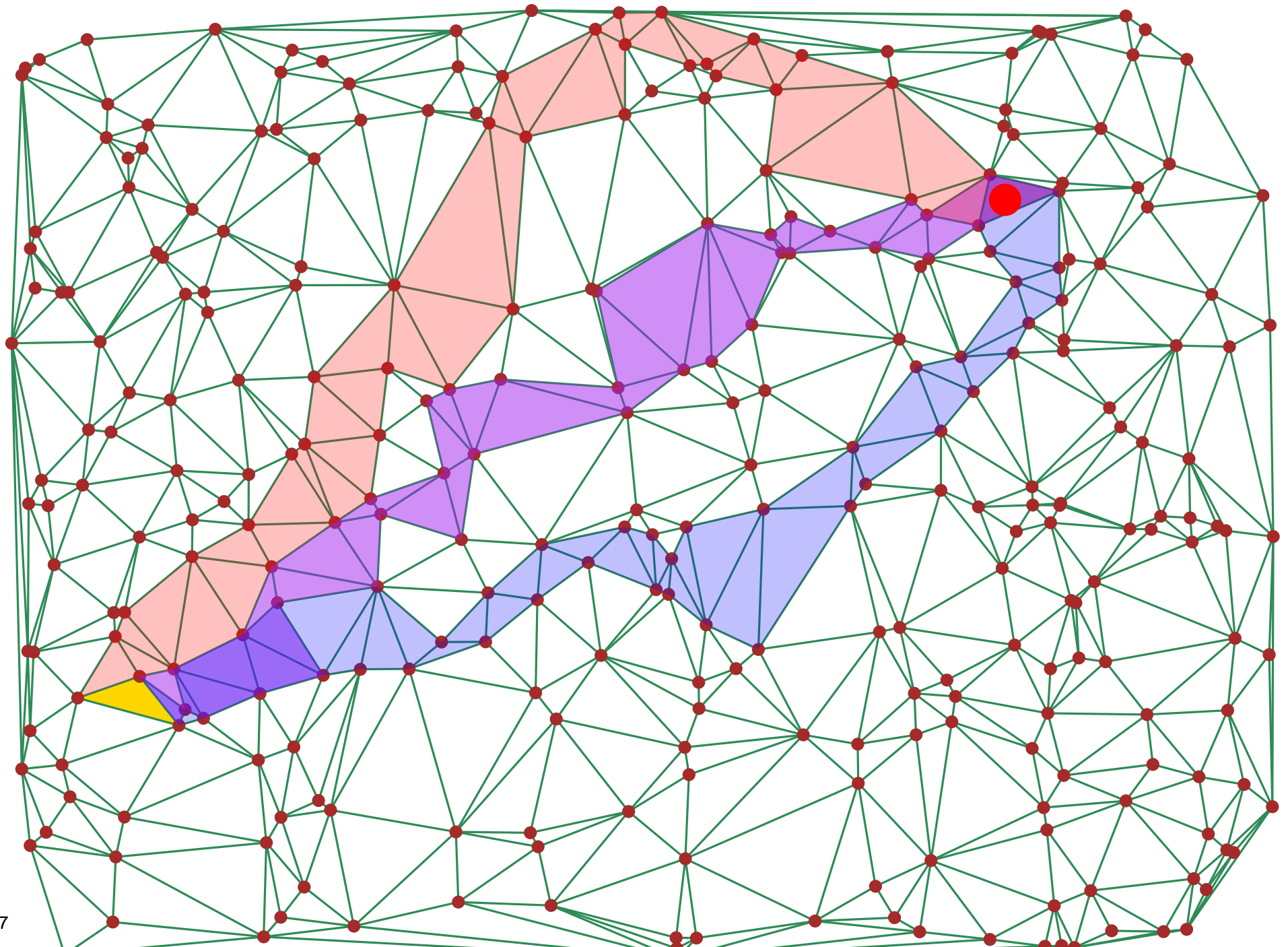


# Locate by visibility walk - walk shape



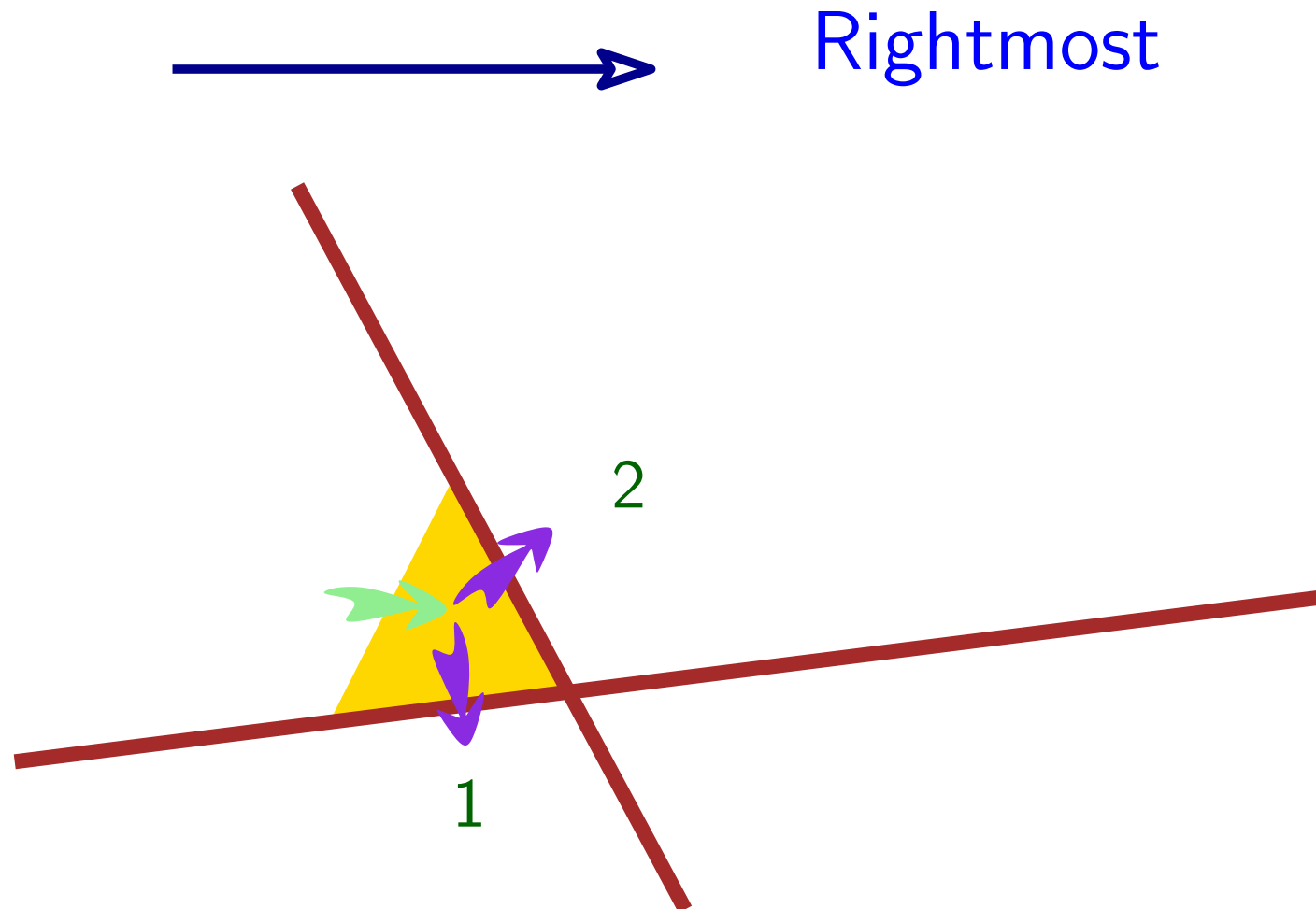


# Locate by visibility walk - walk shape



Locate by visibility walk - walk shape

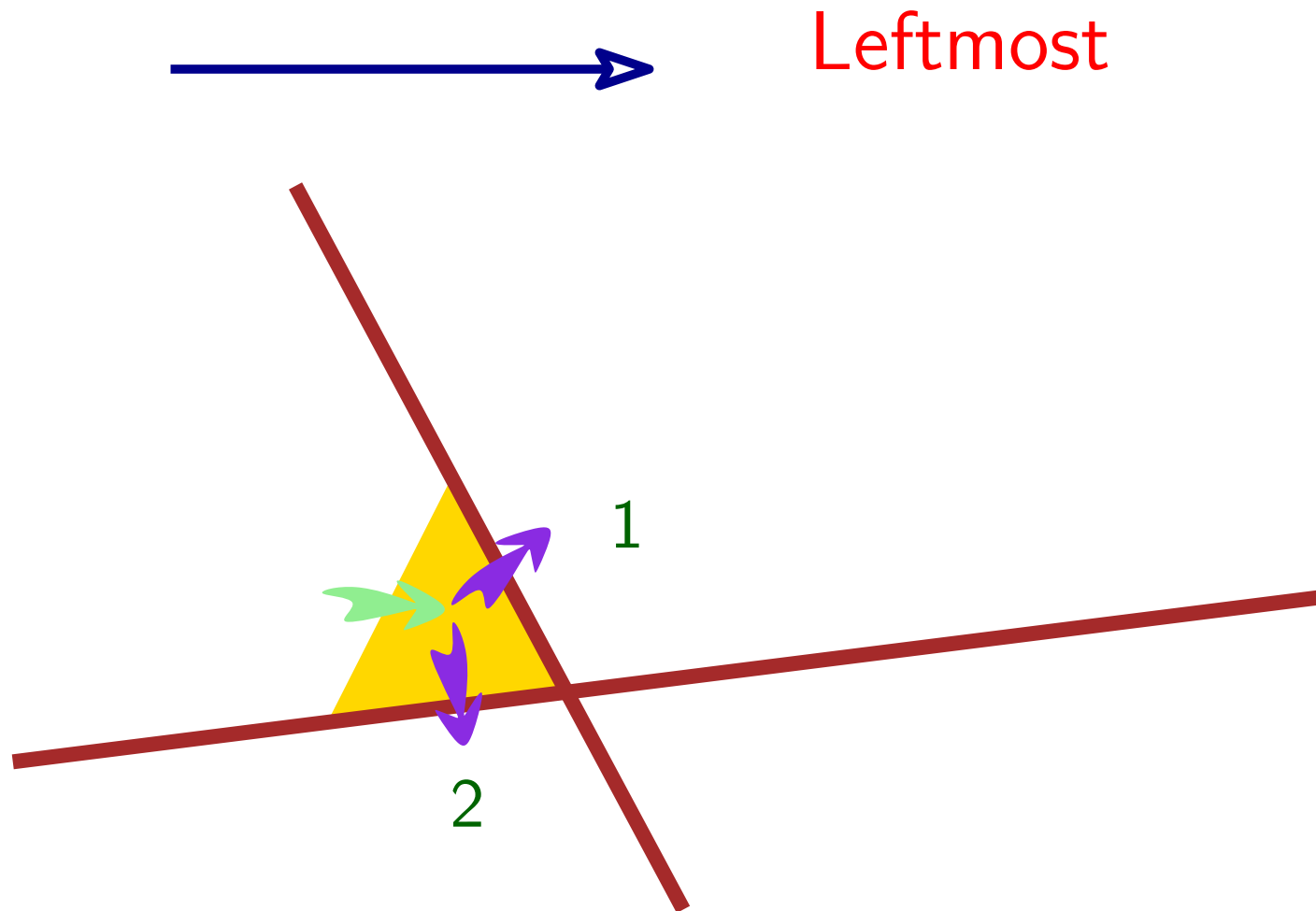
Turn counterclockwise from previous





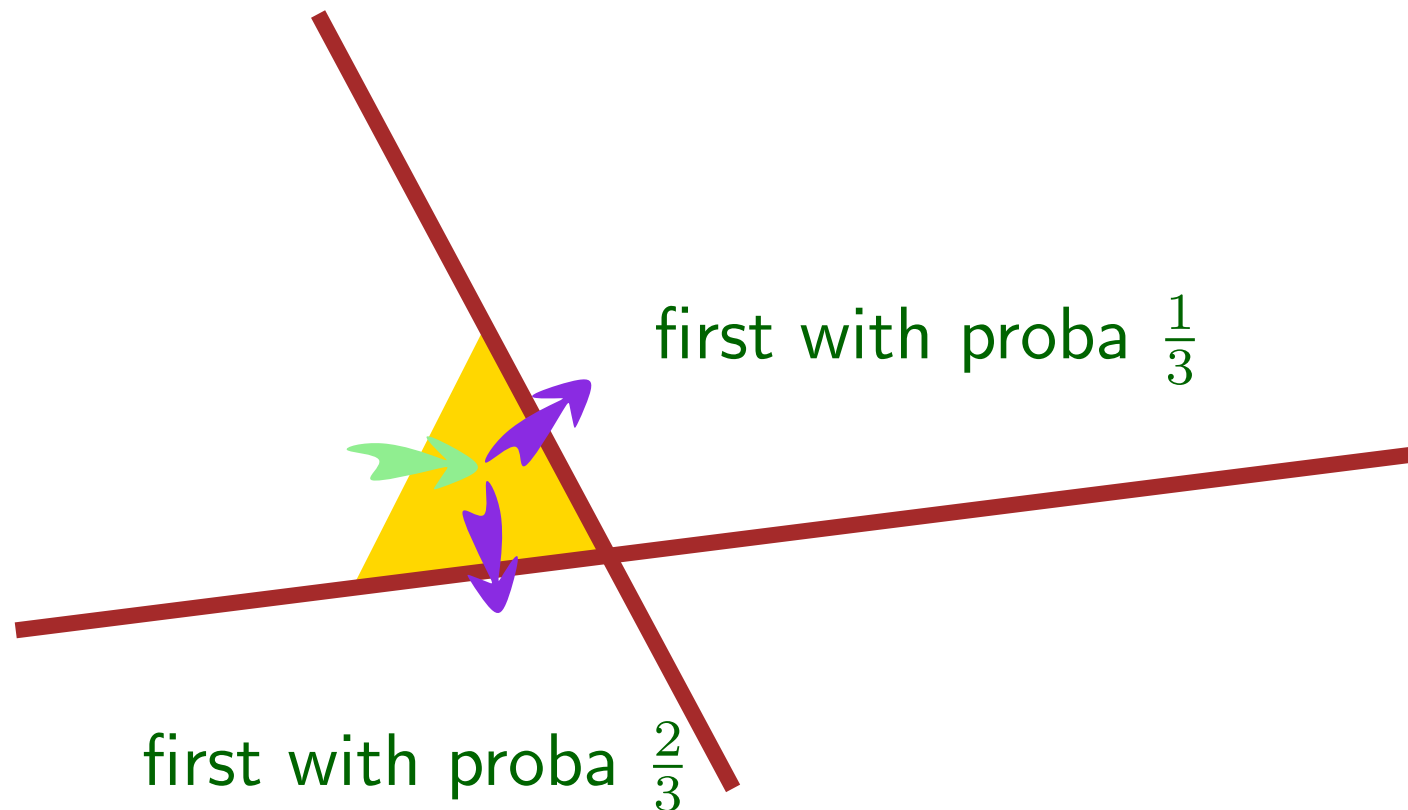
Locate by visibility walk - walk shape

Turn clockwise from previous



Locate by visibility walk - walk shape

Balance left and right turns

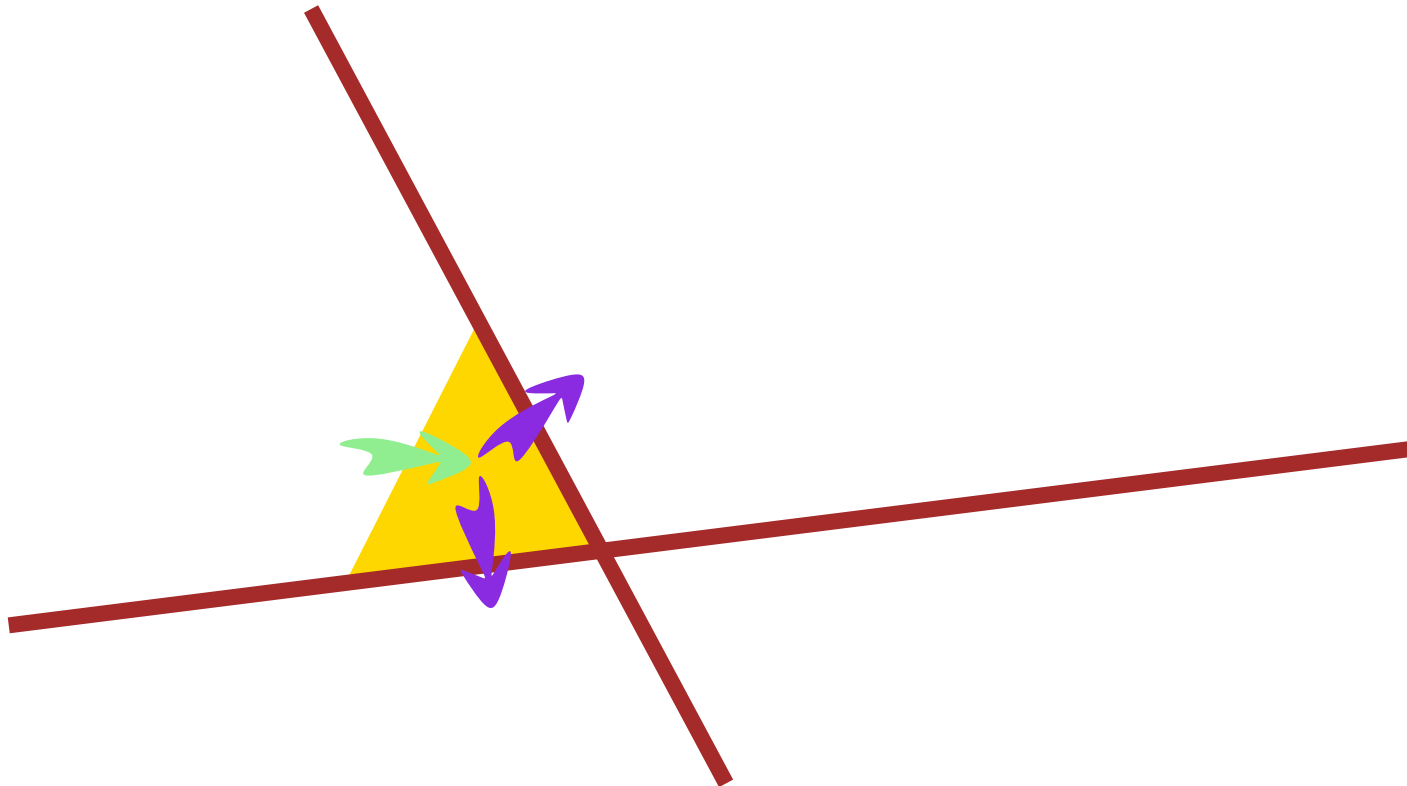


Locate by visibility walk - walk shape

Walk in Delaunay 1 Mpoints

Leftmost 220  $\mu$ seconds

Balanced 188  $\mu$ seconds



## Locate by visibility walk

Walk in Delaunay 1 Mpoints

Straight walk	324 $\mu$ seconds
Visibility walk	285 $\mu$ seconds
Structural filtering	220 $\mu$ seconds
Balanced walk	188 $\mu$ seconds



One word on robustness issues  
Basic incremental algorithm  
Locate by walk

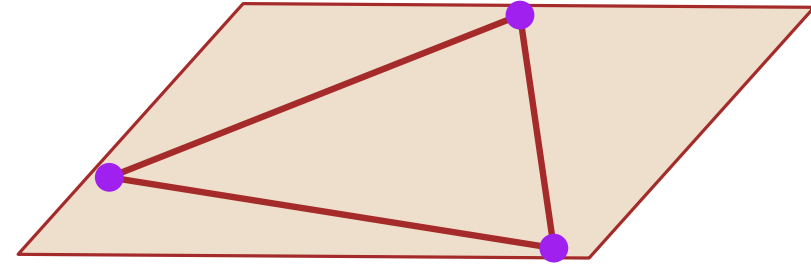
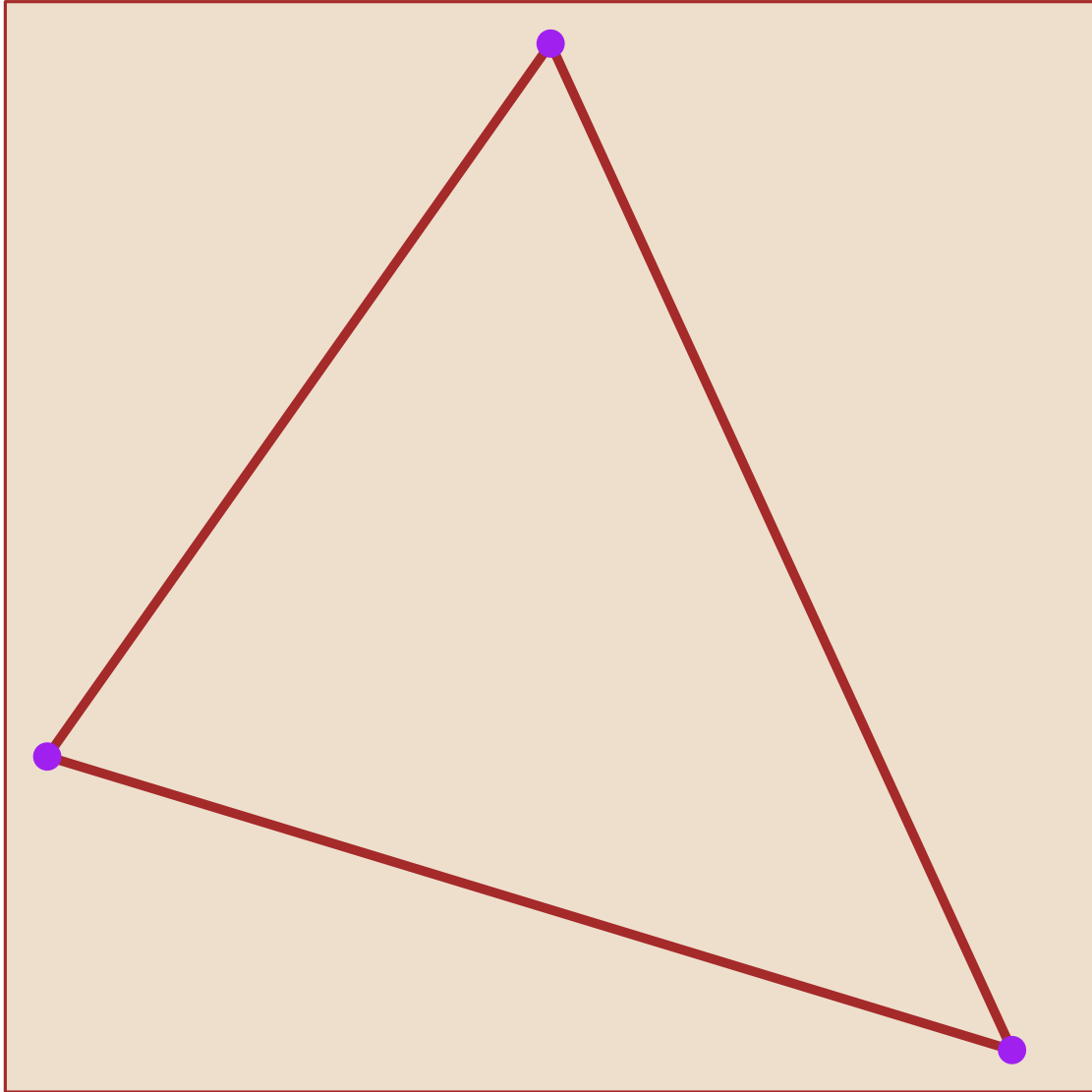
Locate using randomized data structures

Delaunay tree

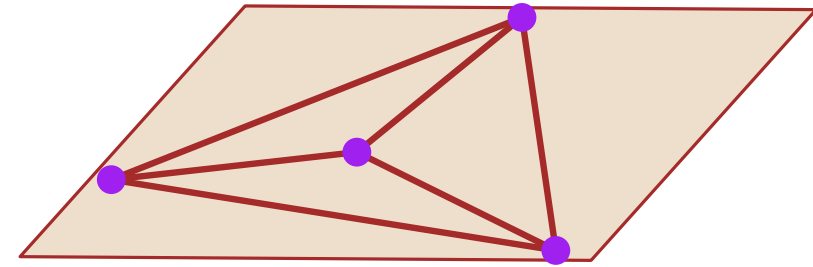
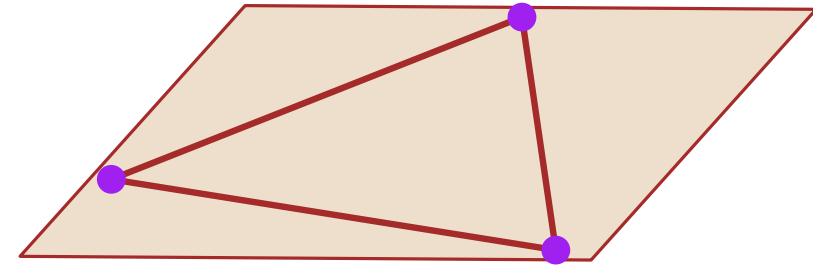
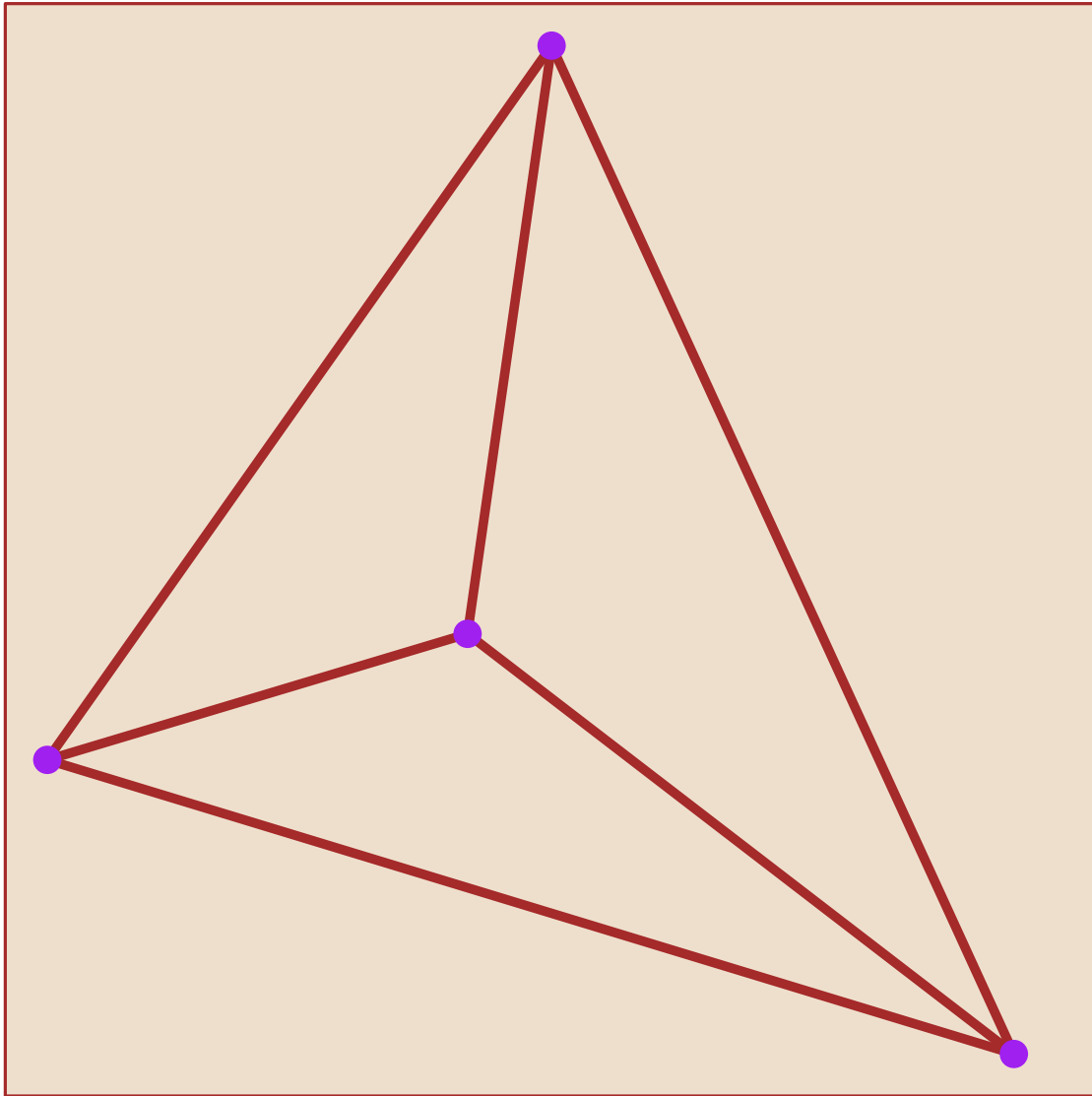
Vertex removal in 2D  
Remarks on CGAL programming  
Conclusion

# Data structures to locate - the Delaunay tree

# Data structures to locate - the Delaunay tree

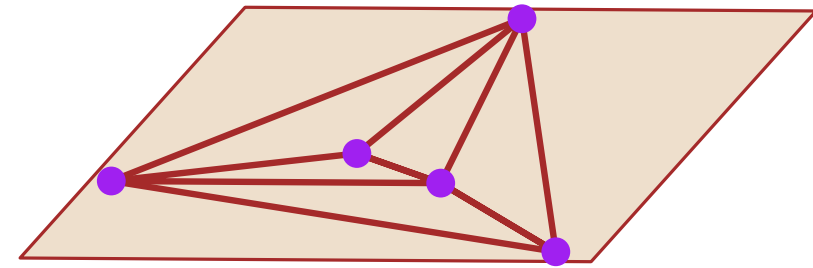
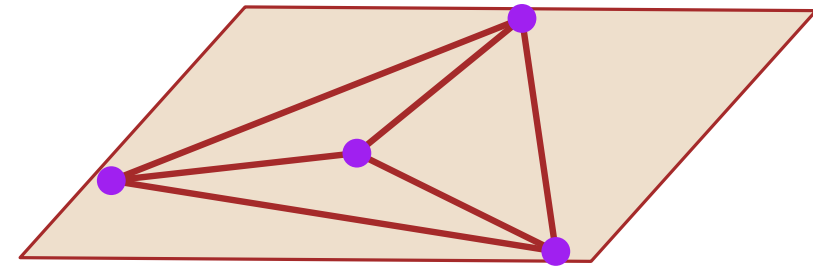
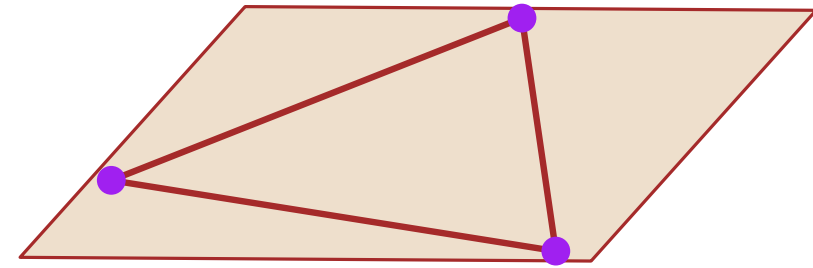
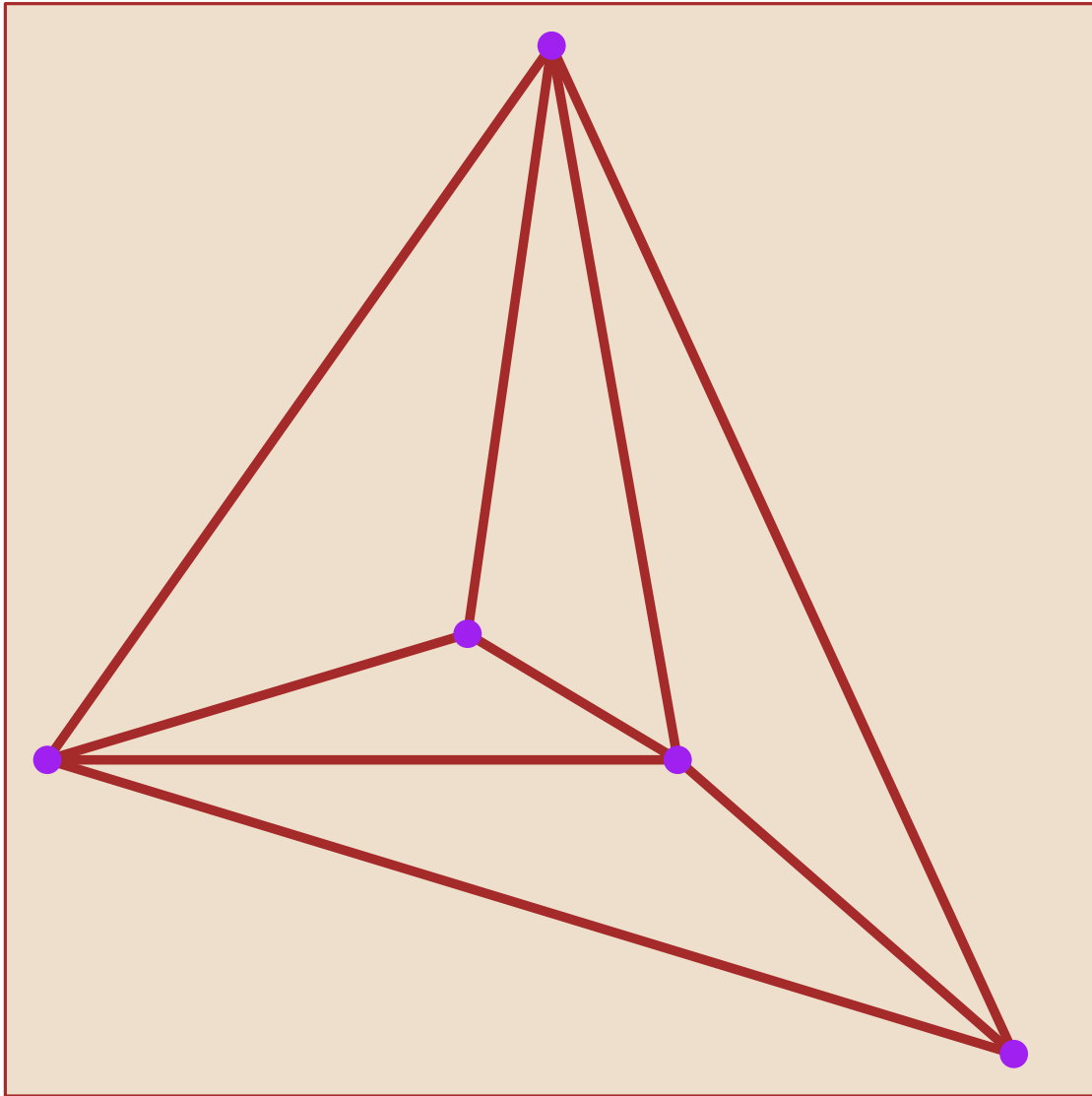


# Data structures to locate - the Delaunay tree

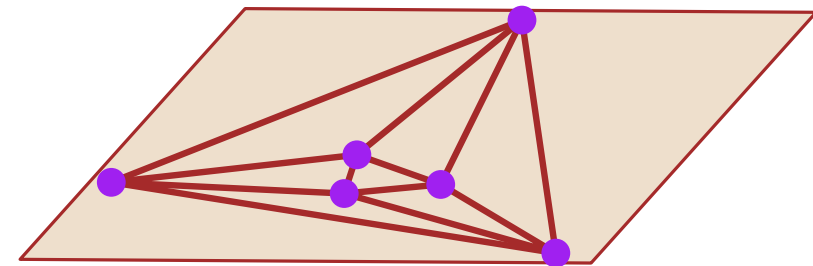
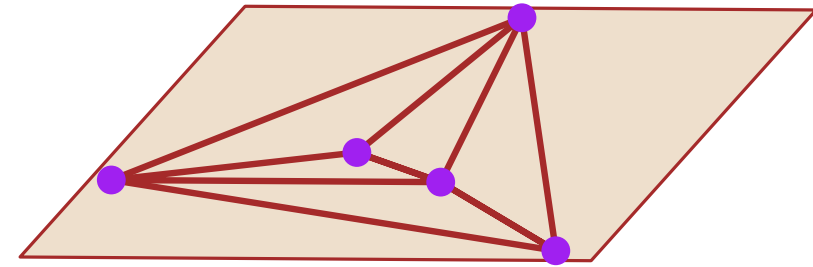
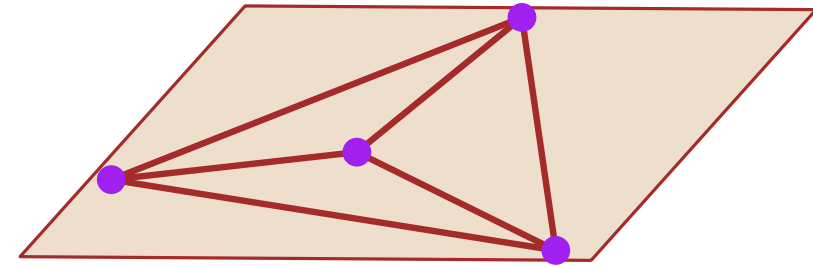
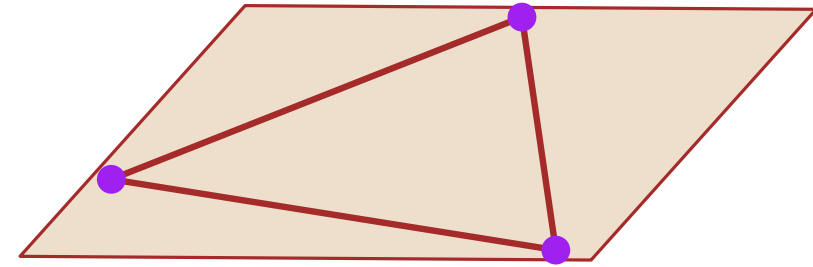
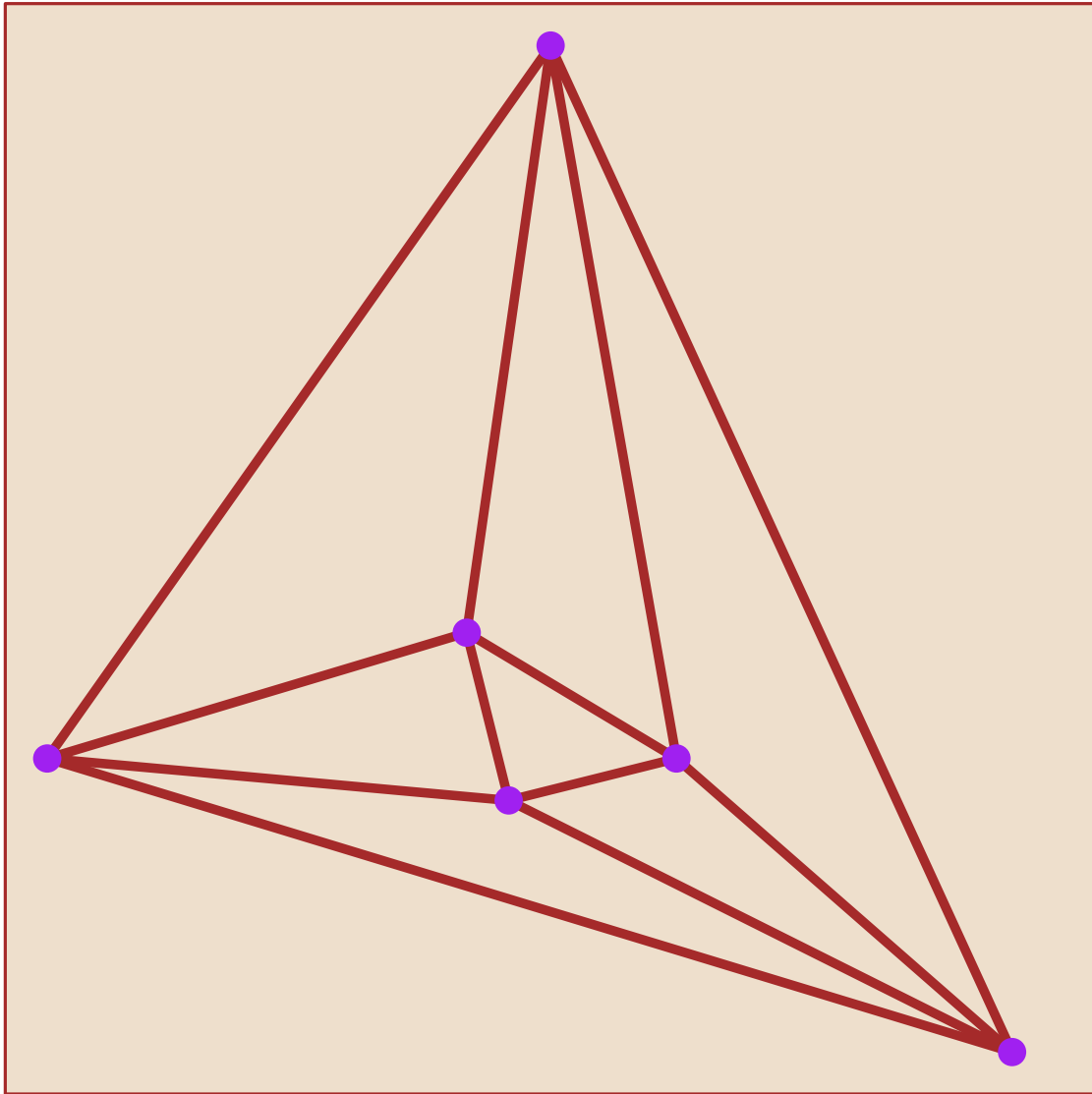




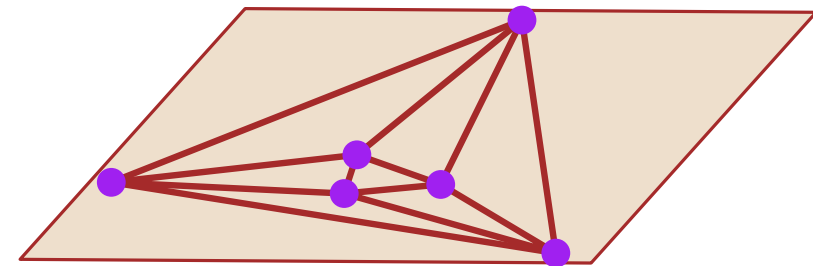
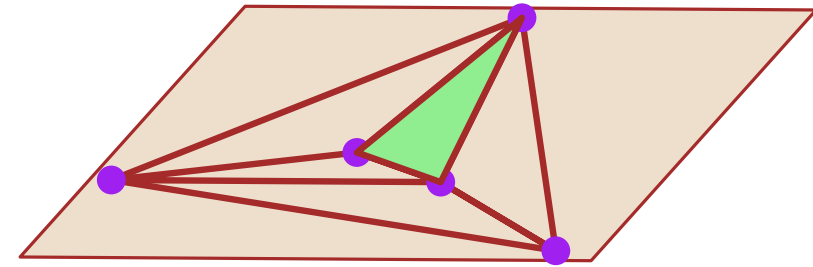
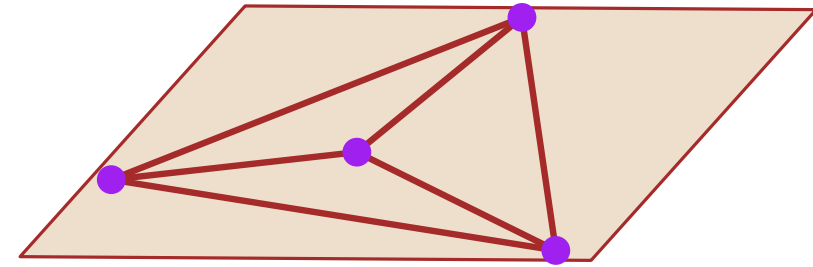
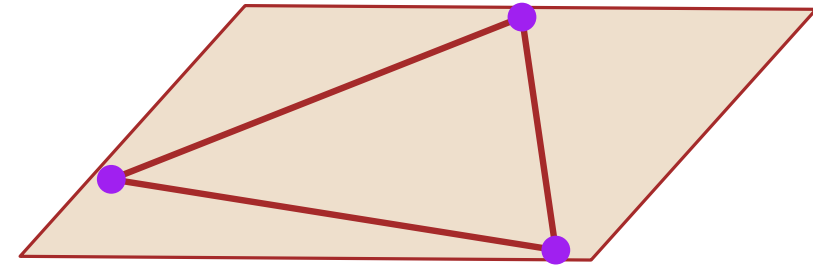
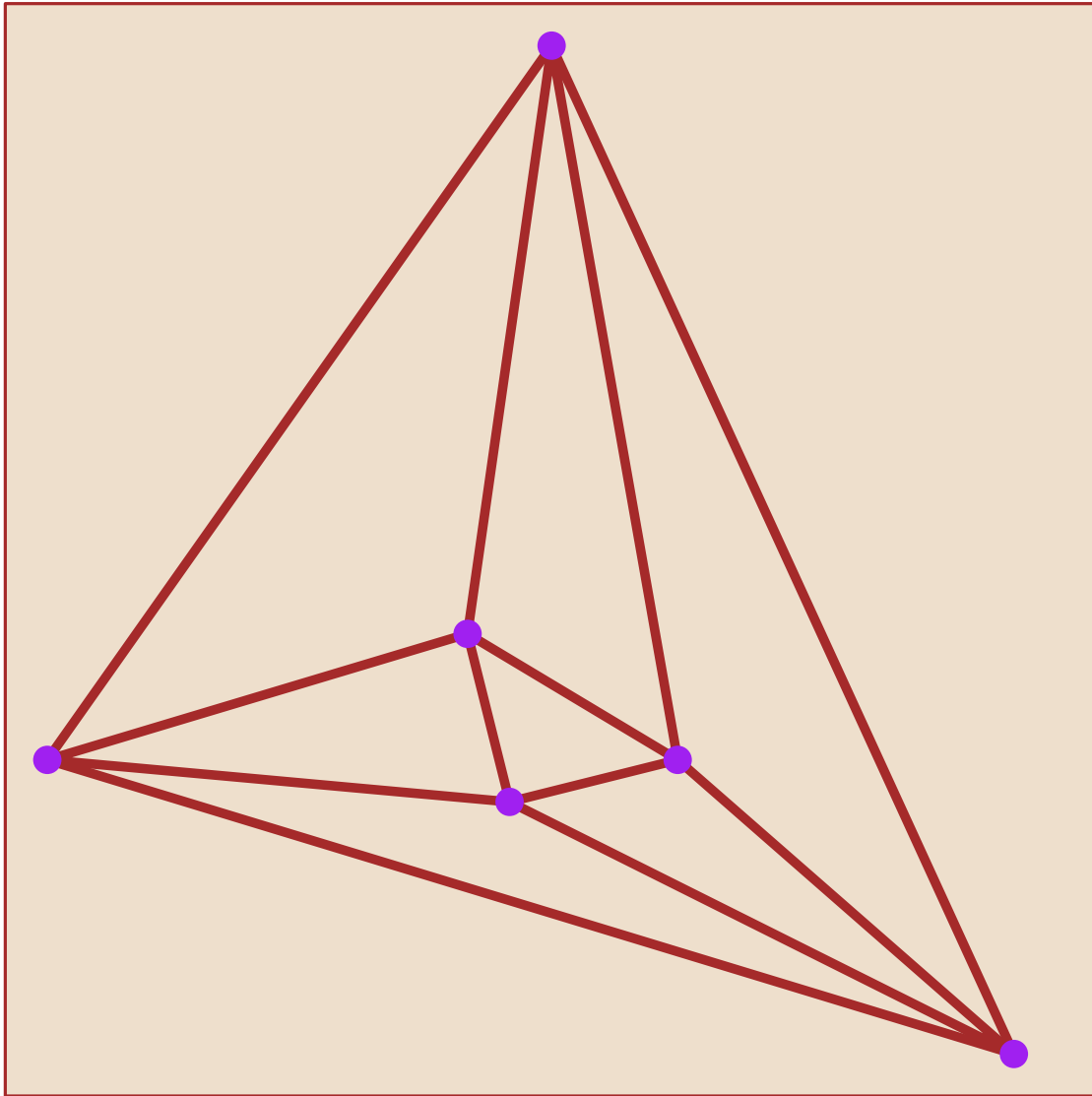
# Data structures to locate - the Delaunay tree



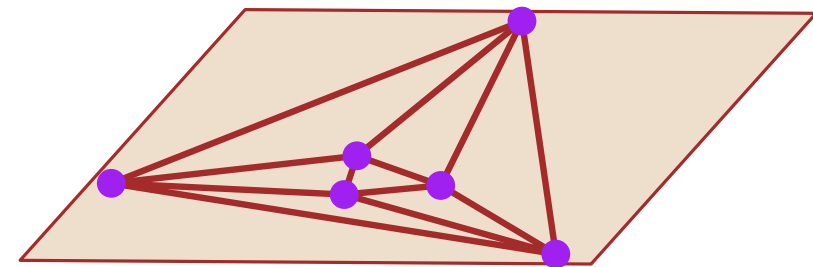
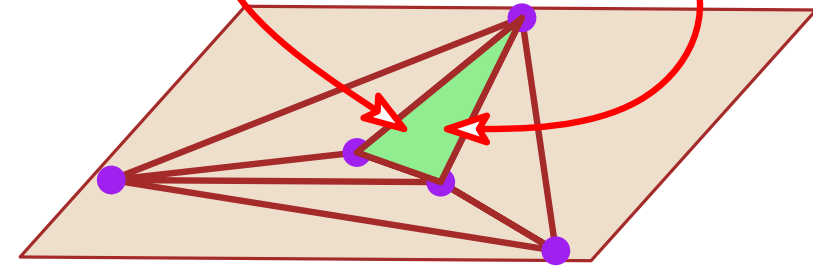
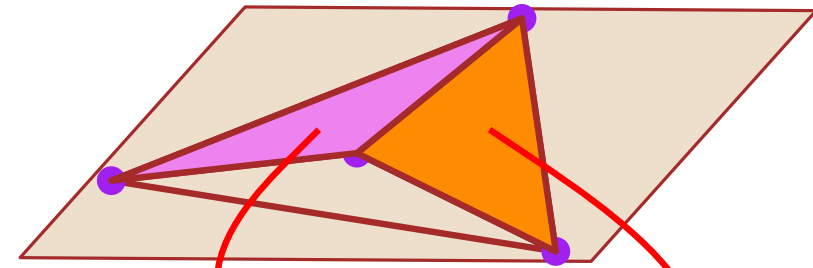
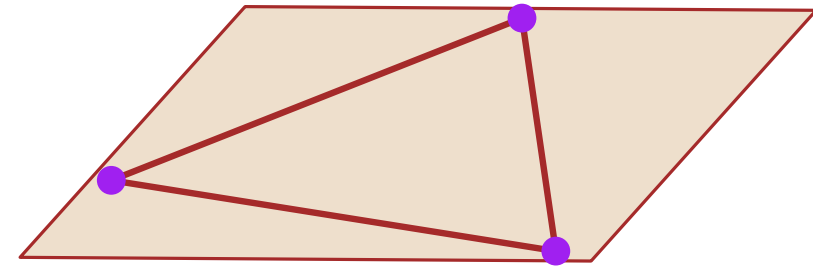
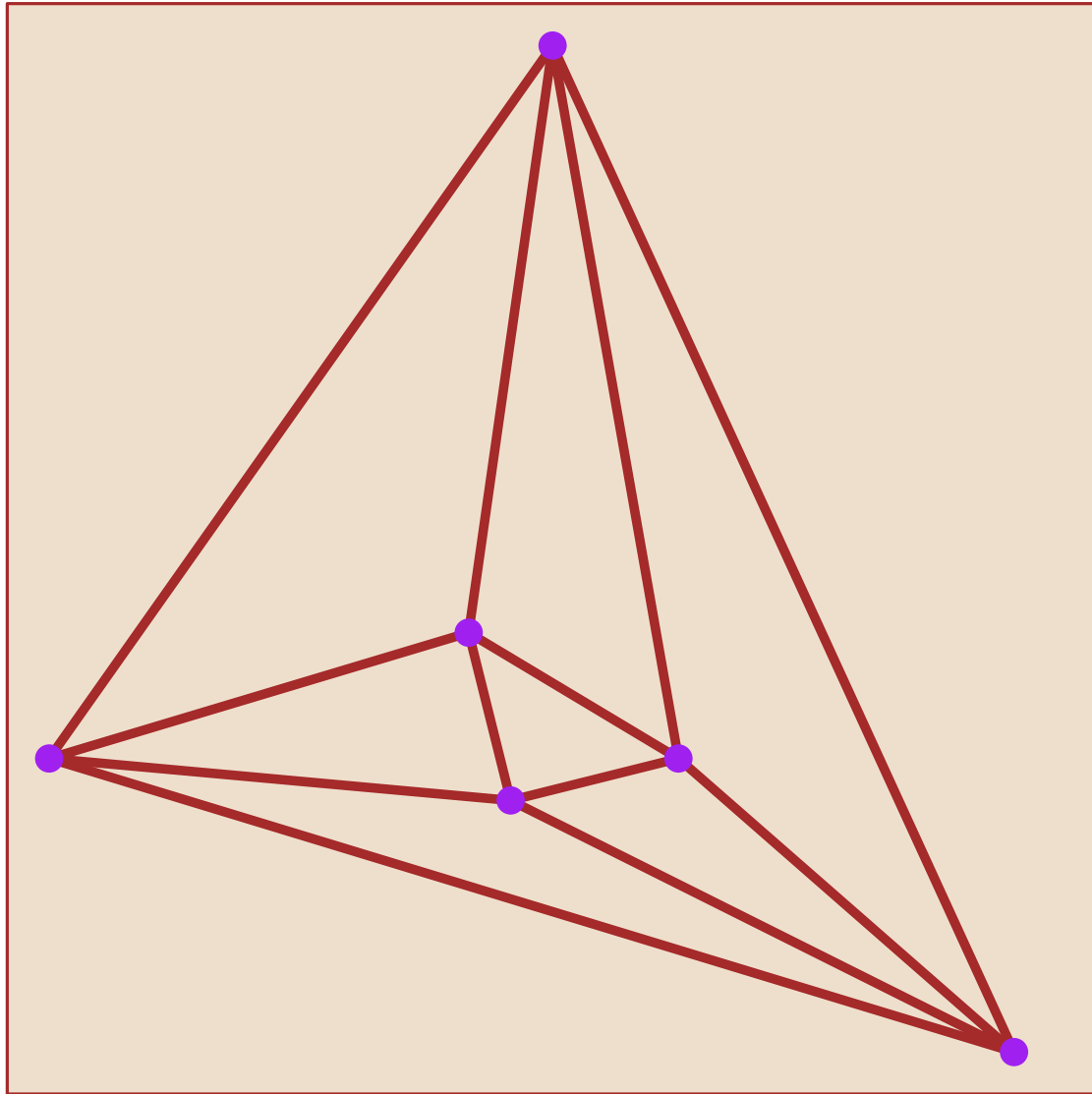
# Data structures to locate - the Delaunay tree



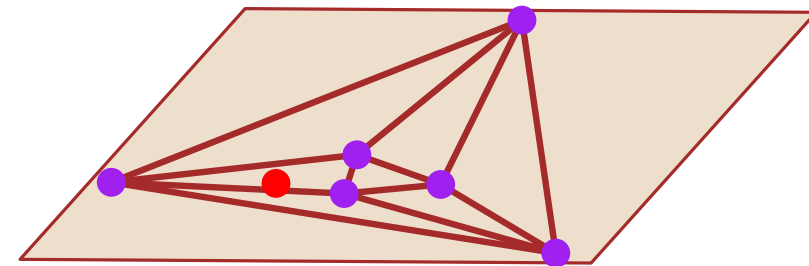
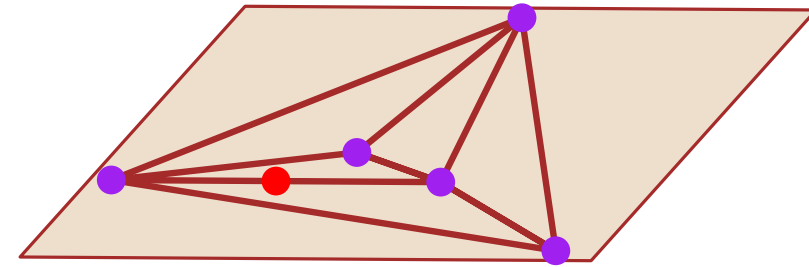
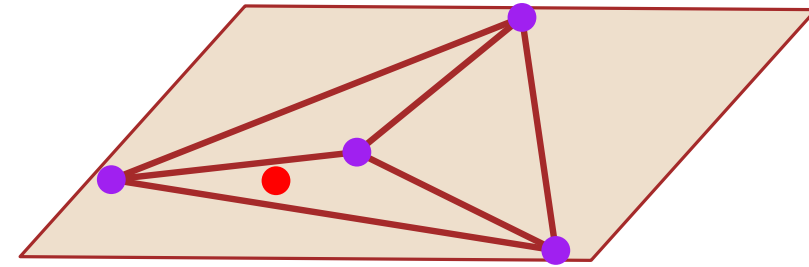
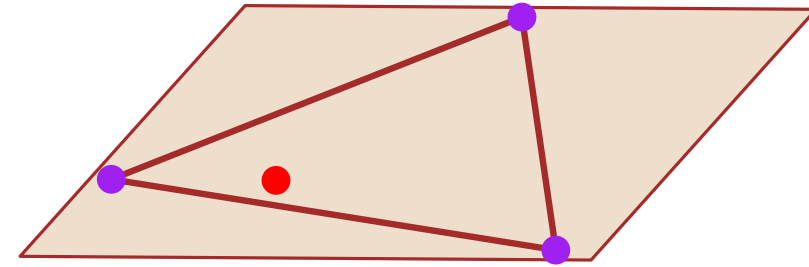
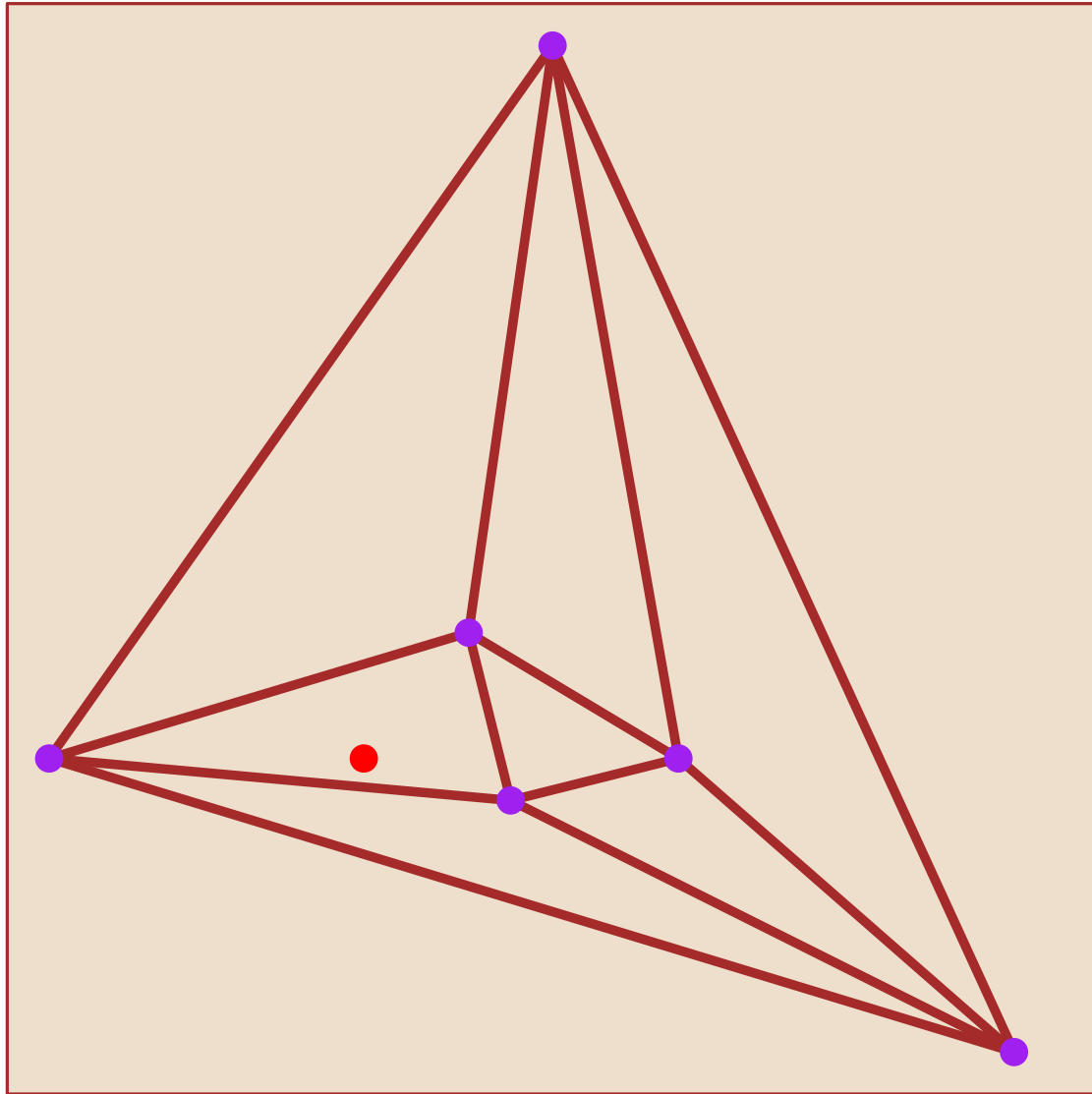
# Data structures to locate - the Delaunay tree



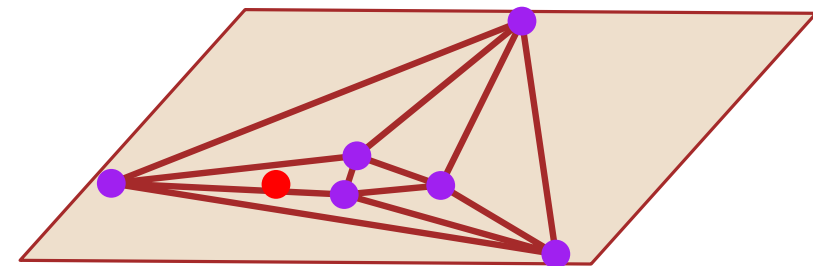
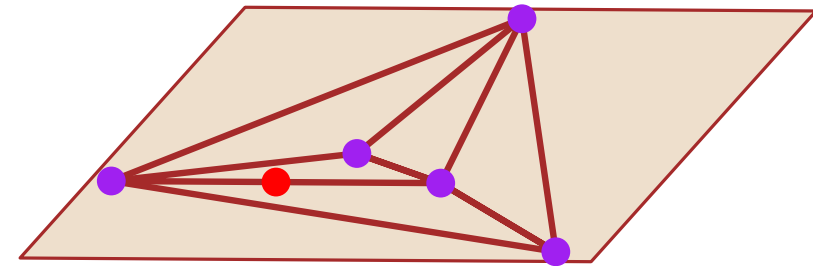
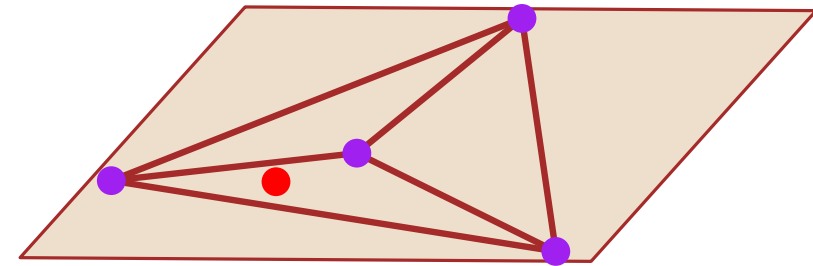
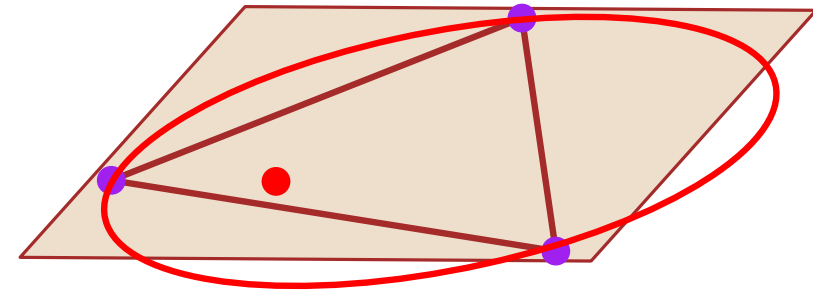
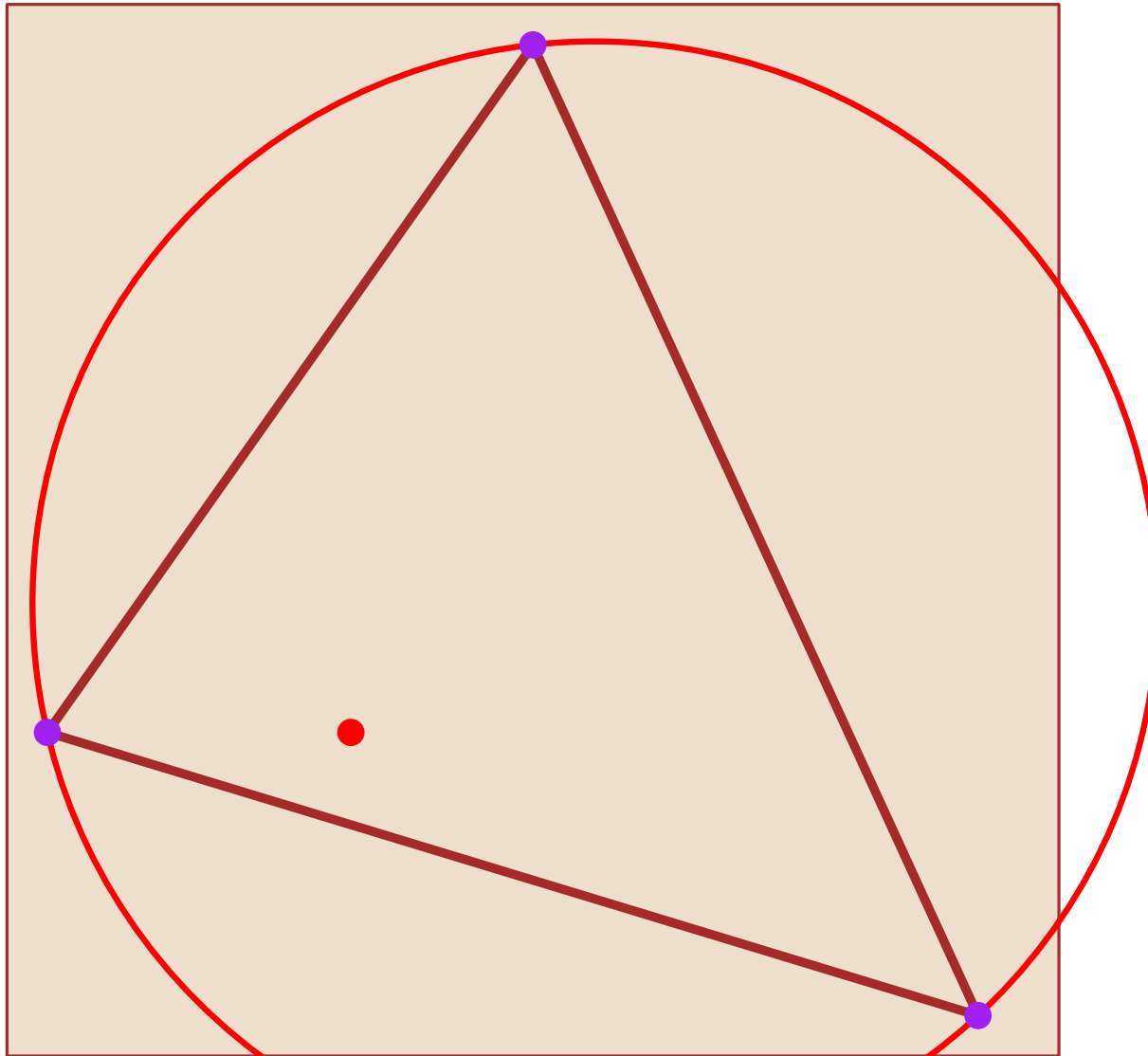
# Data structures to locate - the Delaunay tree



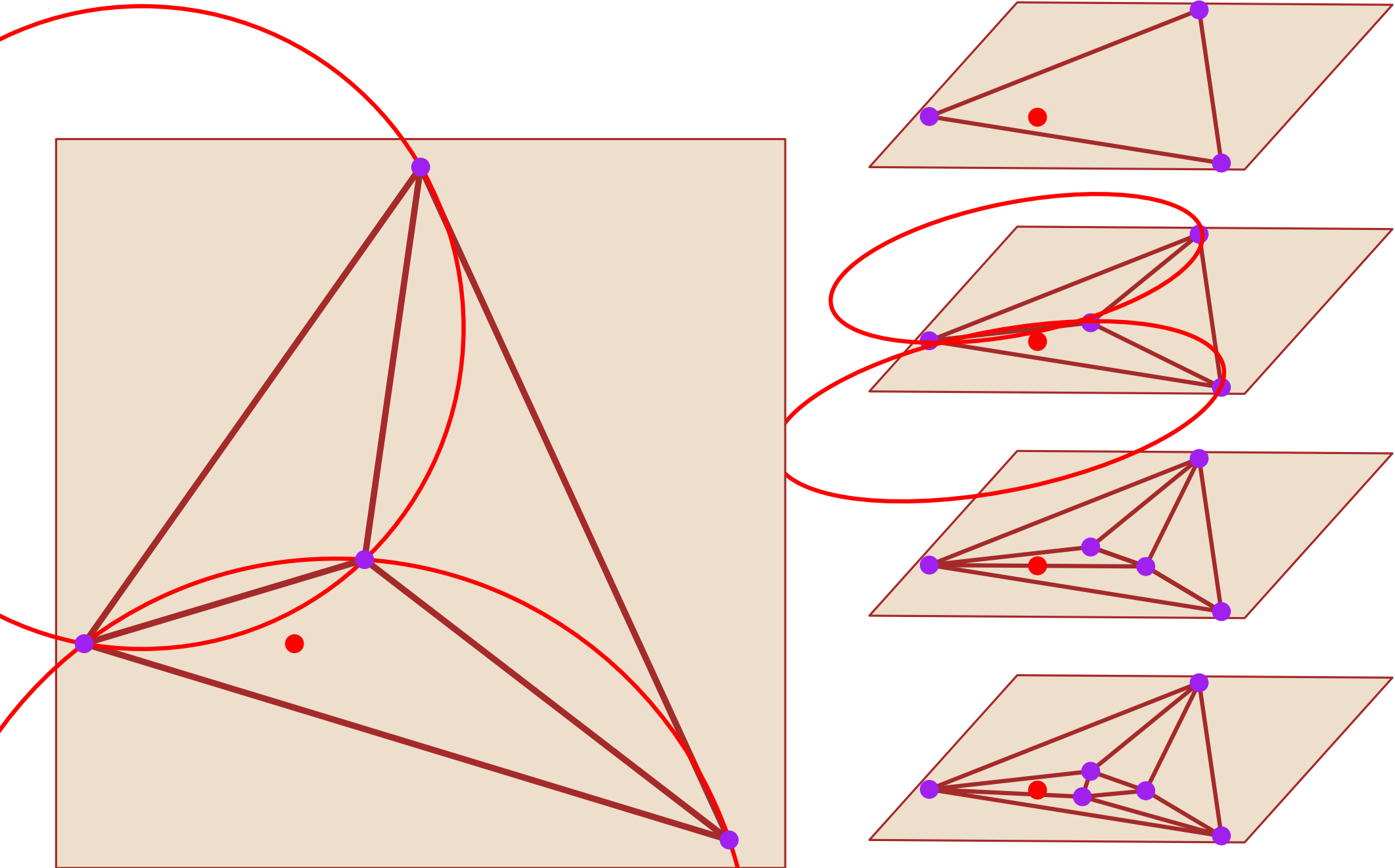
# Data structures to locate - the Delaunay tree



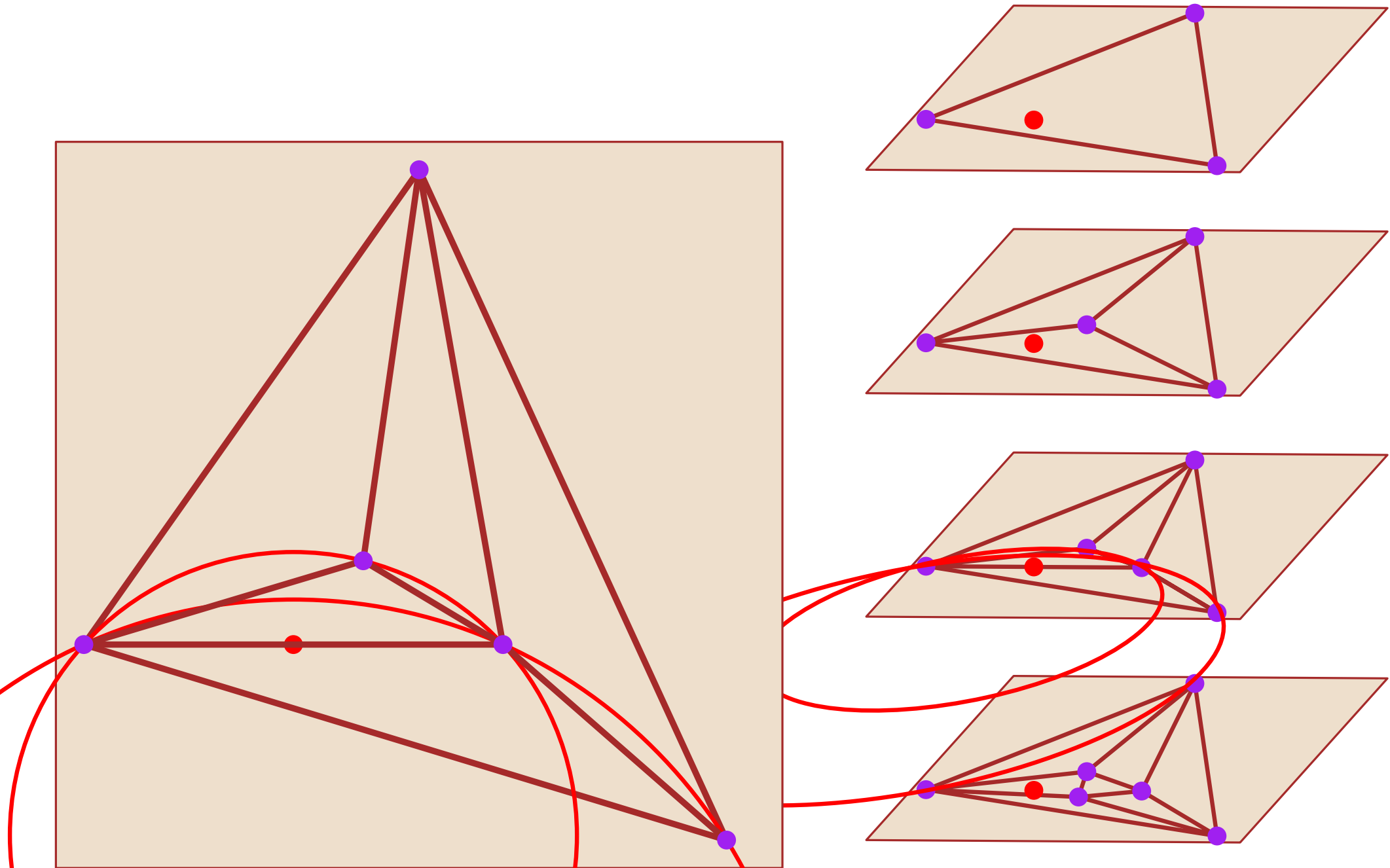
# Data structures to locate - the Delaunay tree



# Data structures to locate - the Delaunay tree

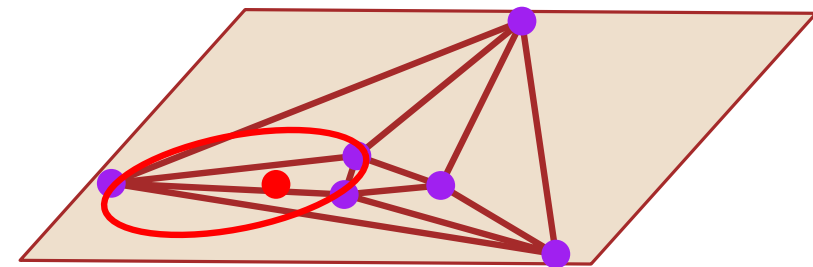
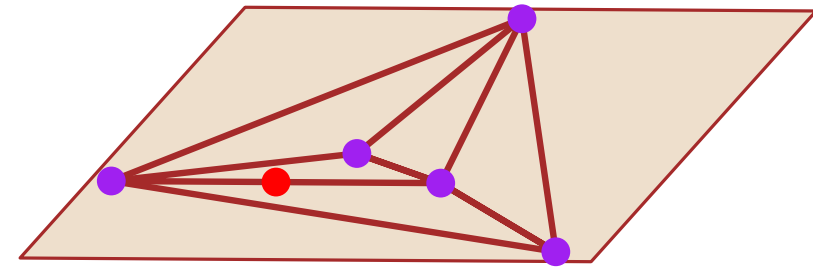
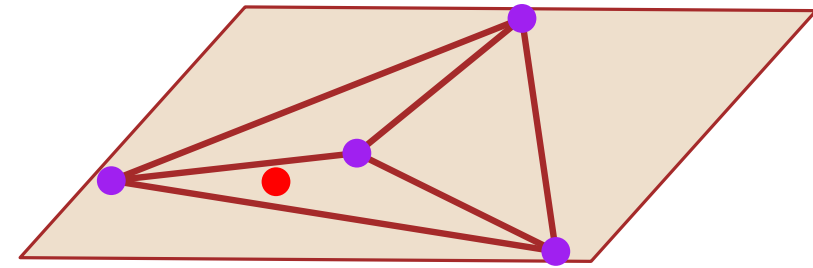
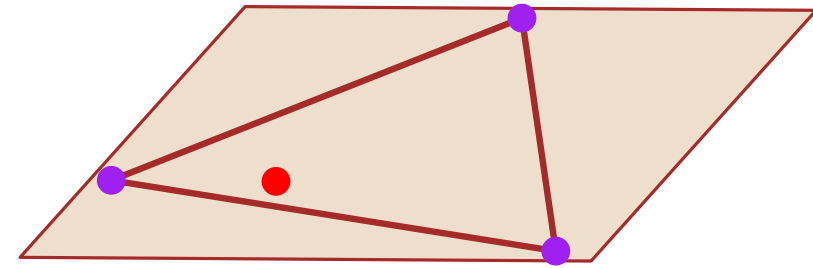
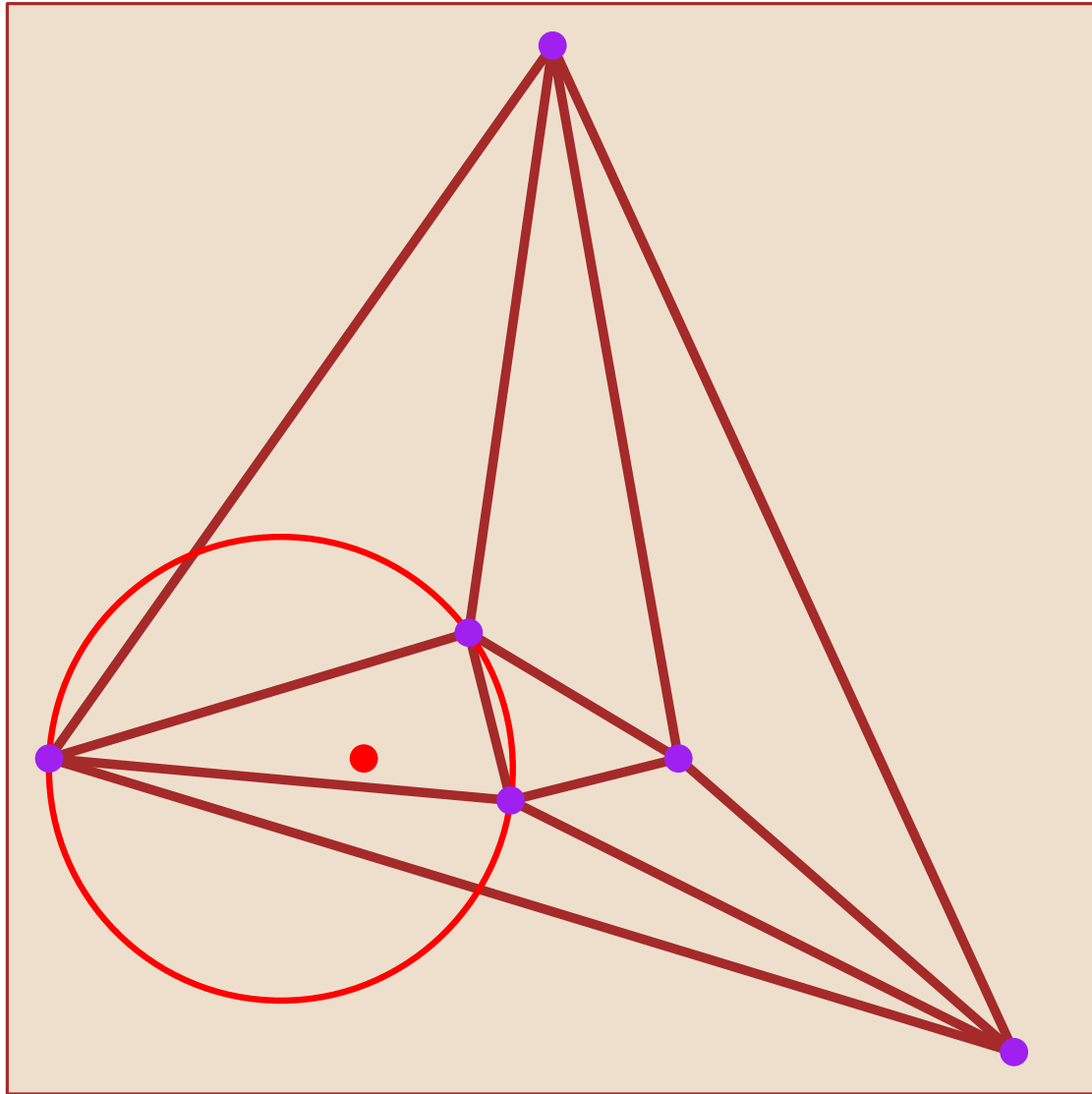


# Data structures to locate - the Delaunay tree





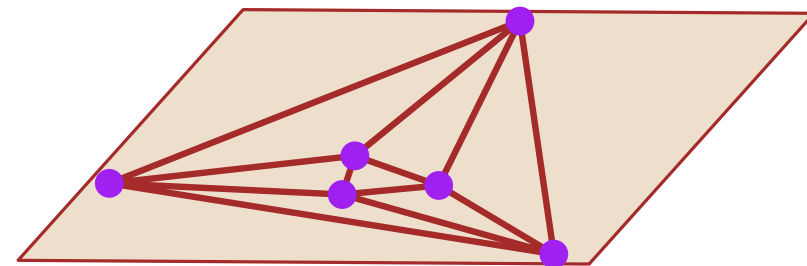
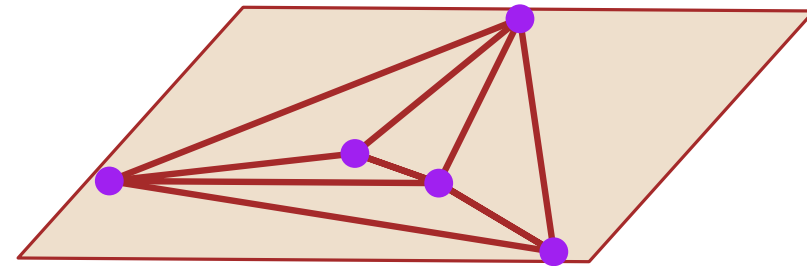
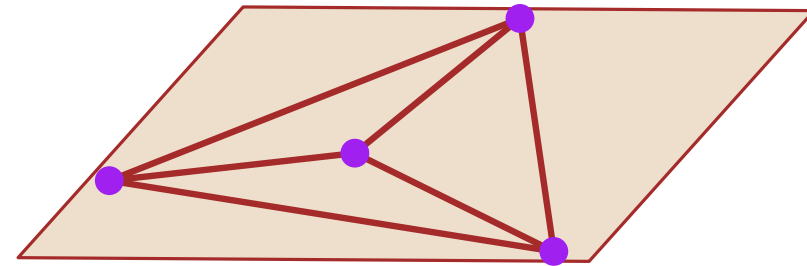
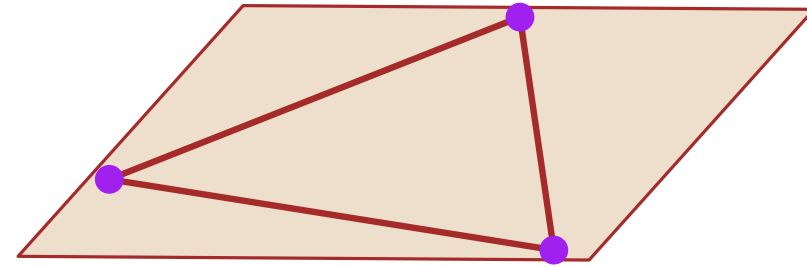
# Data structures to locate - the Delaunay tree



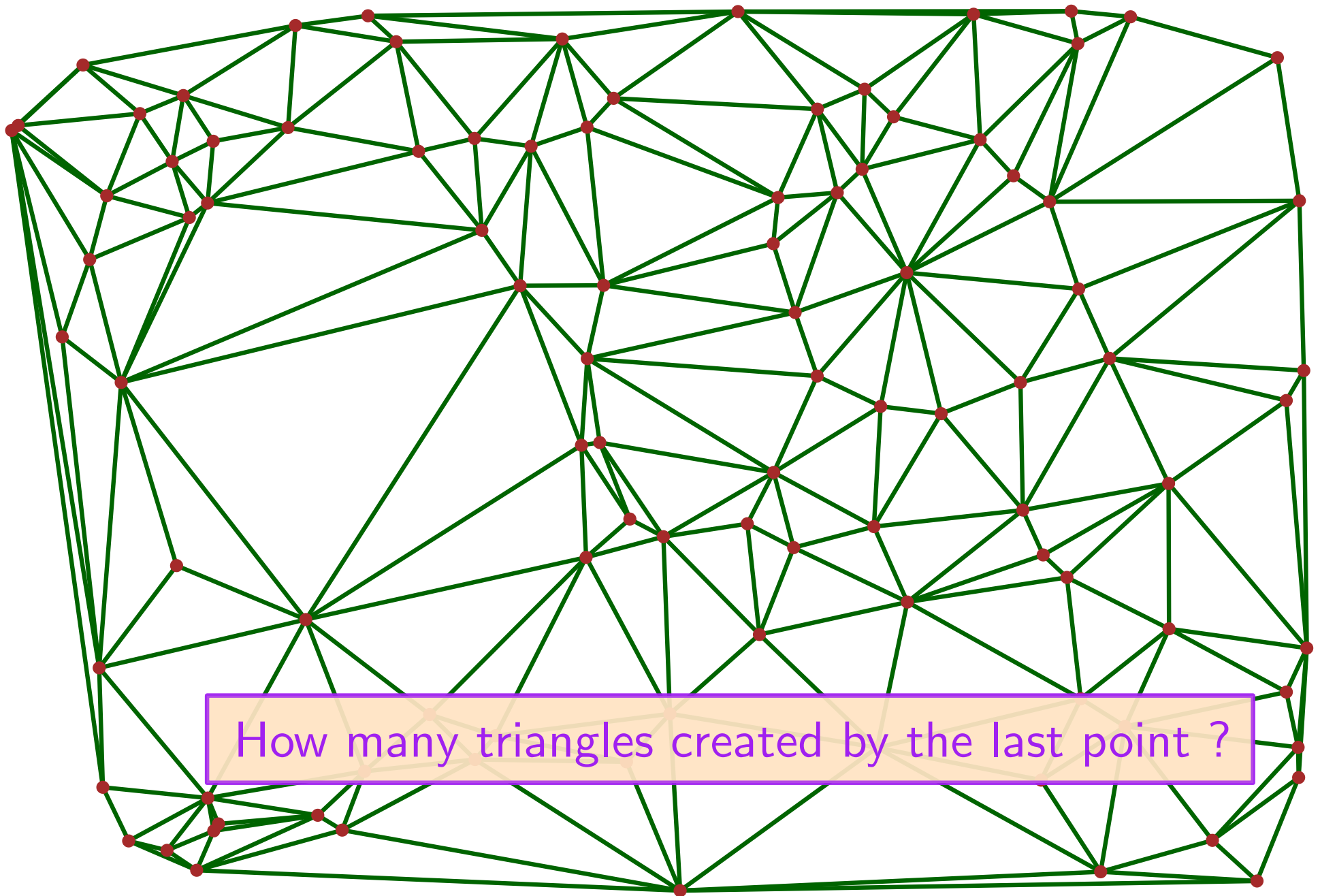
# Data structures to locate - the Delaunay tree

locate based on incircle predicate

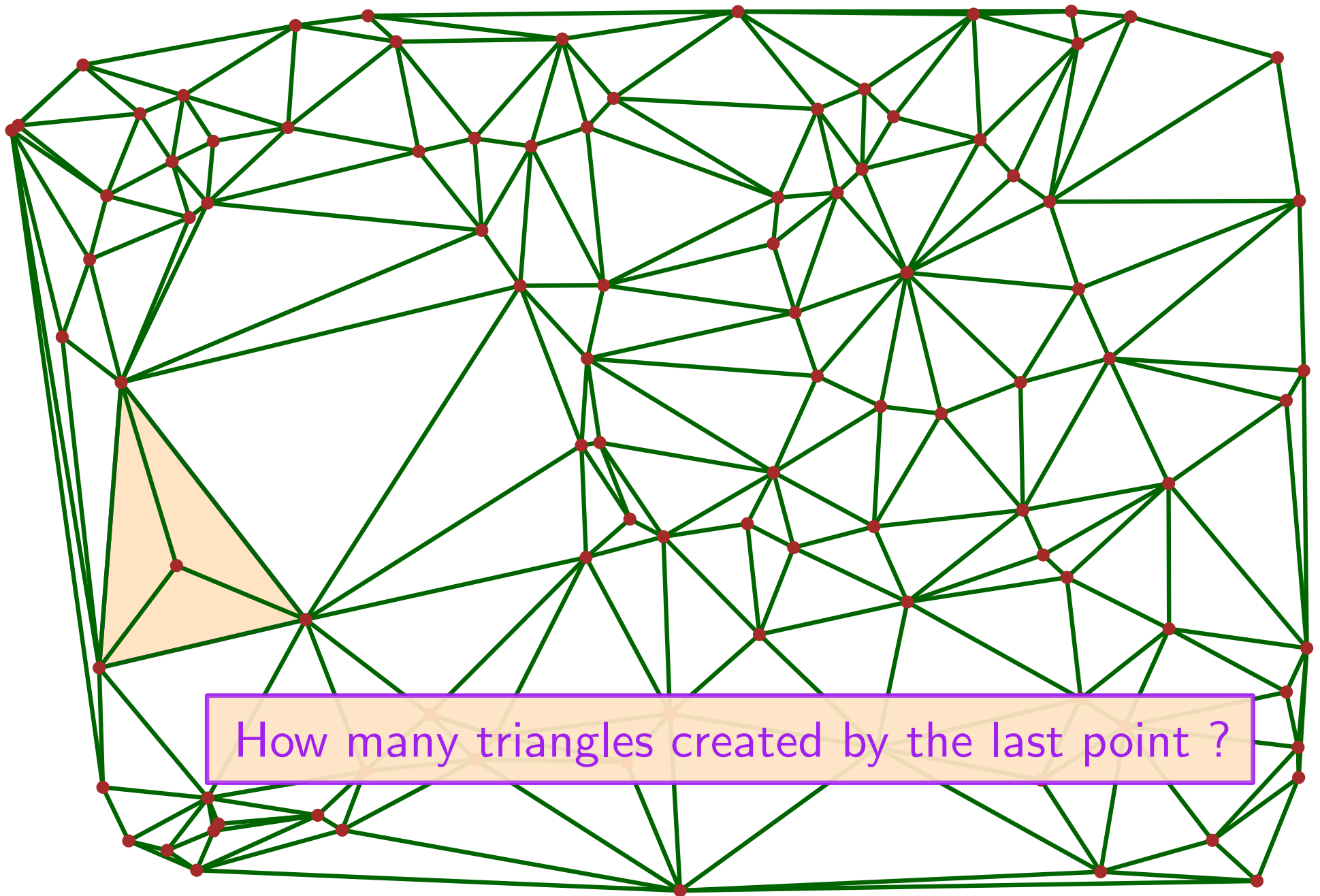
# triangles in the Delaunay tree



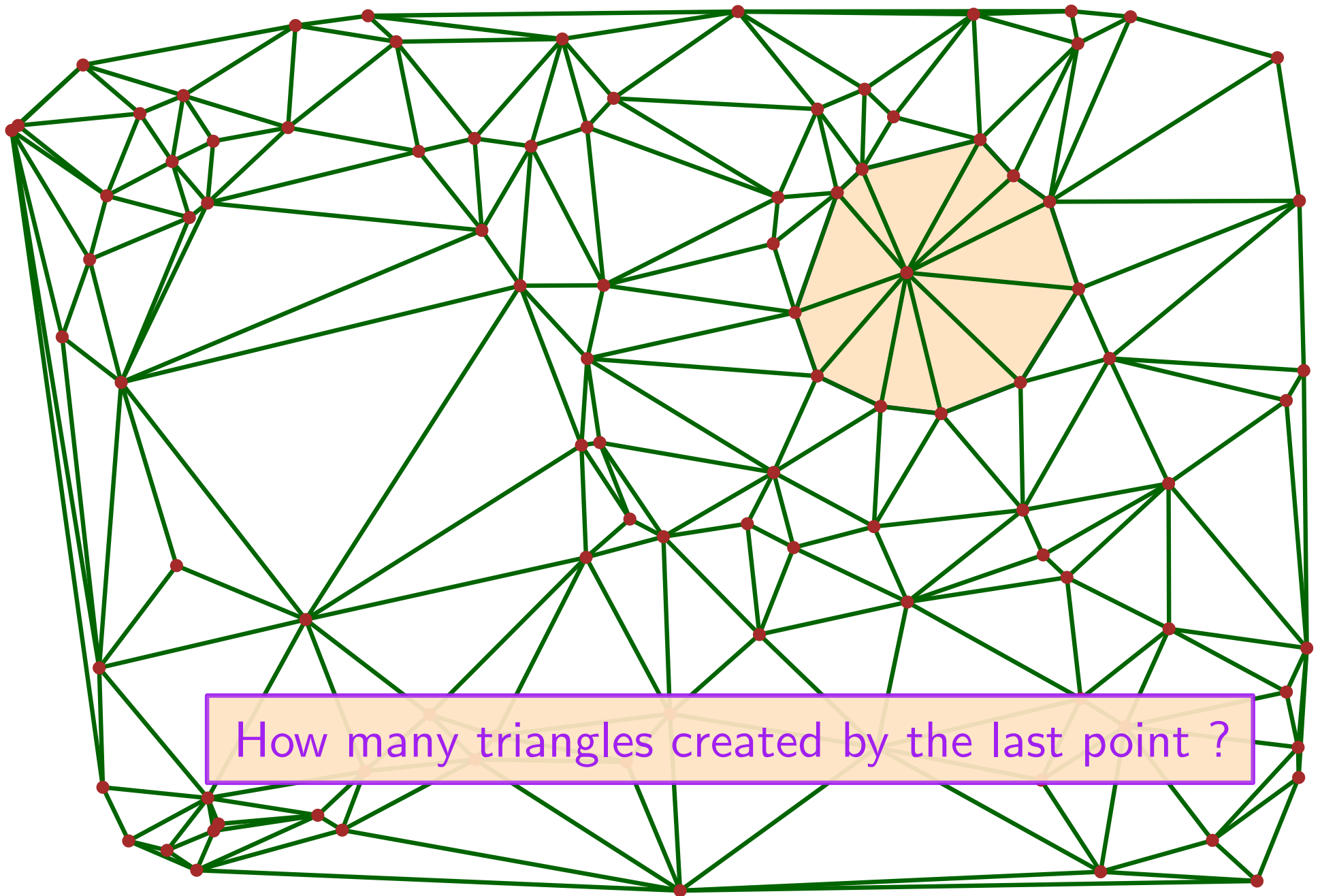
## Data structures to locate - the Delaunay tree



## Data structures to locate - the Delaunay tree



## Data structures to locate - the Delaunay tree

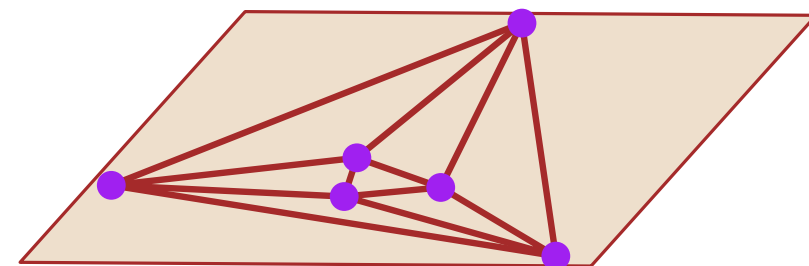
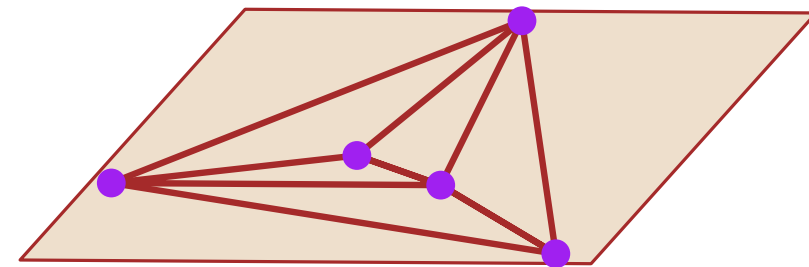
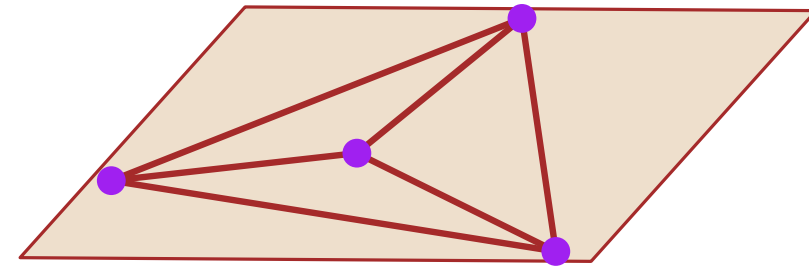
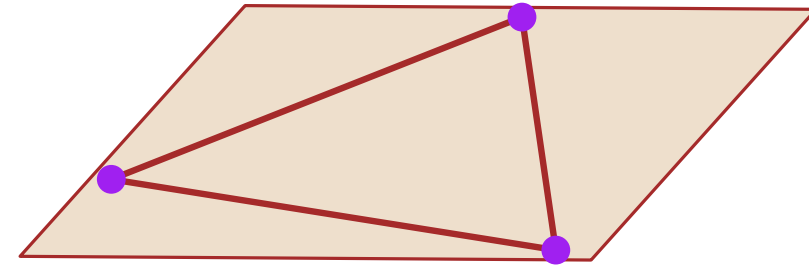


# Data structures to locate - the Delaunay tree

locate based on incircle predicate

# triangles in the Delaunay tree

$$= 6n \text{ (randomized)}$$





One word on robustness issues  
Basic incremental algorithm  
Locate by walk

**Locate using randomized data structures**

Delaunay tree

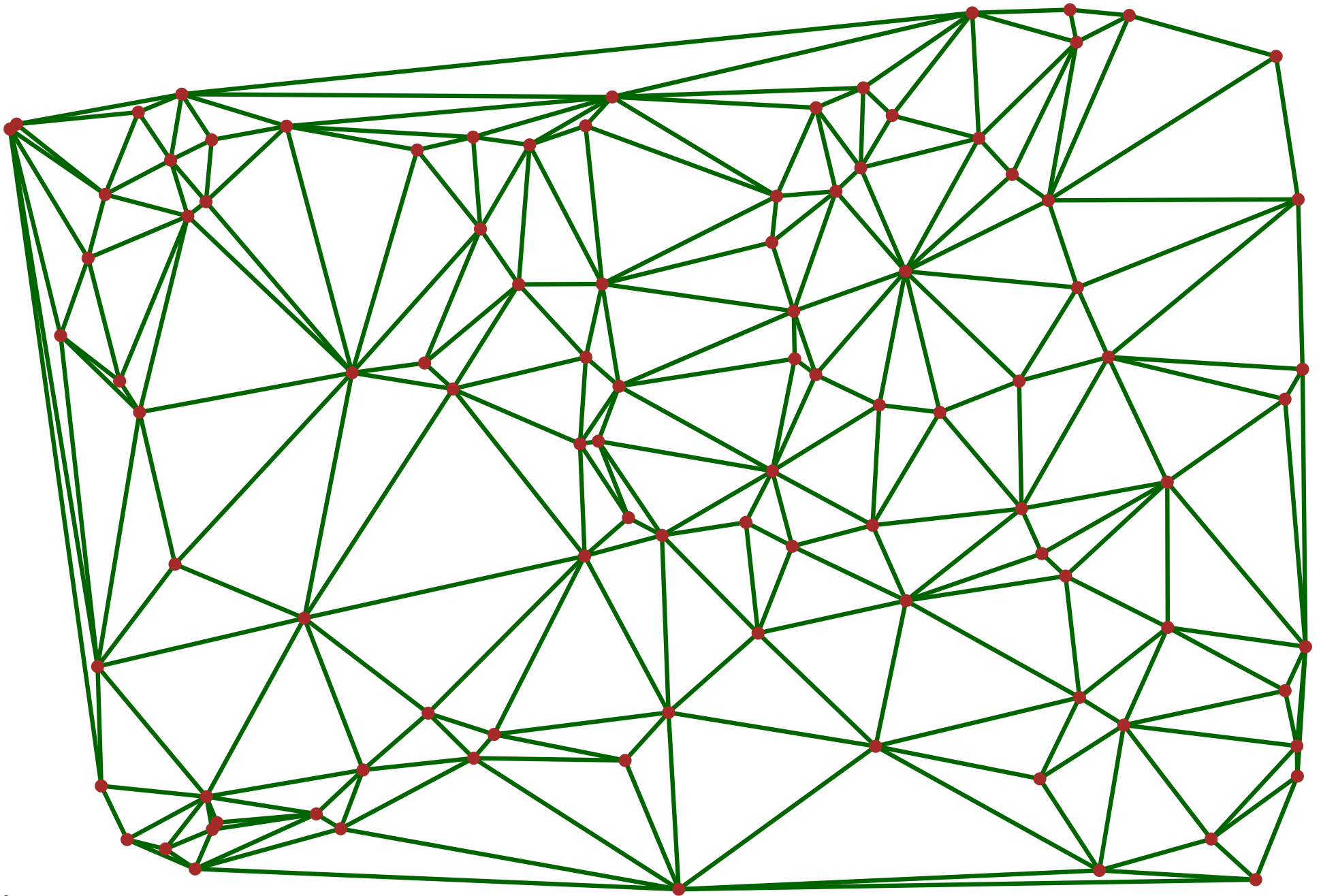
Delaunay hierarchy

Vertex removal in 2D  
Remarks on CGAL programming  
Conclusion

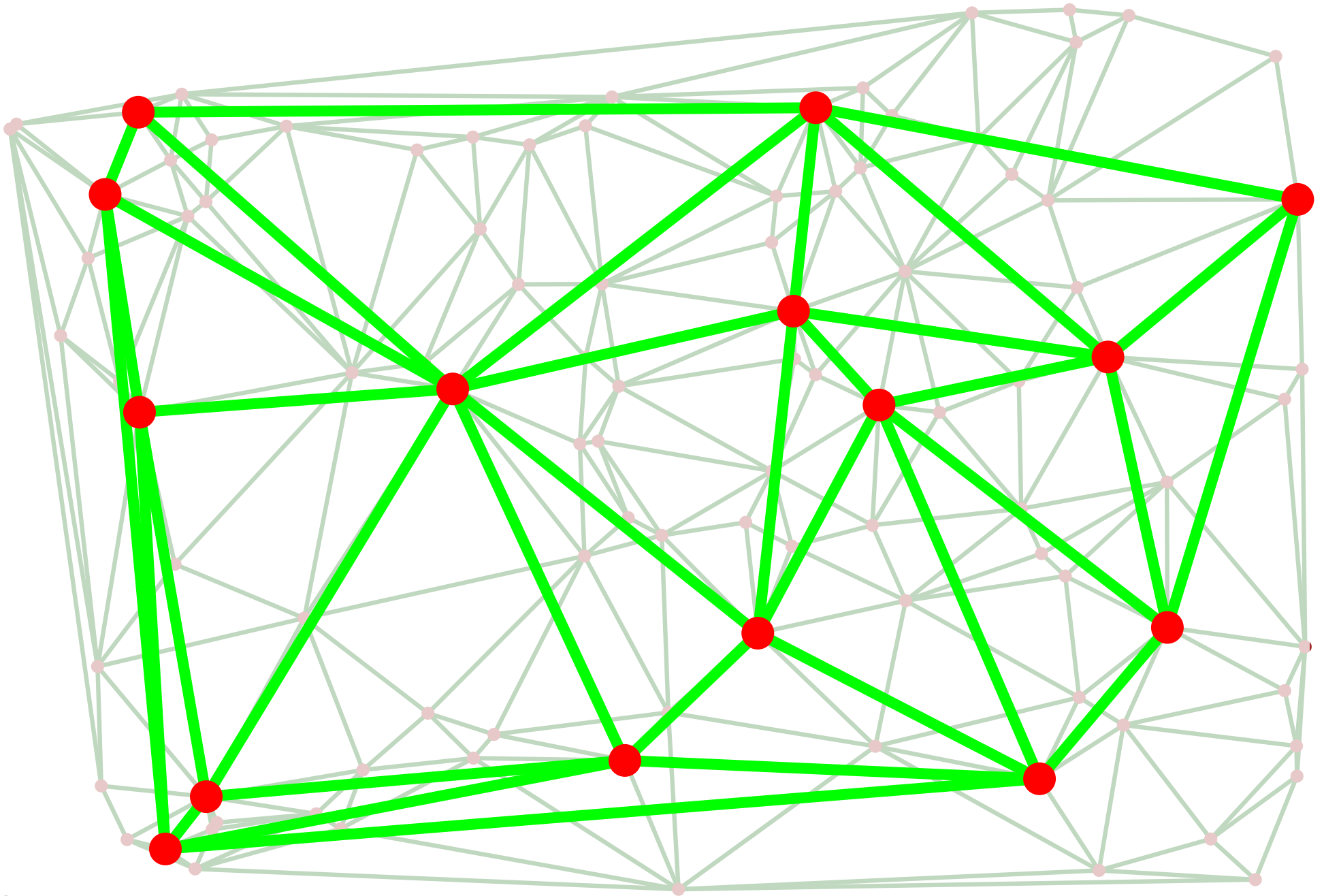
# Data structures to locate - the Delaunay hierarchy



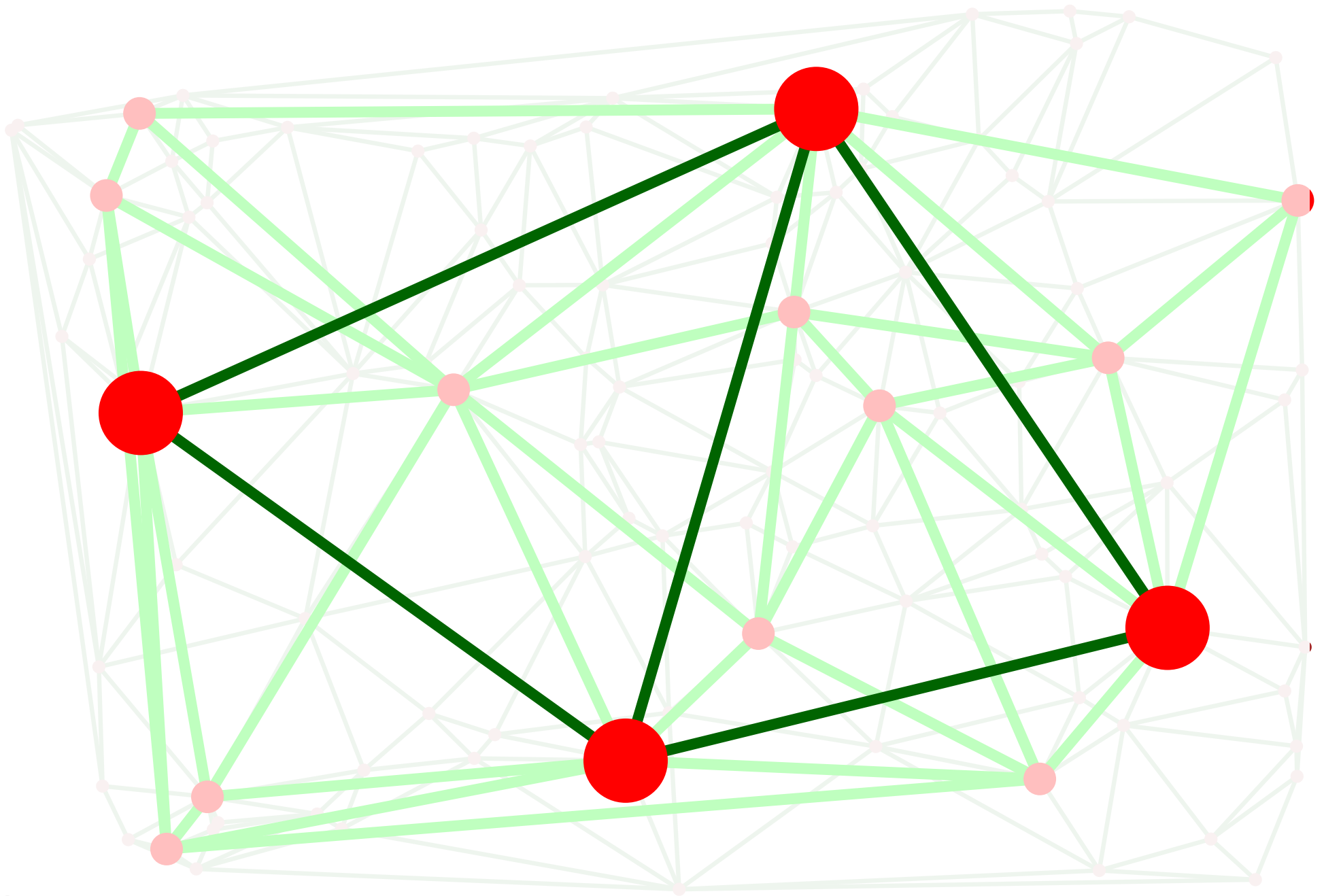
# Data structures to locate - the Delaunay hierarchy



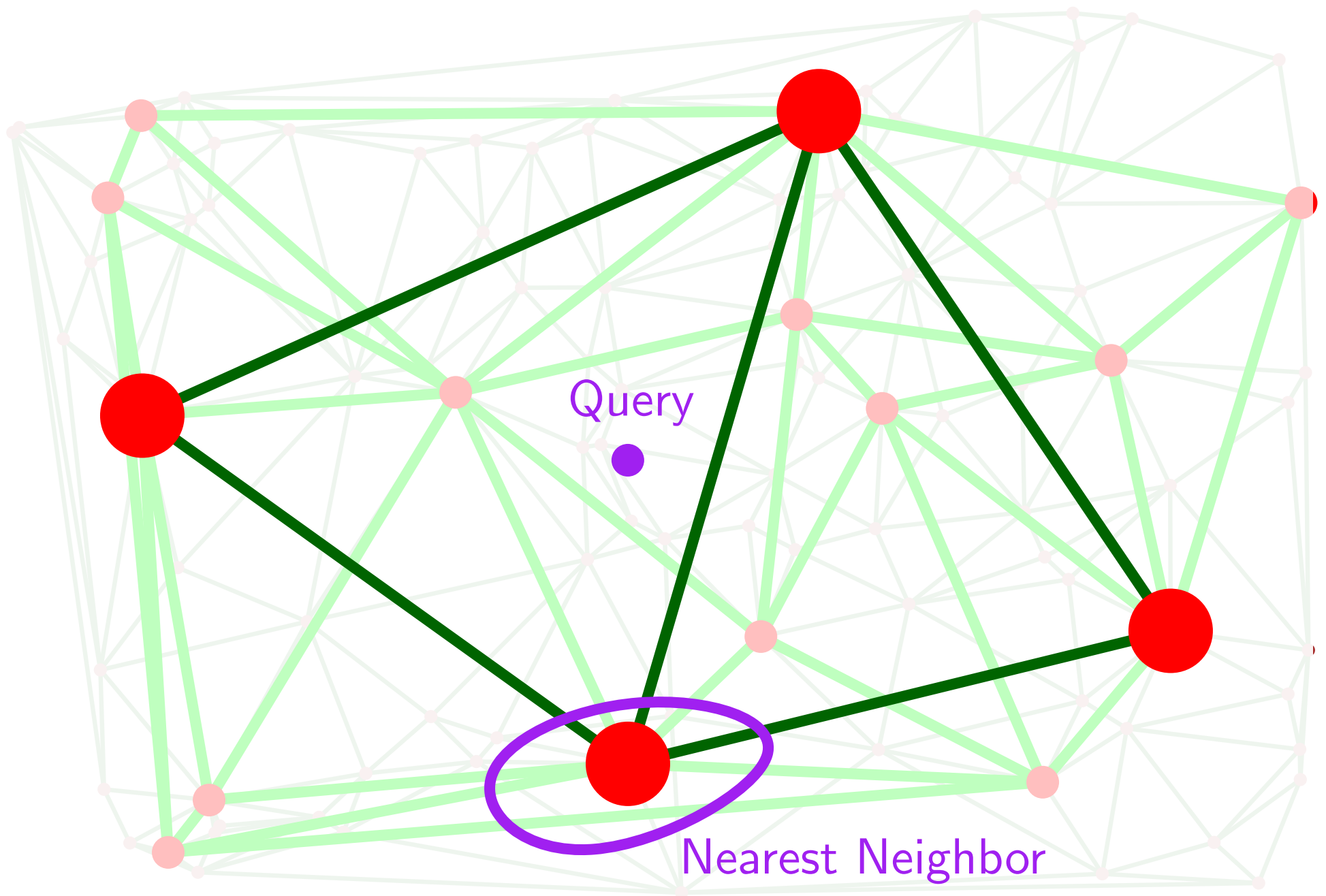
# Data structures to locate - the Delaunay hierarchy



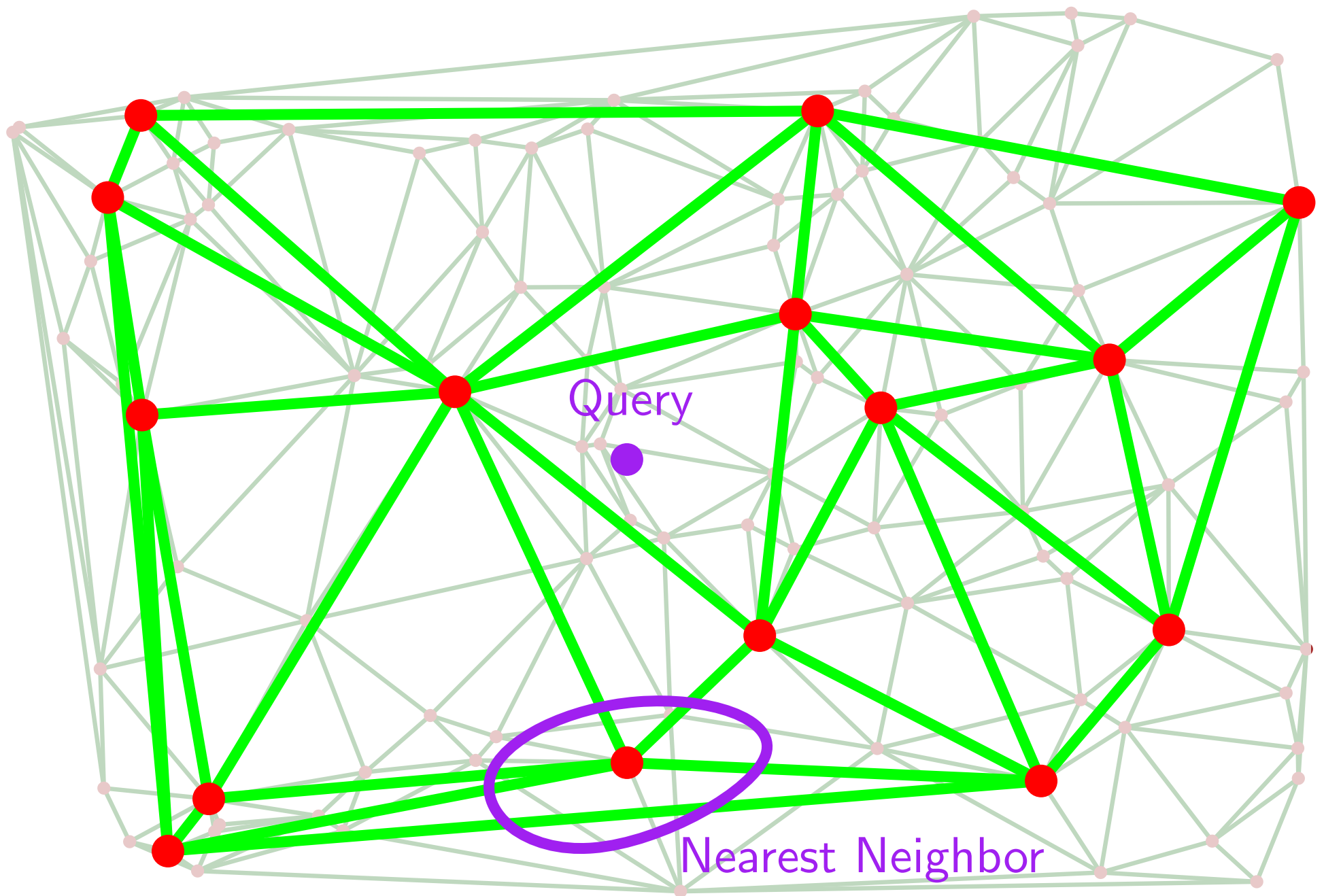
# Data structures to locate - the Delaunay hierarchy



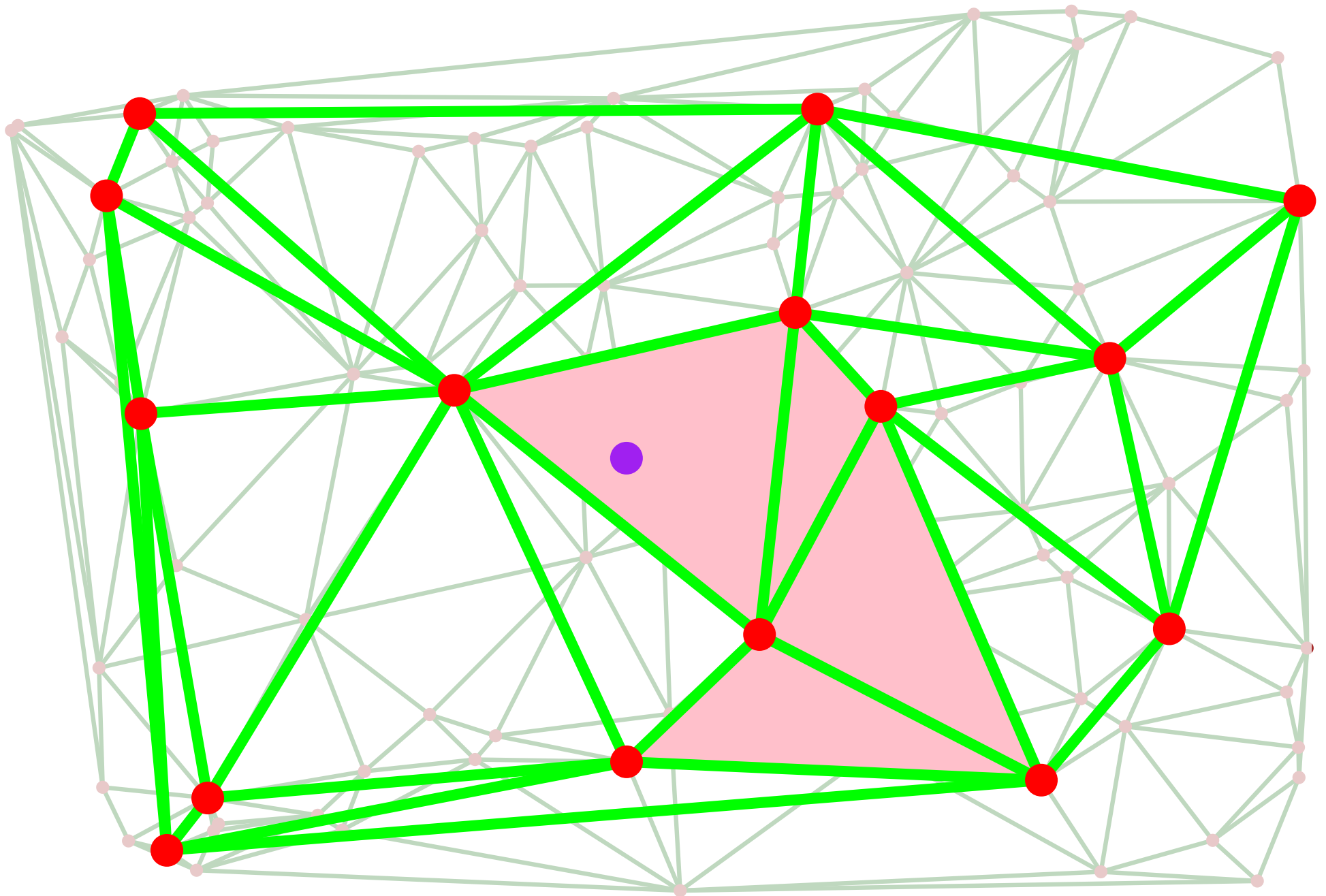
# Data structures to locate - the Delaunay hierarchy



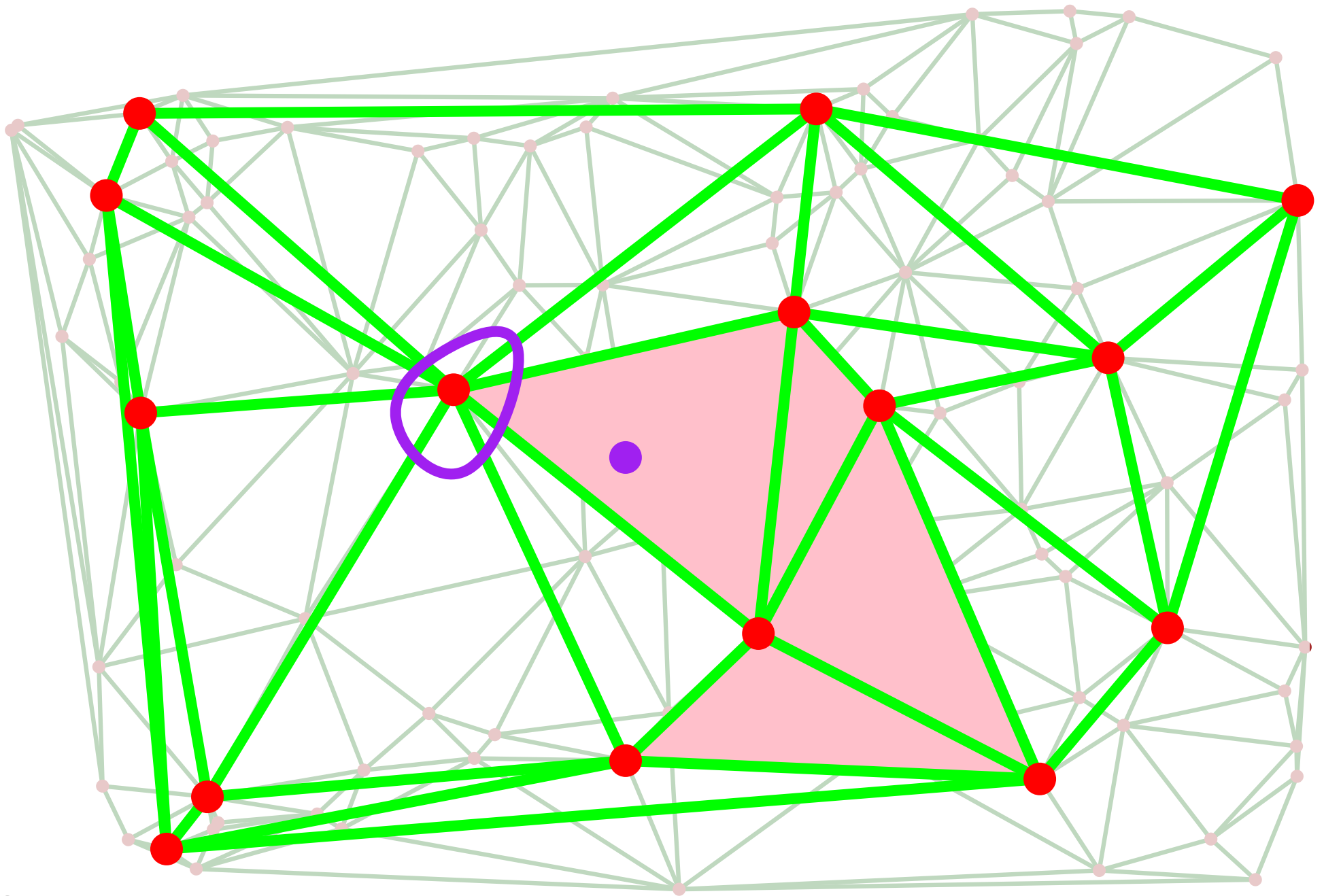
# Data structures to locate - the Delaunay hierarchy



# Data structures to locate - the Delaunay hierarchy

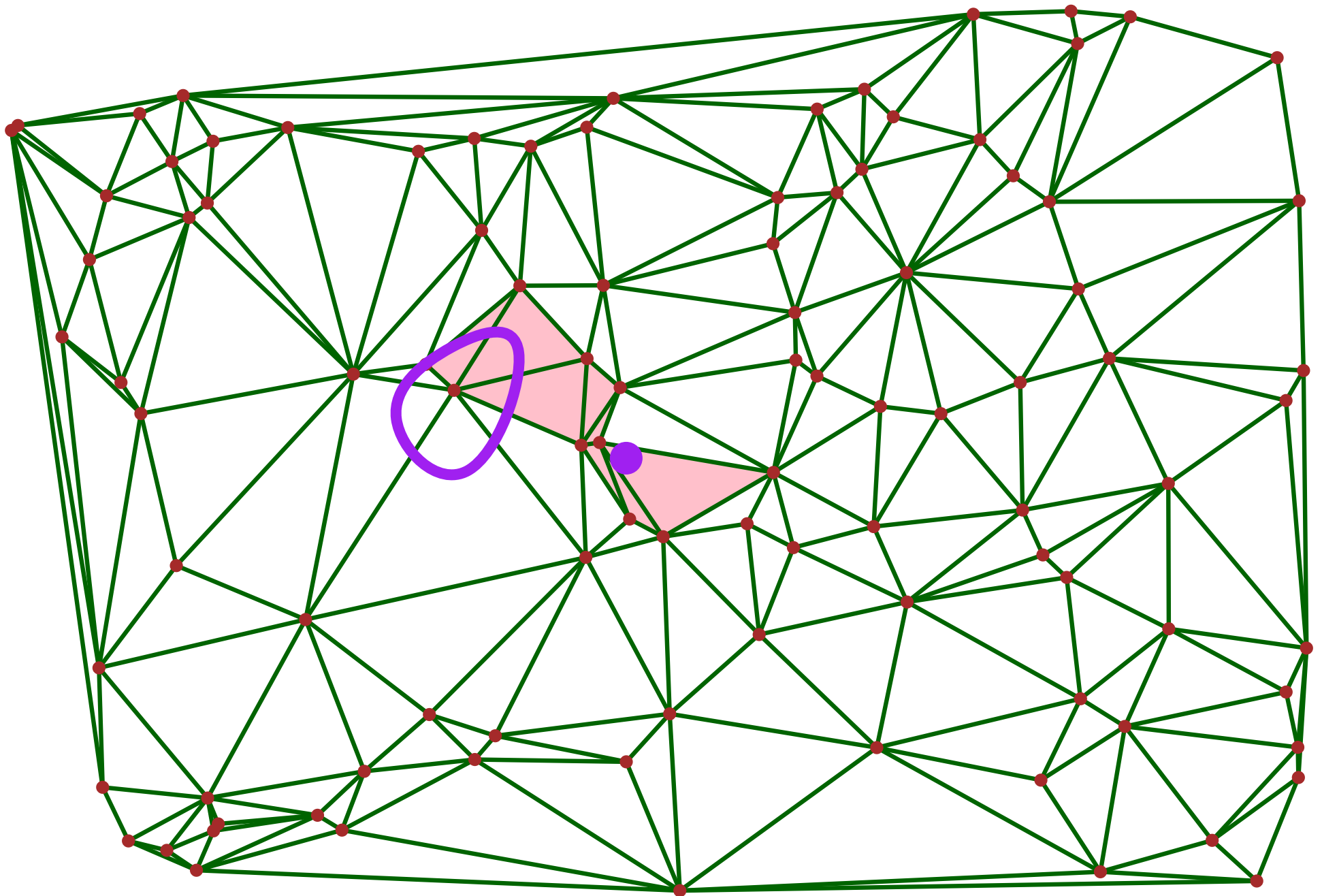


# Data structures to locate - the Delaunay hierarchy





# Data structures to locate - the Delaunay hierarchy





# Data structures to locate - the Delaunay hierarchy

## The Delaunay tree

locate based on incircle predicate

# triangles in the Delaunay tree

$= 6n$  (randomized)

# Data structures to locate - the Delaunay hierarchy

## The Delaunay hierarchy

based on orientation predicate

# triangles in the hierarchy

can be chosen

$$= 1.03 \times 2n \text{ (expected)}$$

## The Delaunay tree

locate based on incircle predicate

# triangles in the Delaunay tree

$$= 6n \text{ (randomized)}$$

# Data structures to locate - the Delaunay hierarchy

## The Delaunay hierarchy

based on orientation predicate

# triangles in the hierarchy

can be chosen

$$= 1.03 \times 2n \text{ (expected)}$$

## The Delaunay tree

locate based on incircle predicate

# triangles in the Delaunay tree

$$= 6n \text{ (randomized)}$$

$$O(n \log n)$$

# Data structures to locate - the Delaunay hierarchy

## The Delaunay hierarchy

based on orientation predicate

# triangles in the hierarchy

can be chosen

$$= 1.03 \times 2n \text{ (expected)}$$

2.3 seconds

## The Delaunay tree

locate based on incircle predicate

# triangles in the Delaunay tree

$$= 6n \text{ (randomized)}$$

17 seconds

50000 random points (original benchmarks in 2000).



One word on robustness issues  
Basic incremental algorithm  
Locate by walk

**Locate using randomized data structures**

Delaunay tree

Delaunay hierarchy

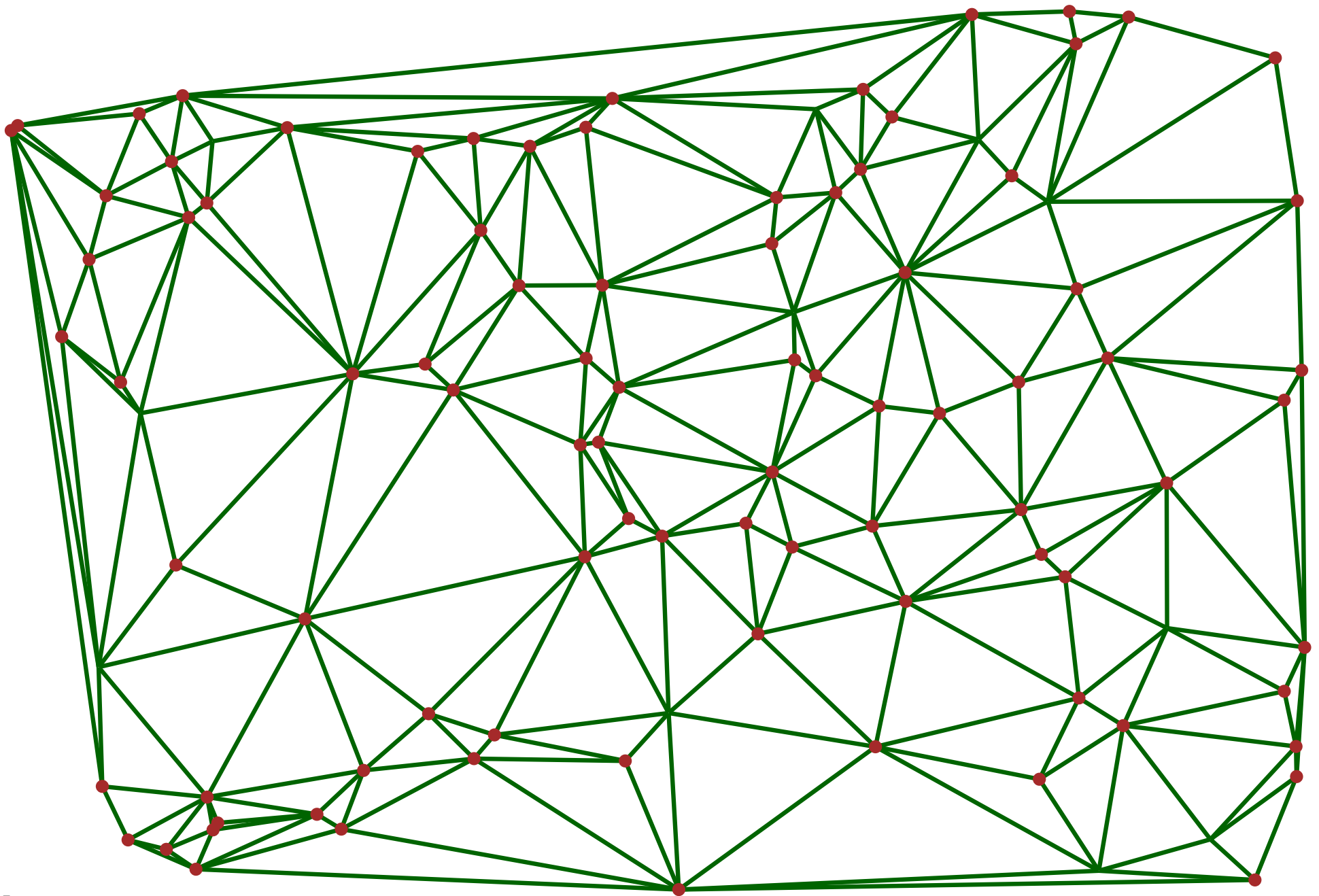
Biased randomized insertion order

Vertex removal in 2D

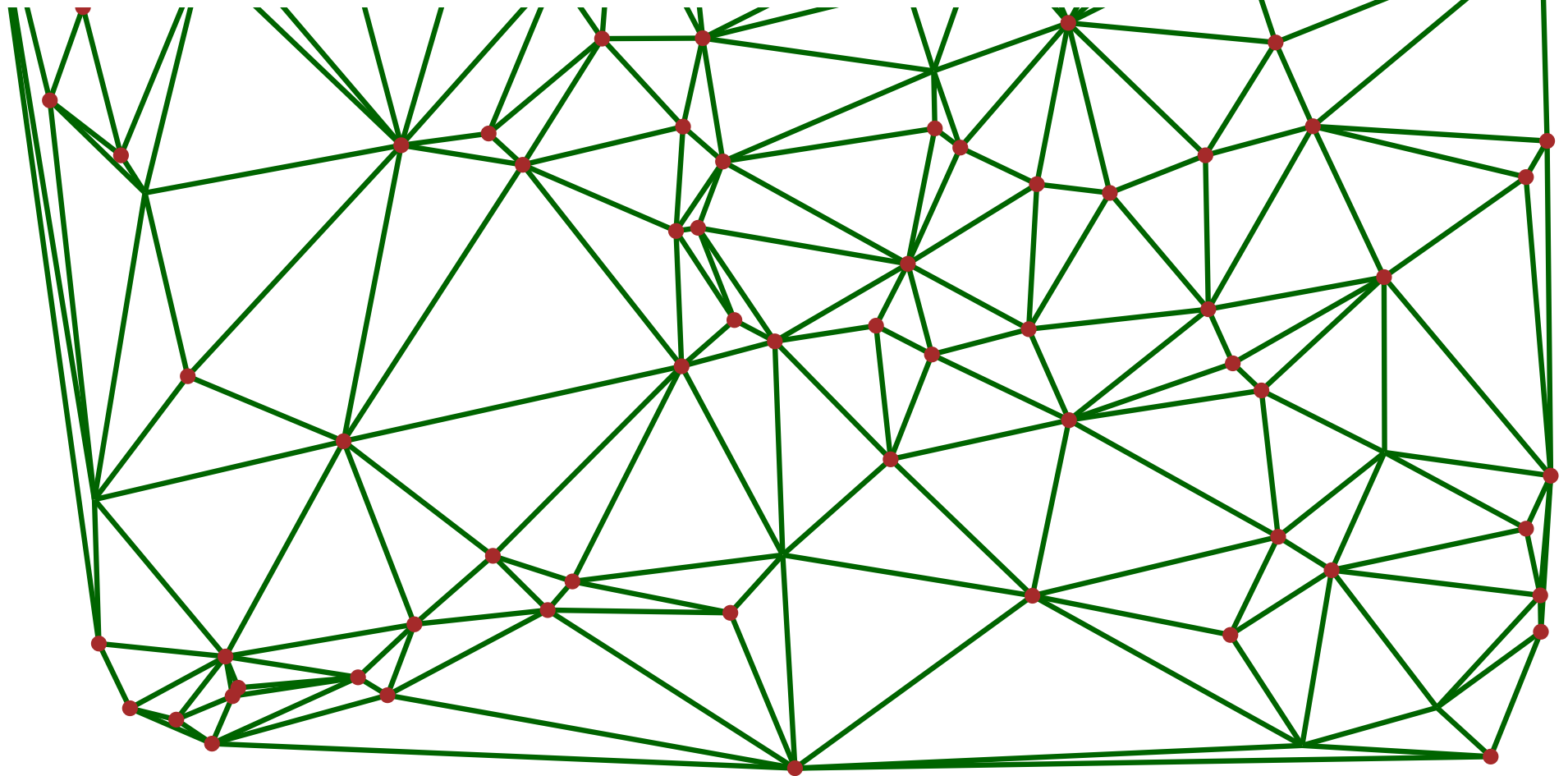
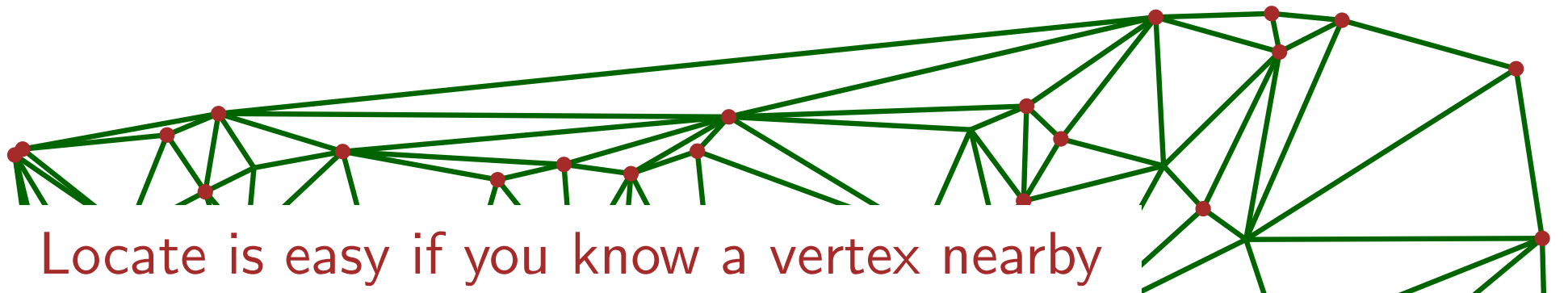
Remarks on CGAL programming

Conclusion

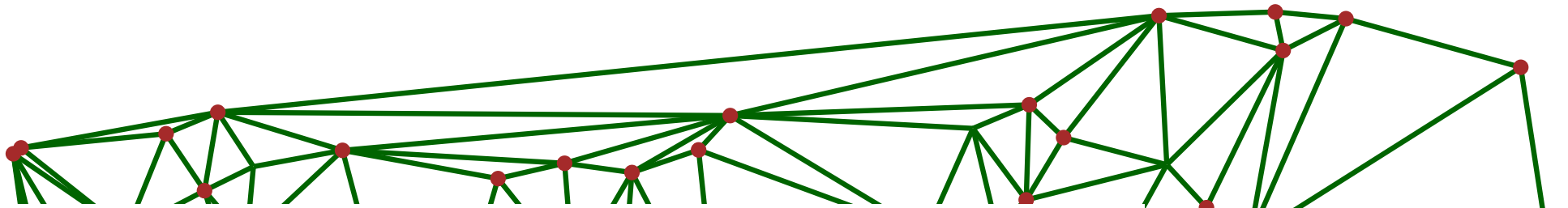
# Data structures to locate - biased random insertion order



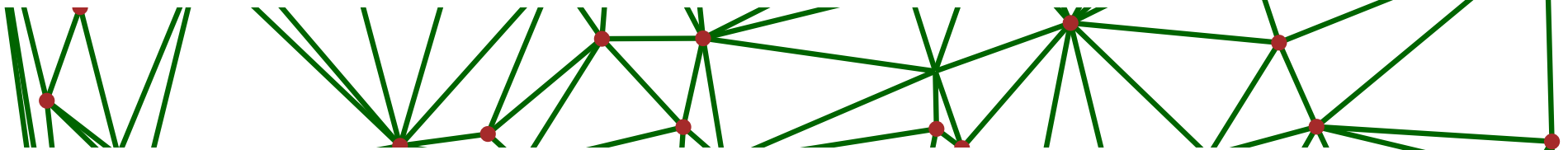
# Data structures to locate - biased random insertion order



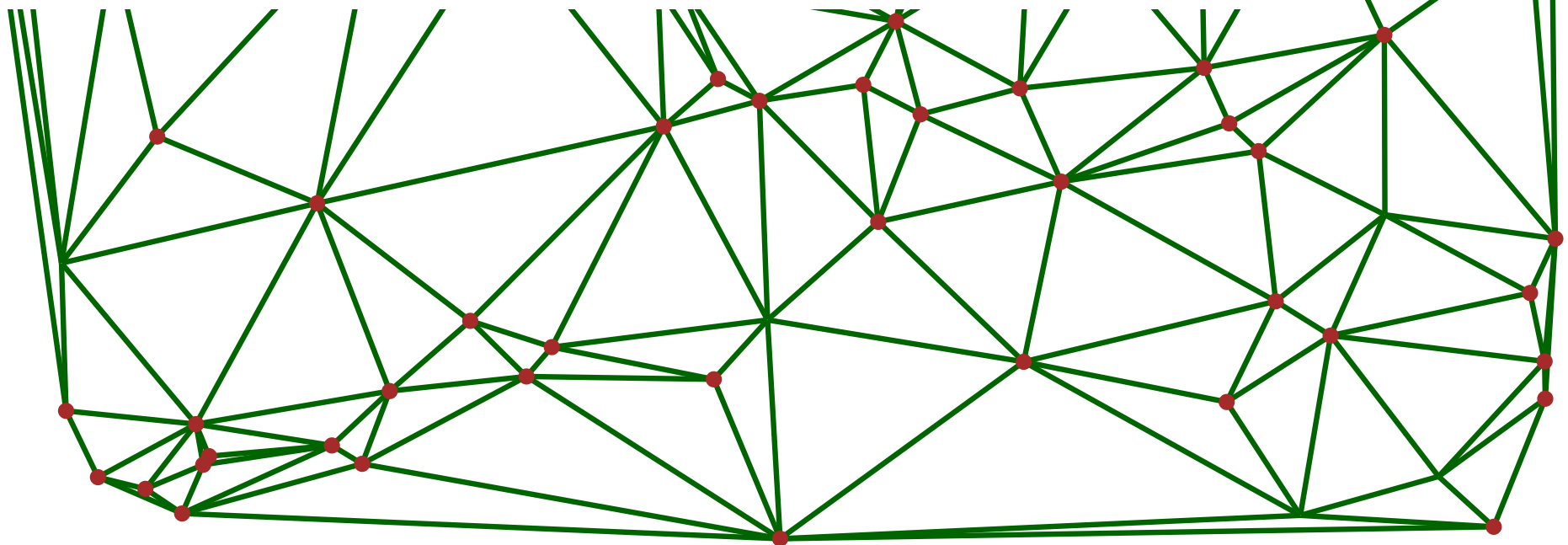
# Data structures to locate - biased random insertion order



Locate is easy if you know a vertex nearby

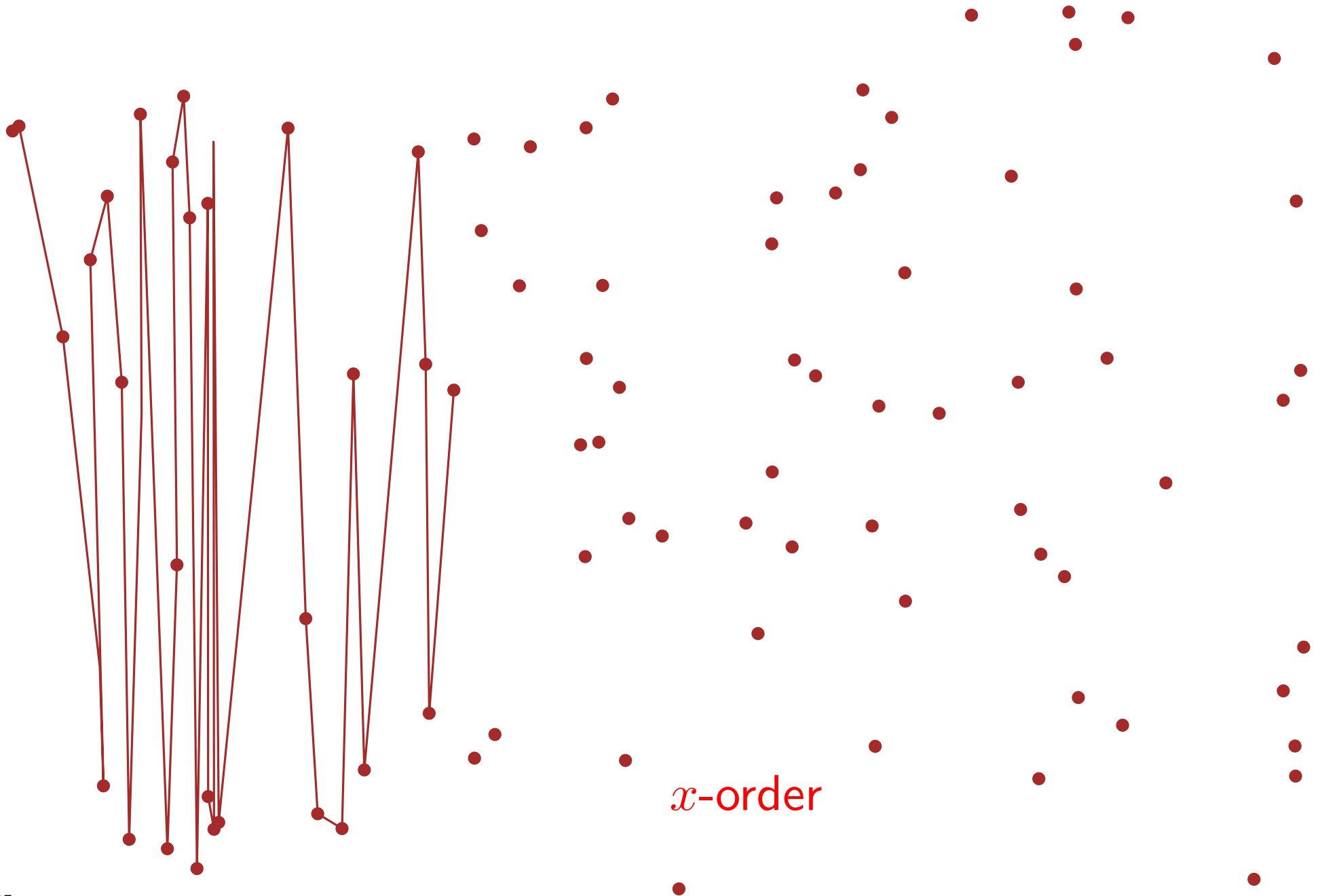


Natural idea: sort the points, locate from previous

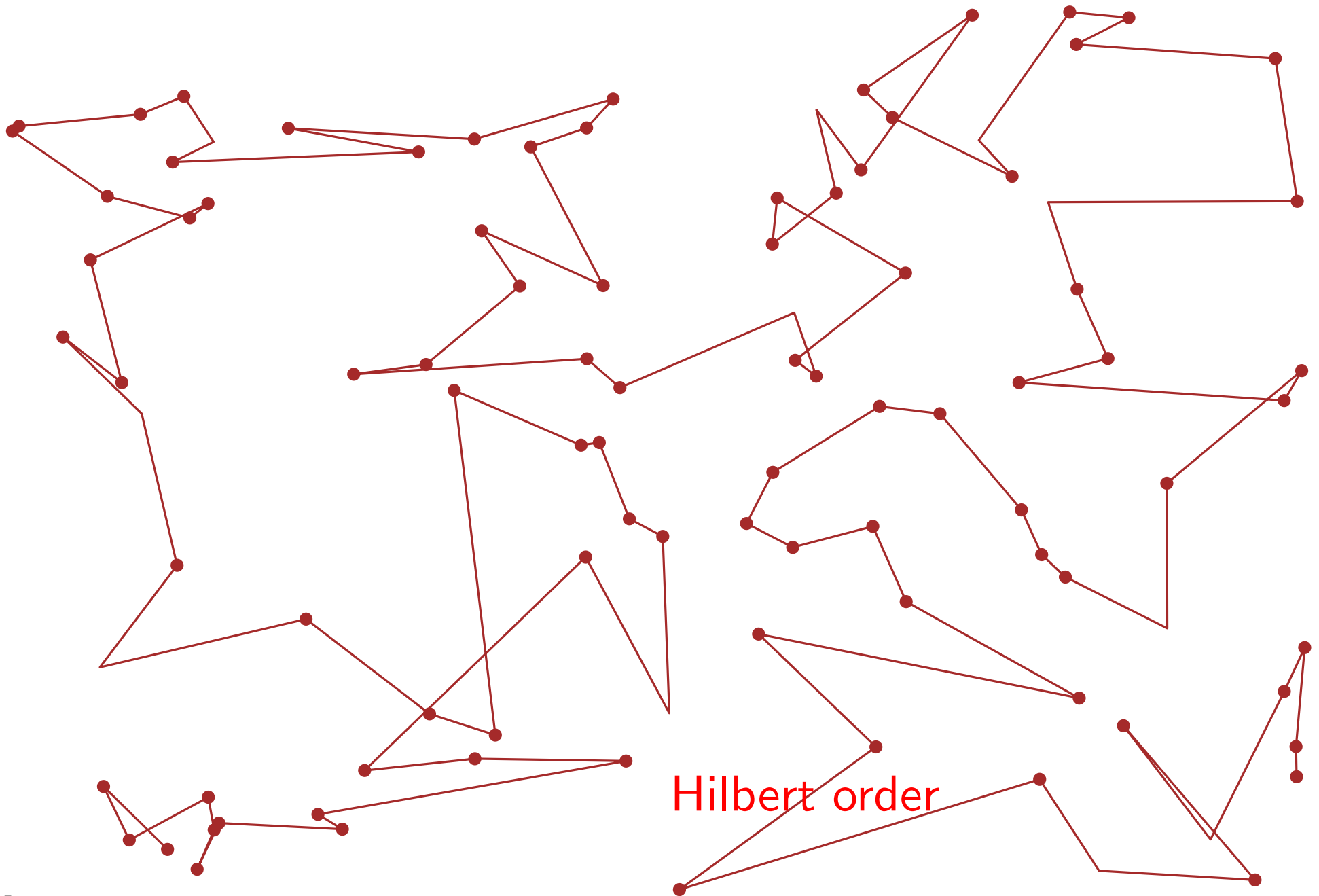




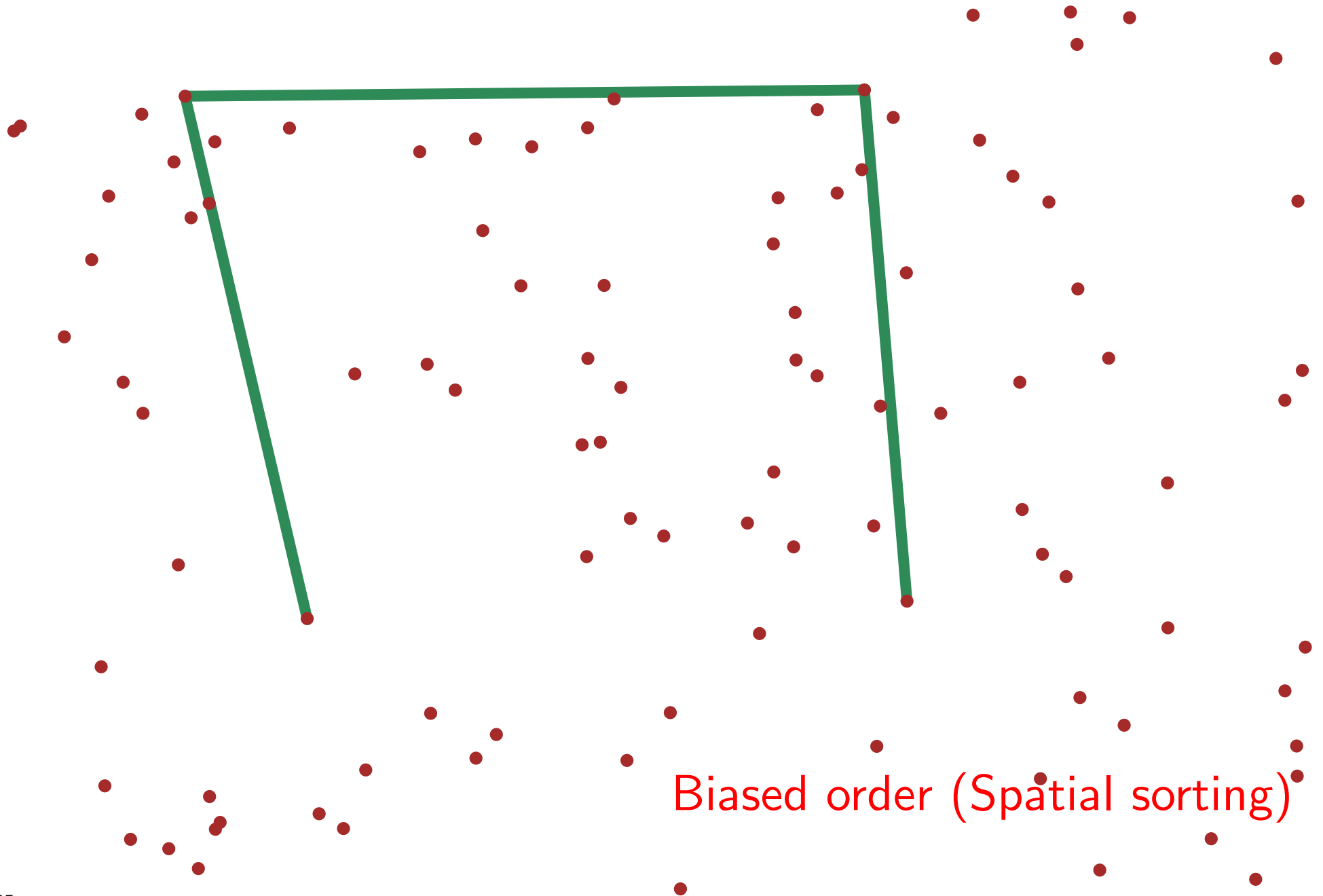
# Data structures to locate - biased random insertion order



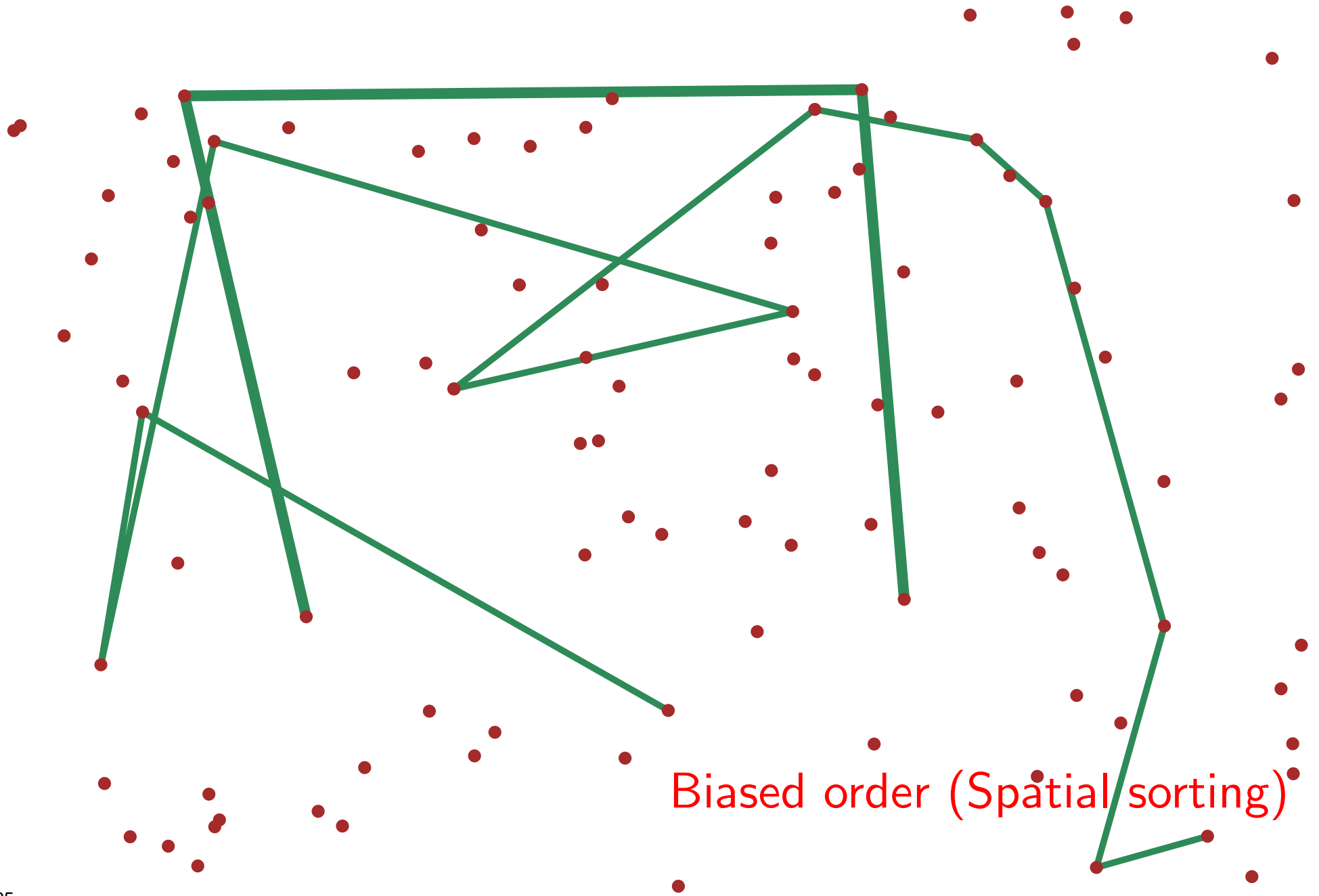
# Data structures to locate - biased random insertion order



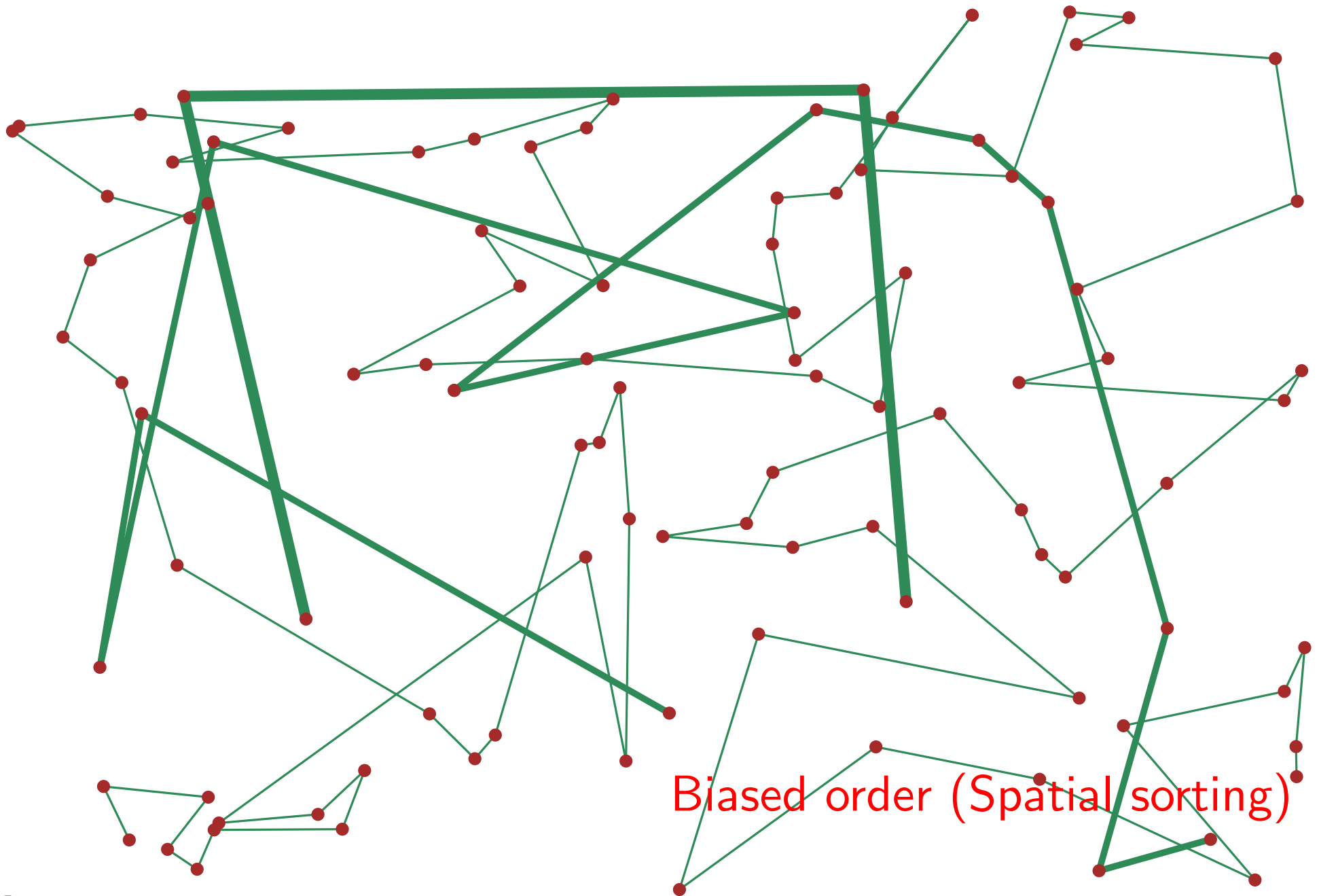
# Data structures to locate - biased random insertion order



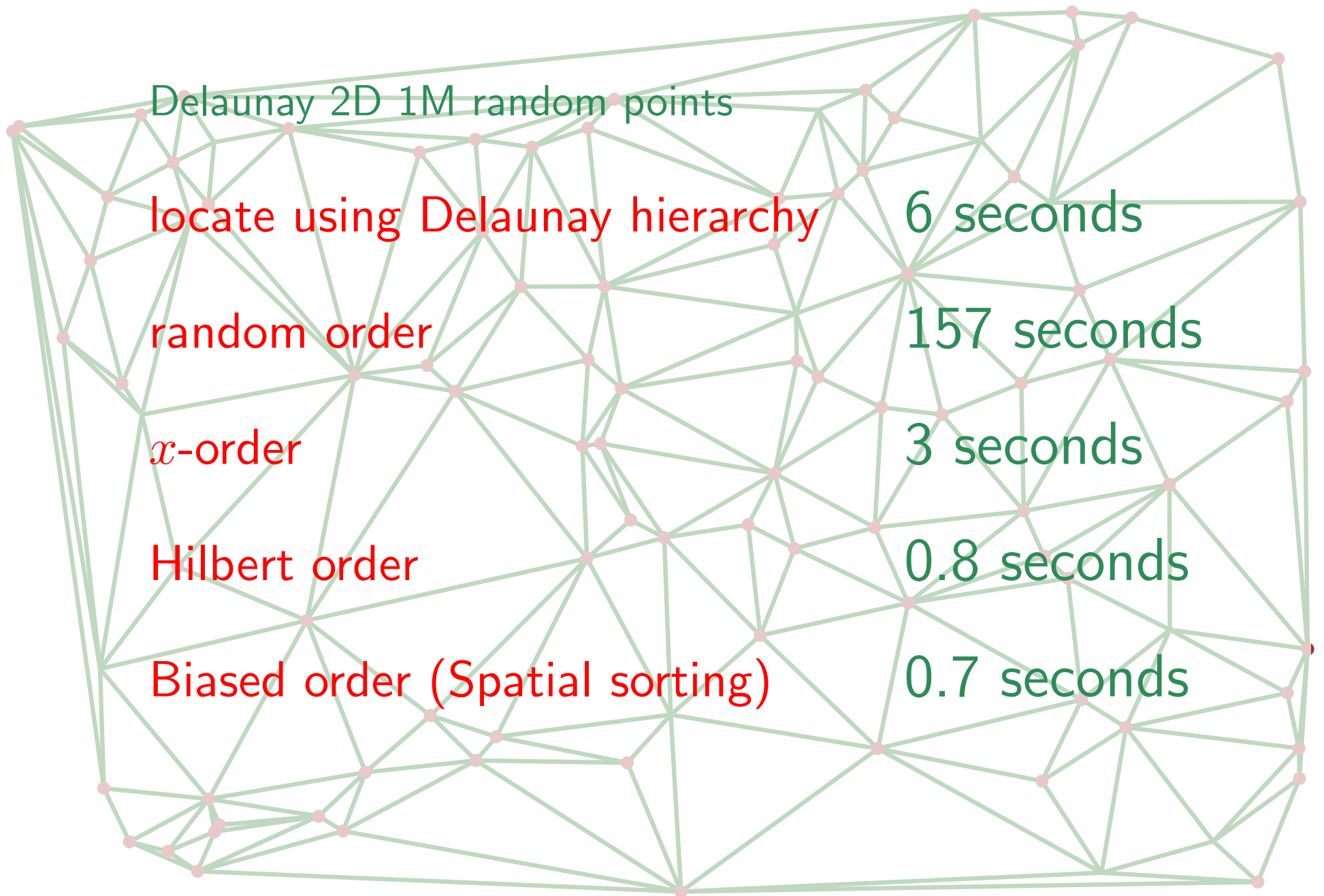
# Data structures to locate - biased random insertion order



# Data structures to locate - biased random insertion order



# Data structures to locate - biased random insertion order



# Data structures to locate - biased random insertion order

Delaunay 2D 100K parabola points

locate using Delaunay hierarchy

0.3 seconds

random order

128 seconds

$x$ -order

632 seconds

Hilbert order

46 seconds

Biased order (Spatial sorting)

0.3 seconds

## Data structures to locate

### Construction of Delaunay 10 M random points

Delaunay tree  $\sim 10$  mn (estimate)

Delaunay hierarchy 90 seconds

Biased random order 10.6 seconds



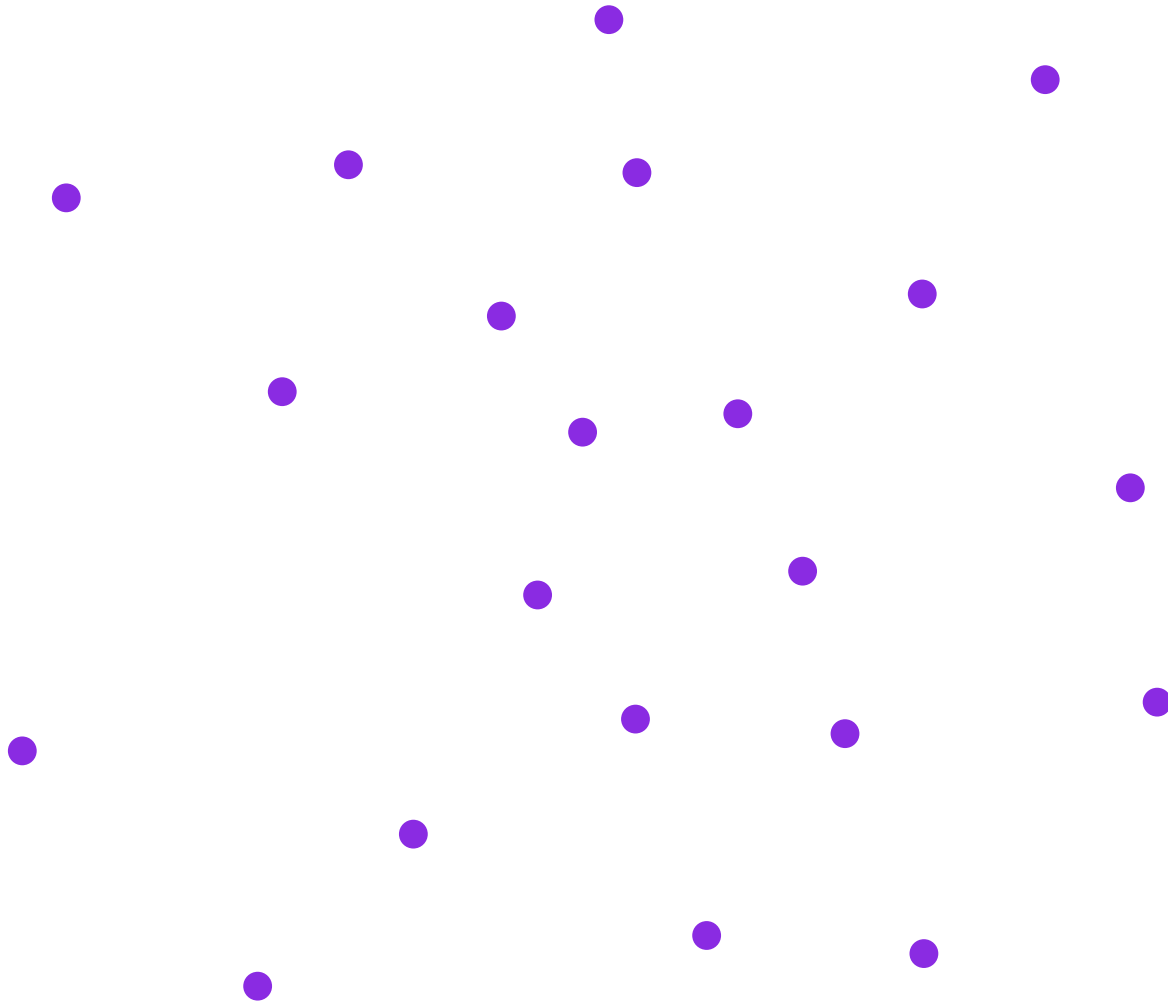


One word on robustness issues  
Basic incremental algorithm  
Locate by walk  
Locate using randomized data structures

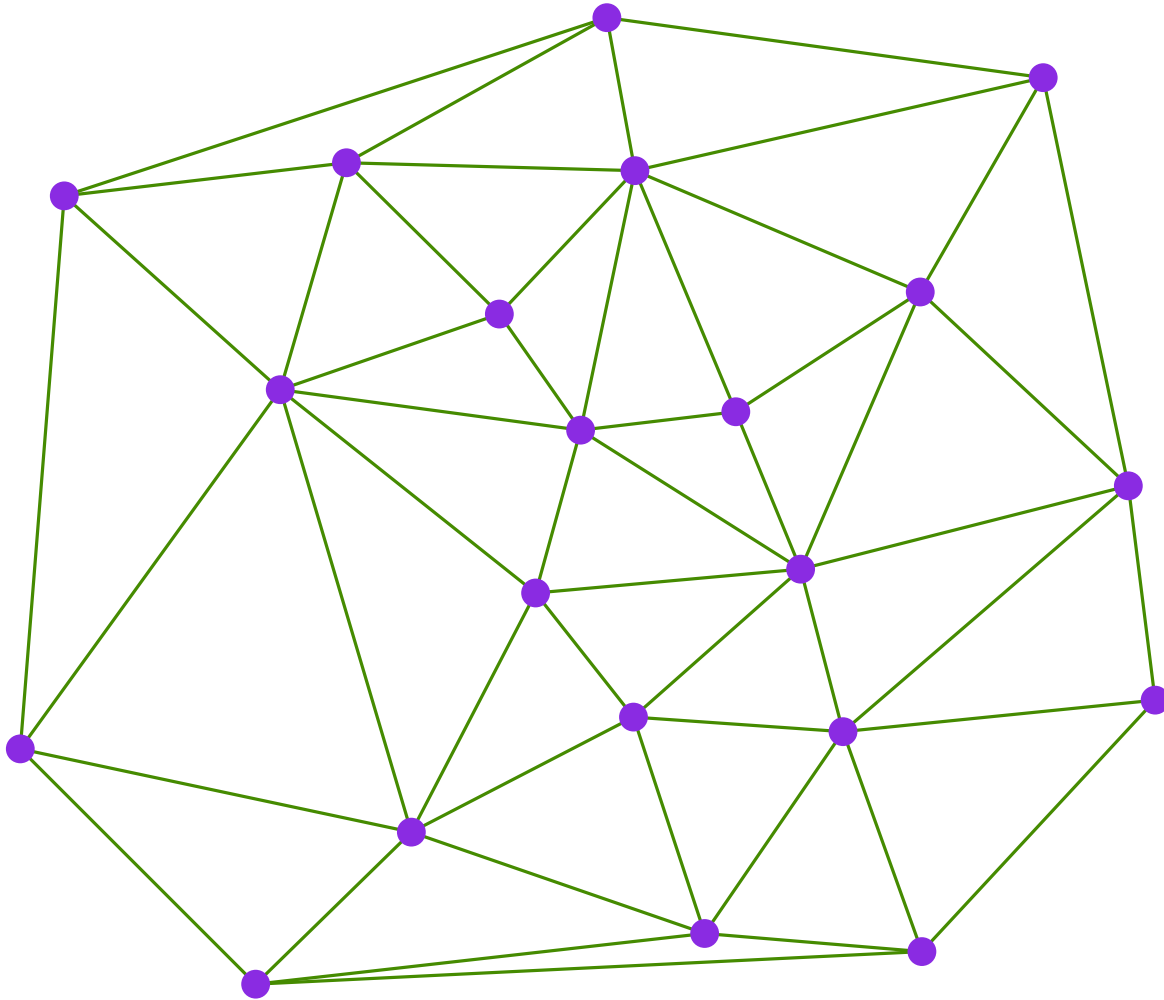
## Vertex removal in 2D

Remarks on CGAL programming  
Conclusion

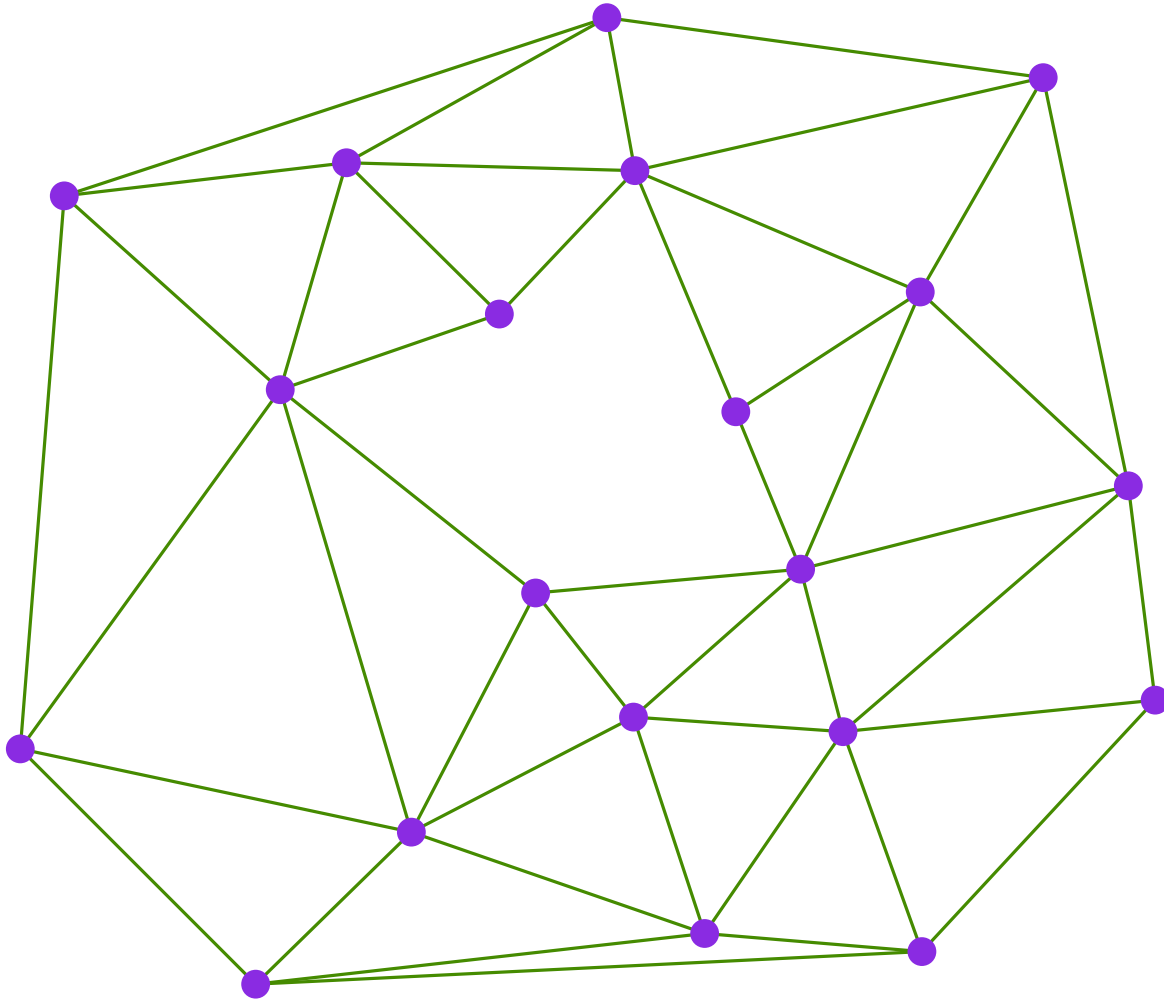
# Vertex removal



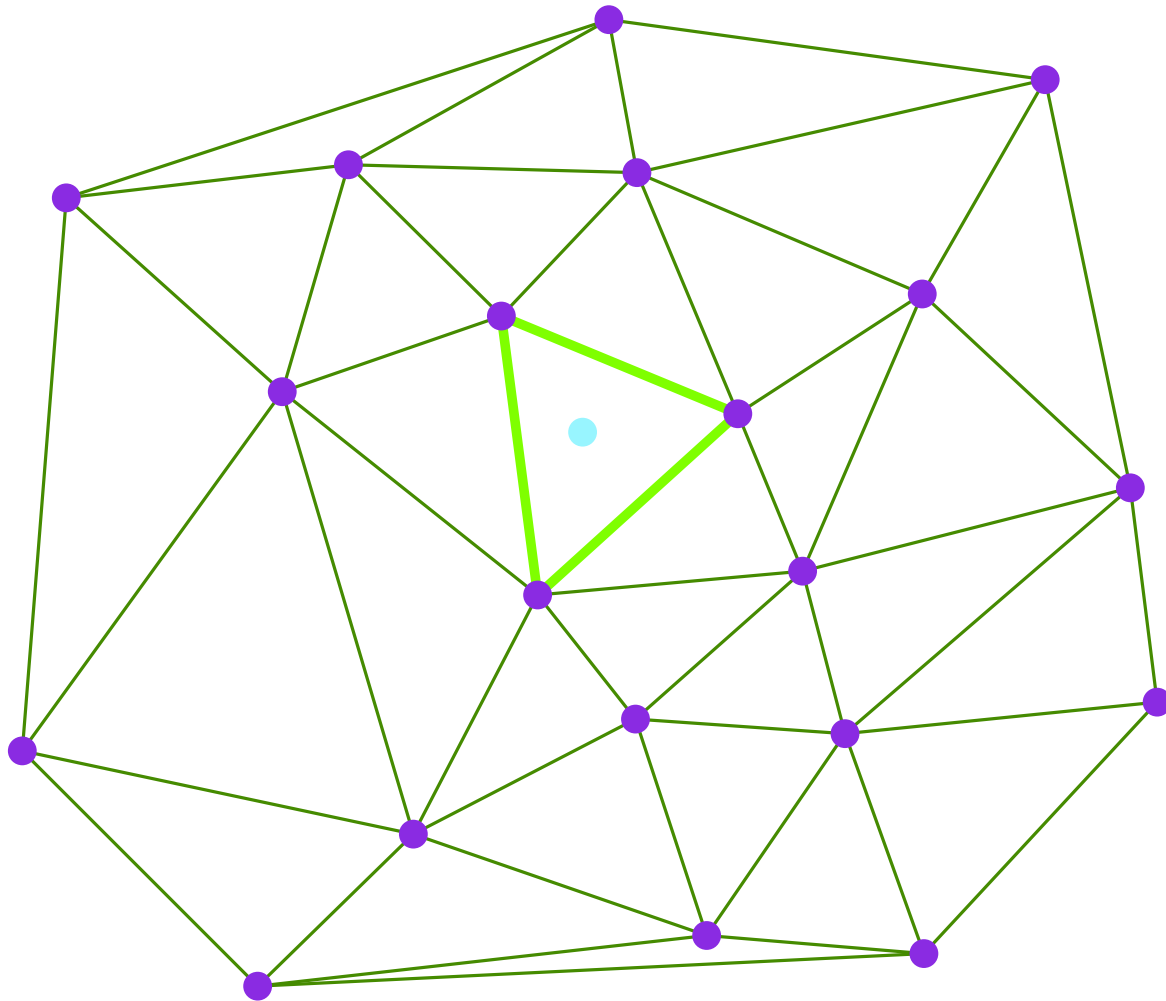
# Vertex removal



# Vertex removal



# Vertex removal





One word on robustness issues  
Basic incremental algorithm  
Locate by walk  
Locate using randomized data structures

## Vertex removal in 2D

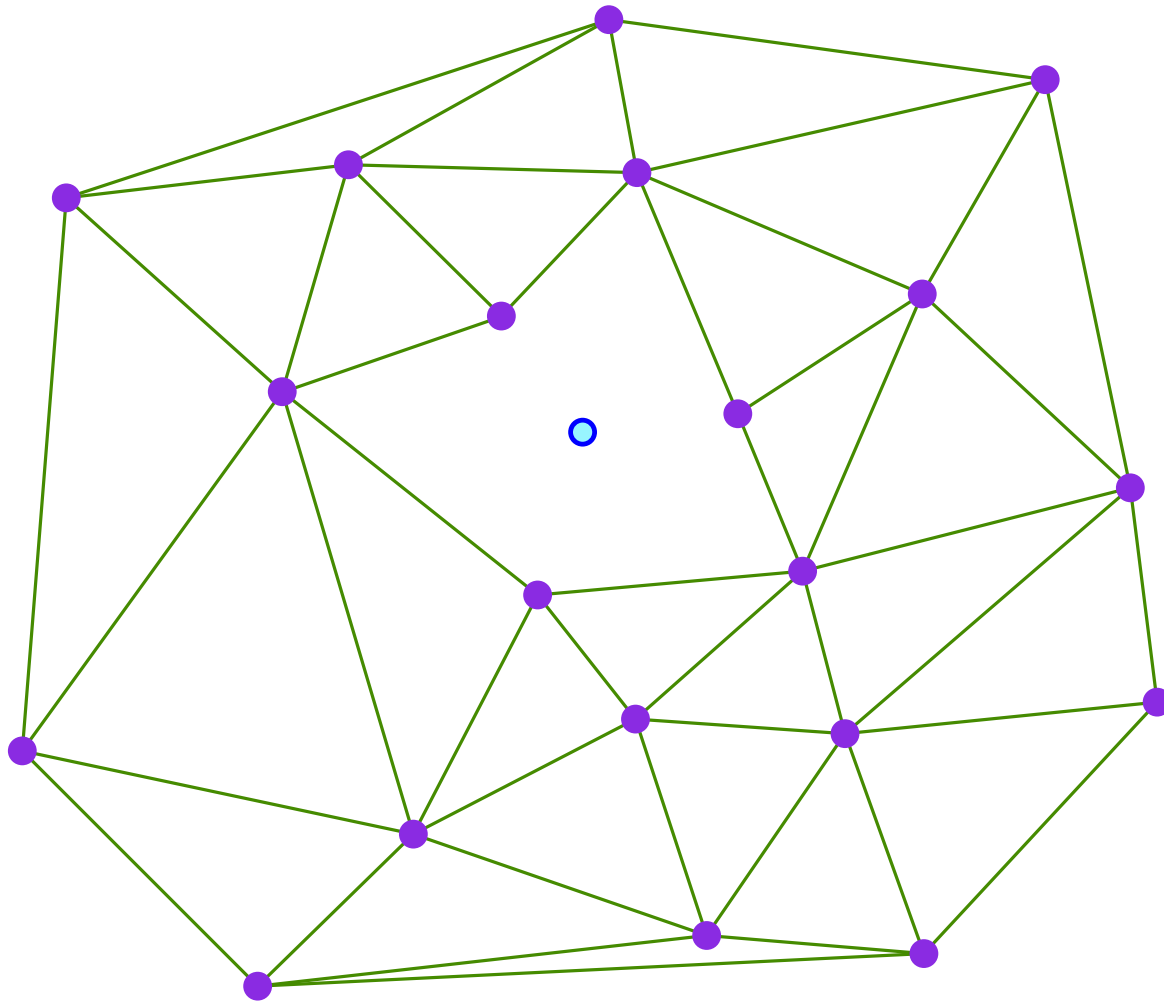
Various algorithms  
Boundary expansion

Remarks on CGAL programming  
Conclusion

# Vertex removal - boundary expansion



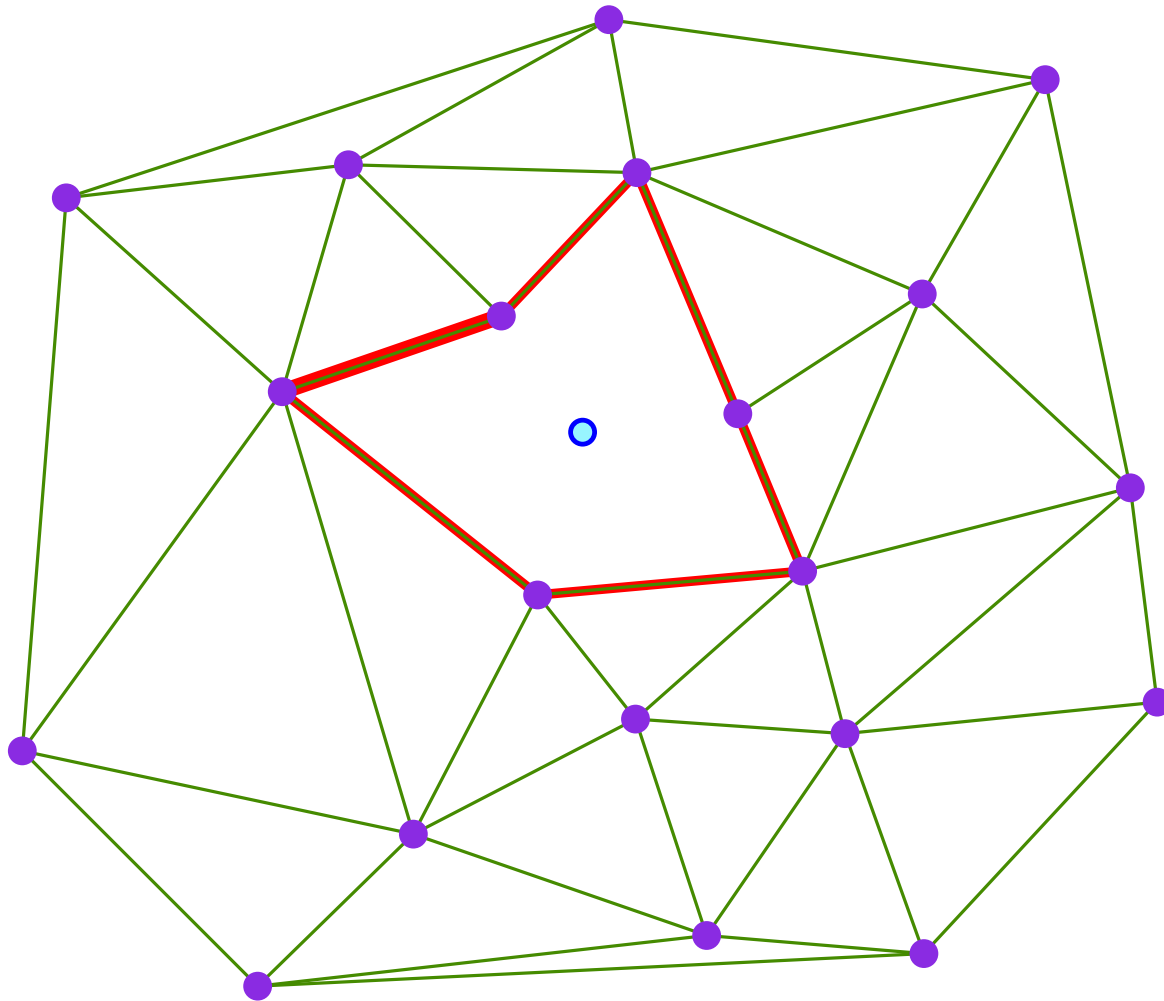
release 3.5, 2D implementation



# Vertex removal - boundary expansion



release 3.5, 2D implementation

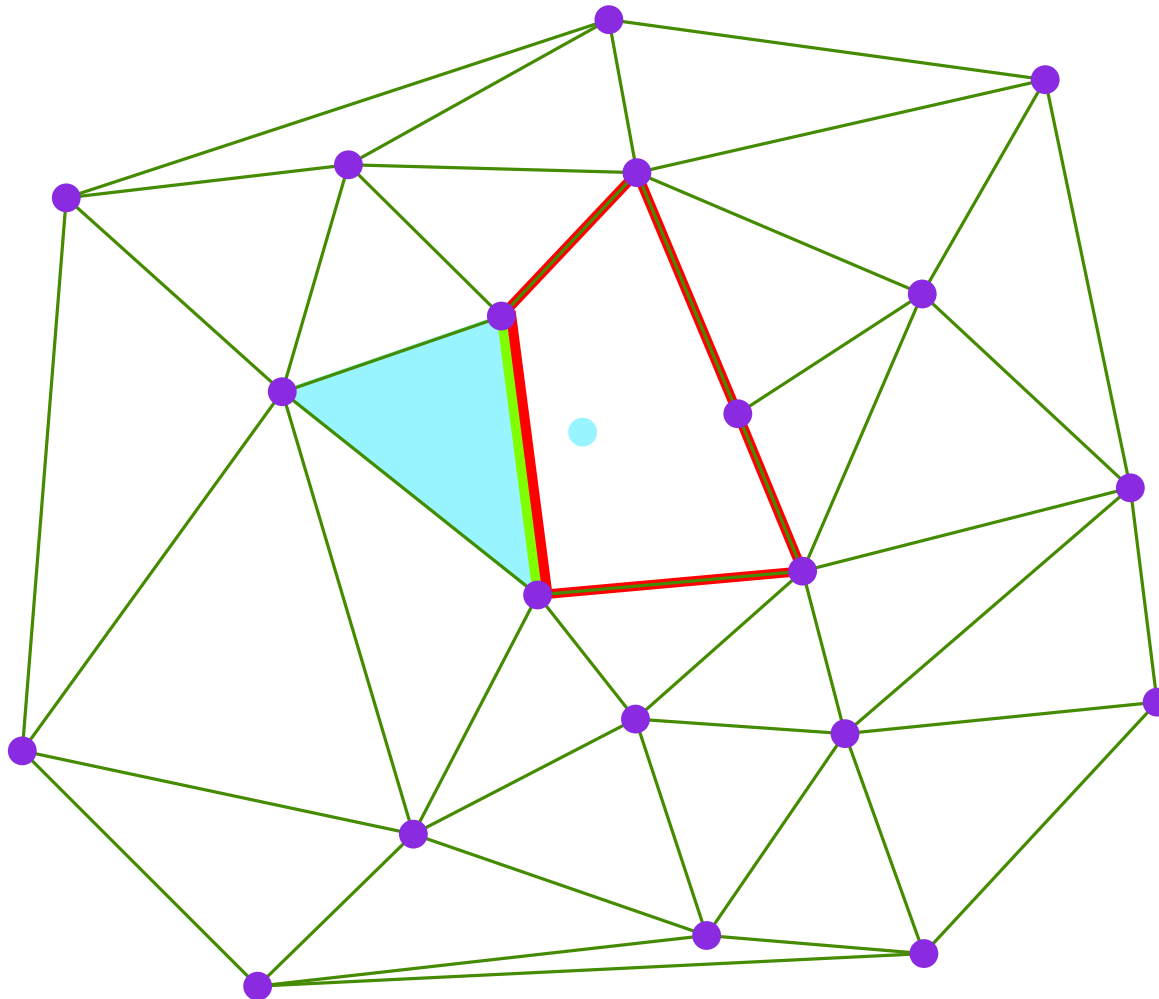




# Vertex removal - boundary expansion



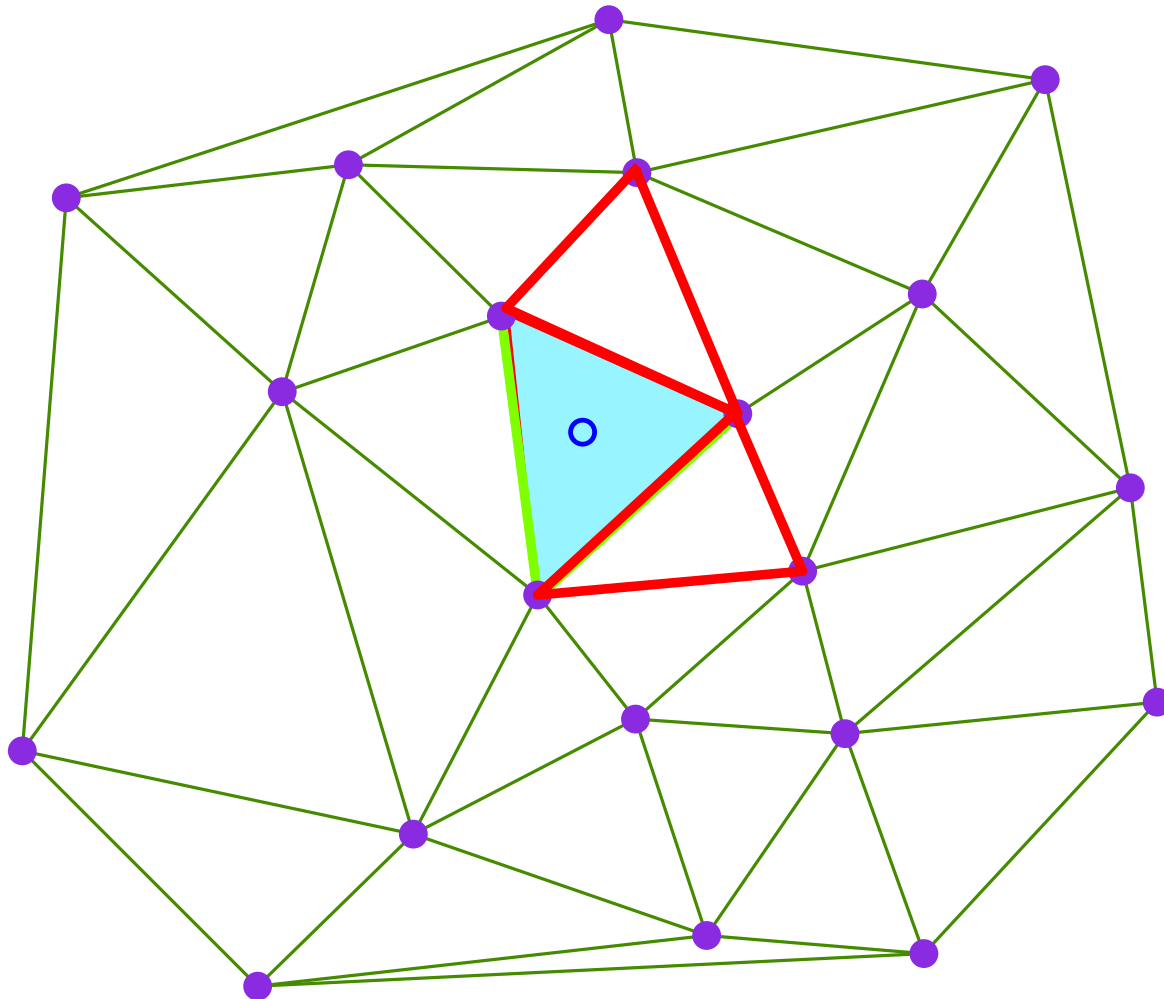
release 3.5, 2D implementation



# Vertex removal - boundary expansion



release 3.5, 2D implementation



# Vertex removal - boundary expansion



release 3.5, 2D implementation






One word on robustness issues  
Basic incremental algorithm  
Locate by walk  
Locate using randomized data structures

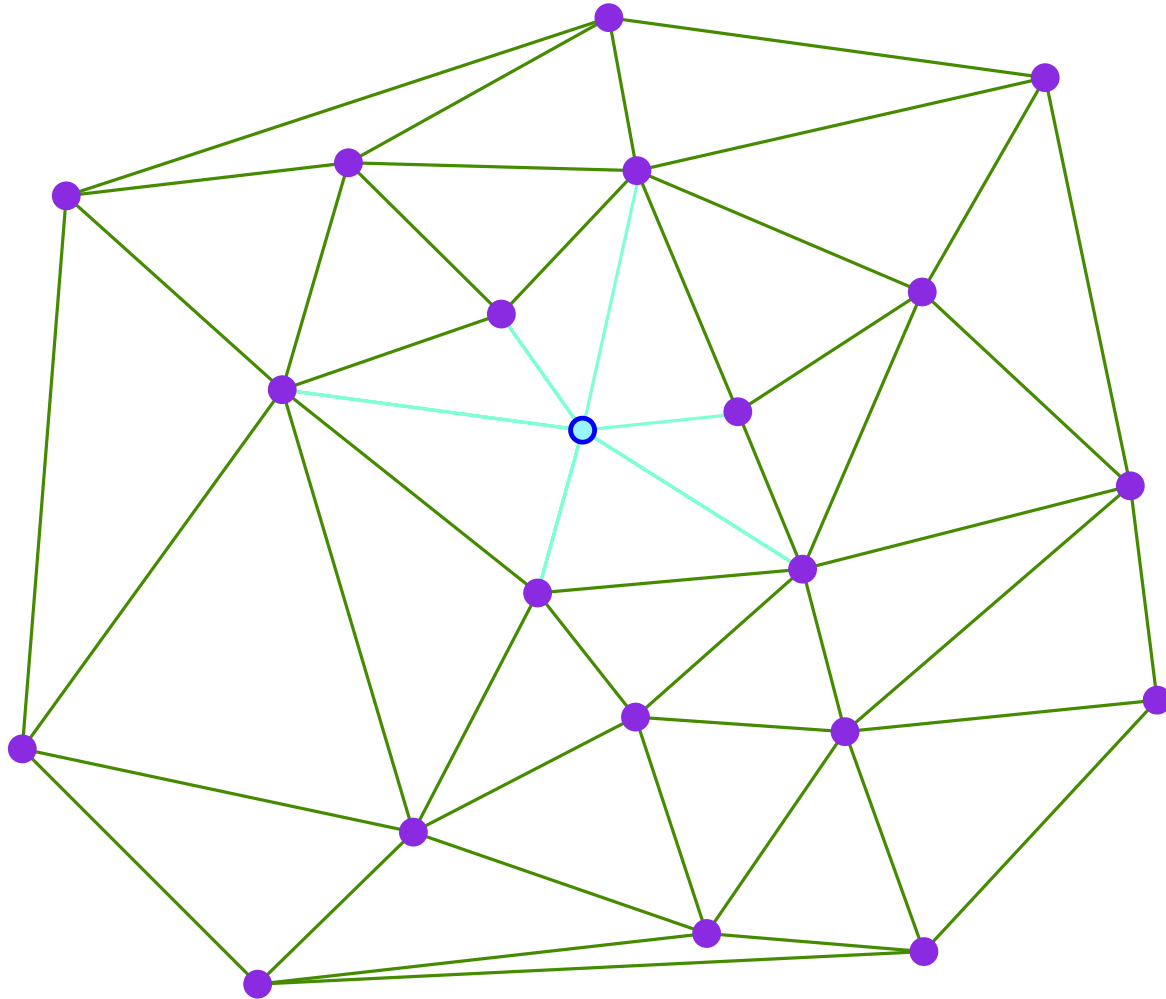
## Vertex removal in 2D

Various algorithms  
Boundary expansion  
Triangulate and sew


Remarks on CGAL programming  
Conclusion

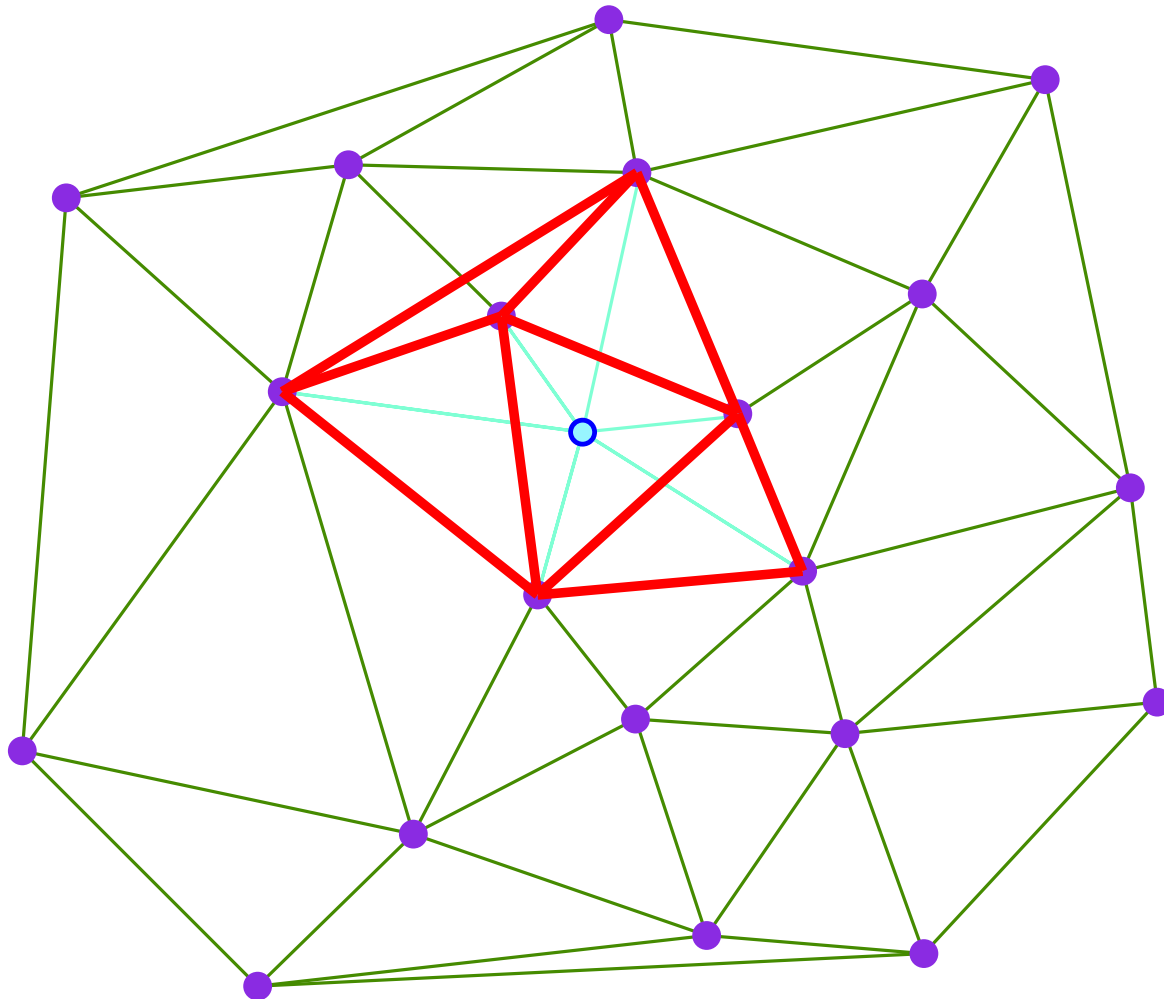
# Vertex removal - triangulate and sew

current  3D implementation




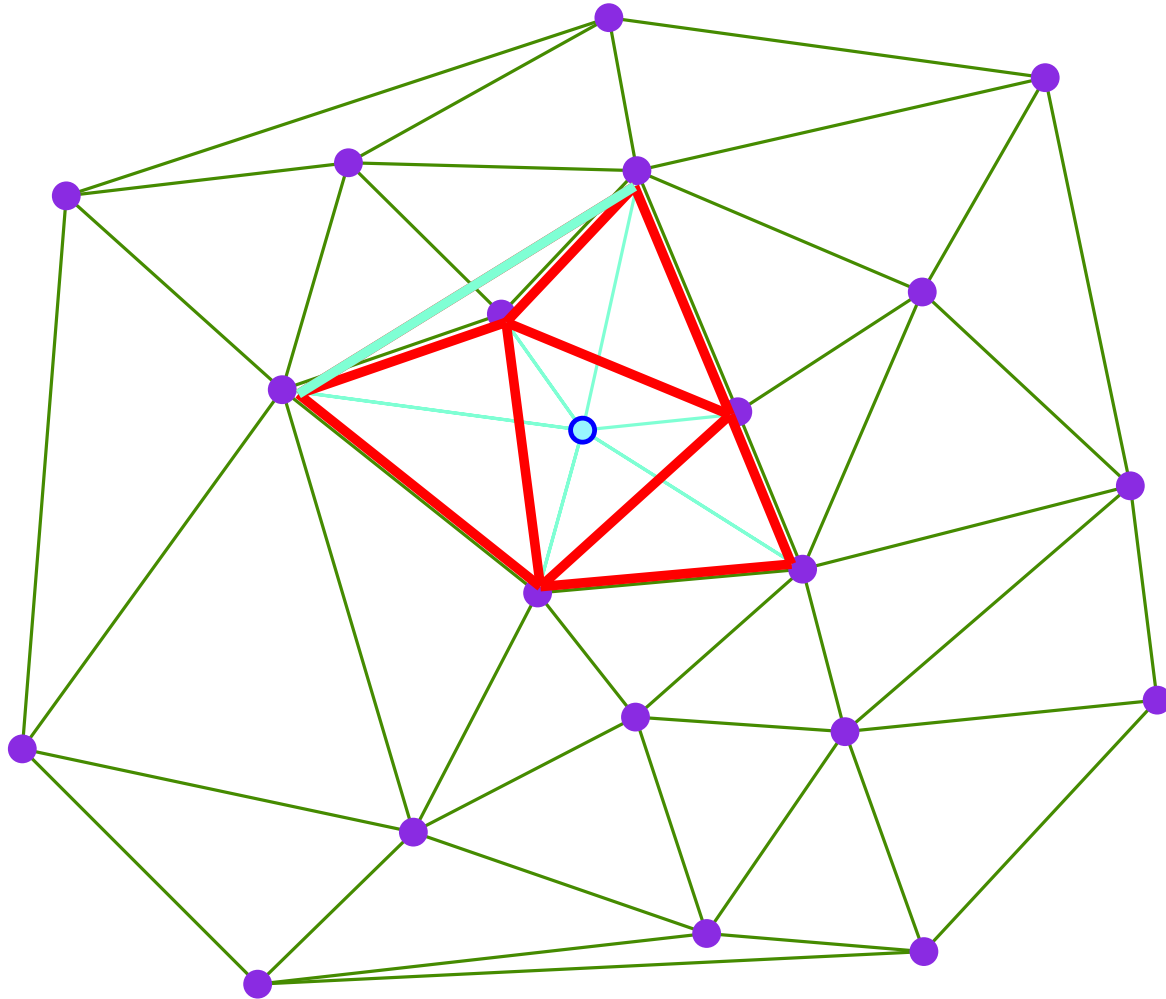
# Vertex removal - triangulate and sew

current  3D implementation




# Vertex removal - triangulate and sew

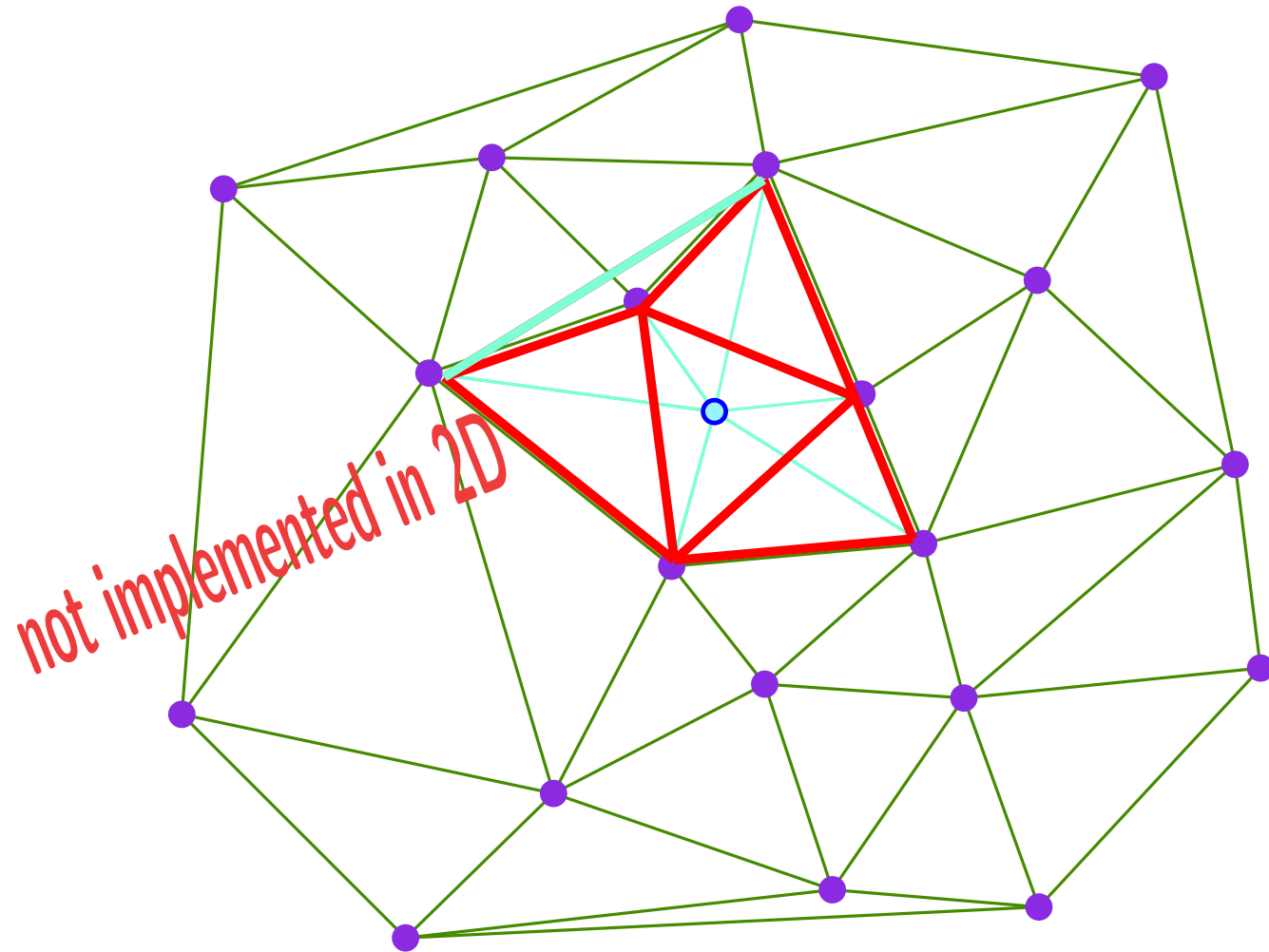
current  3D implementation



delete extra triangles and sew

# Vertex removal - triangulate and sew

current  3D implementation



delete extra triangles and sew





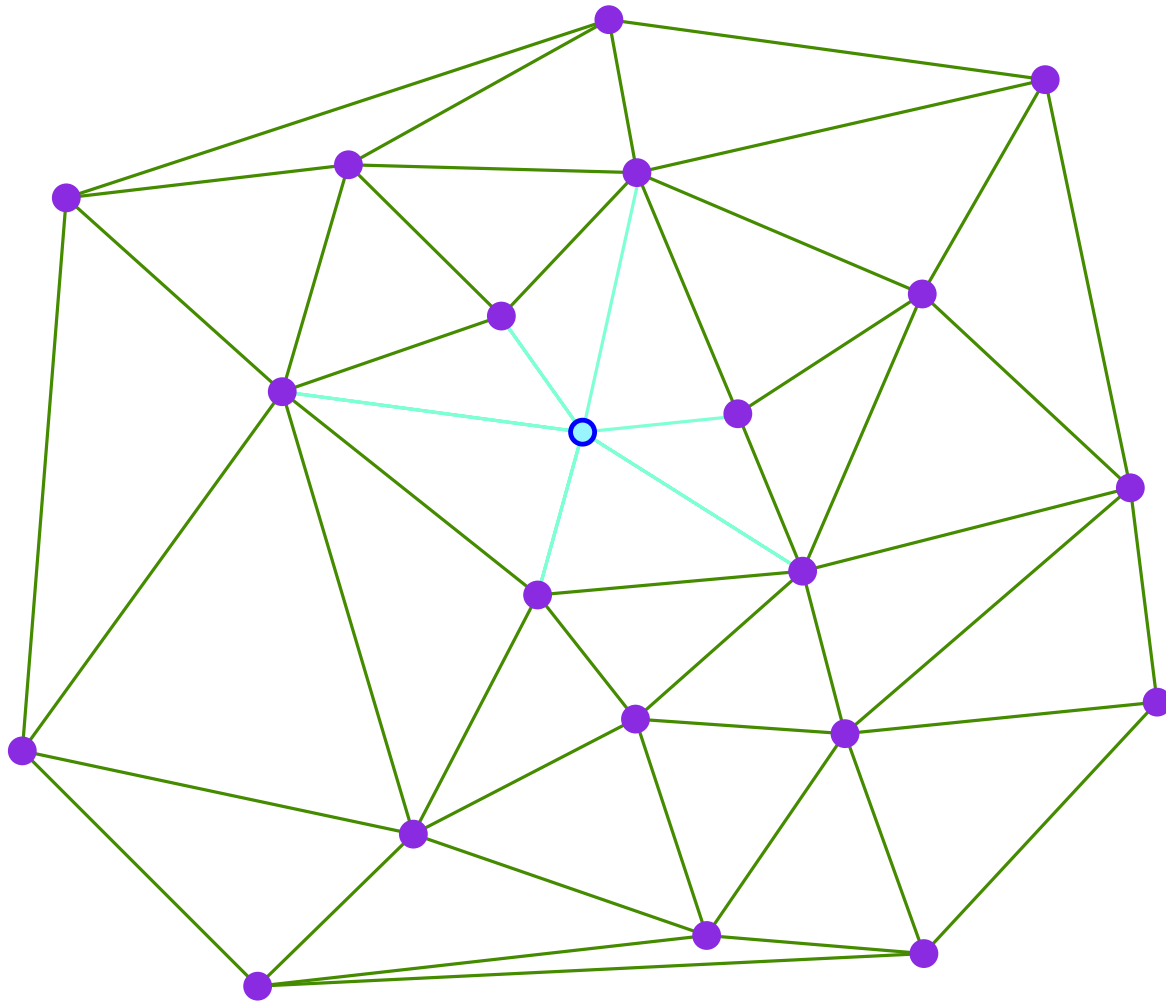
One word on robustness issues  
Basic incremental algorithm  
Locate by walk  
Locate using randomized data structures

## Vertex removal in 2D

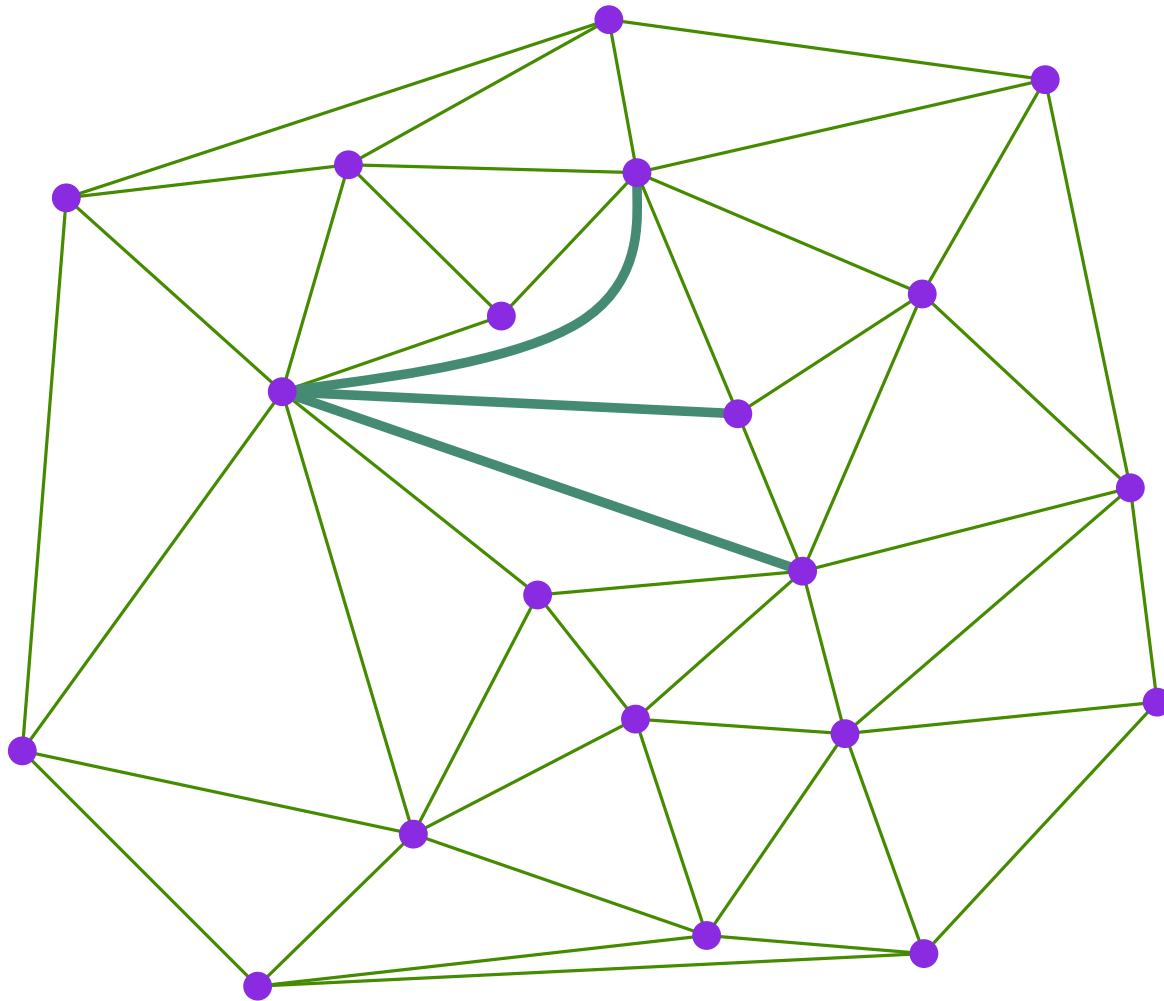
Various algorithms  
Boundary expansion  
Triangulate and sew  
Flip the hole

Remarks on CGAL programming  
Conclusion

# Vertex removal - flip the hole

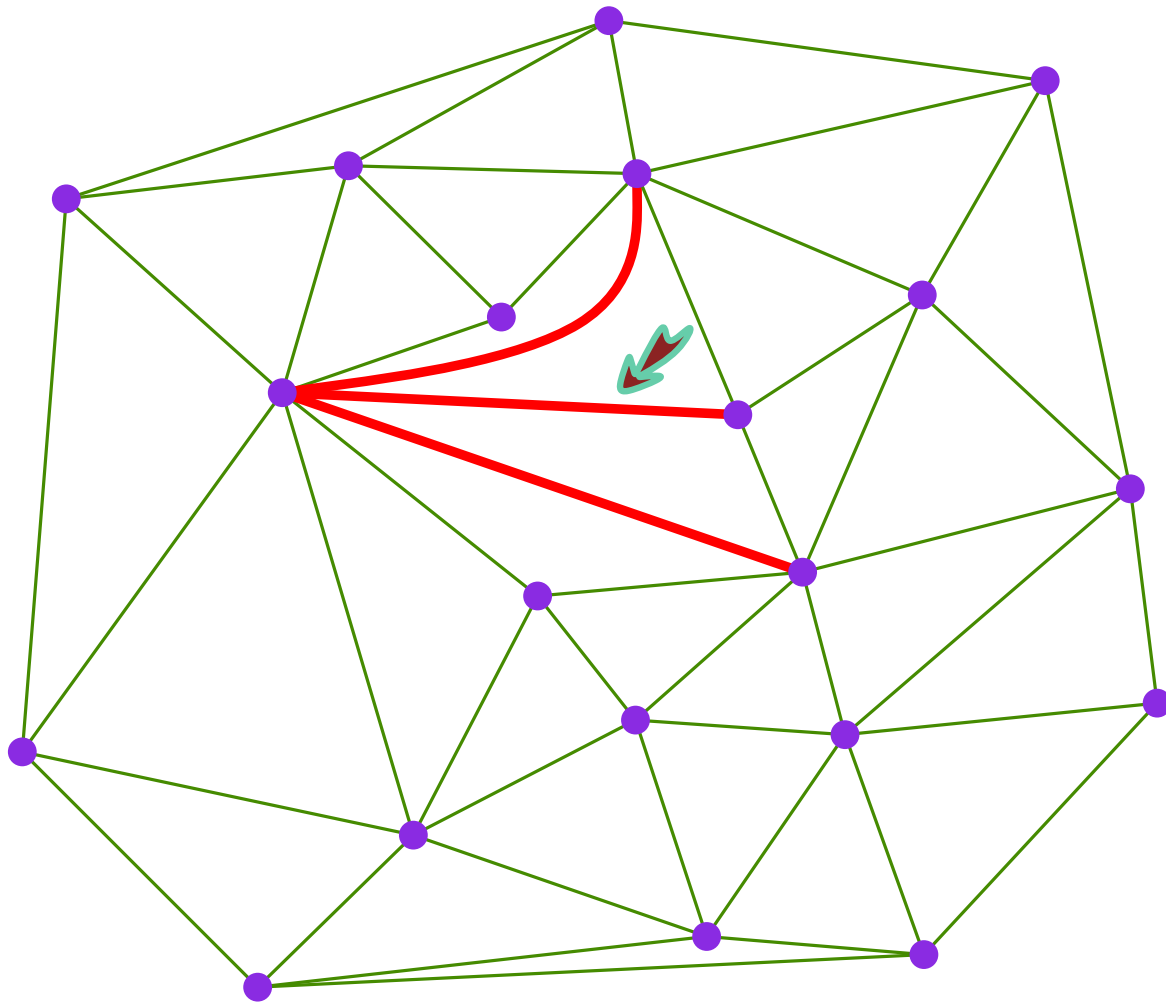


## Vertex removal - flip the hole



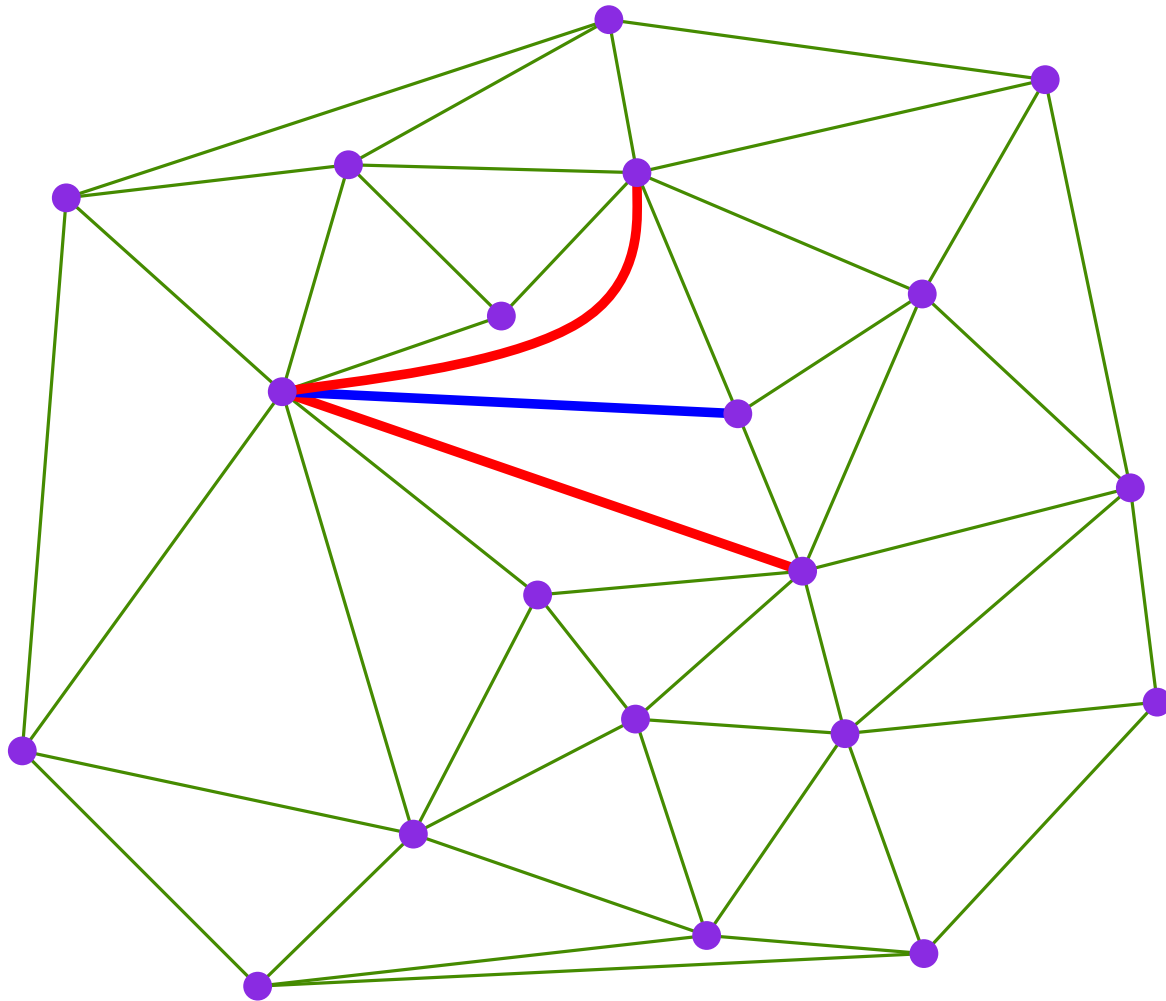
triangulate from any vertex

# Vertex removal - flip the hole

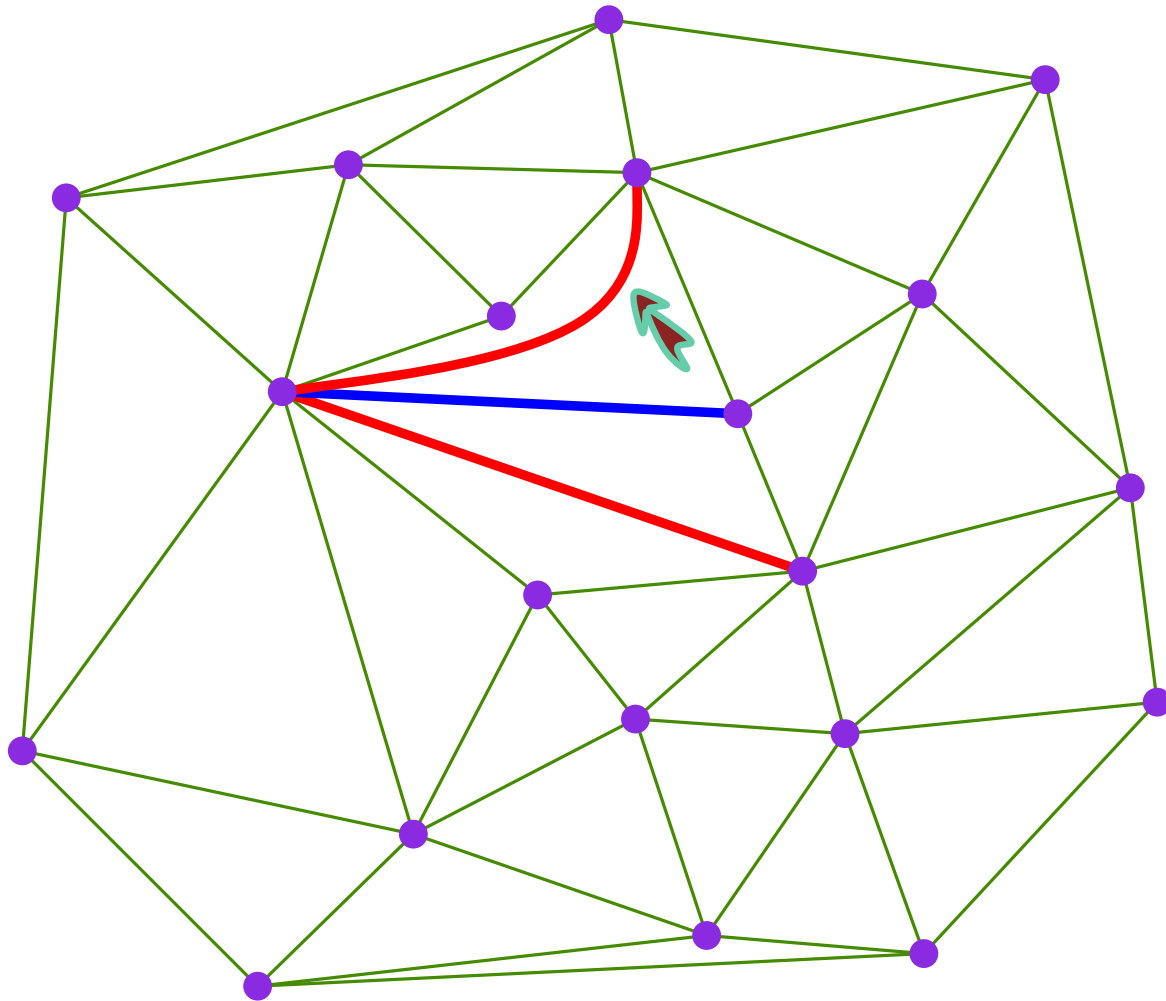


queue of edges to be checked

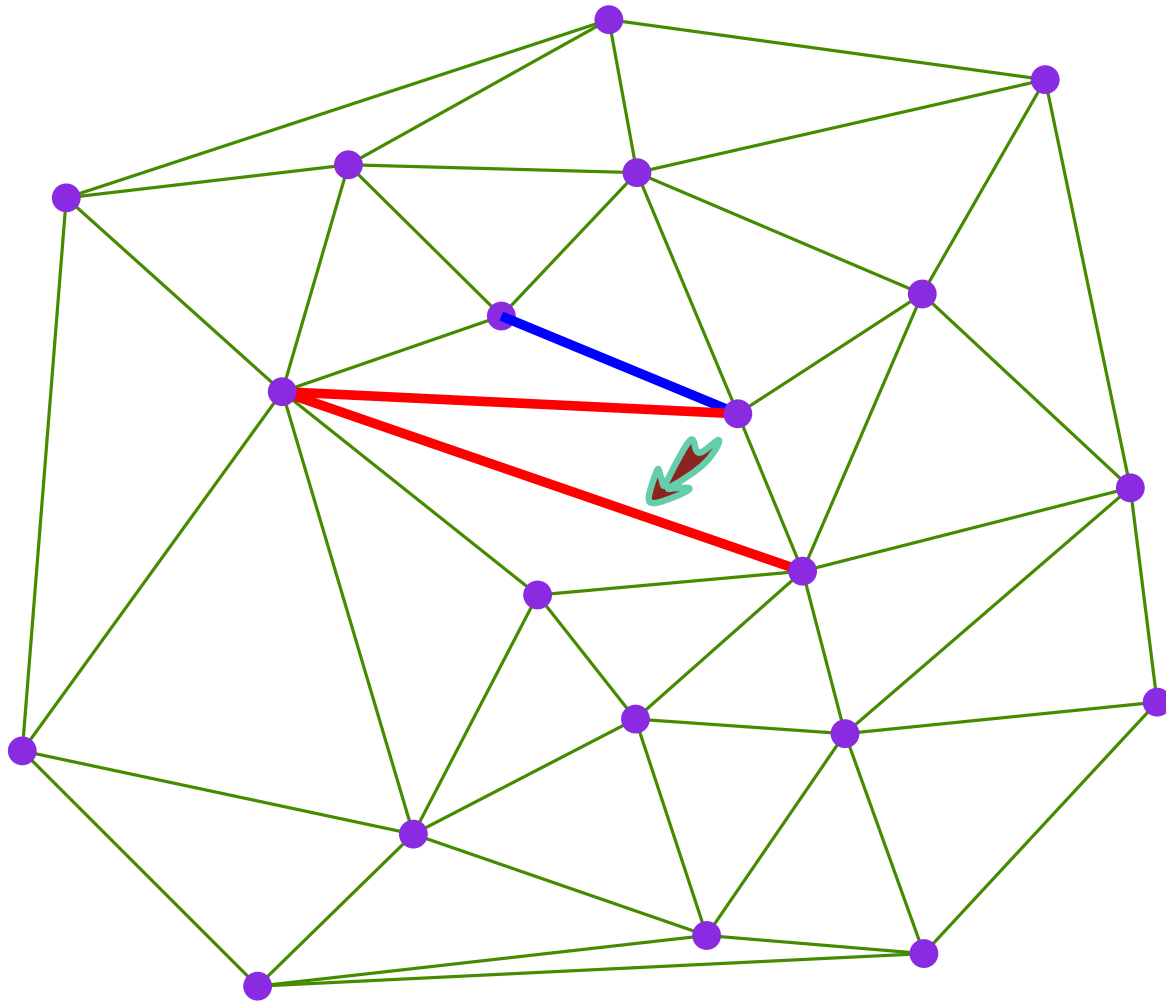
# Vertex removal - flip the hole



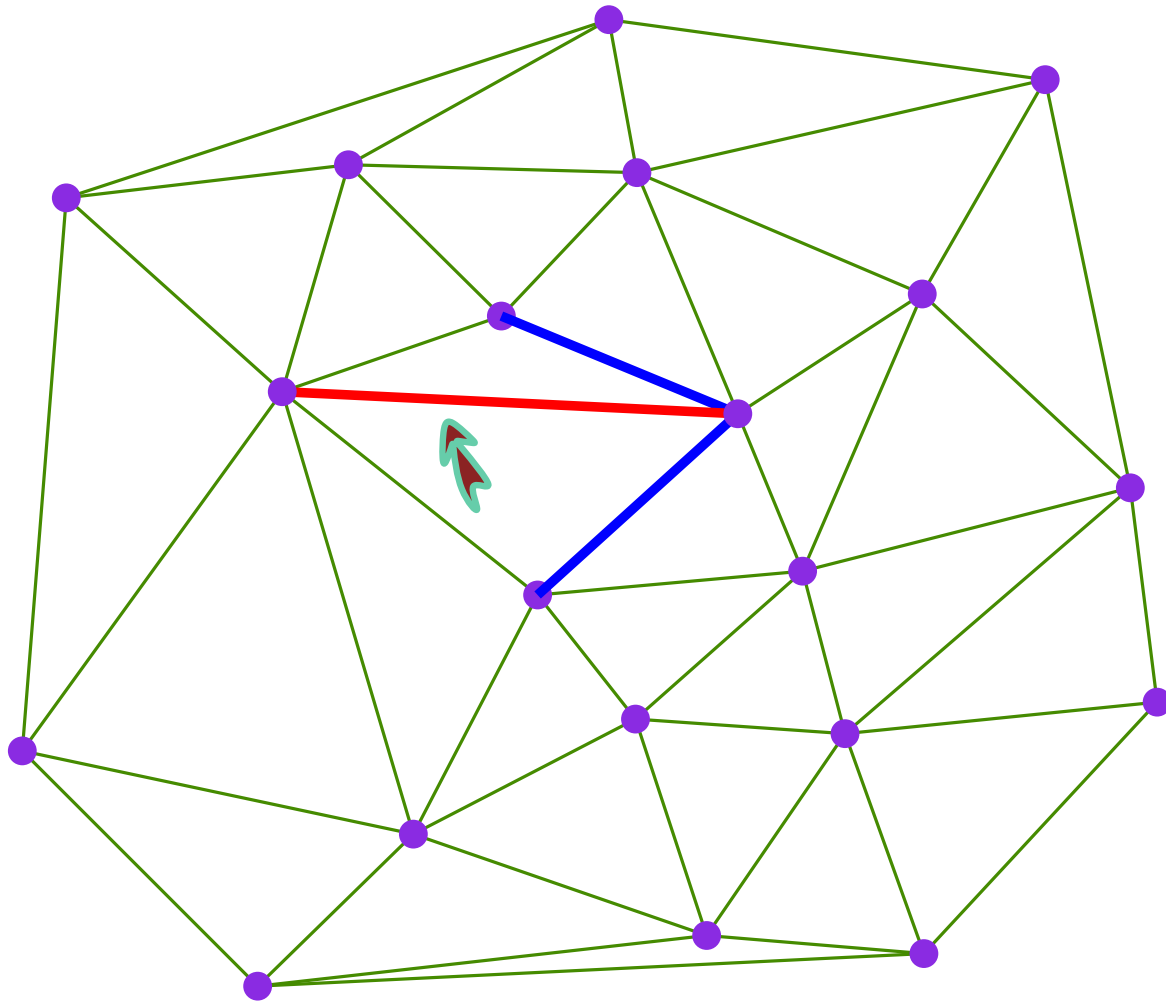
# Vertex removal - flip the hole



# Vertex removal - flip the hole

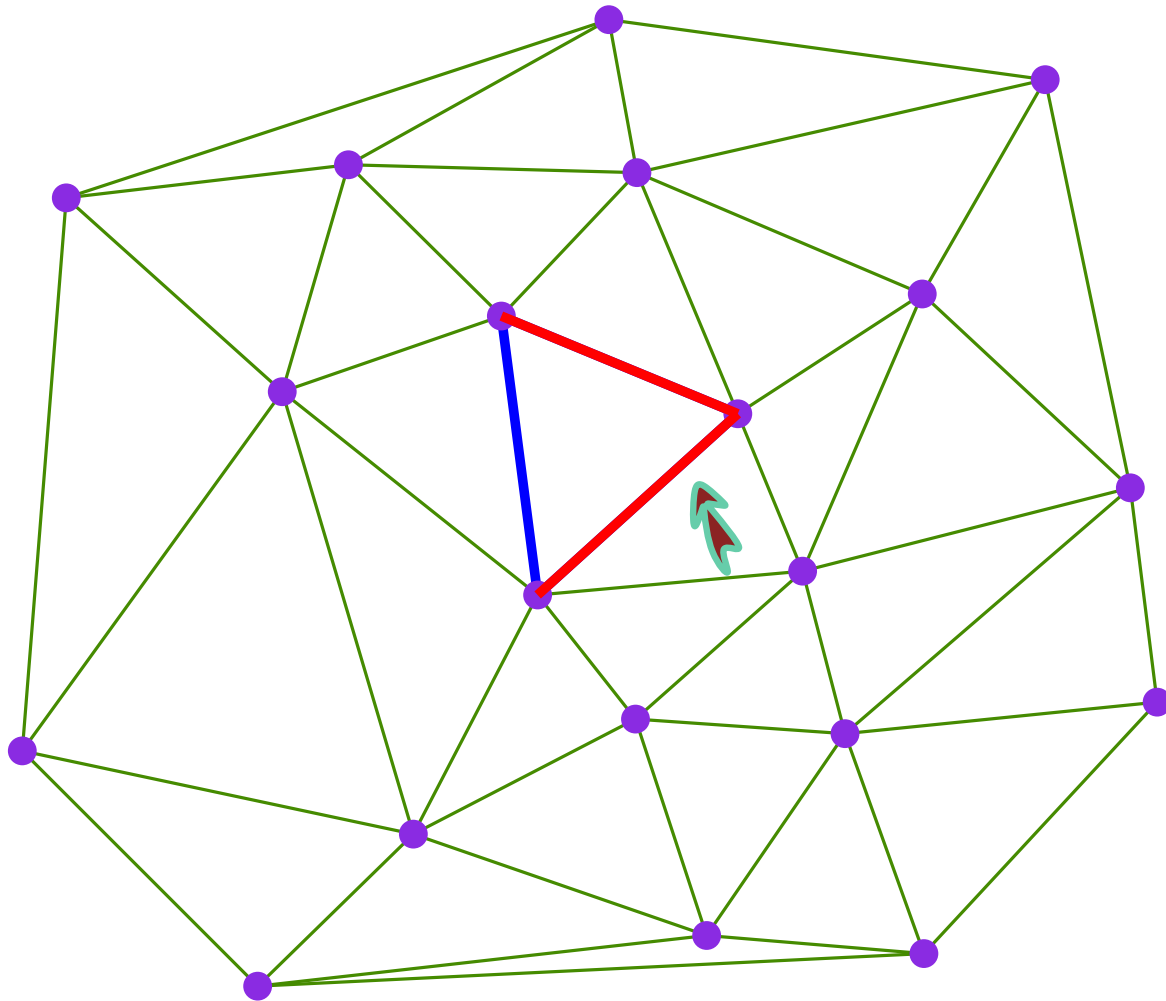


# Vertex removal - flip the hole

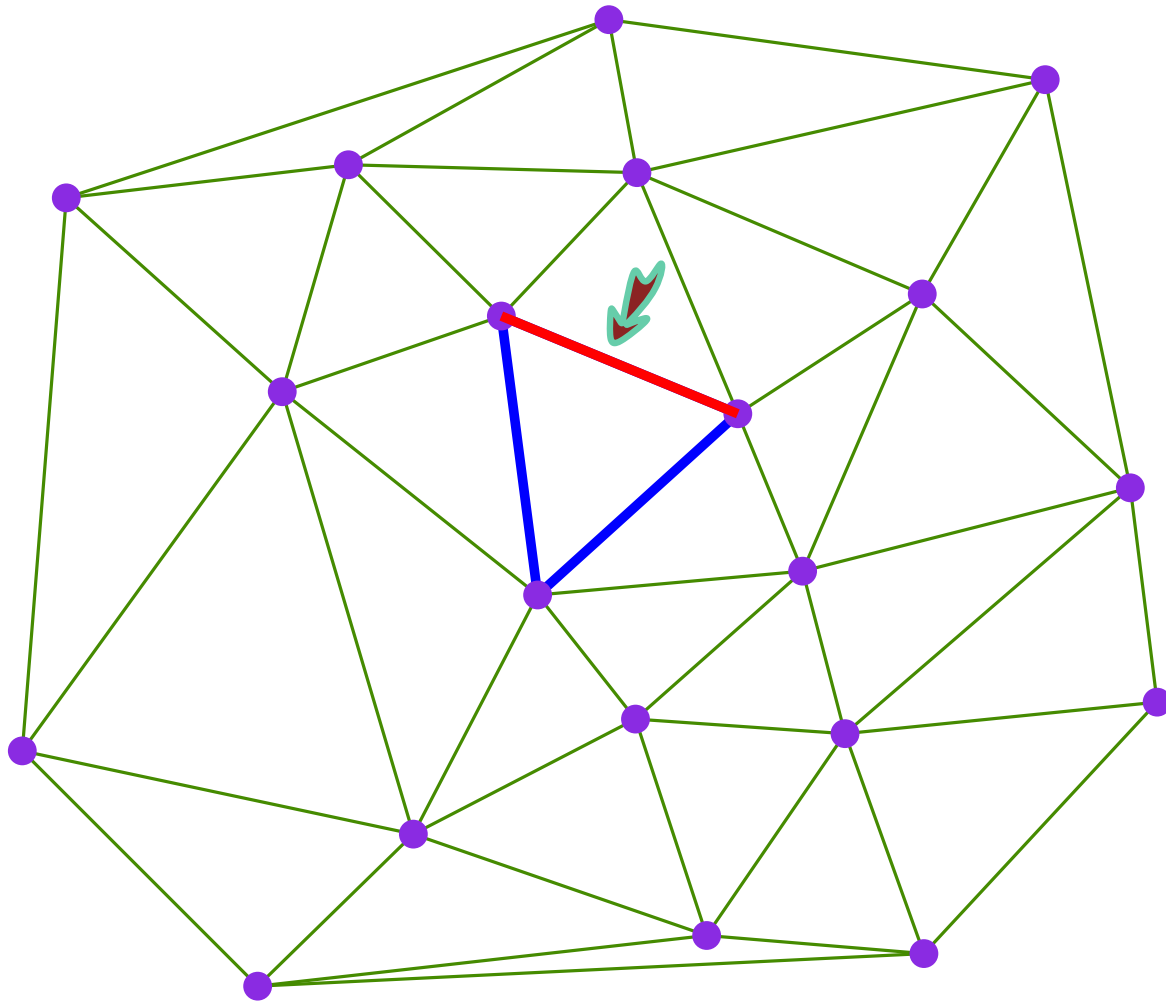




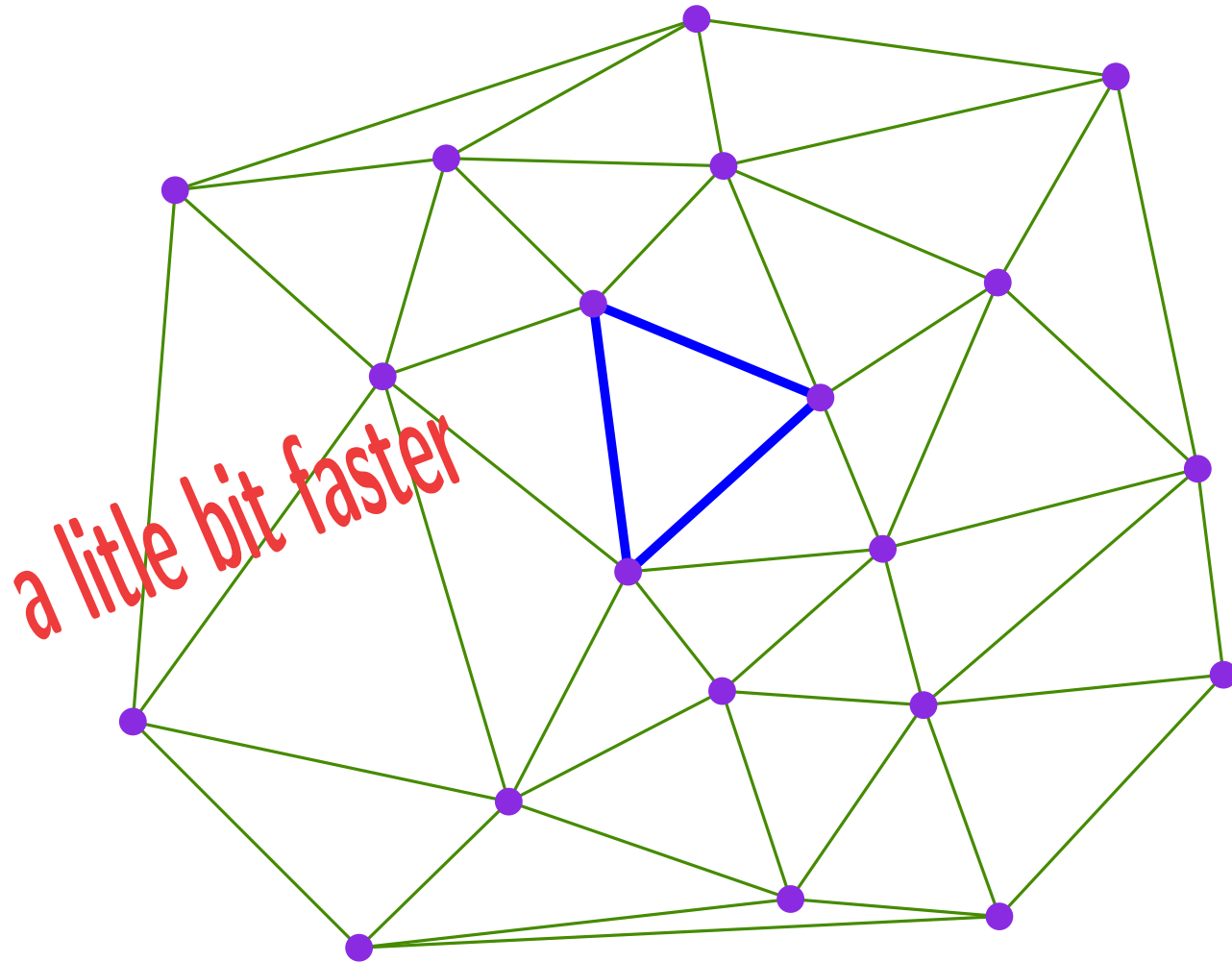
# Vertex removal - flip the hole



# Vertex removal - flip the hole



# Vertex removal - flip the hole





One word on robustness issues  
Basic incremental algorithm  
Locate by walk  
Locate using randomized data structures

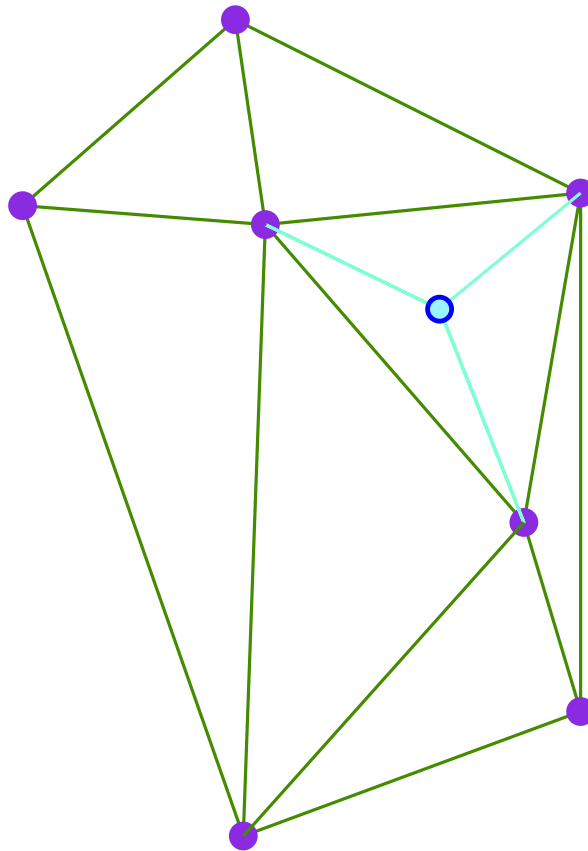
## Vertex removal in 2D

Various algorithms  
Low degree optimization

Remarks on CGAL programming  
Conclusion

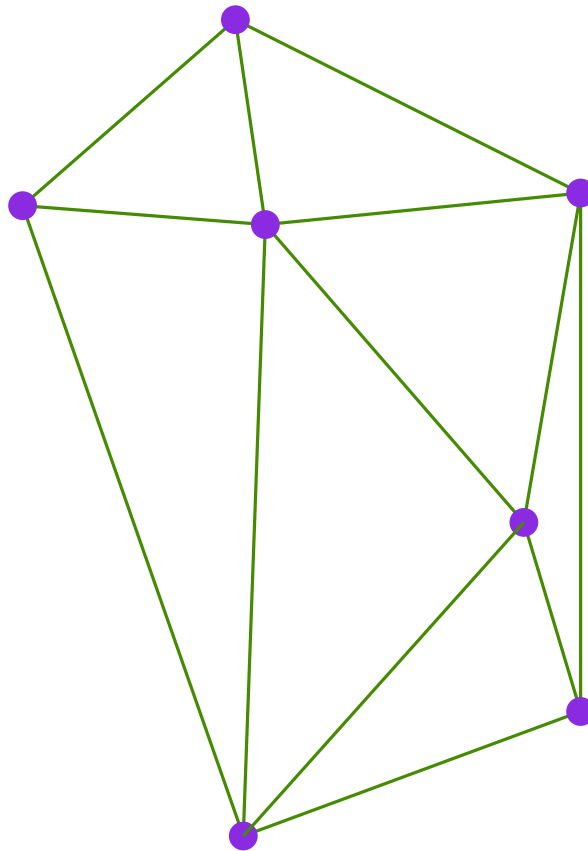
# Vertex removal - low degree optimization

degree 3



# Vertex removal - low degree optimization

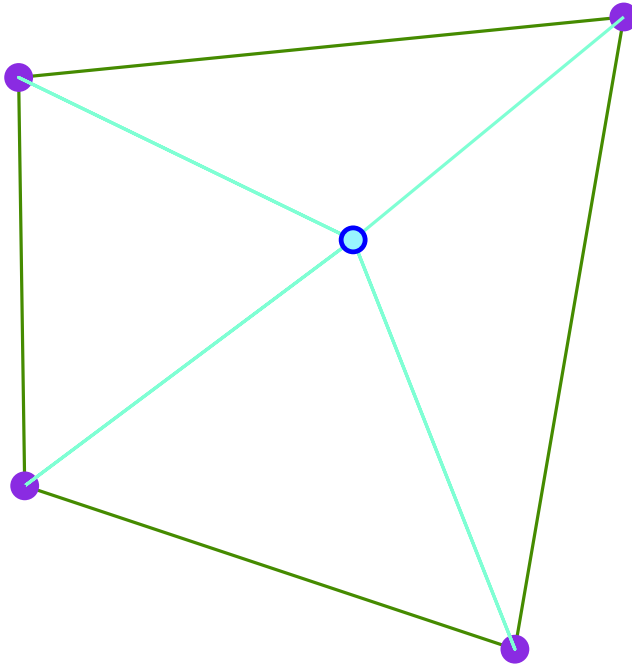
degree 3



almost nothing to do

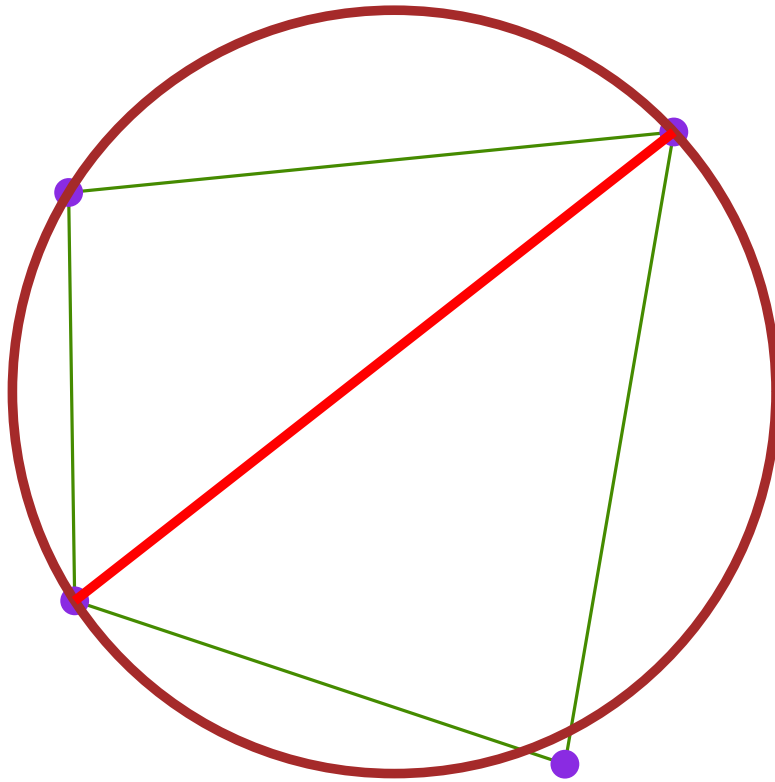
# Vertex removal - low degree optimization

degree 4



# Vertex removal - low degree optimization

degree 4

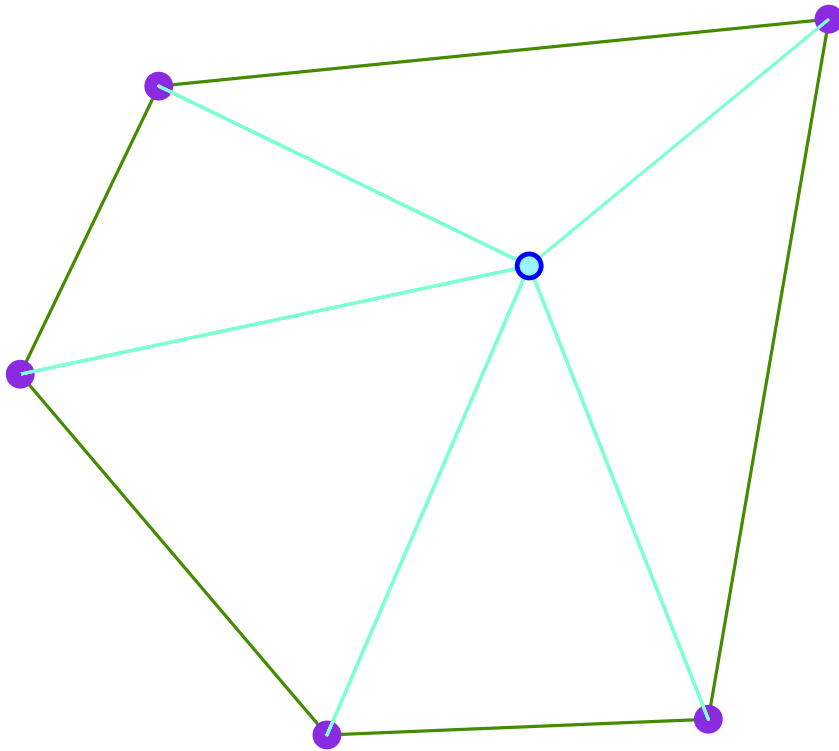


just one incircle test to decide



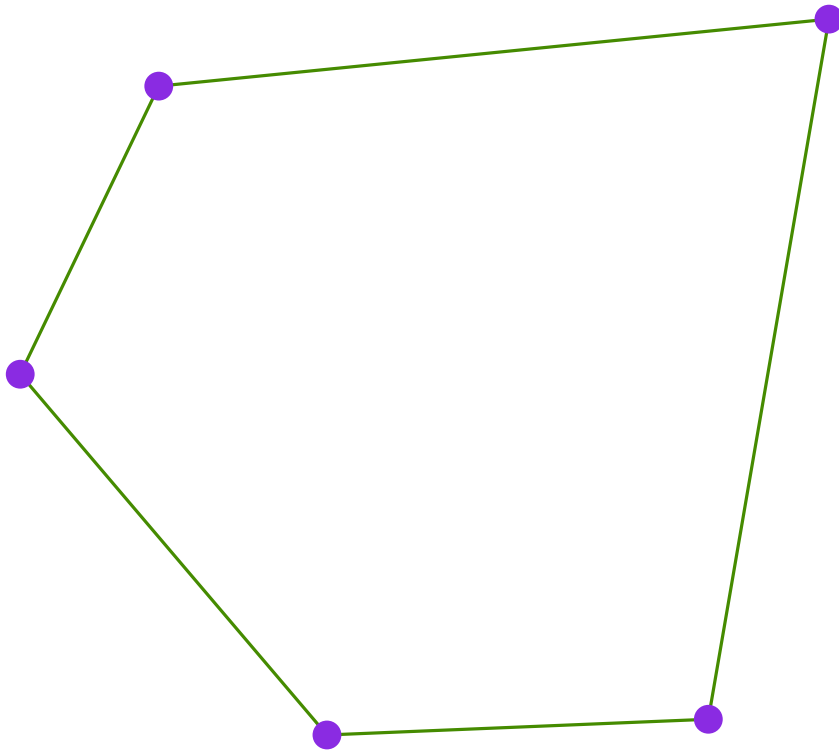
# Vertex removal - low degree optimization

degree 5



# Vertex removal - low degree optimization

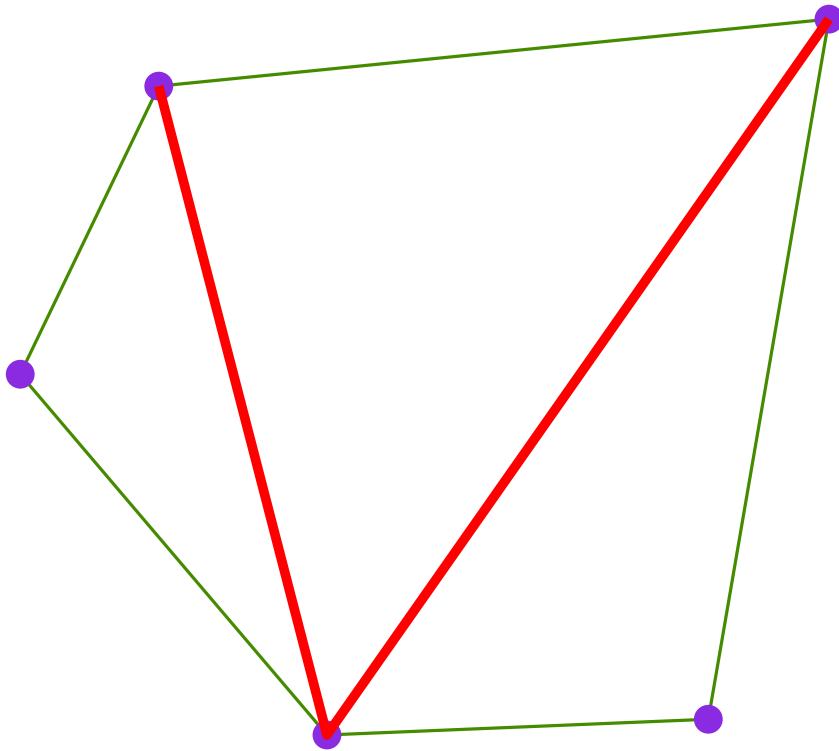
degree 5



"star" the pentagon from the right vertex

# Vertex removal - low degree optimization

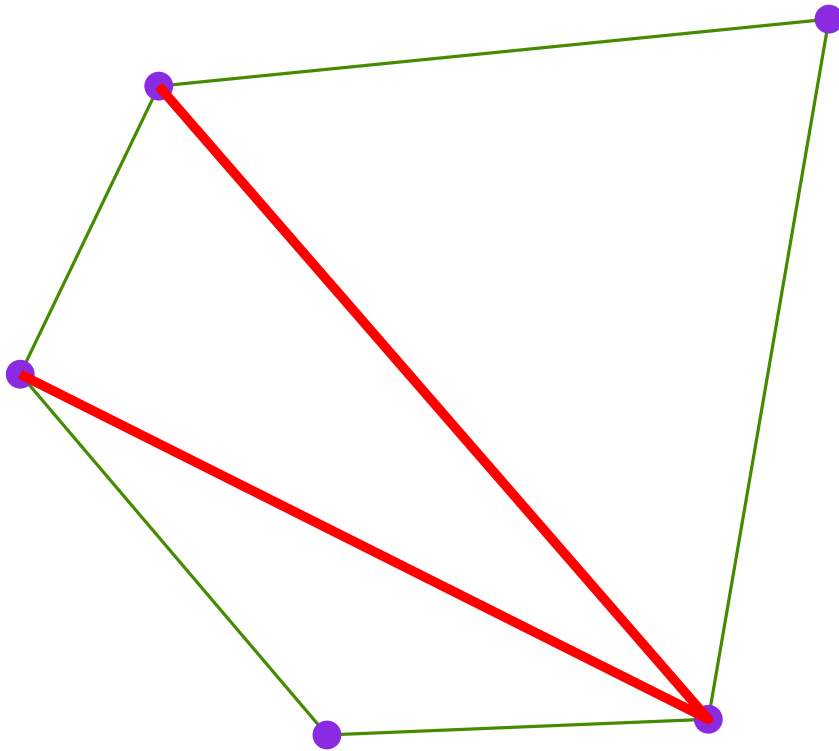
degree 5



"star" the pentagon from the right vertex

# Vertex removal - low degree optimization

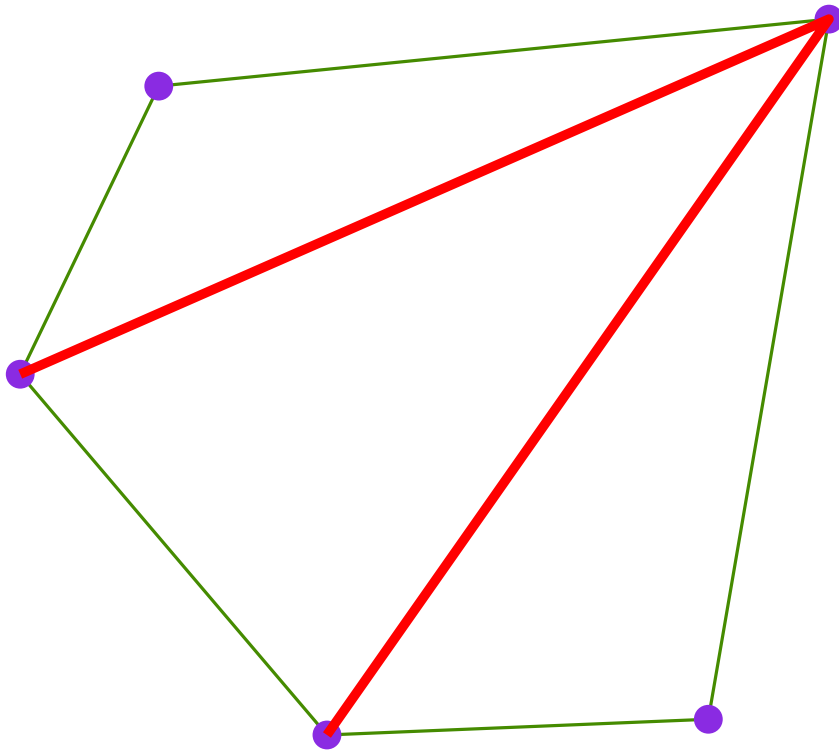
degree 5



"star" the pentagon from the right vertex

# Vertex removal - low degree optimization

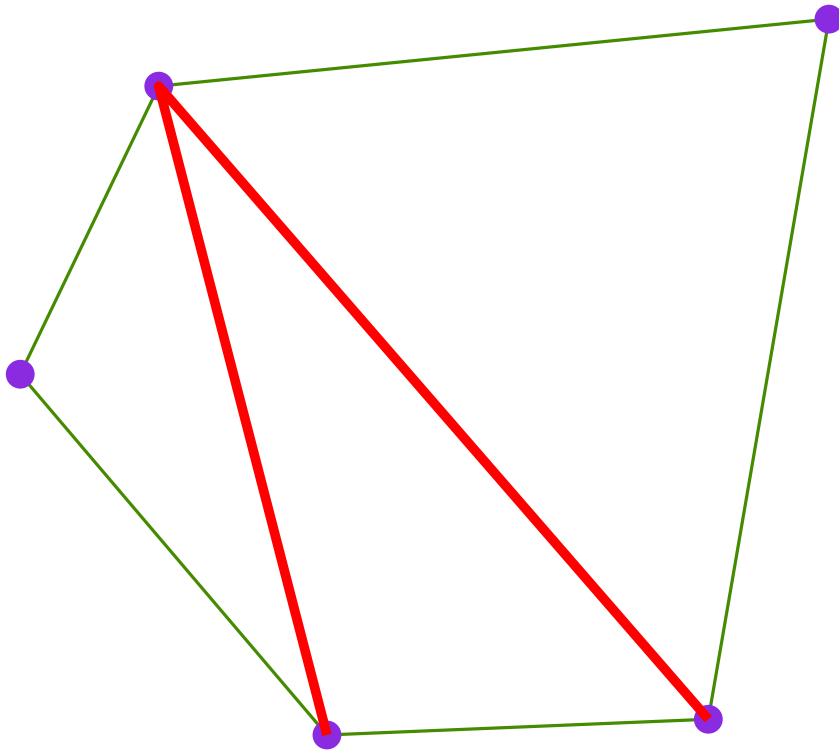
degree 5



"star" the pentagon from the right vertex

# Vertex removal - low degree optimization

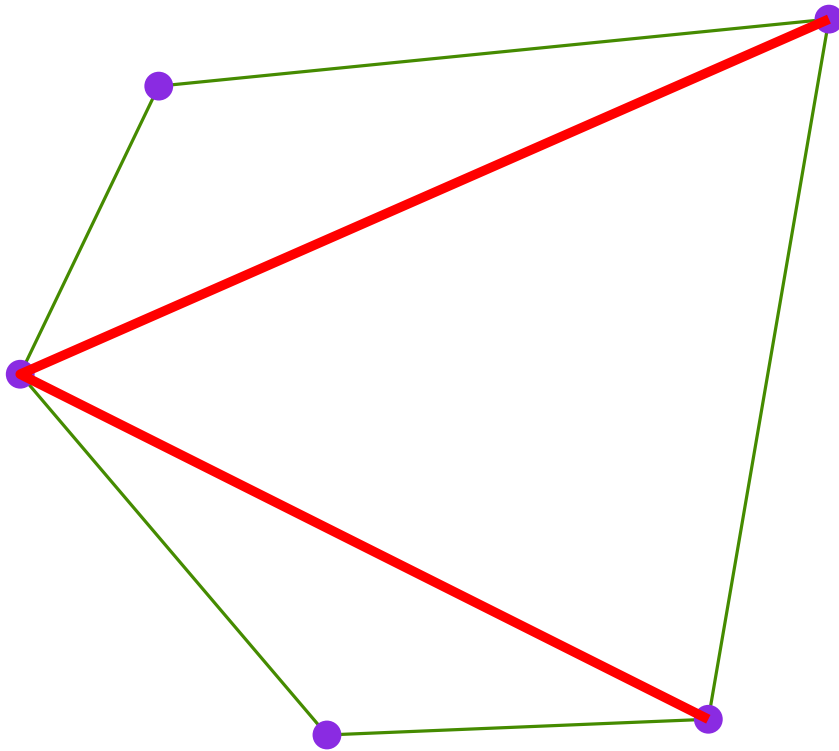
degree 5



"star" the pentagon from the right vertex

# Vertex removal - low degree optimization

degree 5



"star" the pentagon from the right vertex

# Vertex removal - low degree optimization

degree 5

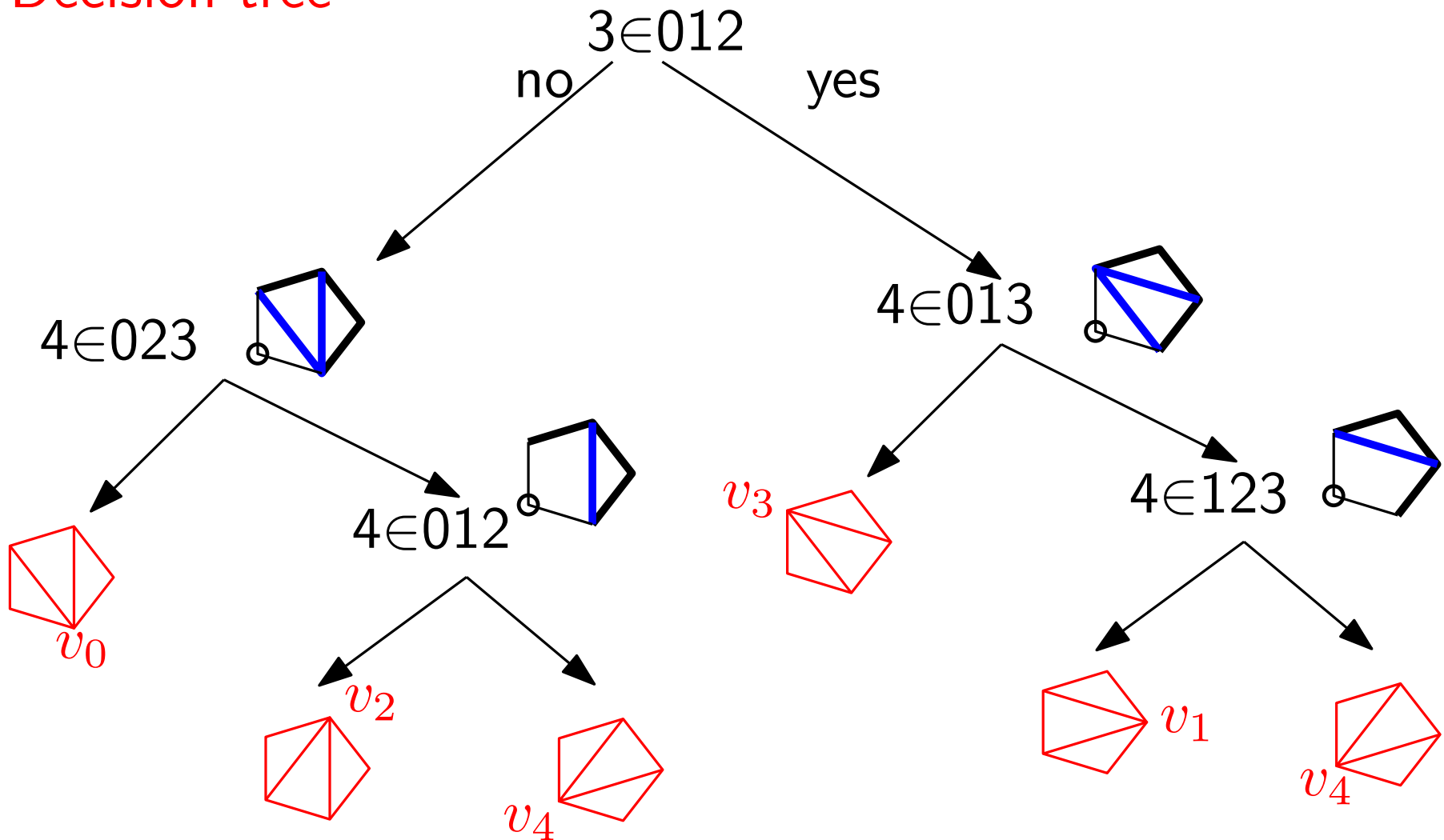
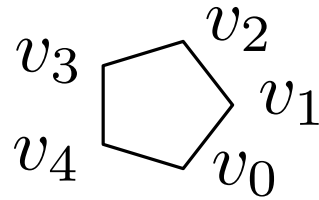
Decision tree



# Vertex removal - low degree optimization

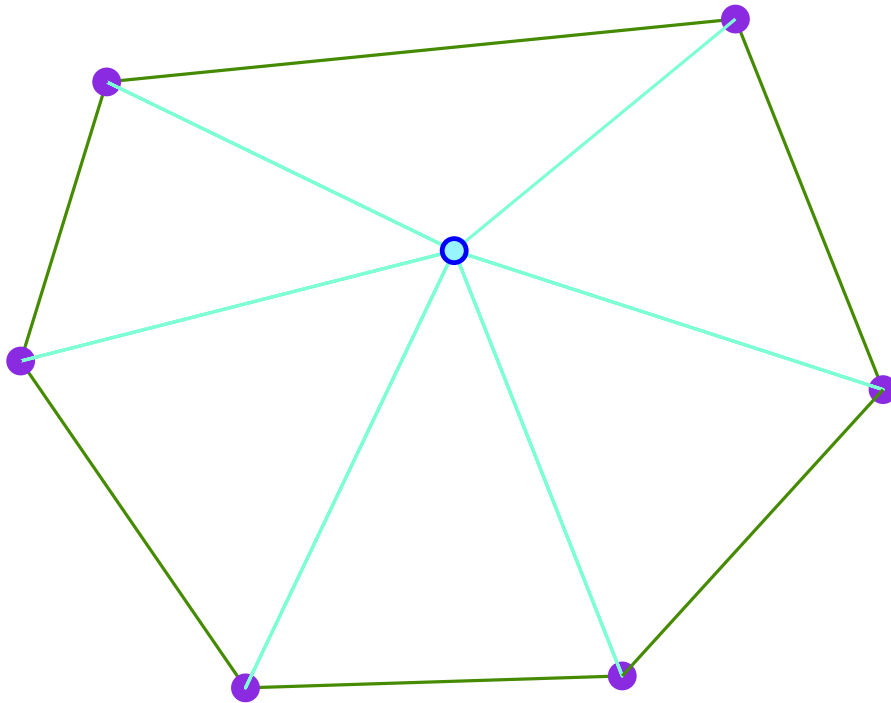
degree 5

Decision tree



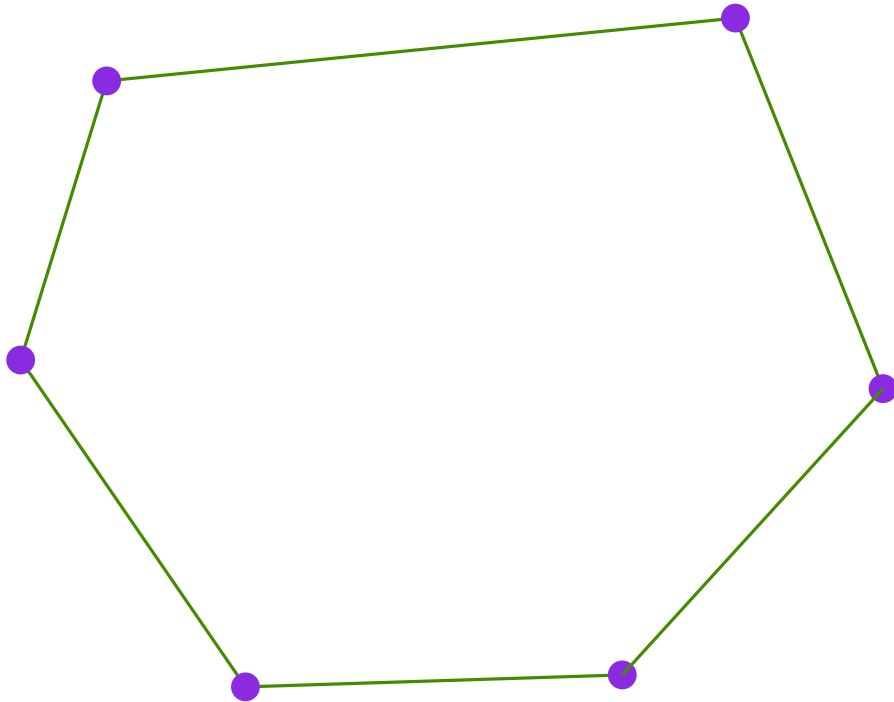
# Vertex removal - low degree optimization

degree 6



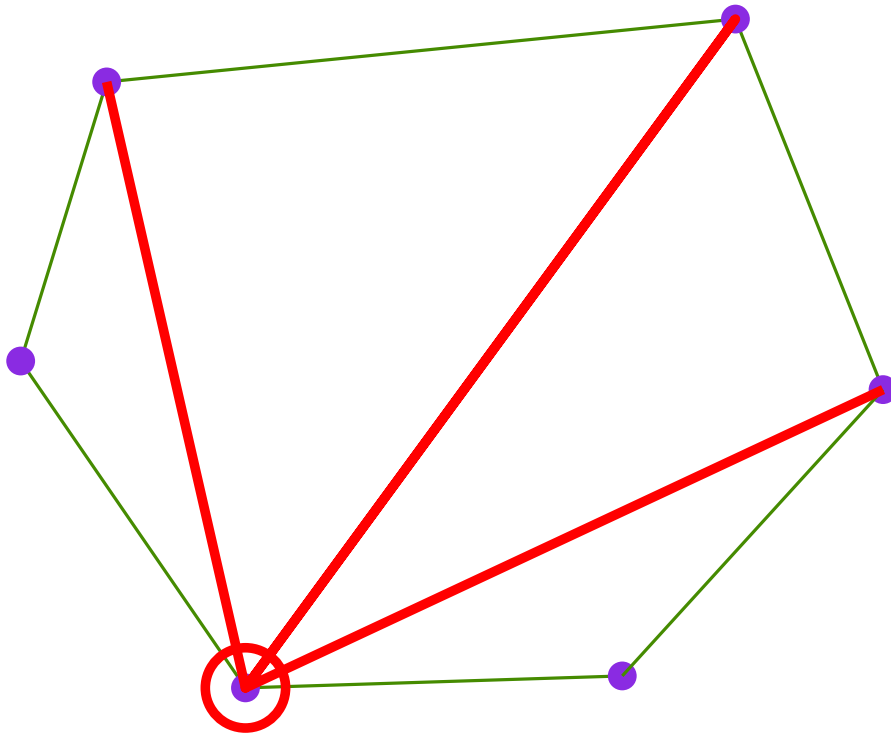
# Vertex removal - low degree optimization

degree 6



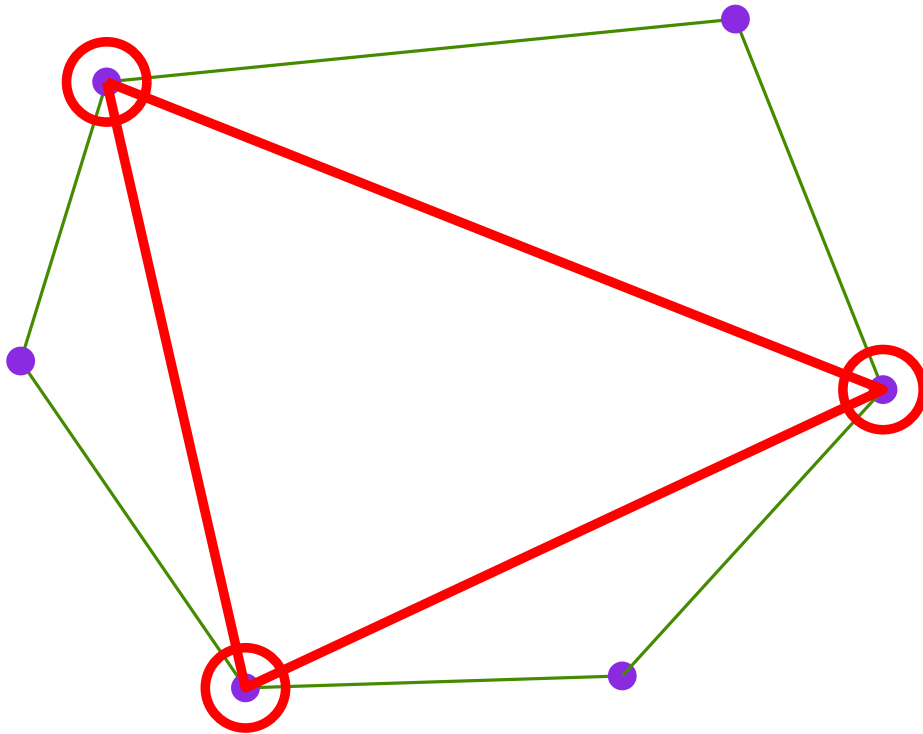
# Vertex removal - low degree optimization

degree 6



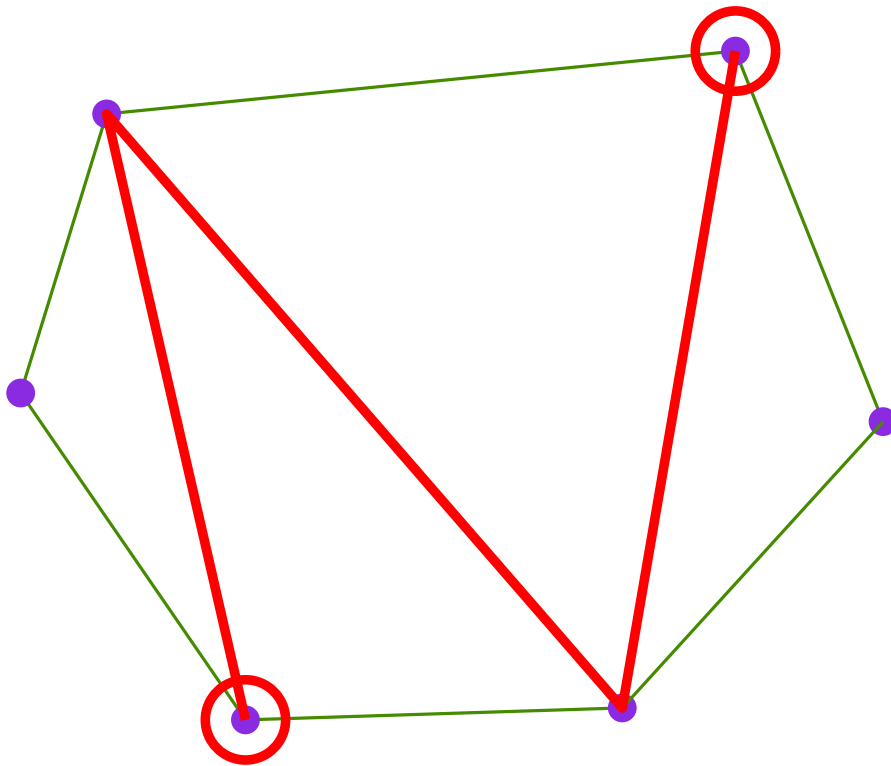
# Vertex removal - low degree optimization

degree 6



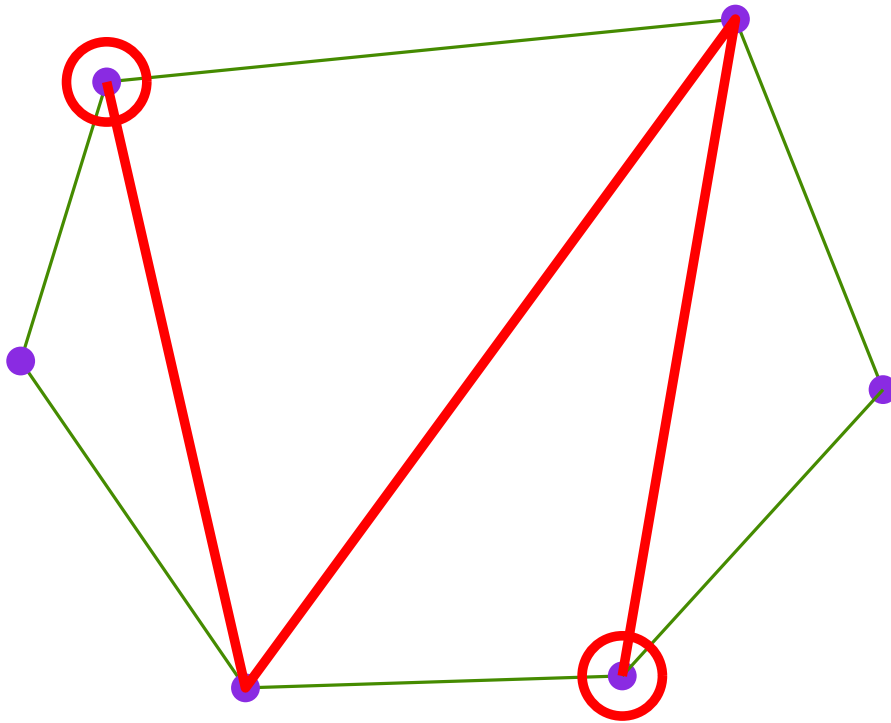
# Vertex removal - low degree optimization

degree 6



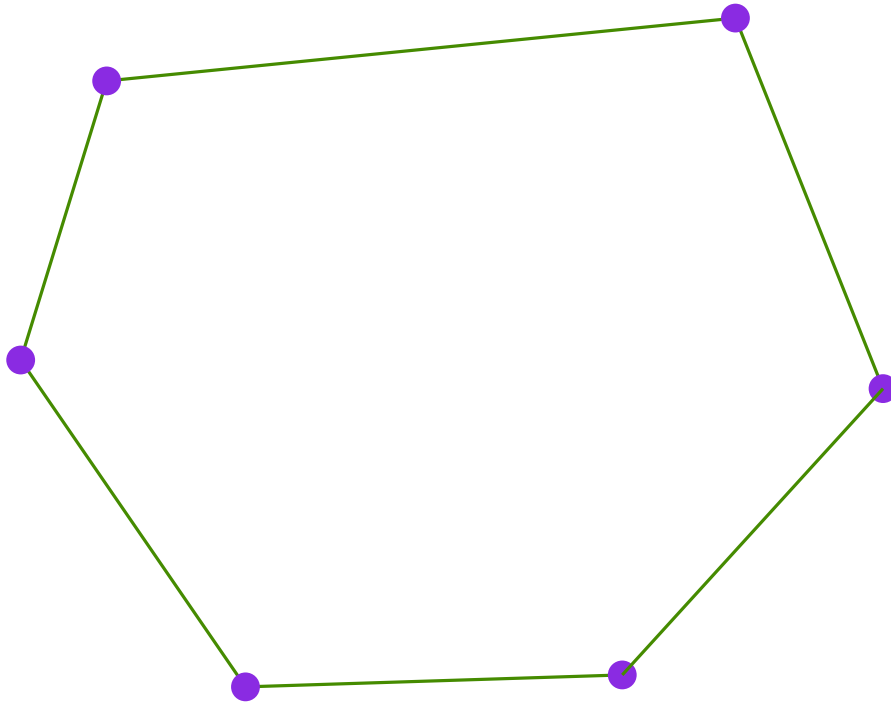
# Vertex removal - low degree optimization

degree 6



# Vertex removal - low degree optimization

degree 6



14 results



# Vertex removal - low degree optimization

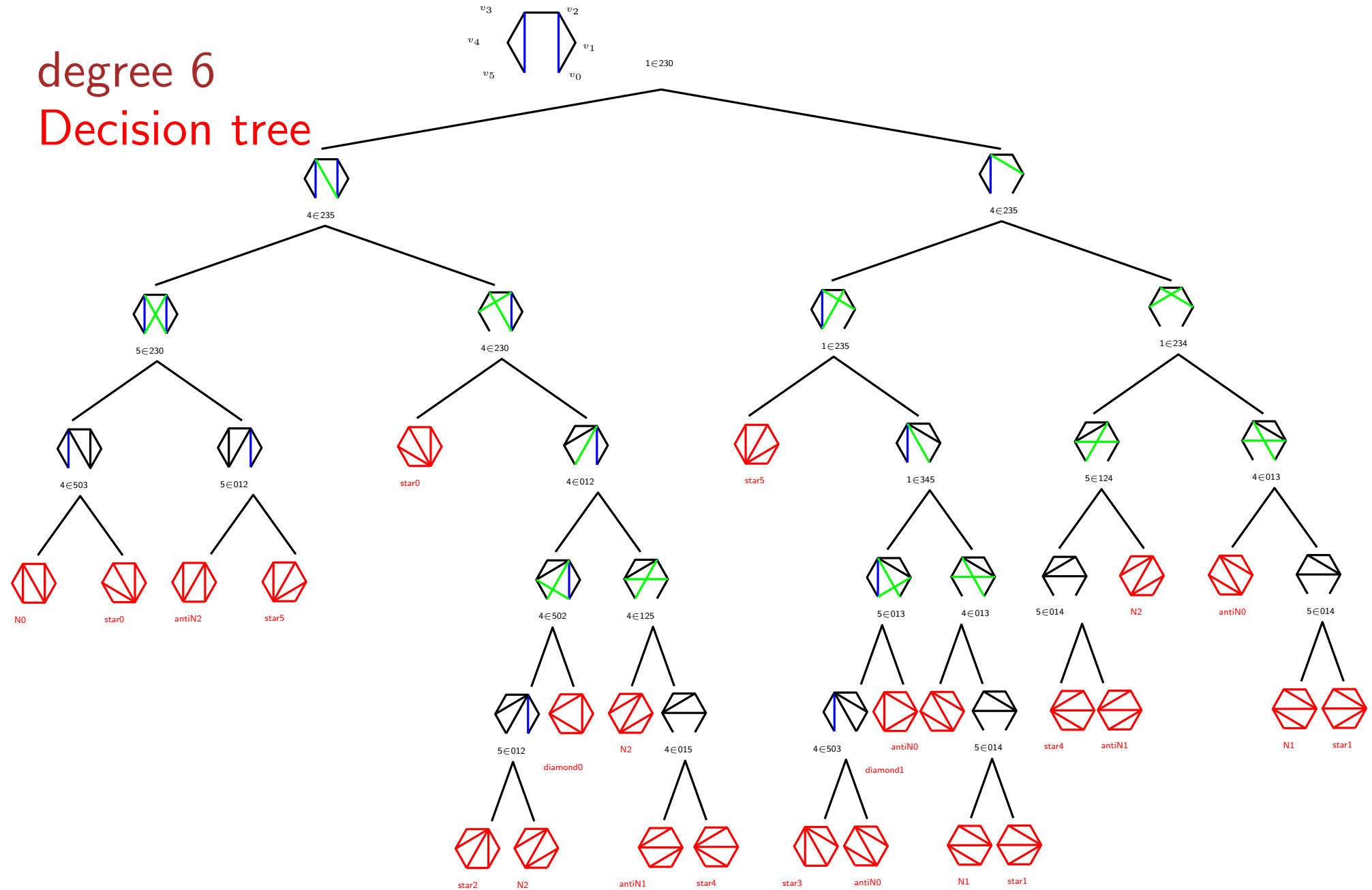
degree 6

Decision tree

# Vertex removal - low degree optimization

degree 6

Decision tree

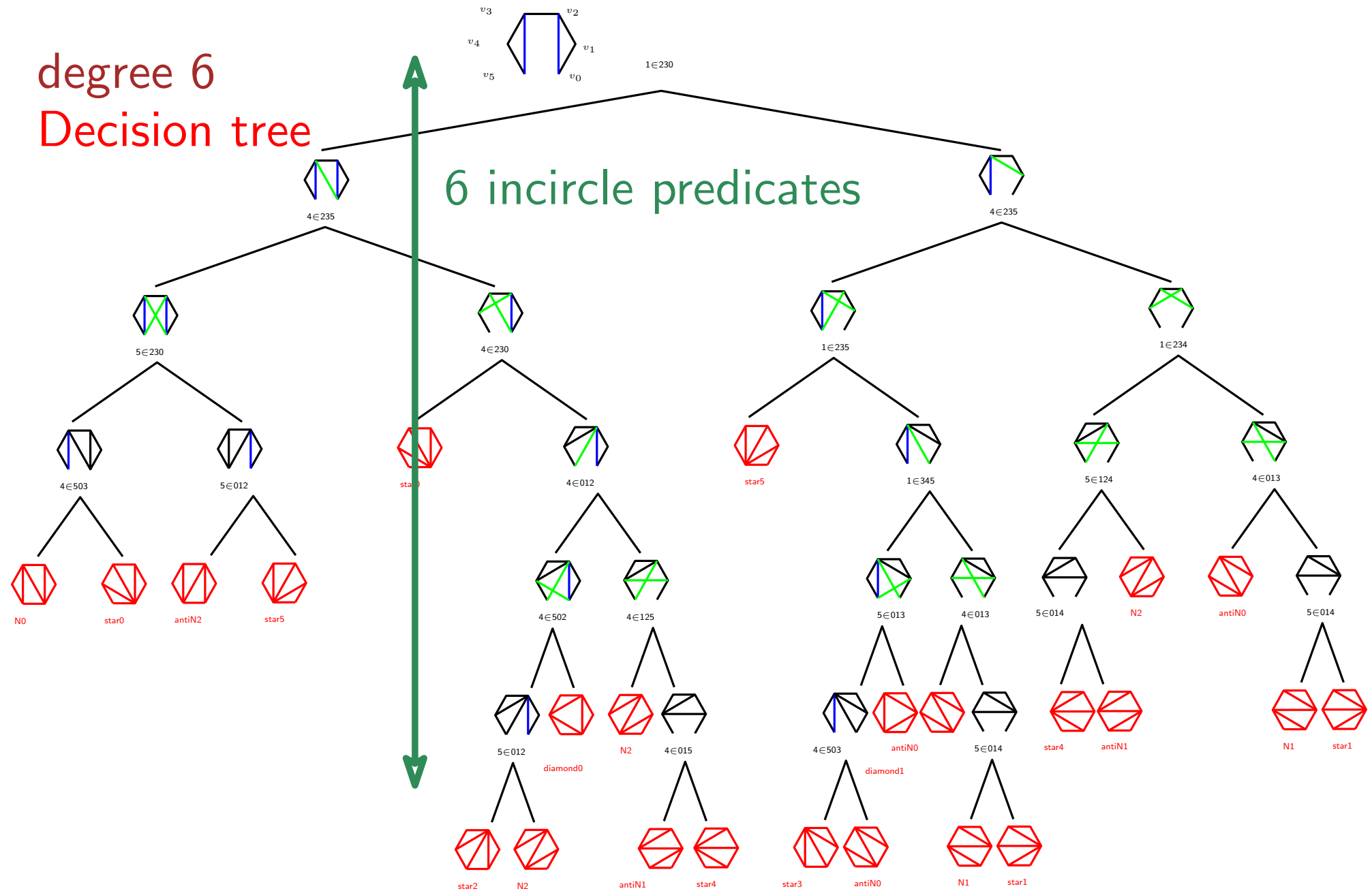


# Vertex removal - low degree optimization

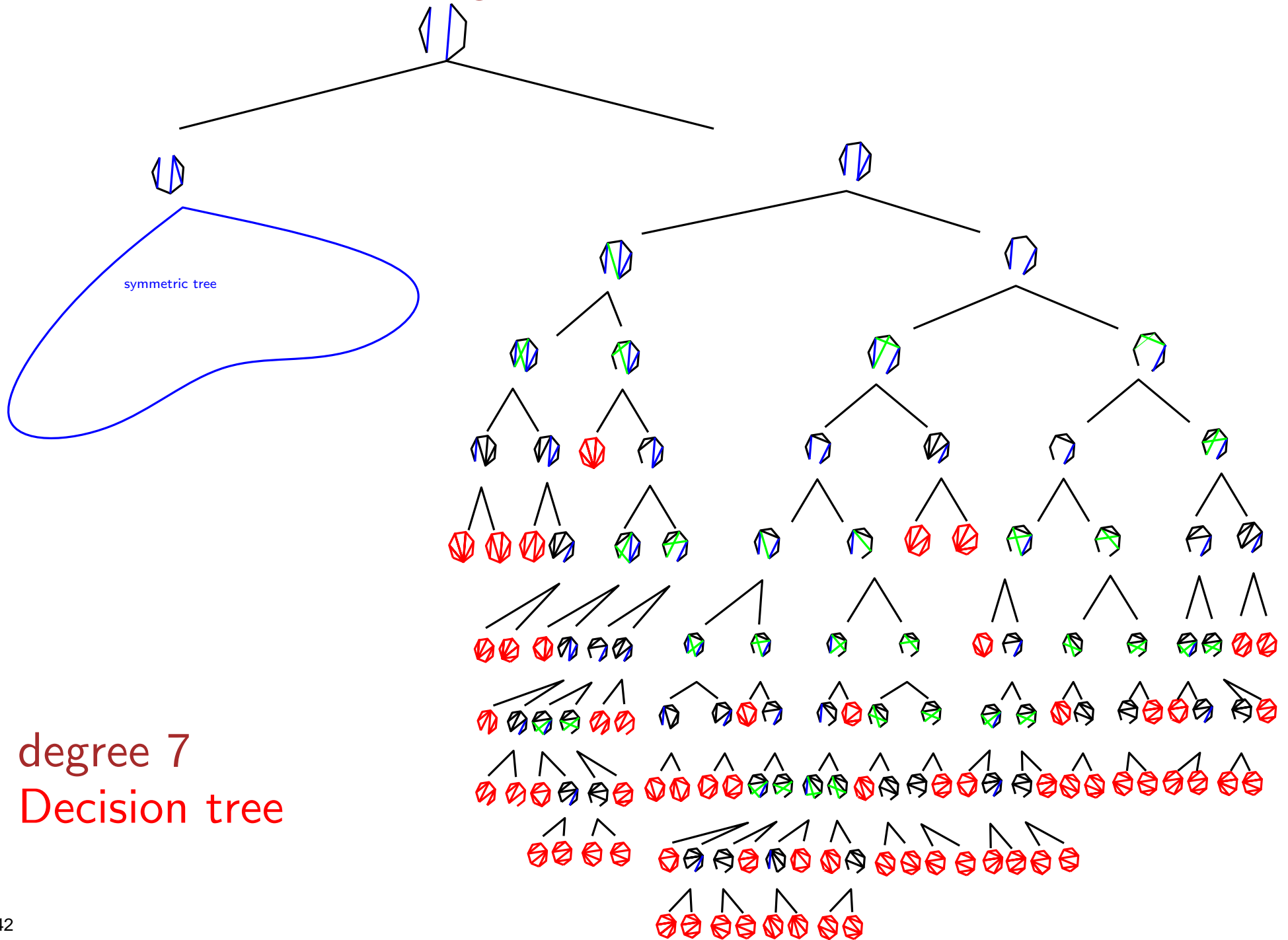
degree 6

Decision tree

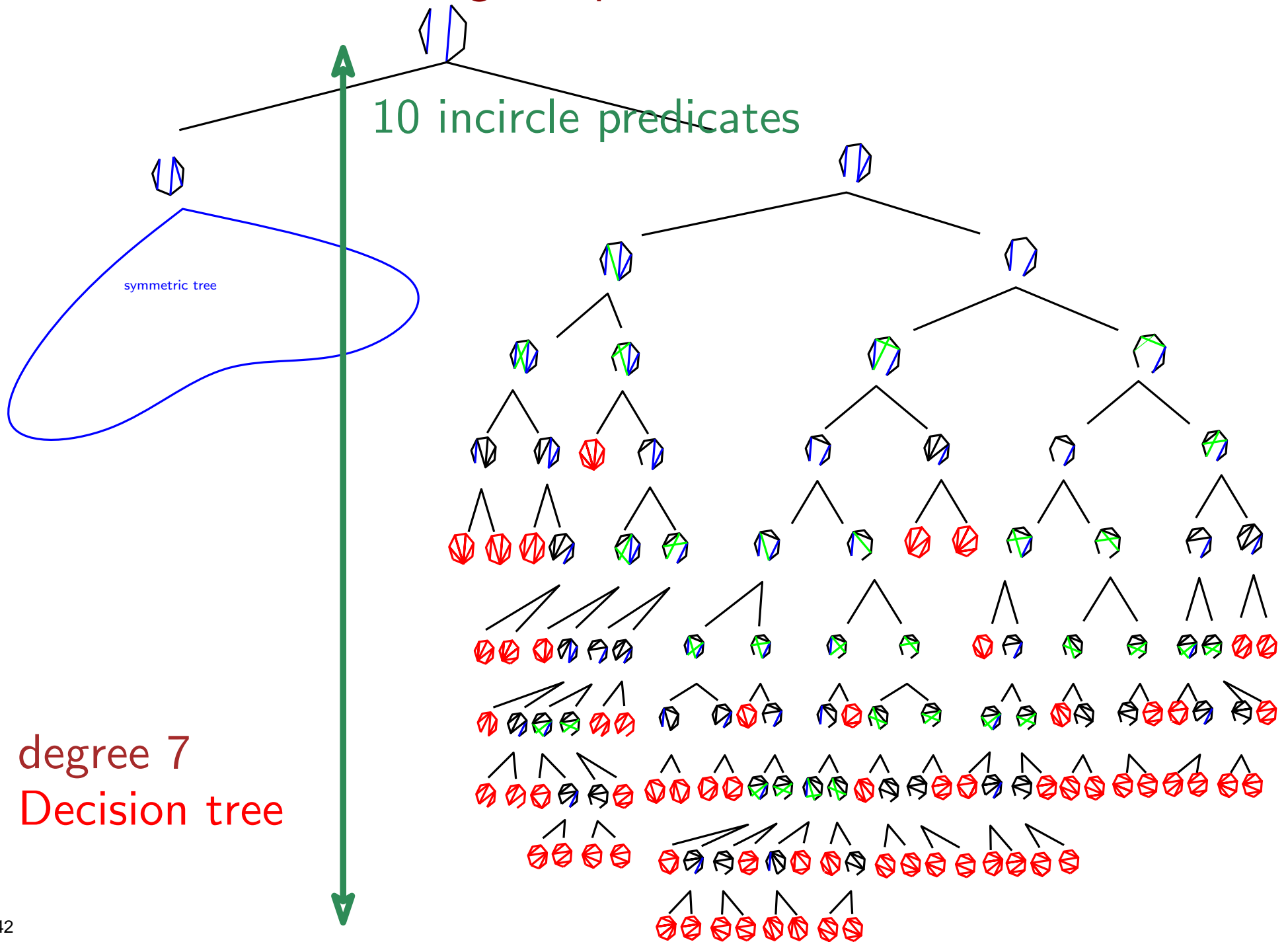
6 incircle predicates



# Vertex removal - low degree optimization



# Vertex removal - low degree optimization



## Vertex removal - low degree optimization

degree	3	4	5	6	7	8*	9
# results	1	2	5	14	42	132	429
# leaves	1	2	6	24	130	$\simeq 500$	
$\lceil \log_2 \#results \rceil$	0	1	3	4	6	8	9
tree height	0	1	3	6	10	$\simeq 14$	
# lines of code	30	40	90	280	700	$\simeq 2500$	

\* not implemented. The sizes of the tree and the code are estimated

# Vertex removal - low degree optimization

## Remarks on implementation

limited memory allocation, use old faces "in place"

re-use as many neighbor links as possible

# Vertex removal - low degree optimization

## Remarks on implementation

limited memory allocation, use old faces "in place"

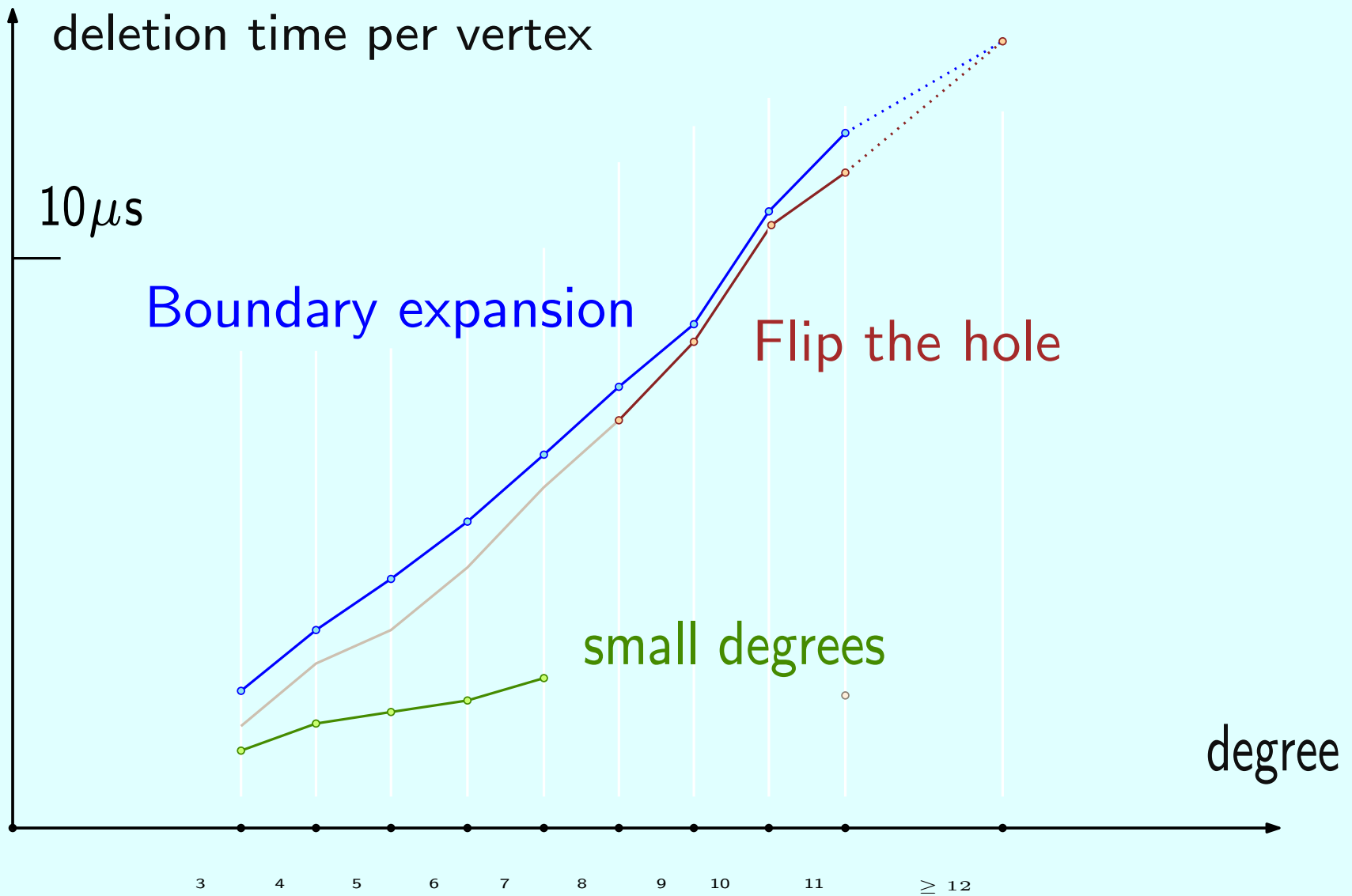
re-use as many neighbor links as possible

tree implementation

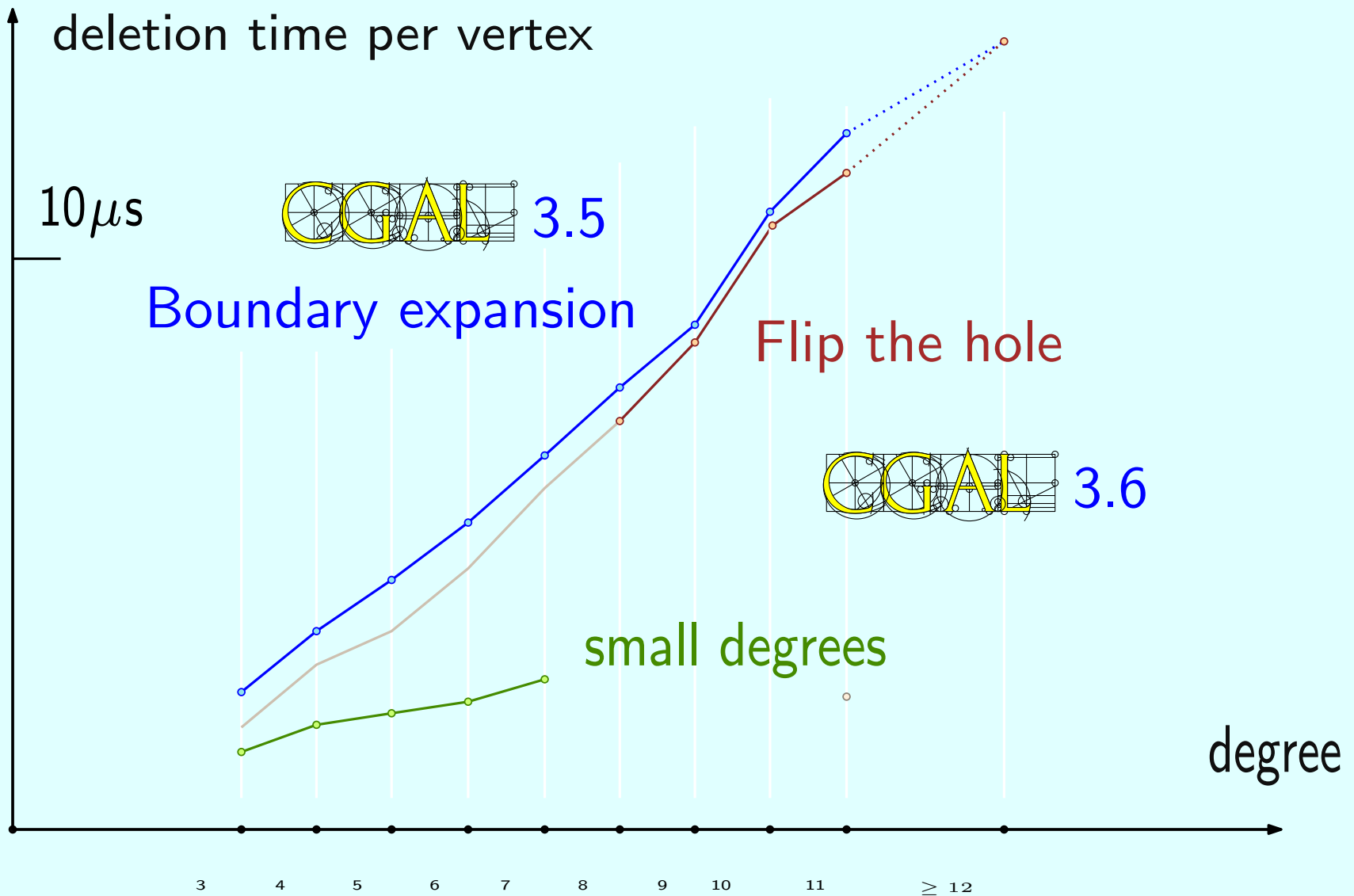
```
if incircle(...)
    if incircle(...)
        if incircle(...) use_this_shape(face0,face1,face2...)
        else               use_other_shape(face2,face3,face4...)
    .....
```



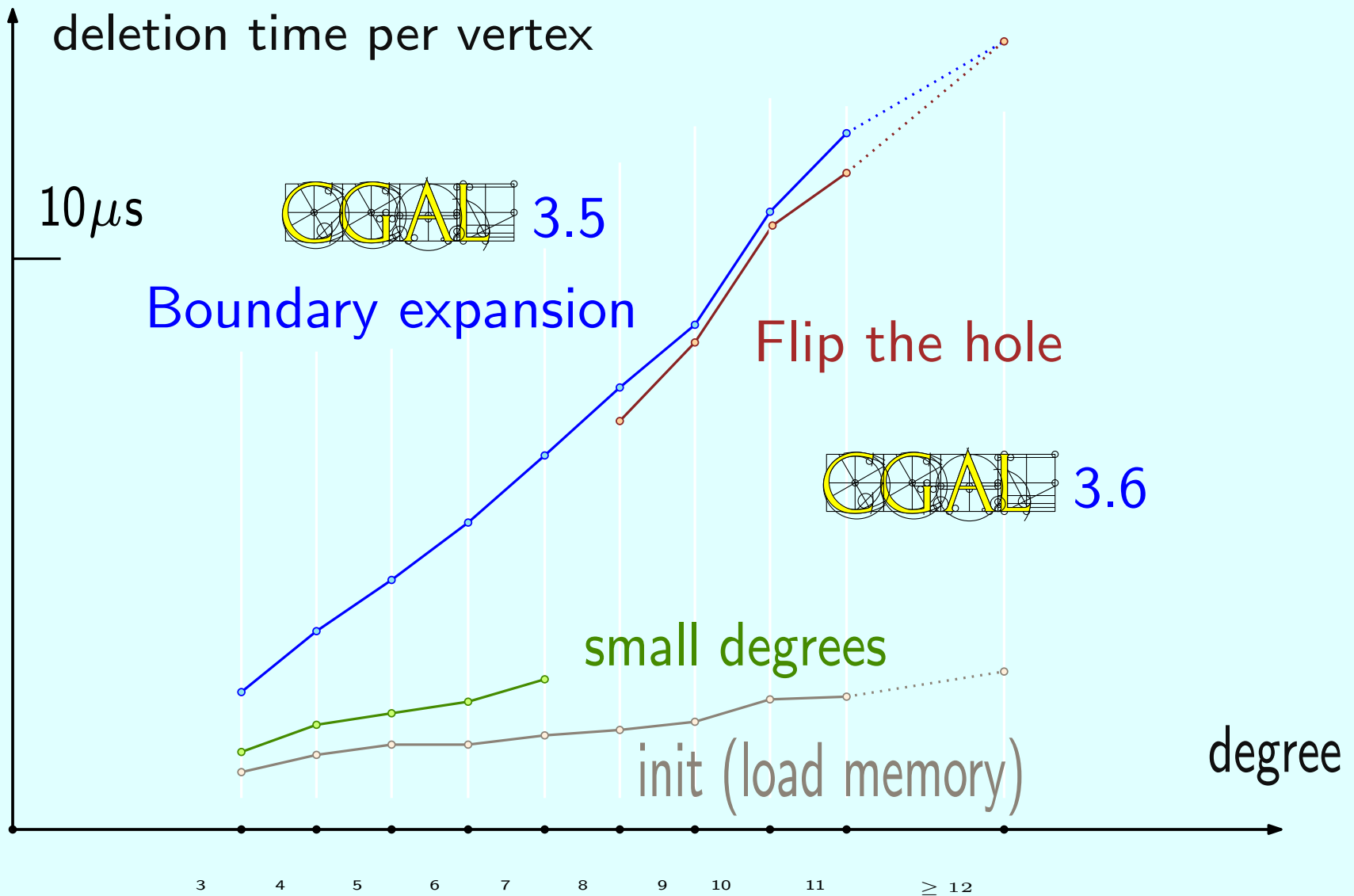
# Vertex removal



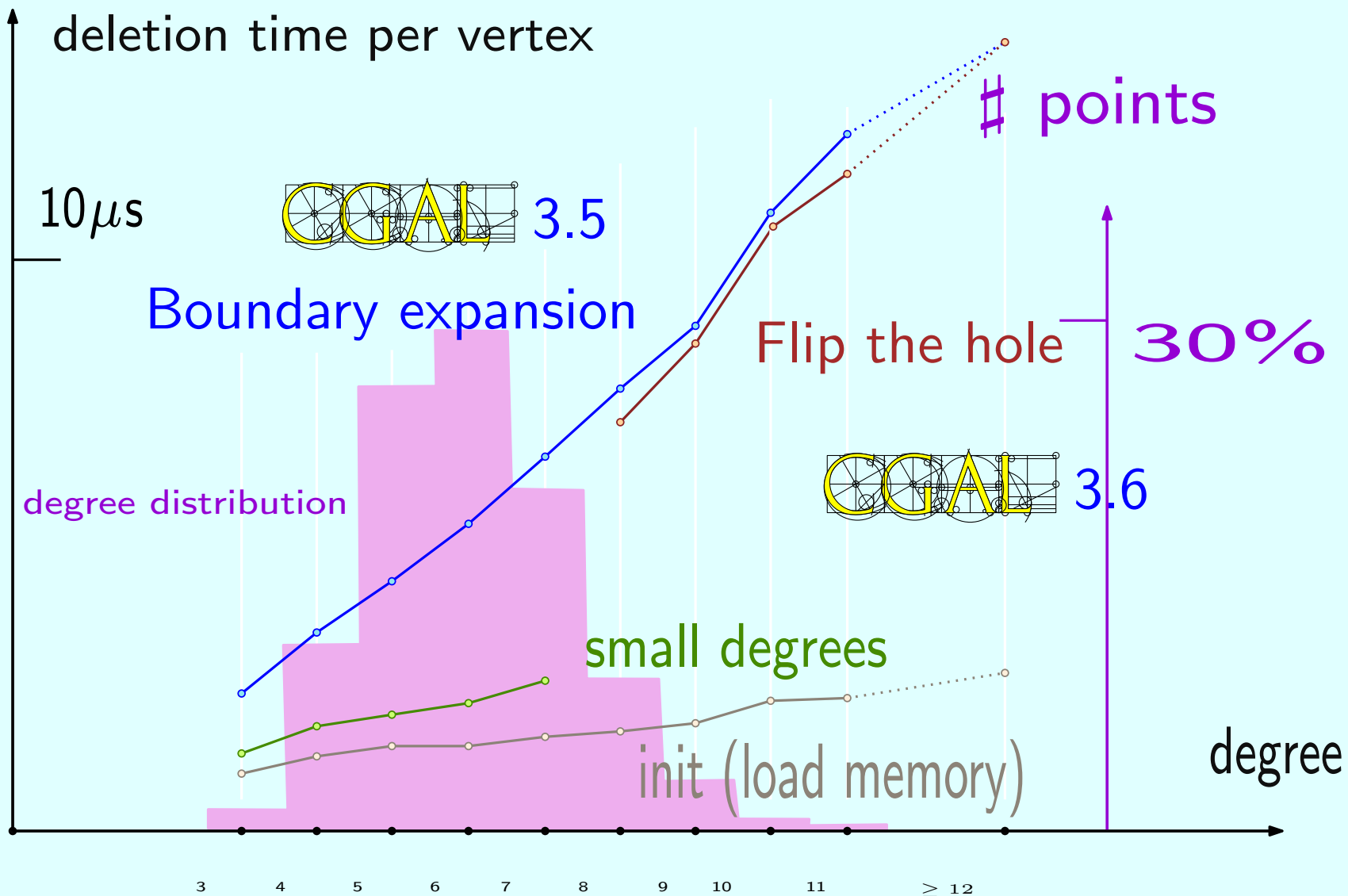
# Vertex removal



# Vertex removal



# Vertex removal





One word on robustness issues

Basic incremental algorithm

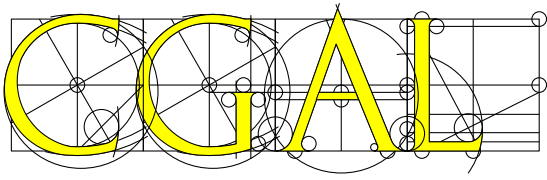
Locate by walk

Locate using randomized data structures

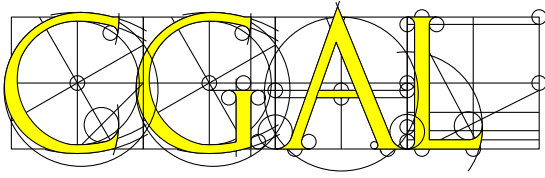
Vertex removal in 2D

Remarks on CGAL programming

Conclusion



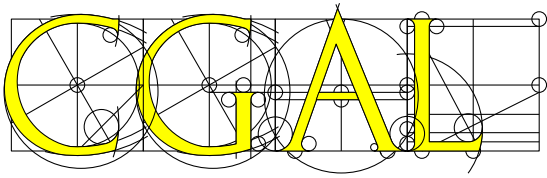
Algorithmic choices



Theoretical efficiency



Algorithmic choices



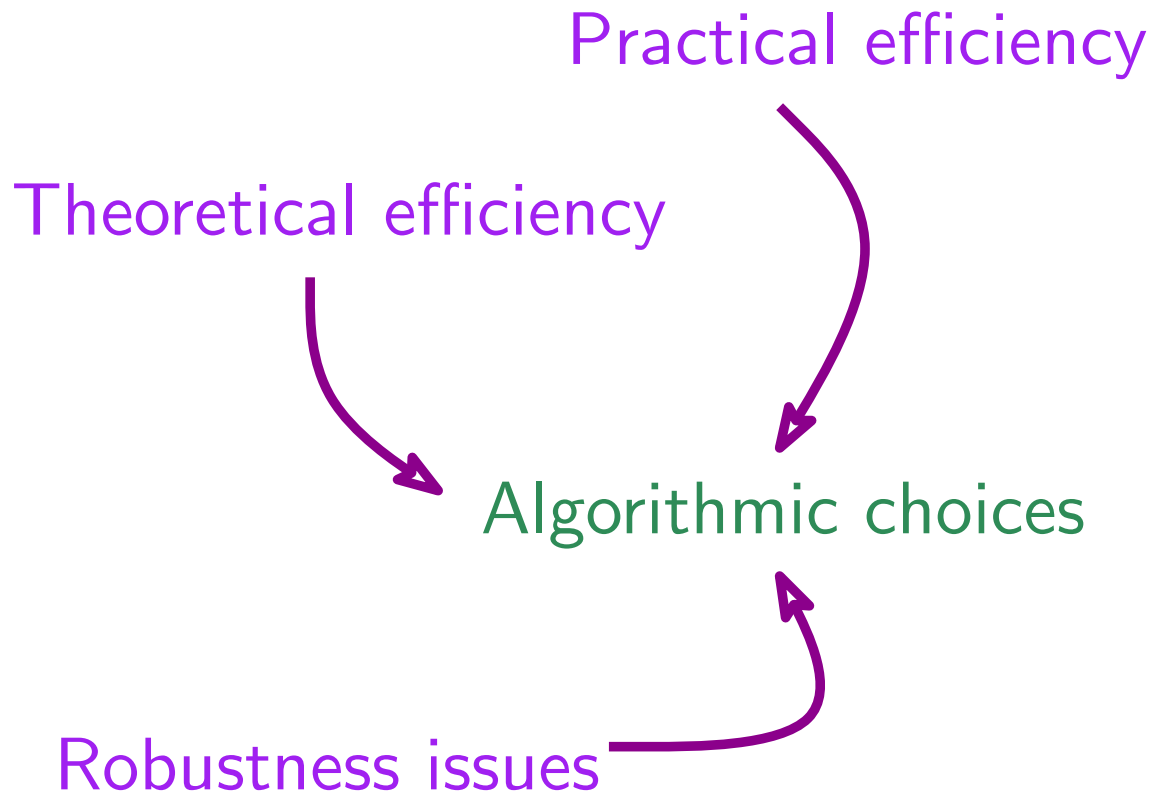
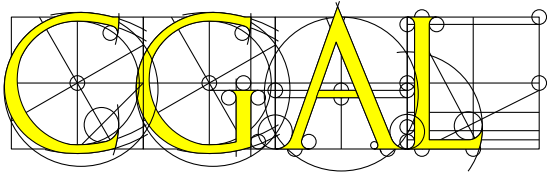
Practical efficiency

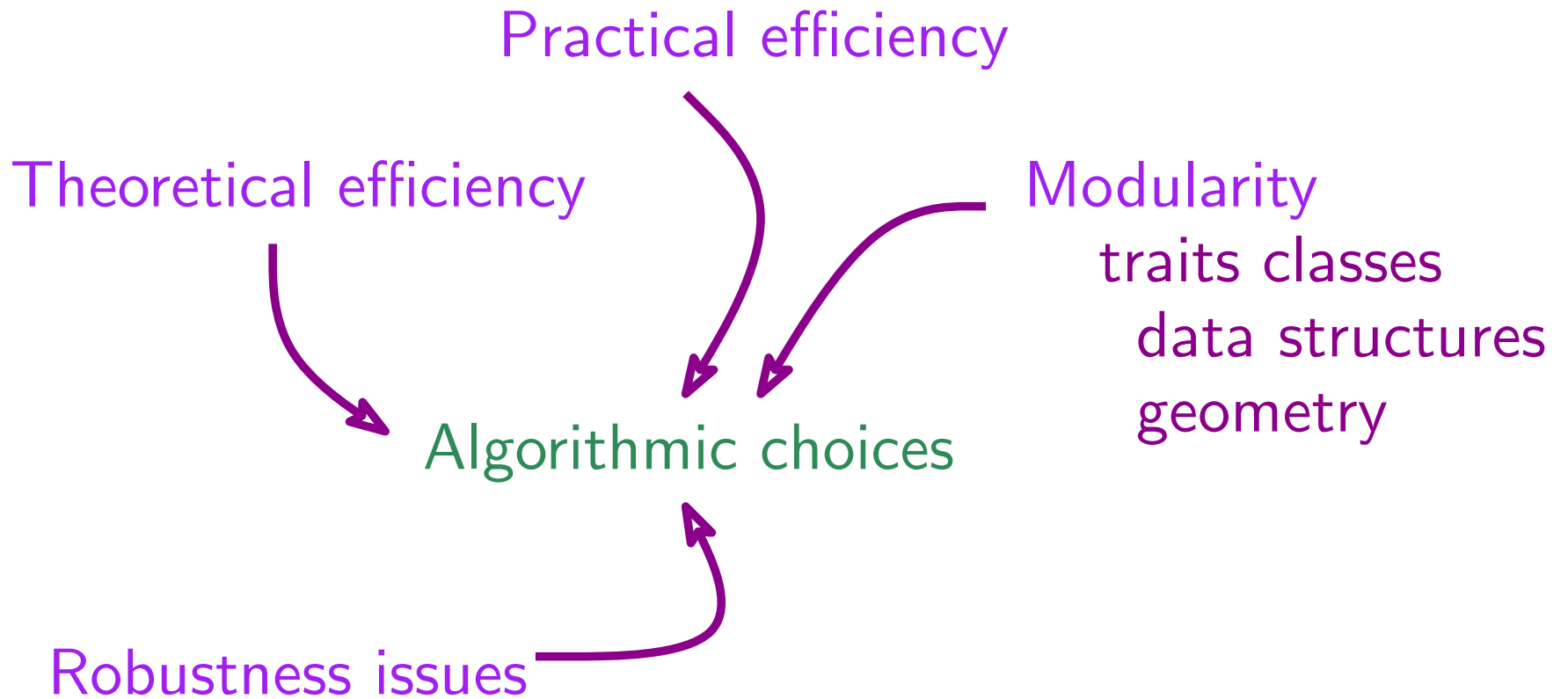
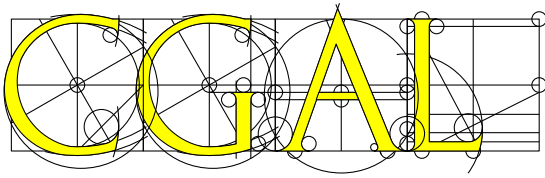
Theoretical efficiency

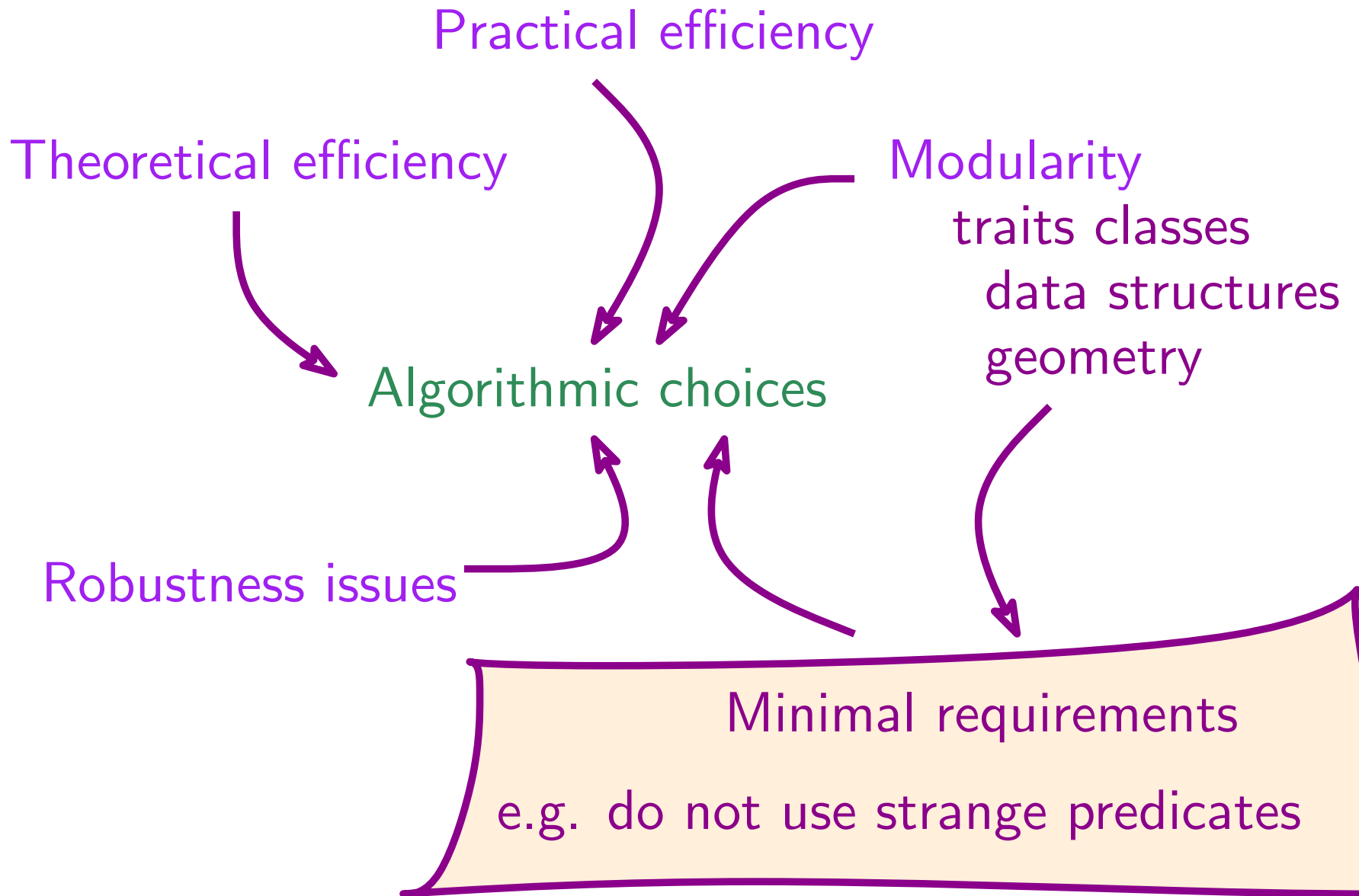
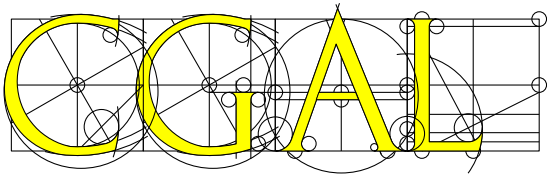
Algorithmic choices

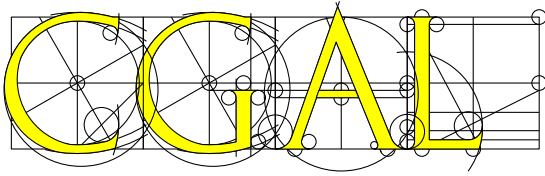






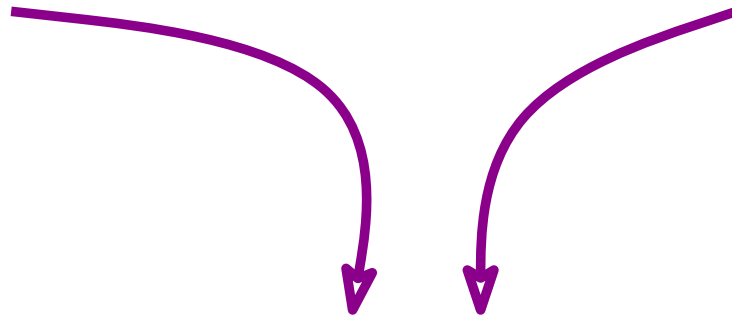




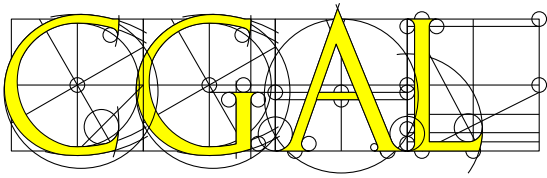


Geometric traits

Triangulation data structure



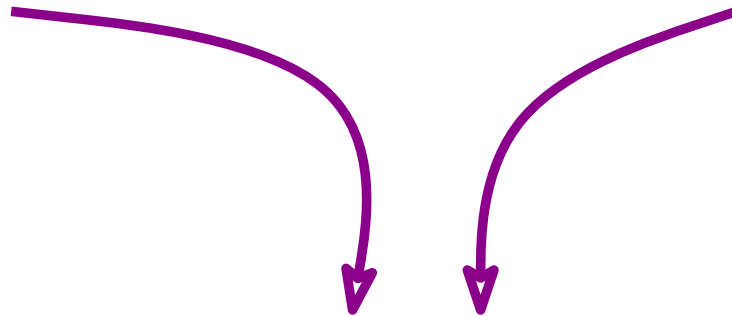
Delaunay triangulation



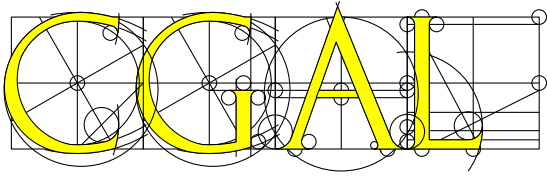
$\mathbb{R}^2$

Geometric traits

Triangulation data structure



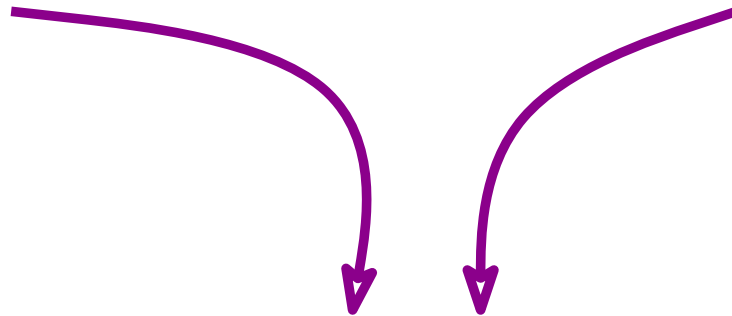
Delaunay triangulation



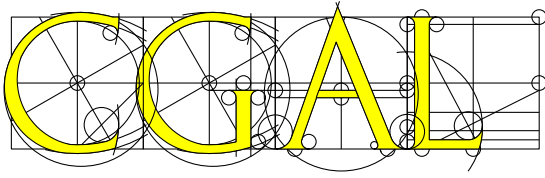
# $\mathbb{R}^3$ projection

Geometric traits

Triangulation data structure



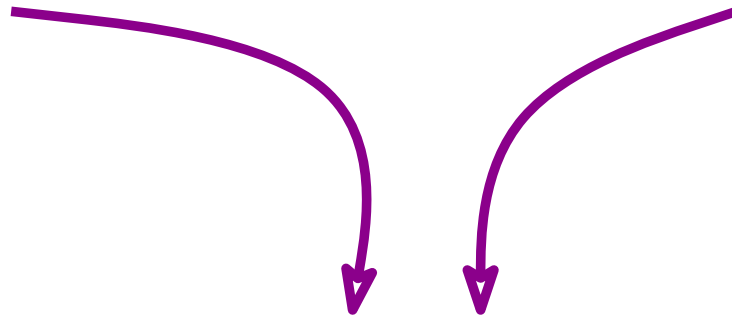
Delaunay triangulation



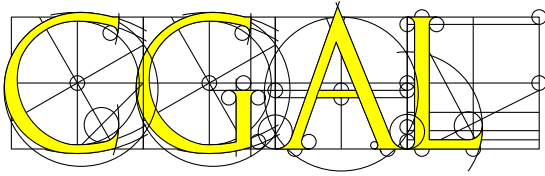
# $\mathbb{R}^2$ other metric

Geometric traits

Triangulation data structure

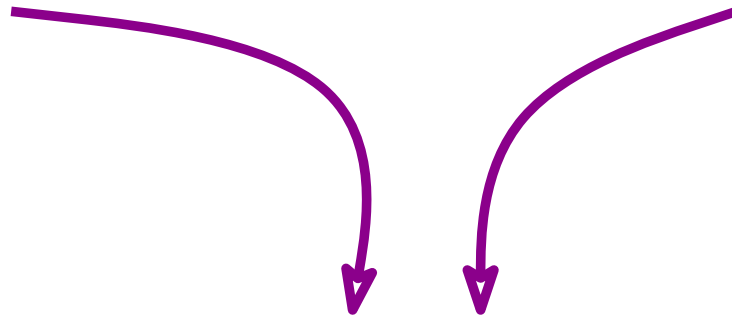


Delaunay triangulation



Geometric traits

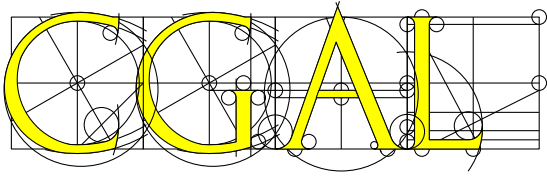
Triangulation data structure



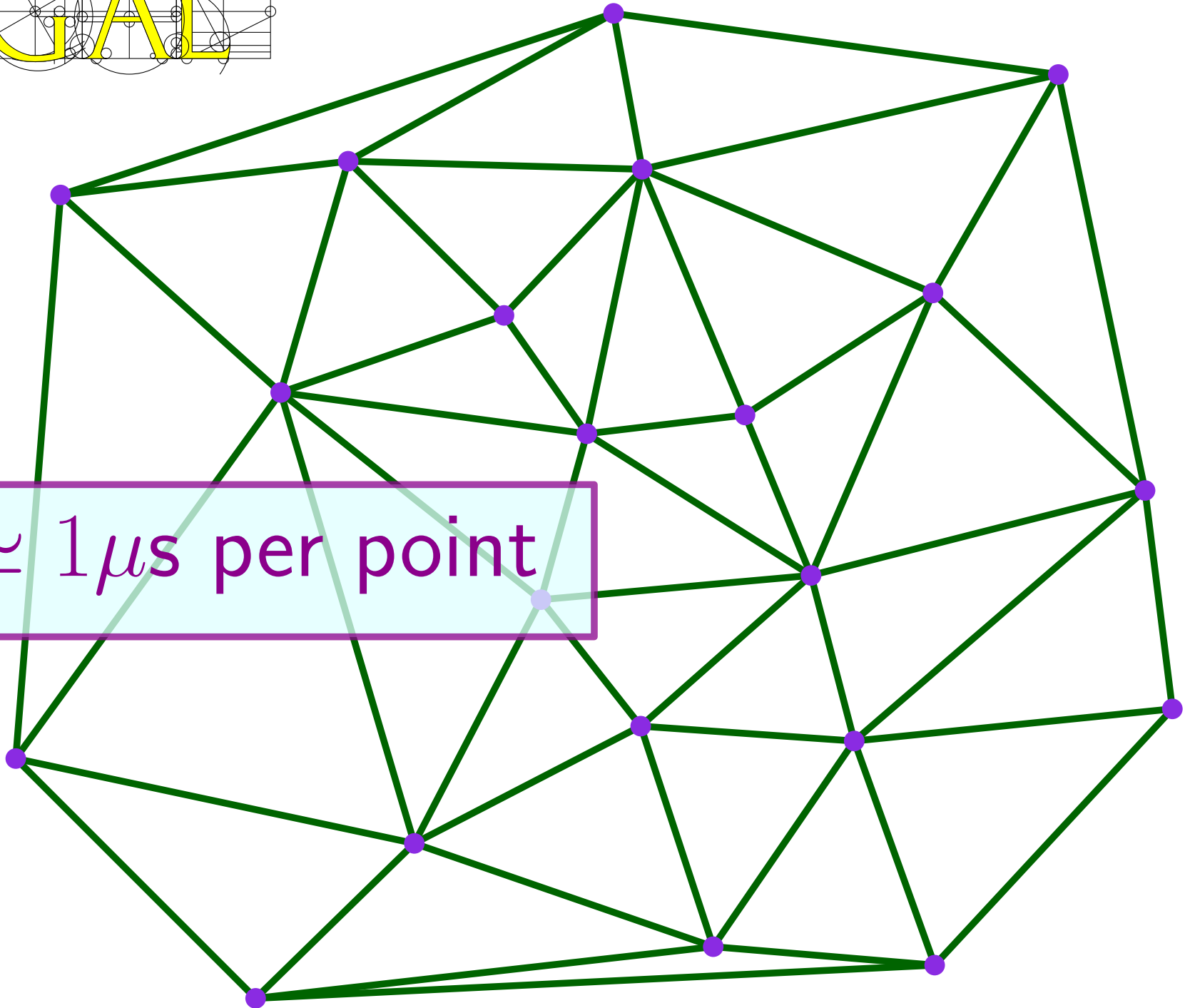
Delaunay triangulation

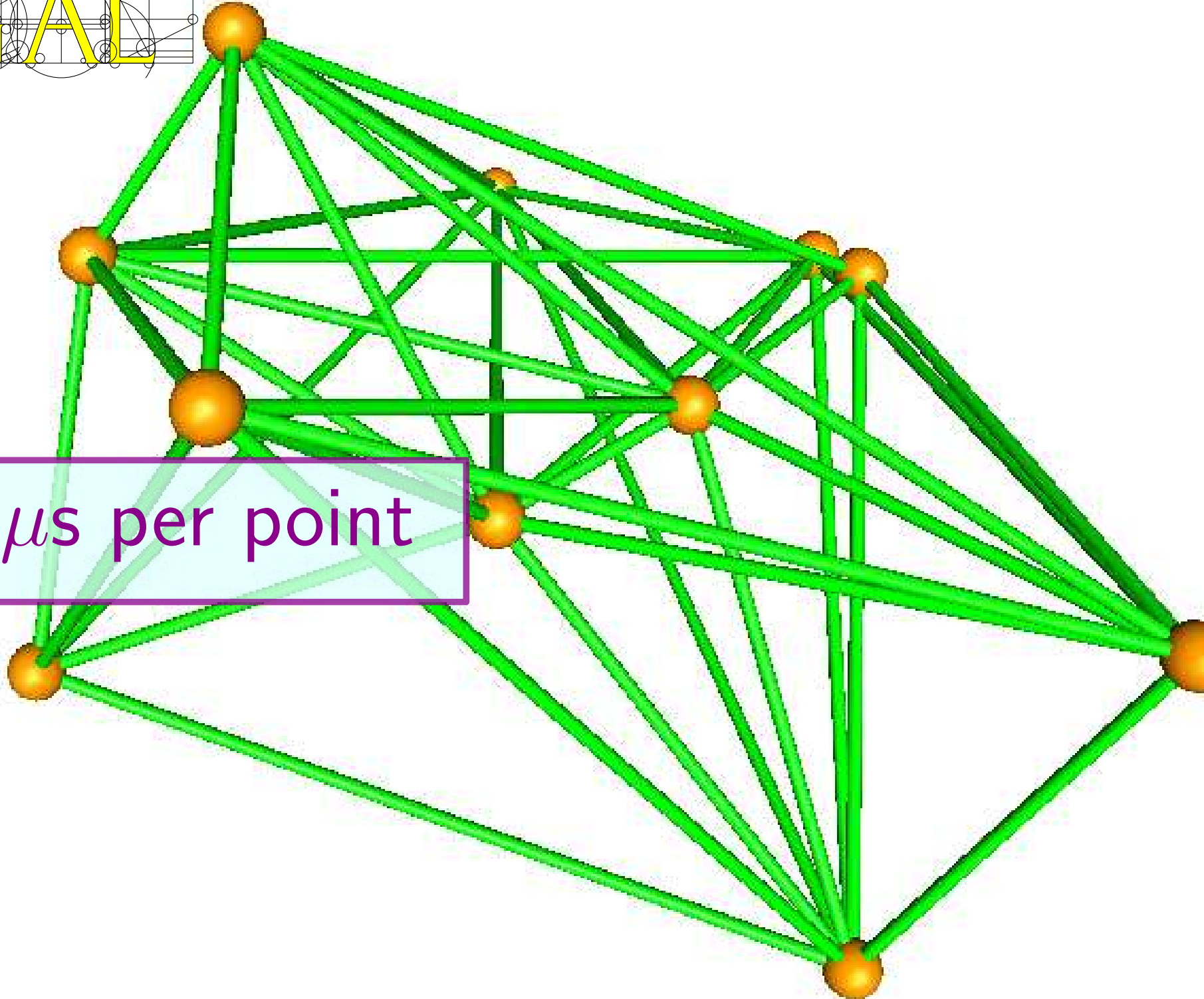
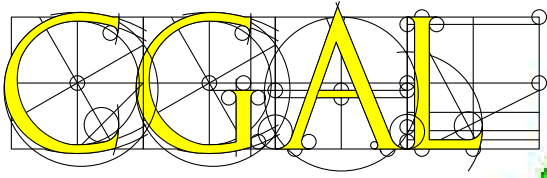
# Periodic triangulation





$\approx 1\mu s$  per point





$\approx 8\mu s$  per point



One word on robustness issues

Basic incremental algorithm

Locate by walk

Locate using randomized data structures

Vertex removal in 2D

Remarks on CGAL programming

Conclusion

Delaunay challenges

# Delaunay challenges

Practical vs worst case size of Delaunay 3D

# Delaunay challenges

## Practical vs worst case size of Delaunay 3D

### Known results

$\Theta(n^2)$  worst case

$\Theta(n)$  random in ball

$\Omega(n)O(n \log n)$  random on polyhedron

$O(n \log n)$  good sample of smooth generic surface

$\Theta(n \log n)$  random on cylinder

# Delaunay challenges

## Practical vs worst case size of Delaunay 3D

### Known results

$\Theta(n^2)$  worst case

Find good models of practical data

$\Theta(n)$  random in ball

(Smooth analysis)

$\Omega(n)O(n \log n)$  random on polyhedron

$O(n \log n)$  good sample of smooth generic surface

$\Theta(n \log n)$  random on cylinder

# Delaunay challenges

Practical vs worst case size of Delaunay 3D

Better algorithm for 3D deletion

10  $\mu s$  to insert

100  $\mu s$  to delete

# Delaunay challenges

Practical vs worst case size of Delaunay 3D

Better algorithm for 3D deletion

One billion points

Needs memory efficient algorithms

Cache effects are already important





# *Questions*