OI-OPPA. European Social Fund

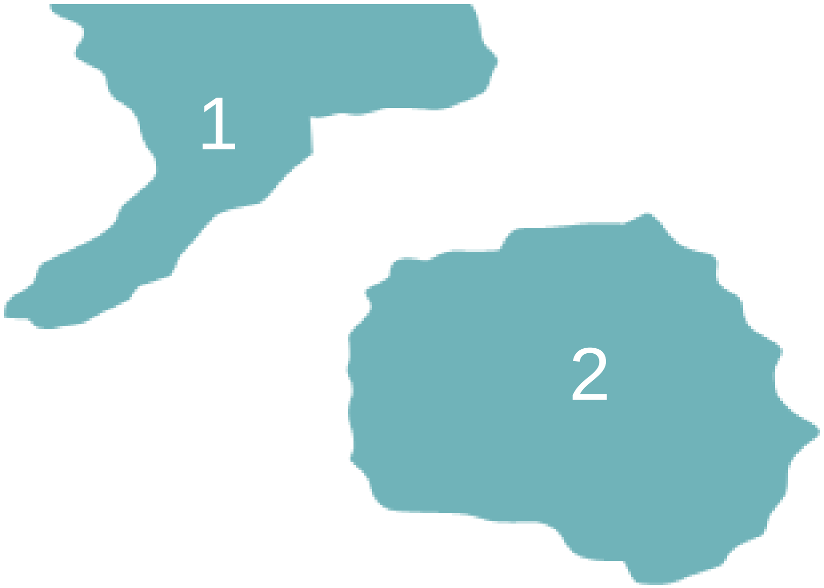
Prague & EU: We invest in your future.

DEPARTMENT OF COMPUTER GRAPHICS AND INTERACTION

Overlay of planar subdivisions

Radek Smetana

Overlay of planar subdivisions



Overlay of planar subdivisions

Overlay of planar subdivisions

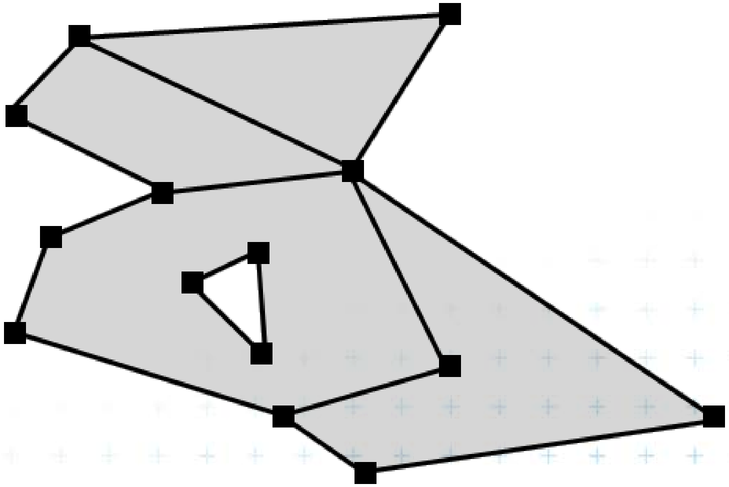
Requirements and Goals

- Representation of planar subdivision
- Computing intersections of line segments
- Merging subdivisions
- Boolean operations

DCGI

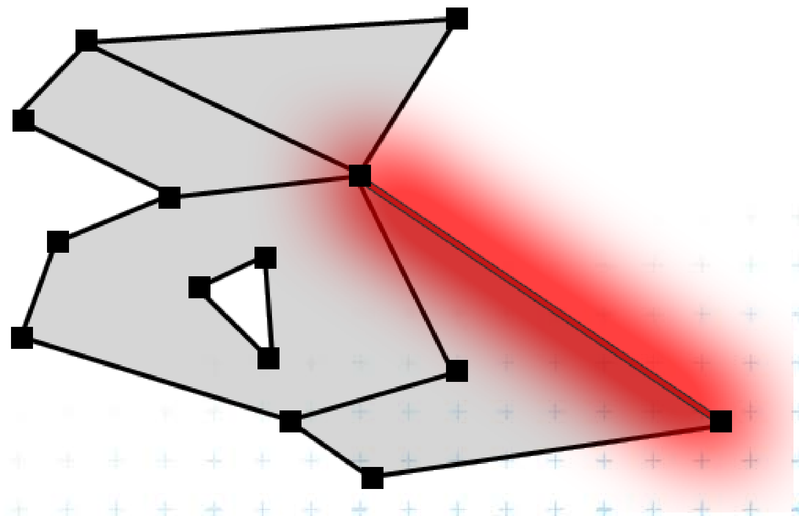
The Doubly-Connected Edge List

- Data structure with basic operations
- Edges are the straight lines, not crossing each other (only in verteces)
- Three collections: verteces, edges and faces



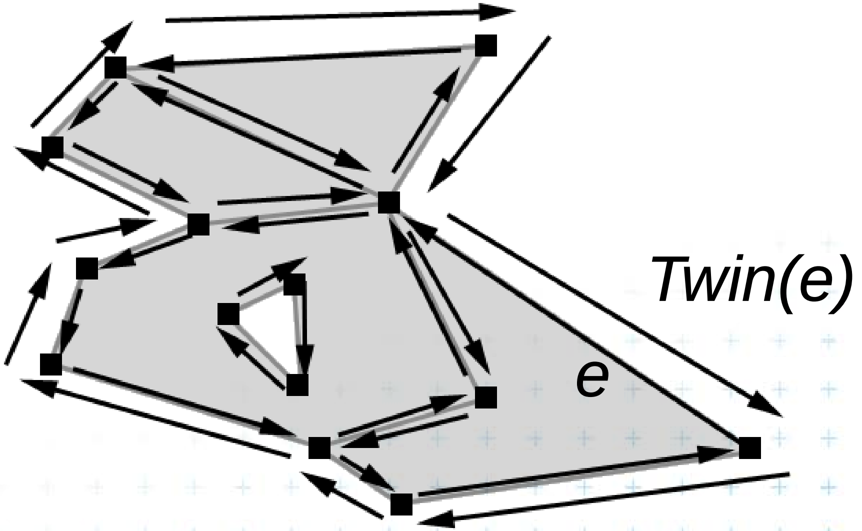
The Doubly-Connected Edge List

- Edge
  - **Open** - endpoints (= verteces) not included
  - Divided to **half-edges** - Two vectors in both directions



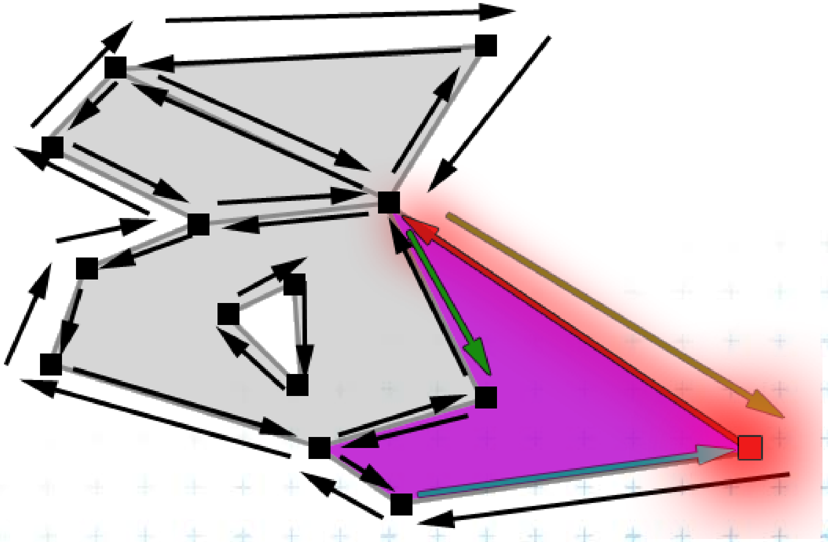
The Doubly-Connected Edge List

- Half-edge
  - Walk around a face in counterclockwise order
  - Holes have oposite direction
  - Half-edge e and Twin (e)



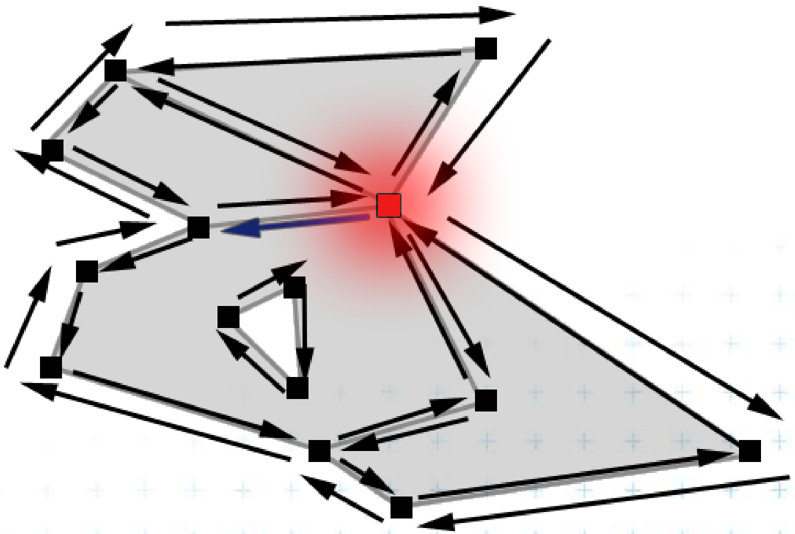
The Doubly-Connected Edge List

- Half-edge
  - e
  - Twin(e)
  - Origin(e)
  - IncidentFace(e)
  - Next(e)
  - Prev(e)



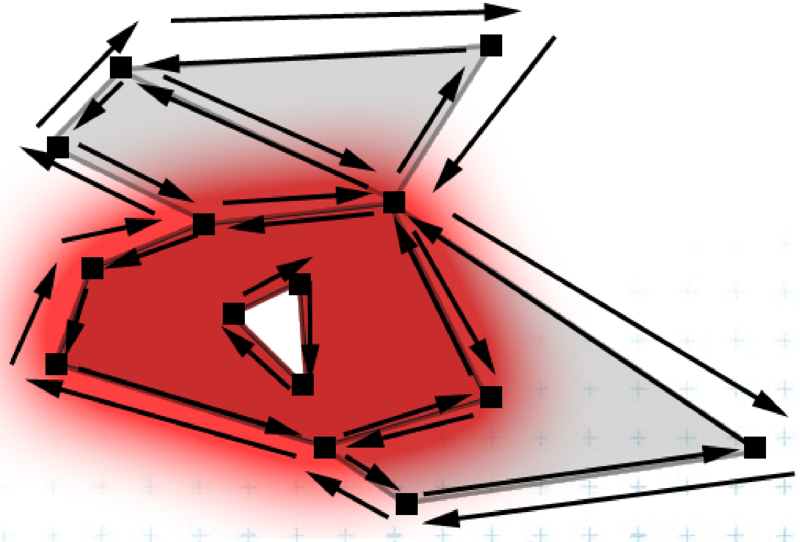
The Doubly-Connected Edge List

- Vertex
  - Coordinates(v)
  - IncidentEdge(v) – (náhodná) arbitary half-edge



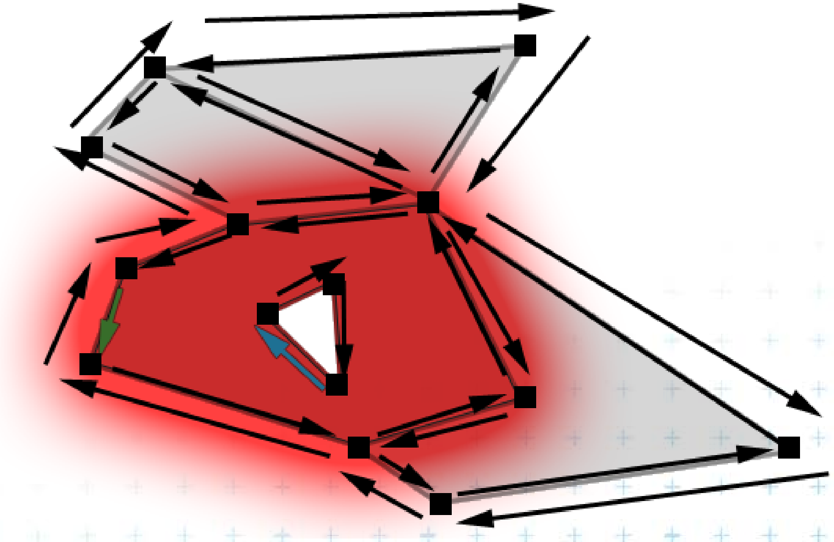
The Doubly-Connected Edge List

- Face
  - Do not contain a point on an edge or a vertex
  - On the left side of the half-edge

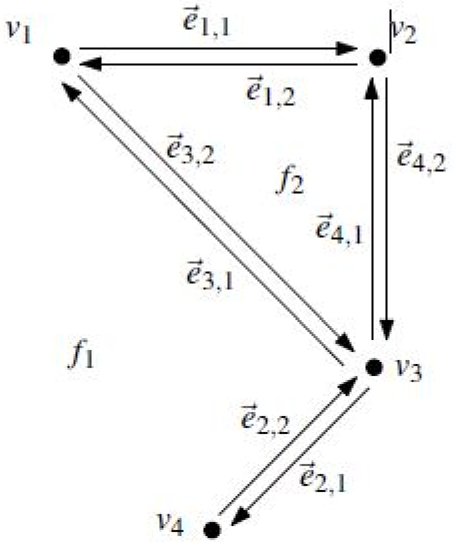


The Doubly-Connected Edge List

- Face
  - OuterComponent (f) - Unbounded has nil
  - InnerComponents (f) – Each hole – one pointer



The Doubly-Connected Edge List



Vertex	Coordinates	IncidentEdge
v_1	$(0,4)$	$\vec{e}_{1,1}$
v_2	$(2,4)$	$\vec{e}_{4,2}$
v_3	$(2,2)$	$\vec{e}_{2,1}$
v_4	$(1,1)$	$\vec{e}_{2,2}$

Face	OuterComponent	InnerComponents
f_1	**nil**	$\vec{e}_{1,1}$
f_2	$\vec{e}_{4,1}$	**nil**

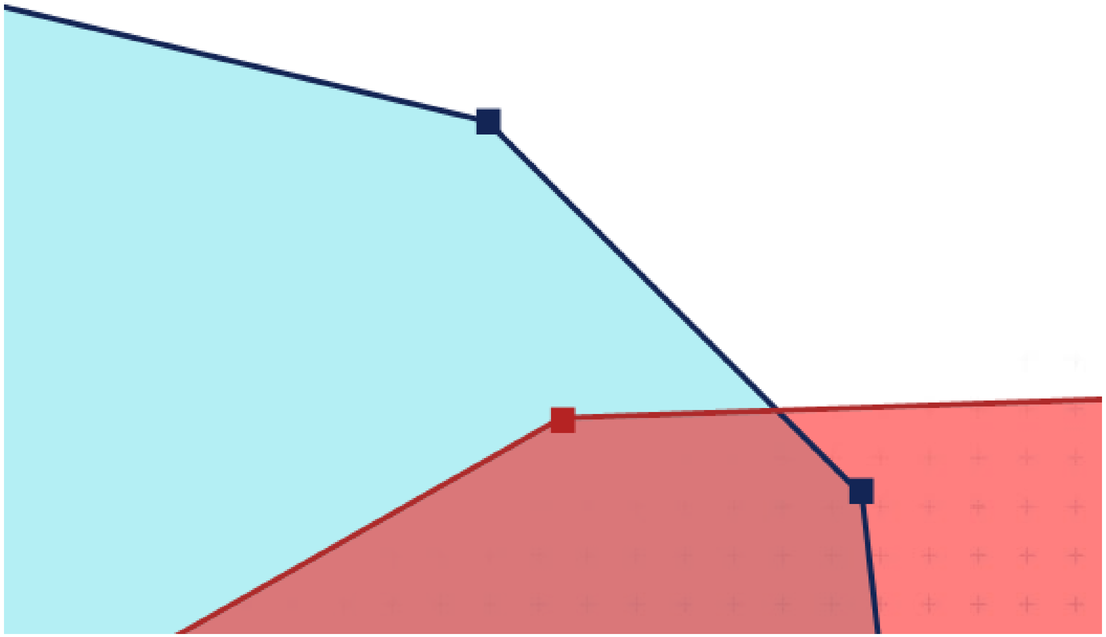
Half-edge	Origin	Twin	IncidentFace	Next	Prev
$\vec{e}_{1,1}$	v_1	$\vec{e}_{1,2}$	f_1	$\vec{e}_{4,2}$	$\vec{e}_{3,1}$
$\vec{e}_{1,2}$	v_2	$\vec{e}_{1,1}$	f_2	$\vec{e}_{3,2}$	$\vec{e}_{4,1}$
$\vec{e}_{2,1}$	v_3	$\vec{e}_{2,2}$	f_1	$\vec{e}_{2,2}$	$\vec{e}_{4,2}$
$\vec{e}_{2,2}$	v_4	$\vec{e}_{2,1}$	f_1	$\vec{e}_{3,1}$	$\vec{e}_{2,1}$
$\vec{e}_{3,1}$	v_3	$\vec{e}_{3,2}$	f_1	$\vec{e}_{1,1}$	$\vec{e}_{2,2}$
$\vec{e}_{3,2}$	v_1	$\vec{e}_{3,1}$	f_2	$\vec{e}_{4,1}$	$\vec{e}_{1,2}$
$\vec{e}_{4,1}$	v_3	$\vec{e}_{4,2}$	f_2	$\vec{e}_{1,2}$	$\vec{e}_{3,2}$
$\vec{e}_{4,2}$	v_2	$\vec{e}_{4,1}$	f_1	$\vec{e}_{2,1}$	$\vec{e}_{1,1}$

[Berg]

DCGI

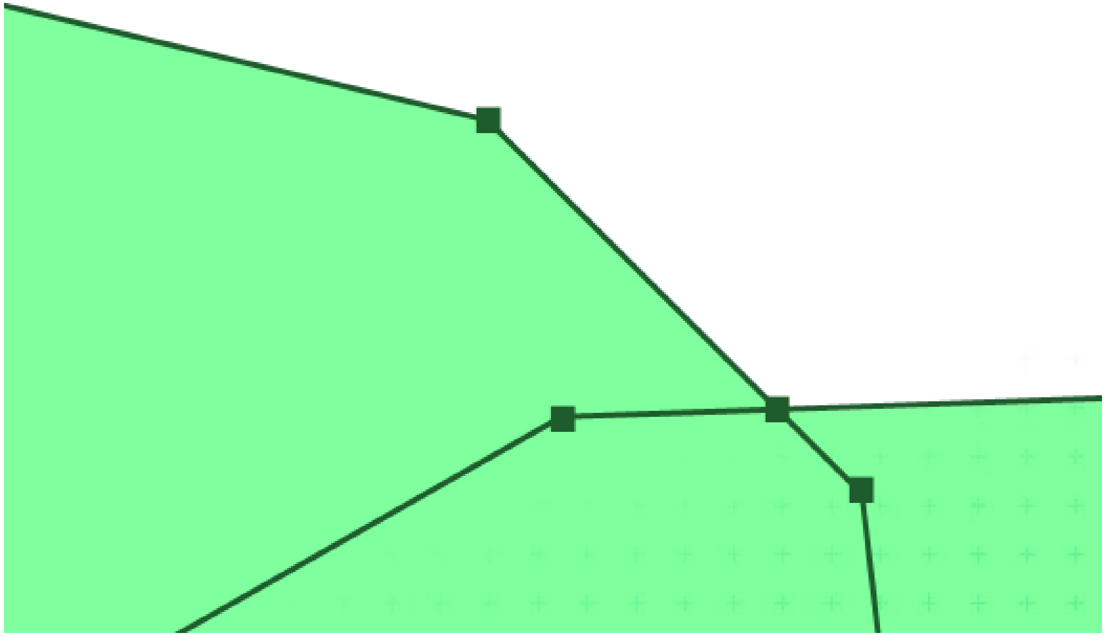
Computing intersections of line segments

- Have to handle intercestion of edges
- For now - Line segments



Computing intersections of line segments

- Have to handle intercestion of edges
- For now - Line segments



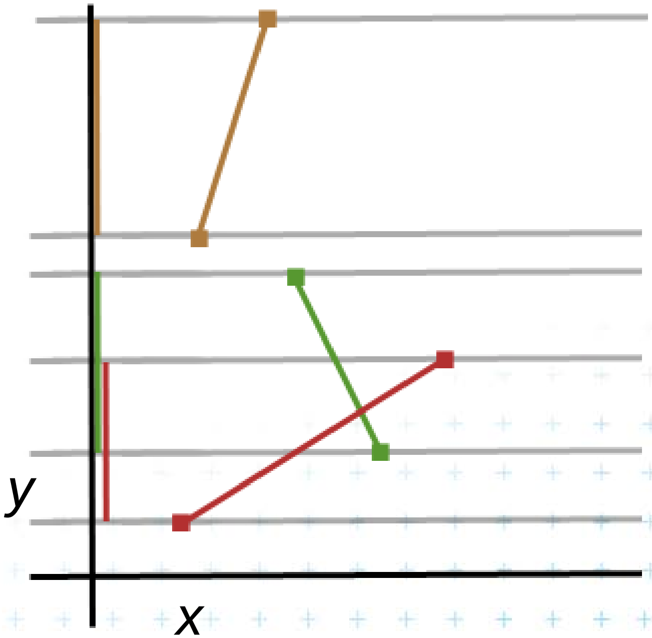
Computing intersections of line segments

- Intersection of two lines – just take equation of the line and make it equal to the second line
- Segments are not infinite
- Indicate if segments has intersection
- Brute force $O(n^2)$
- When each segment has intersection - $\Omega(n^2)$

DCGI

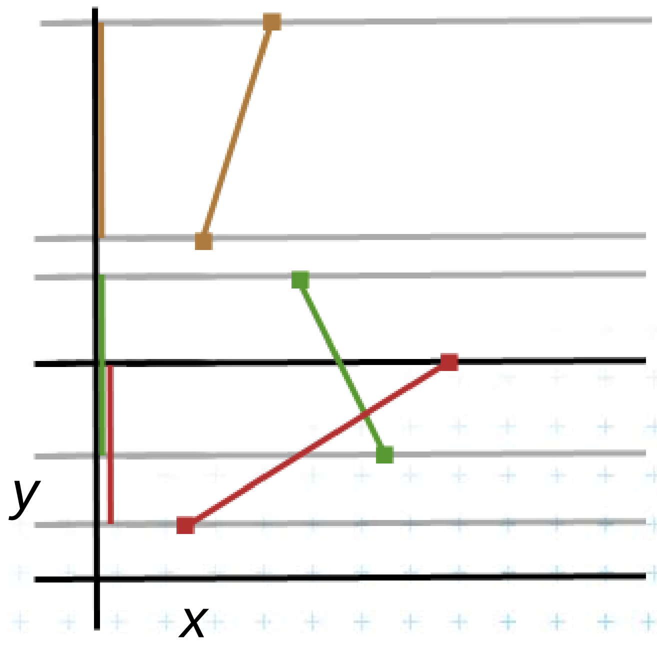
Plane sweep algorithm (to find intersections)

- Output sensitive algorithm
- Basic thoughts
  - Projection on y-axis



Plane sweep algorithm (to find intersections)

- Output sensitive algorithm
- Basic thoughts
  - Projection on y-axis
  - Divide by sweep line by x-axis
  - Test only horizontal neighbors



Plane sweep algorithm (to find intersections)

- Event queue Q
  - Event – endpoints of segments (two for each) or possible intersections
  - Balanced binary search tree implemetation
    - dynamic structure with removing and adding events on the fly
  - **Ordering:** p is first if $p.y > q.y$ holds or $py = qy$ and $p.x < q.x$
  - If event is the endpoint, where its segment starts, this **segment** is **stored** as well

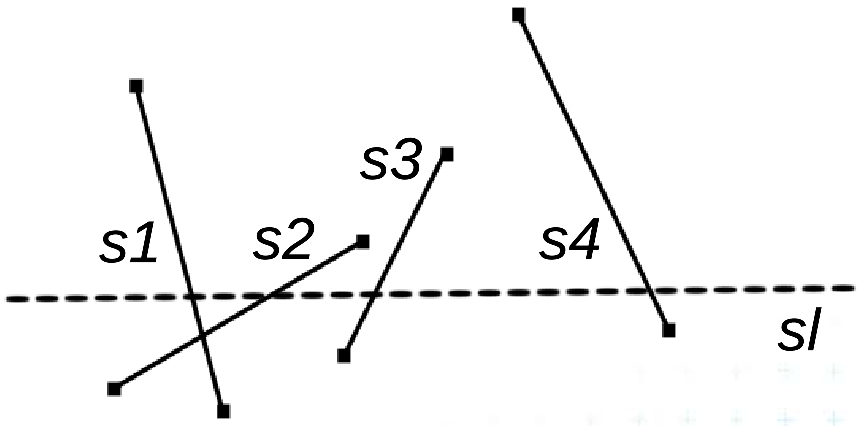
DCGI

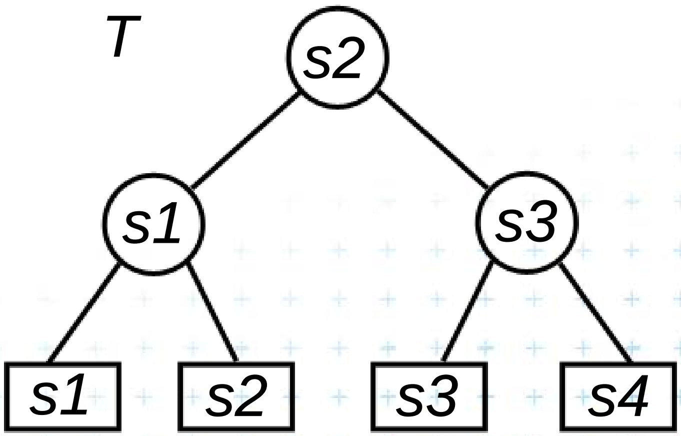
Plane sweep algorithm (to find intersections)

- Ordered sequence of segments *T*
  - To create order of segments defined by intersecting sweep line = dynamic structure of neighbors
  - Balanced binary search tree
  - Left-to-right order
  - Leaves store segments itself
  - Internal node store the segment from the **rightmost leaf** in its **left subtree** to guide search

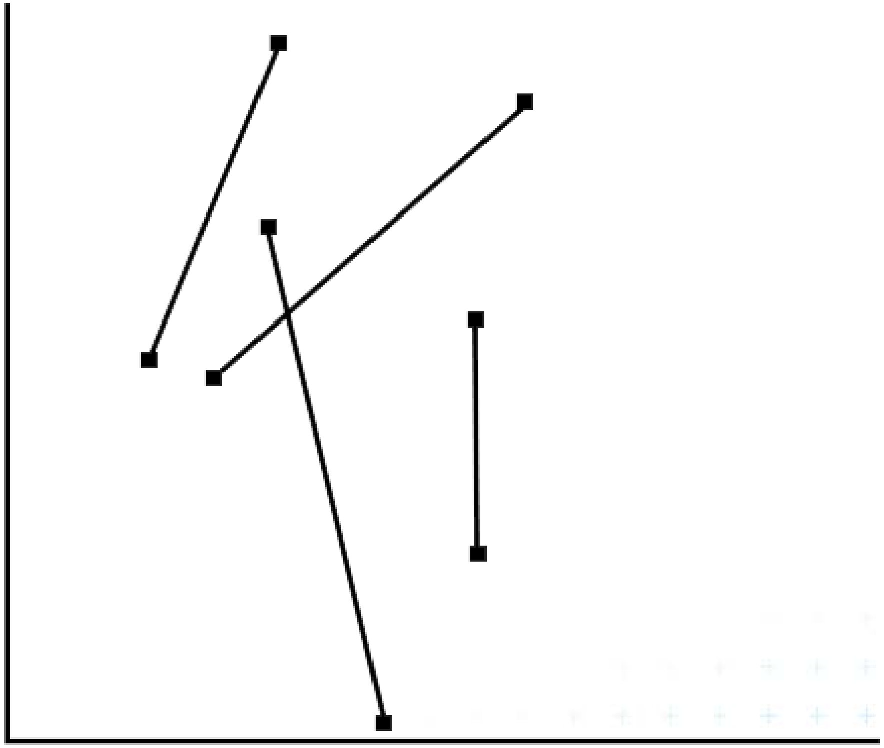
Plane sweep algorithm (to find intersections)

- Ordered sequence of segments *T*
  - Testing in each internal node position of the searched point
  - Result is the leaf or immediately to the left of it

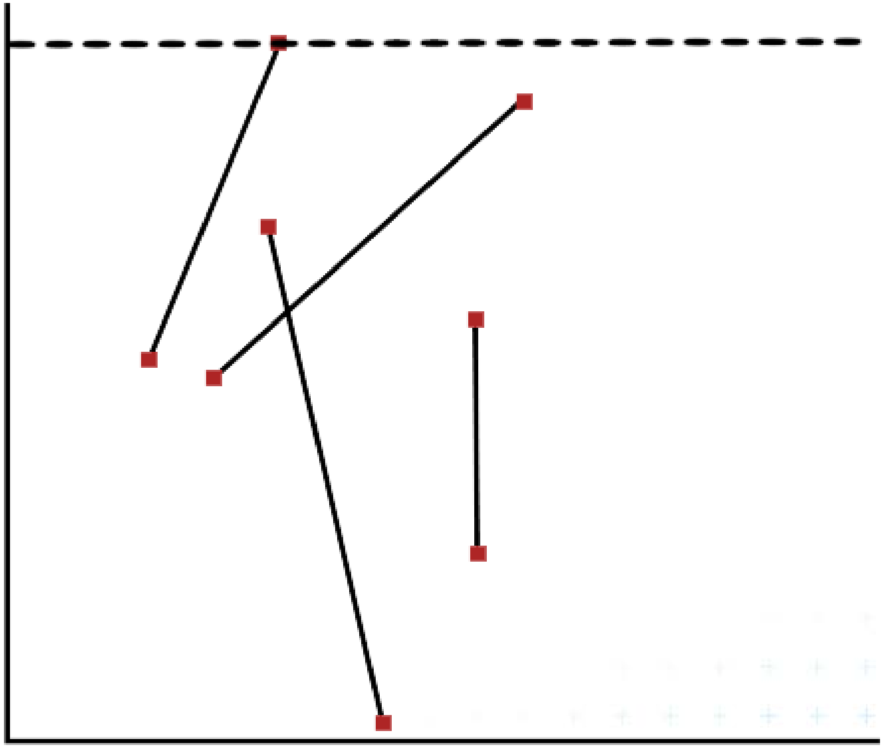




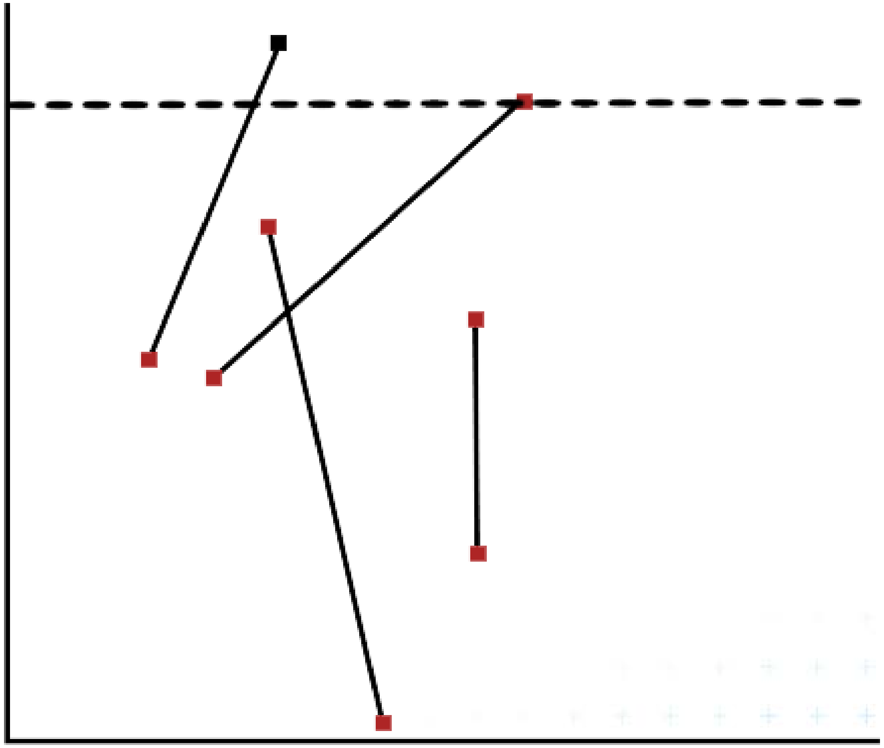
Plane sweep algorithm (to find intersections)



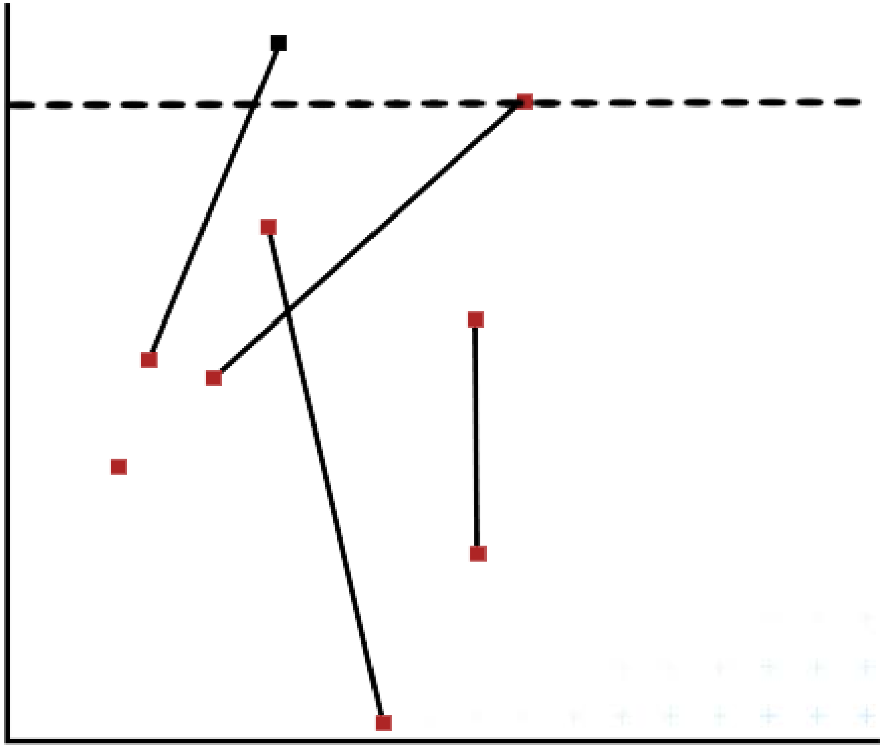
Plane sweep algorithm (to find intersections)



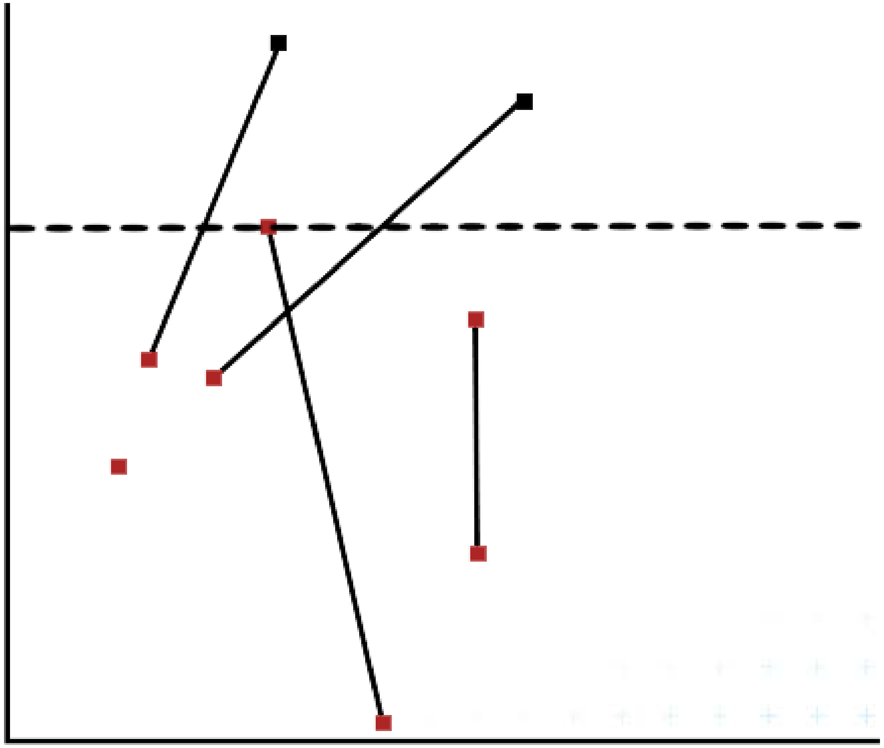
Plane sweep algorithm (to find intersections)



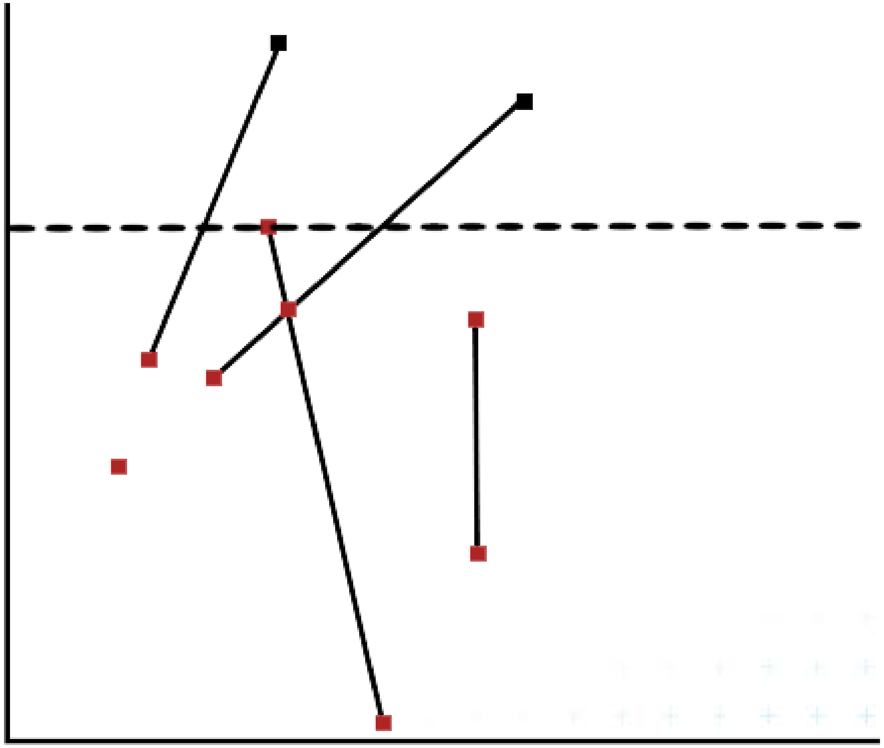
Plane sweep algorithm (to find intersections)



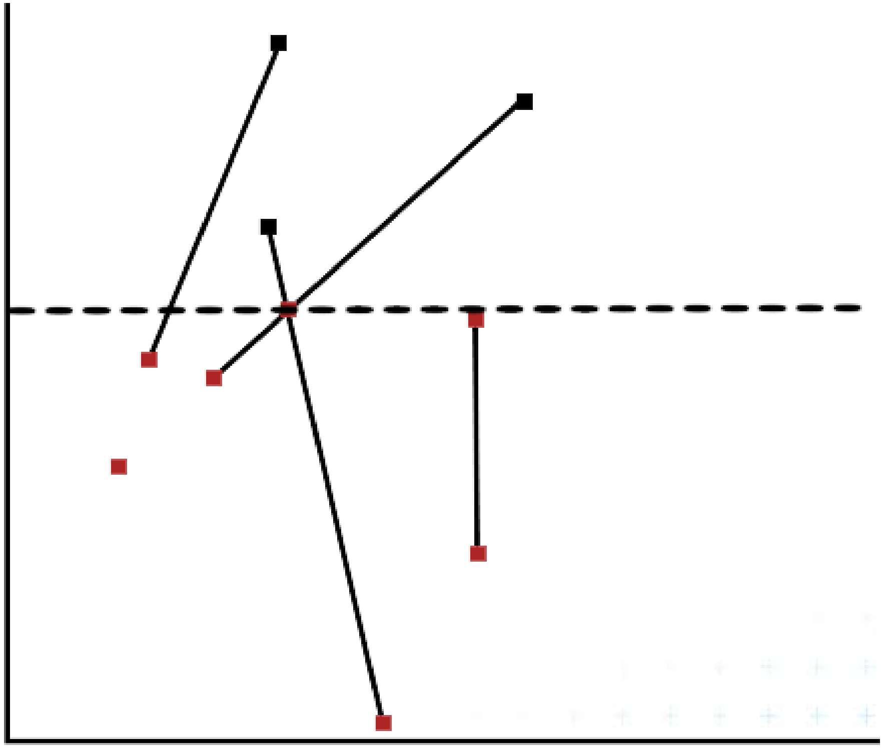
Plane sweep algorithm (to find intersections)



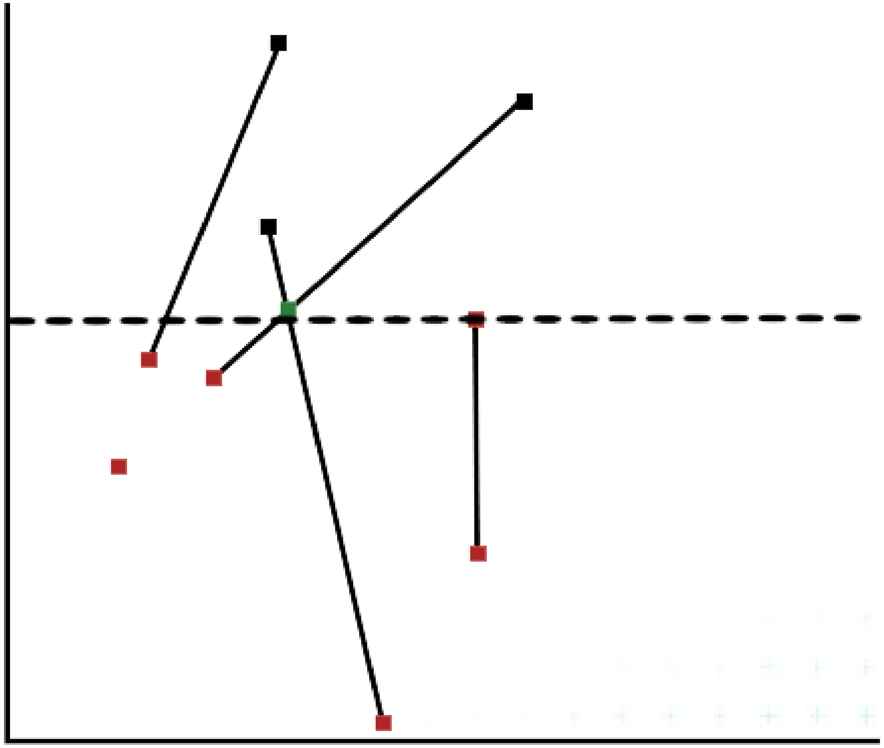
Plane sweep algorithm (to find intersections)



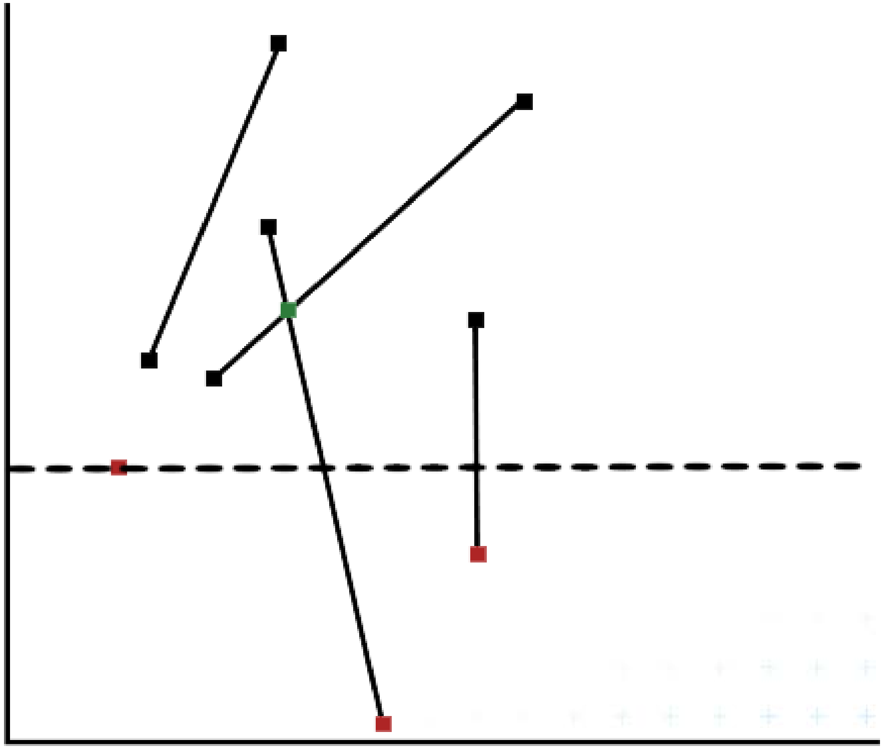
Plane sweep algorithm (to find intersections)



Plane sweep algorithm (to find intersections)



Plane sweep algorithm (to find intersections)



Plane sweep algorithm (to find intersections)

Create event queue Q with endpoints of all segments

(when an upper endpoint is inserted, the corresponding segment should be stored with it)

Initialize an empty status structure T.

while Q is not empty

Determine the next event point **p** in Q and delete it.

Let U(**p**) be the set of segments whose upper endpoint is **p**

Find all segments stored in **T** that contain **p**; they are adjacent in **T**

Let L(**p**) denote the subset of segments found whose lower endpoint is **p**

Let C(**p**) denote the subset of segments found that contain **p** in their interior

if L(**p**) ∪ U(**p**) ∪ C(**p**) contains more than one segment

then Report **p** as an intersection, together with L(**p**), U(**p**), and C(**p**)

Delete the segments in L(p) ∪ C(p) from T

Insert the segments in U(p) ∪ C(p) into T (below sweep, =reversing order of C(p))

if U(p) ∪ C(p) = empty

then Let sl and sr be the left and right neighbors of p in T

FindEvent(sl , sr, p) - (see below)

else Let s be the leftmost segment of U(p)∪C(p) in T

Let sl be the left neighbor of s in T

FindEvent (sl , s, p) – (see below)

Let s be the rightmost segment of U(p)∪C(p) in T

Let sr be the right neighbor of s in T

FindEvent (s, sr, p) – if intersect is below the sweep line and was not added yet- add event point to T

Plane sweep algorithm (to find intersections)

- Event queue *O(n log n)*
- Deletions, insertions and neighbor finding on Q take *O(log n)* time each
- The running time is O(n log n + I log n), where I is the number of intersections

Merging subdivisions

- Two subdivisions – *S1* and *S2*
- Looking for *O (S1, S2)*
- Using plane sweep algorithm
- Closed edges – like segments
- Preserve names

DCGI

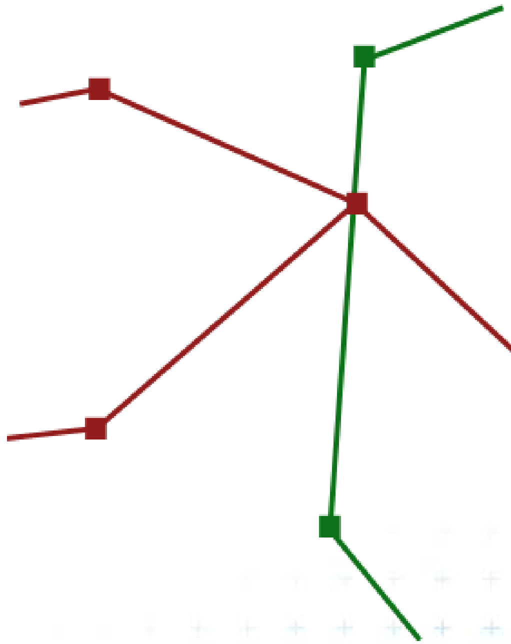
Merging subdivisions - edges

- Copy S1 and S2 into doubly-connected edge list D
  - Not valid, transform into O (S1, S2)
- T has edges; D has half-edges
- Intercesion points are counted when event involves edges of both subdivisions
- By dividing – the directions are preserved

Merging subdivisions - edges

- Two new edges; four new half-edges but

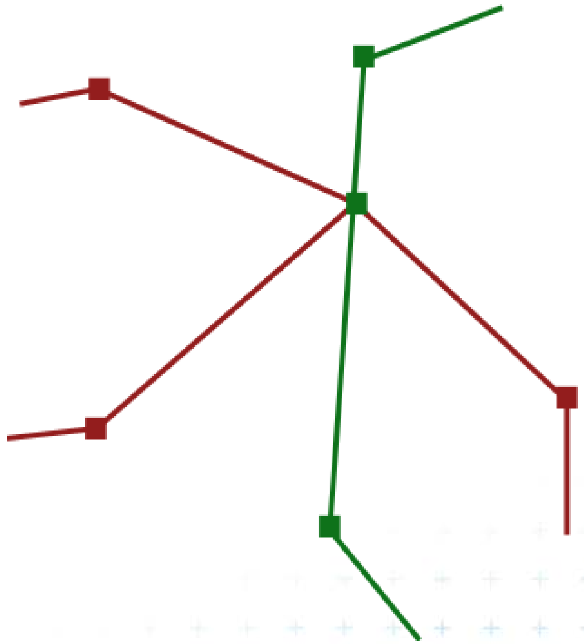
  two new records
- Set *Twin(e)*, *Next(e)*, *Last(e)*
- Linear time depending on degree of spliting point



Merging subdivisions - edges

- Two new edges; four new half-edges but

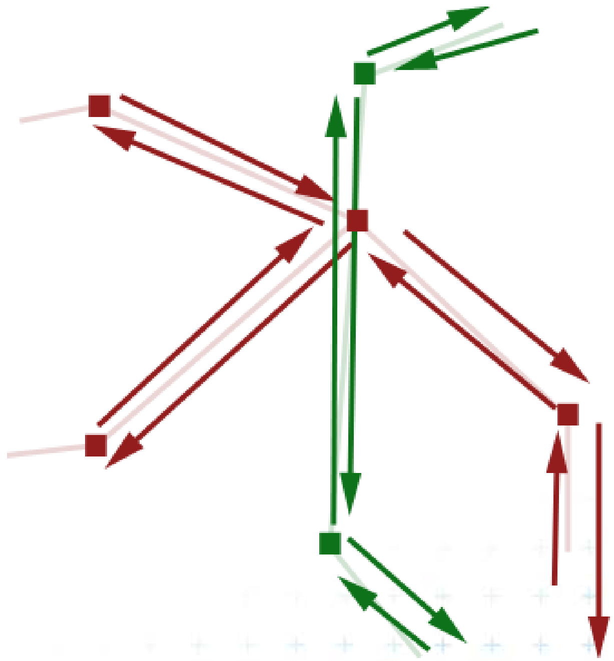
  two new records
- Set *Twin(e)*, *Next(e)*, *Last(e)*
- Linear time depending on degree of spliting point



Merging subdivisions - edges

- Two new edges; four new half-edges but

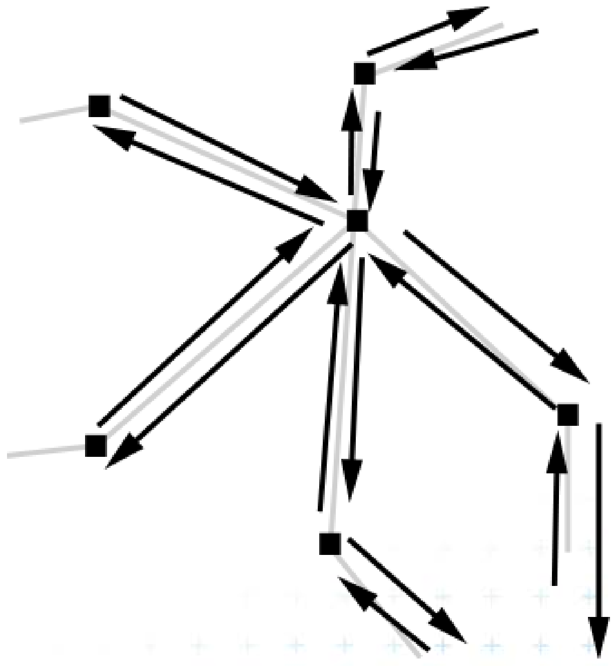
  two new records
- Set *Twin(e)*, *Next(e)*, *Last(e)*
- Linear time depending on degree of spliting point



Merging subdivisions - edges

- Two new edges; four new half-edges but

  two new records
- Set *Twin(e)*, *Next(e)*, *Last(e)*
- Linear time depending on degree of spliting point

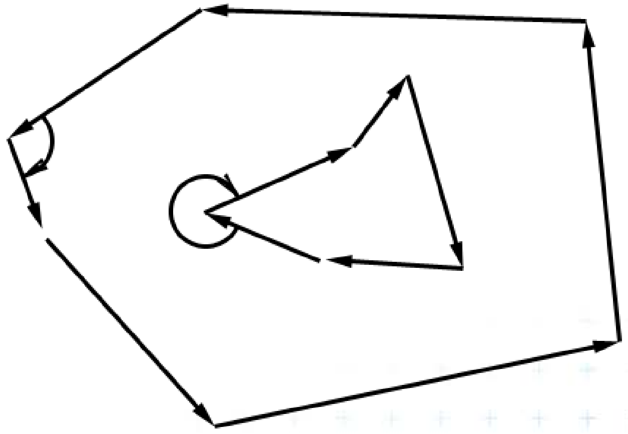


Merging subdivisions - faces

- New face records
- *OuterComponent(f),* set of *InnerComponents(f)* for new faces
- *IncidentFace()* for half-edges in their boundaries
- Label with the names of the faces in the old subdivisions that contain it.

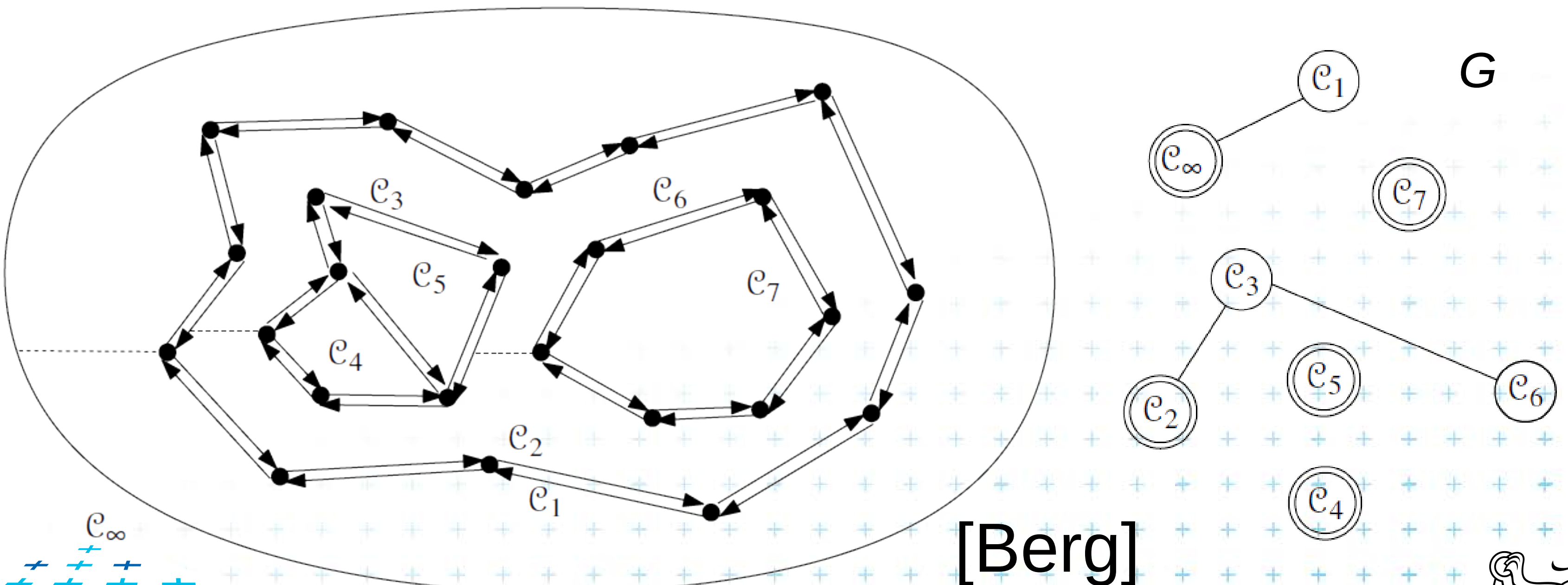
Merging subdivisions - faces

- Need to know holes of faces
- Cycle is hole, when on its leftmost vertex lies angle bigger than 180° (or the lowest when there are more of them)



Merging subdivisions - faces

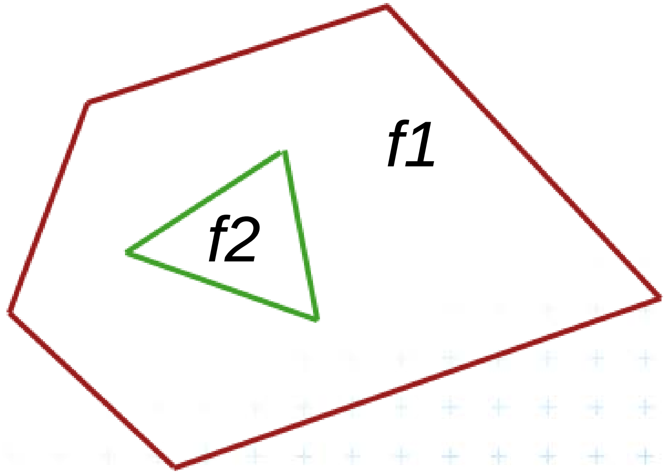
- Graph *G*
  - Node is the boundary cycle
  - Connection if one of the cycles is the boundary of a hole and the other cycle has a half-edge immediately to the left of the leftmost vertex of that hole cycle



[Berg]

Merging subdivisions - faces

- Labeling faces
  - Need to know in which face from both merged subdivisions new face lies
  - Sweep line algorithm again

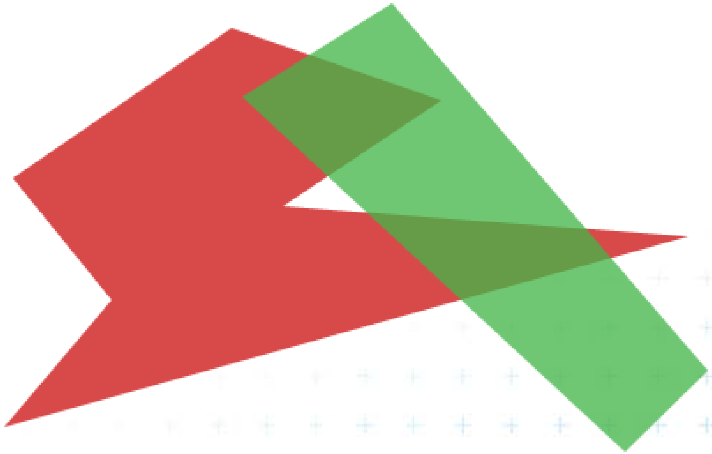


Merging subdivisions

- Copying list $O\ (n)$
- The plane sweep takes $O(n \log n + k \log n)$
- Fill in the face records takes time linear in the complexity of $O(S1,S2)$
- Labeling takes $O(n \log n + k \log n)$
- Construction in $O(n \log n + k \log n)$, where k is the complexity of the overlay

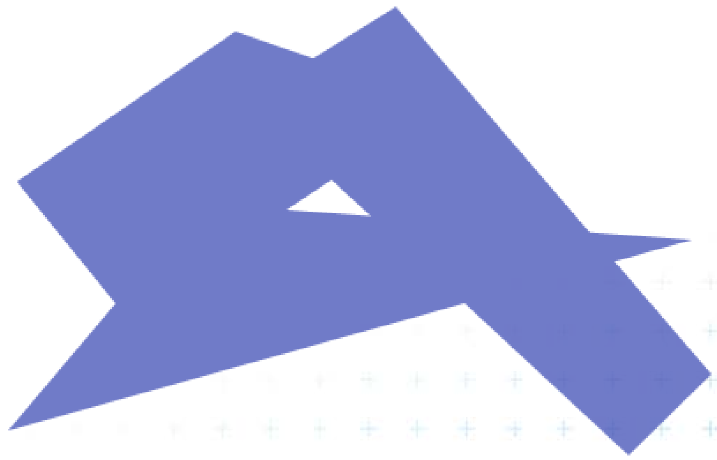
Boolean operations

- Labels of faces
  - Which face did belonge which subdivision of the operation
- Operations: union, difference, intersection



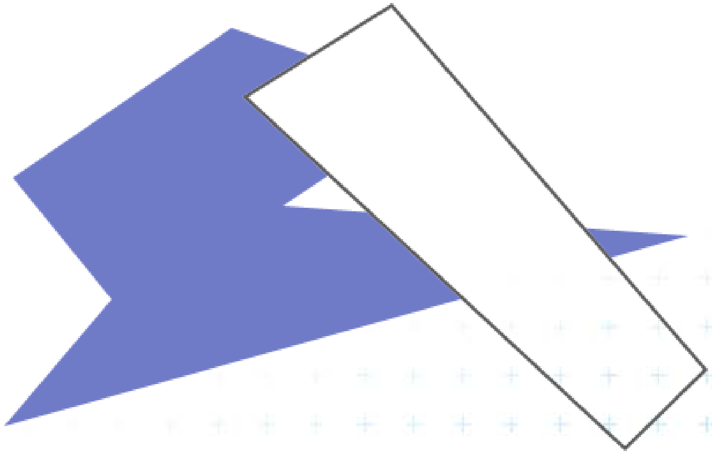
Boolean operations

- Labels of faces
  - Which face did belonge which subdivision of the operation
- Operations: **union**, difference, intersection



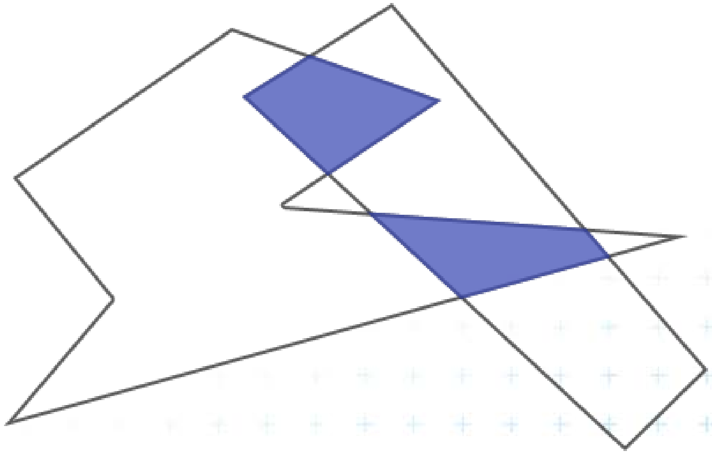
Boolean operations

- Labels of faces
  - Which face did belonge which subdivision of the operation
- Operations: union, **difference**, intersection



Boolean operations

- Labels of faces
  – Which face did belonge which subdivision of the operation
- Operations: union, difference, **intersection**



Thank for your attention

Radek Smetana, 7. 11. 2012

DCGI

Sources

- **[Berg] Mark de Berg**, Otfried Cheong, Marc van Kreveld, Mark Overmars: Computational Geometry: Algorithms and Applications, Springer- Verlag, 3rd rev. ed. 2008. 386 pages, 370 fig. ISBN: 978-3-540-77973-5, Chapter 2
- **Jiří Žára**, slide template

DCGI

OI-OPPA. European Social Fund
Prague & EU: We invest in your future.