Lecture 2b: Subdivision representation and map overlay

Computational Geometry

Utrecht University

Motivation

Motivation

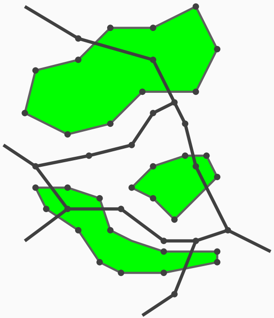
Map overlay

Map overlay

Map overlay is the combination of two (or more) map layers

It is needed to answer questions like:

- What is the total length of roads through forests?
- What is the total area of corn fields within 1 km from a river?
- What area of all lakes occurs at the geological soil type “rock”?

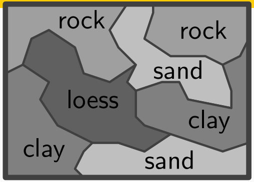


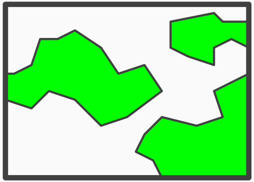
Map overlay

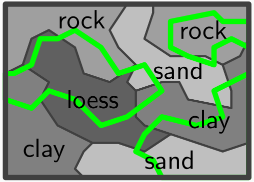
To solve map overlay questions, we need (at the least) intersection points from two sets of line segments (possibly, boundaries of regions)

Map overlay

To solve map overlay questions, we also need to be able to represent subdivisions







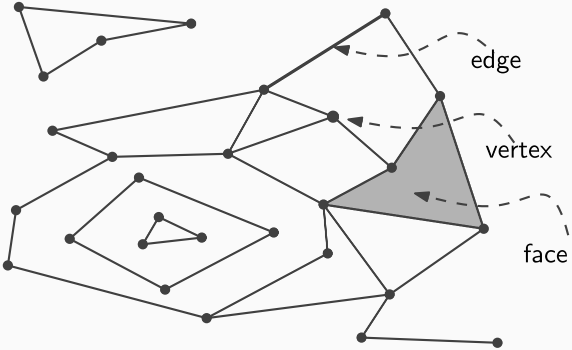
Doubly-connected edge list

Doubly-connected edge list

Subdivisions

Subdivisions

A planar subdivision is a structure induced by a set of line segments in the plane that can only intersect at common endpoints. It consists of vertices, edges, and faces



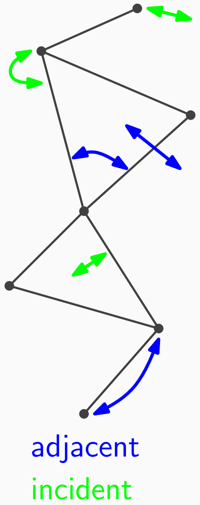
Subdivisions

Vertices are the endpoints of the line segments

Edges are the interiors of the line segments

Faces are the interiors of connected two-dimensional regions that do not contain any point of any line segment

Objects of *the same* dimensionality are adjacent or not; objects of *different* dimensionality are incident or not

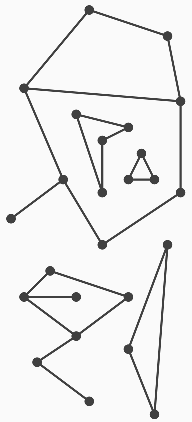


Subdivisions

Exactly one face is unbounded, the outer face

Every other face is bounded and has an outer boundary consisting of vertices and edges

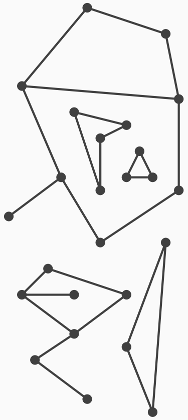
Any face has zero or more inner boundaries



Subdivisions

Vertices, edges, and faces form a partition of the plane

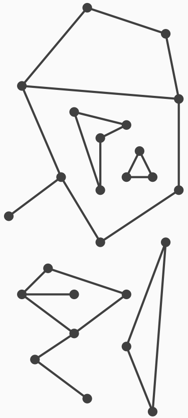
If a planar subdivision is induced by n line segments, it has exactly n edges, and at most $2n$ vertices



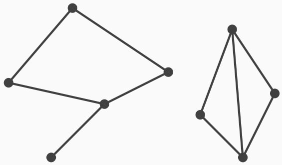
Subdivisions

And how many faces?

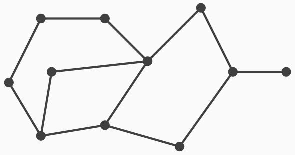
Observe: Every face is bounded by at least 3 edges, and every edge bounds at most 2 faces $\Rightarrow$
$F \leq 2E/3 = 2n/3 = O(n)$



Euler's formula for planar graphs: If S is a planar subdivision with V vertices, E edges, and F faces, then $V - E + F \geq 2$, with equality iff the vertices and edges of S form a connected set



$$V = 9,\ E = 10,\ F = 4$$
$$V - E + F = 3$$



$$V = 11,\ E = 13,\ F = 4$$
$$V - E + F = 2$$

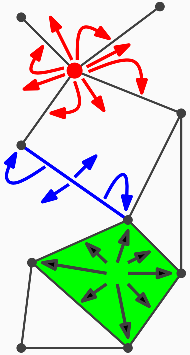
Doubly-connected edge list

Representing subdivisions

Representing subdivisions

A subdivision representation has a vertex-object class, an edge-object class, and a face-object class

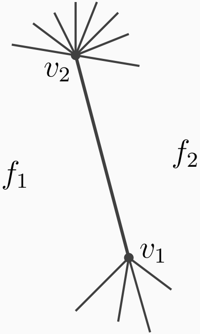
It is a pointer structure where objects can reach incident (or adjacent) objects easily



Representing subdivisions

Use the **edge** as the central object

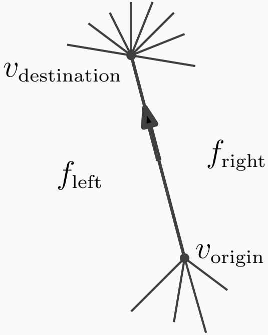
For any edge, exactly two vertices are incident, exactly two faces are incident, and zero or more other edges are adjacent



Representing subdivisions

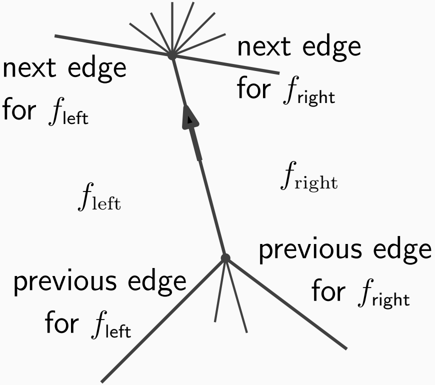
Use the **edge** as the central object, and give it a direction

Now we can speak of Origin, Destination, Left Face, and Right Face



Representing subdivisions

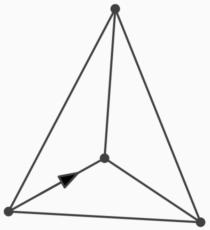
Four edges are of special interest

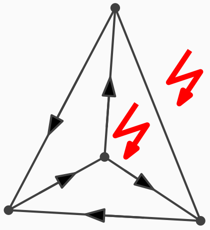


Representing subdivisions

It would be nice if we could traverse a boundary cycle by continuously following the next edge for f_{left} or f_{right}

. . . but, no consistent edge orientation needs to exist



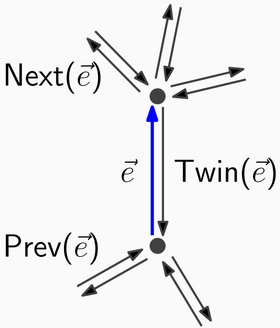


Representing subdivisions

We apply a trick/hack/impossibility: split every edge *length-wise(!)* into two half-edges

Every half-edge:

- has exactly one half-edge as its Twin
- is directed opposite to its Twin
- is incident to only one face (left)



Doubly-connected edge list

DCEL structure

The doubly-connected edge list

The doubly-connected edge list is a subdivision representation structure with an object for every vertex, every half-edge, and every face

A vertex object stores:

- Coordinates
- **IncidentEdge** (some half-edge leaving it)

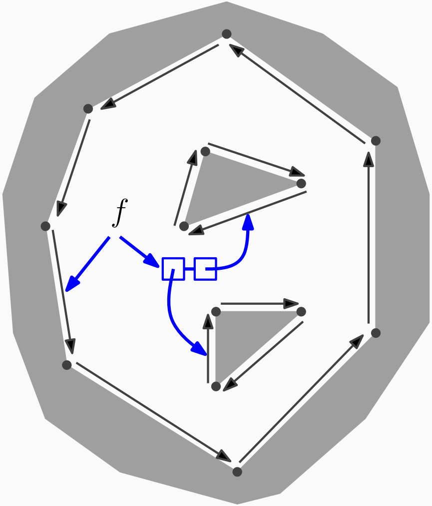
A half-edge object stores:

- **Origin** (vertex)
- **Twin** (half-edge)
- **IncidentFace** (face)
- **Next** (half-edge in cycle of the incident face)
- **Prev** (half-edge in cycle of the incident face)

The doubly-connected edge list

A face object stores:

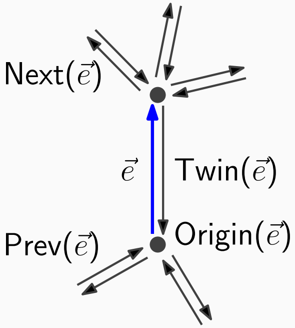
- **OuterComponent** (half-edge of outer cycle)
- **InnerComponents** (list of half-edges for the inner cycles bounding the face)

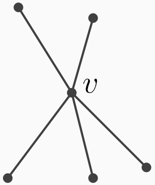


The doubly-connected edge list

Question: A half-edge $\vec{e}$ can directly access its **Origin**, and get the coordinates of one endpoint. How can it get the coordinates of its other endpoint?

Question: For a vertex v, how do we find all adjacent vertices?





The doubly-connected edge list

A vertex object stores:

- Coordinates
- **IncidentEdge**
- *Any attributes, mark bits*

A face object stores:

- **OuterComponent** (half-edge of outer cycle)
- **InnerComponents** (half-edges for the inner cycles)
- *Any attributes, mark bits*

A half-edge object stores:

- **Origin** (vertex)
- **Twin** (half-edge)
- **IncidentFace** (face)
- **Next** (half-edge in cycle of the incident face)
- **Prev** (half-edge in cycle of the incident face)
- *Any attributes, mark bits*

The doubly-connected edge list

Question: For a face f, how do we find all adjacent face names, assuming they are stored in an attribute? Write the code using the proper names like **Next**, **OuterComponent**, etc.

Map overlay

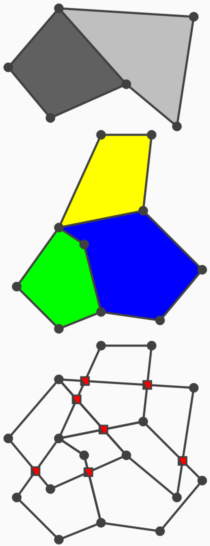
Map overlay

Initialization

Map overlay problem

The map overlay problem for two subdivisions S_1 and S_2 is to compute a subdivision S that is the overlay of S_1 and S_2

All edges of S are (parts of) edges from S_1 and S_2. All vertices of S are also in S_1 or S_2, or intersections of edges from S_1 and S_2

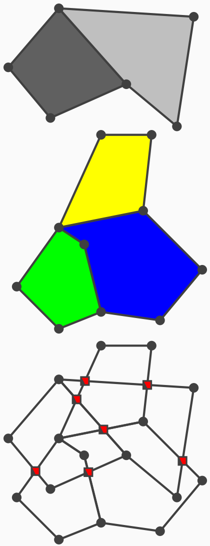


Map overlay

We start by making a copy of S_1 and of S_2 whose vertex and half-edge objects we will re-use

At first we do not compute face object information in the overlay

The output should be a doubly-connected edge list (DCEL) of the overlay

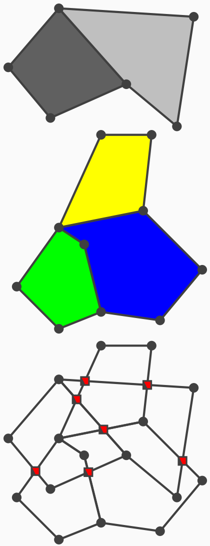


Map overlay

Approach: plane sweep

Need to define status, events, event handling

Need status structure, event list, and DCEL for the output



Map overlay

Overlay algorithm

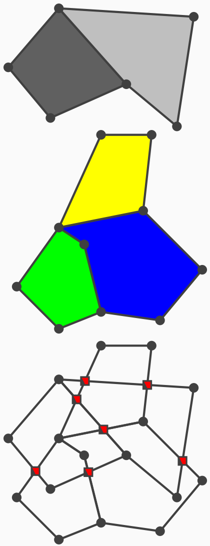
Map overlay

Status: the edges of S_1 and S_2 intersecting the sweep line in the left-to-right order

Events happen:

- At the vertices of S_1 and S_2
- At intersection points from S_1 and S_2

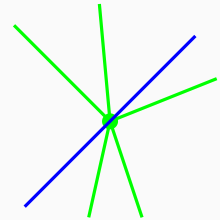
The event list is basically the same as for line segment intersection

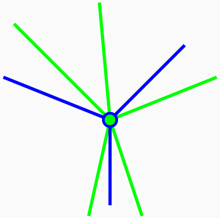


Overlay events

Six types of events:

- A vertex of S_1
- A vertex of S_2
- An intersection point of one edge from S_1 and one edge from S_2
- An edge of S_1 goes through a vertex of S_2
- An edge of S_2 goes through a vertex of S_1
- A vertex of S_1 and a vertex of S_2 coincide



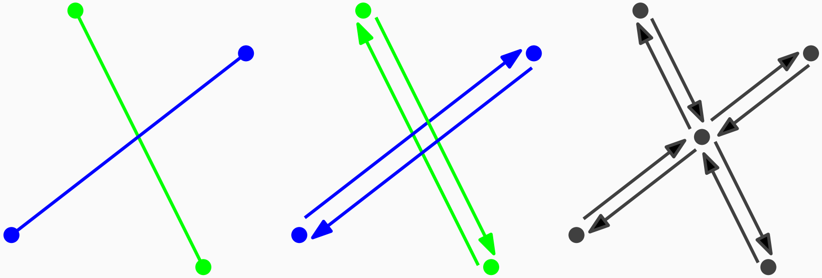


Overlay events

Consider the event:

an intersection point of one edge from S_1 and one edge from S_2

geometry | DCELs before | DCEL after

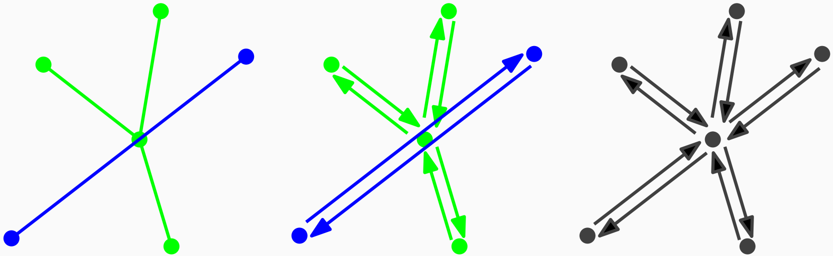


Overlay events

Consider the event:

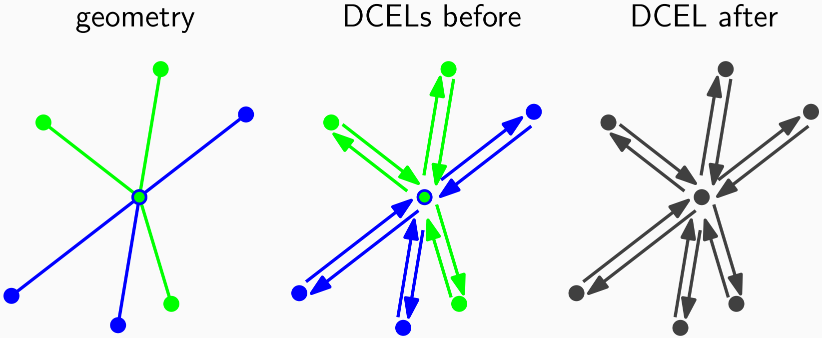
an edge from S_1 goes through a vertex of S_2

geometry DCELs before DCEL after



Consider the event:

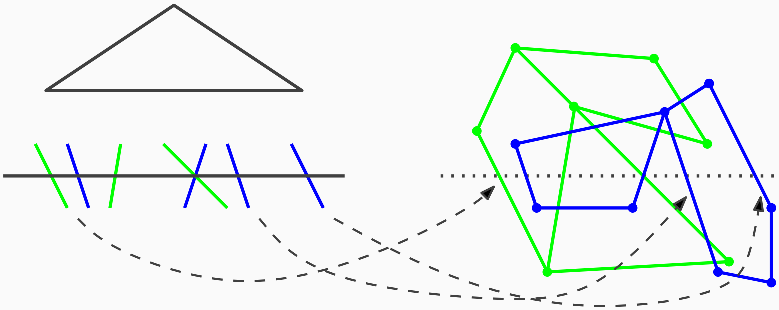
a vertex of S_1 and a vertex of S_2 coincide



Overlay events

When we take an event from the event queue Q, we need quick access to the DCEL to make the necessary changes

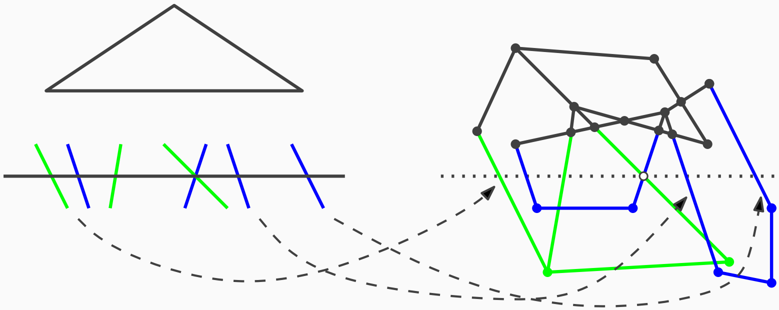
We keep a pointer from each leaf in the status structure to one of the representing half-edges in the DCEL



Overlay events

When we take an event from the event queue Q, we need quick access to the DCEL to make the necessary changes

We keep a pointer from each leaf in the status structure to one of the representing half-edges in the DCEL

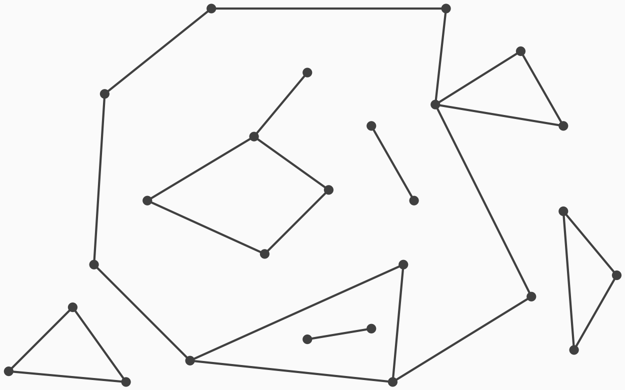


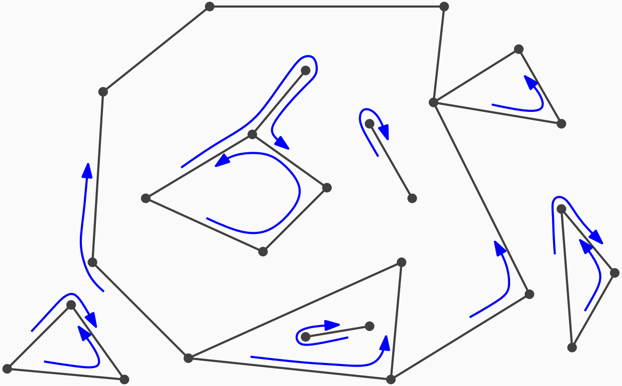
Overlay so far

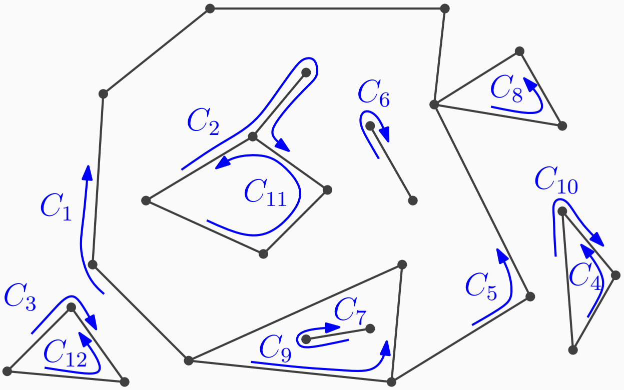
The sweep algorithm gives us all vertices and half-edges of the overlay, and pointers between these objects

Next we need face objects and their connection into the doubly-connected edge list structure

Question: Which variables of vertex, edge, and face objects do we still have to set?

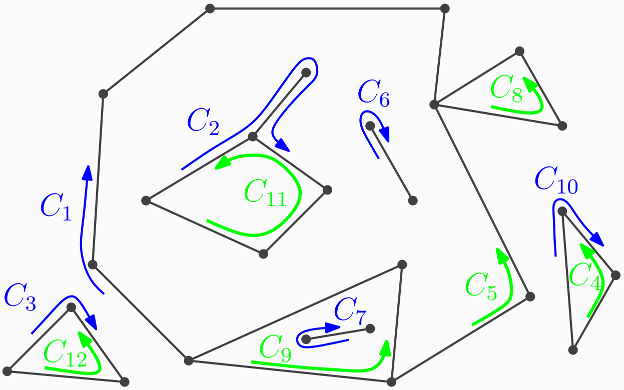


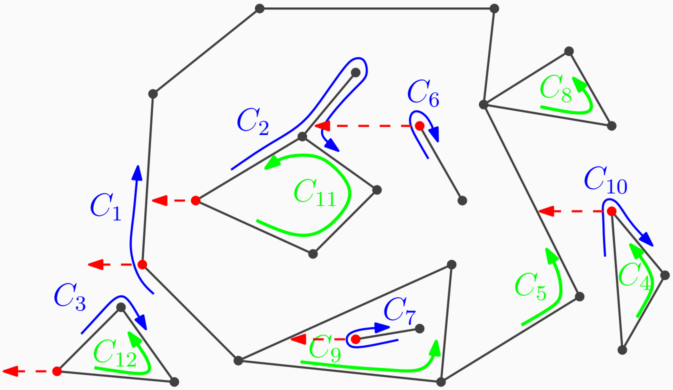



C_1
C_2
C_3
C_4
C_5
C_6
C_7
C_8
C_9
C_{10}
C_{11}
C_{12}

Face information

- Determine all cycles of half-edges, and whether they are inner or outer boundaries of the incident face
- Make a face object for each outer boundary, plus one for the unbounded face, and set the **OuterComponent** variable of each face. Set the **IncidentFace** variable for every half-edge in an outer boundary cycle


C_1
C_2
C_3
C_4
C_5
C_6
C_7
C_8
C_9
C_{10}
C_{11}
C_{12}



$C_1 \rightarrow$

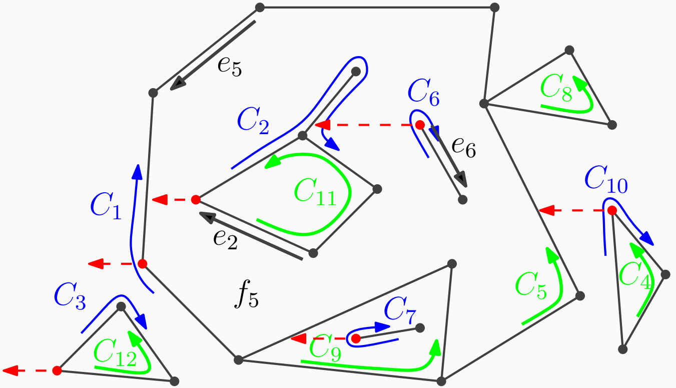
$C_3 \rightarrow$

$C_2 \rightarrow C_5$

$C_7 \rightarrow C_9$

$C_6 \rightarrow C_2$

$C_{10} \rightarrow C_1$



$C_1 \to$
$C_3 \to$
$C_2 \to C_5$
$C_7 \to C_9$
$C_6 \to C_2$
$C_{10} \to C_1$

f_5: outer e_5; inner e_2, e_6

Face information

- Determine the leftmost vertex of each inner boundary cycle
- For all of these leftmost vertices, determine the edge horizontally left of it, take the downward half-edge of it, and its cycle (by plane sweep) to set **InnerComponents** for all faces and **IncidentFace** for half-edges in inner boundary cycles

Efficiency

Every event takes $O(\log n)$ or $O(m + \log n)$ time to handle, where m is the sum of the degrees of any vertex from S_1 and/or S_2 involved

The sum of the degrees of all vertices is exactly twice the number of edges in the output

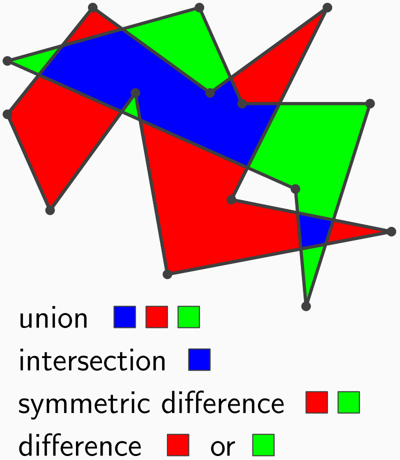
Theorem: Given two planar subdivisions S_1 and S_2, their overlay can be computed in $O(n \log n + k \log n)$ time, where k is the number of vertices of the overlay

Map overlay

Boolean operations, placement space

Boolean operations on polygons

Boolean operations on two polygons with n vertices take $O(n \log n + k \log n)$ time, where k is the number of intersection points

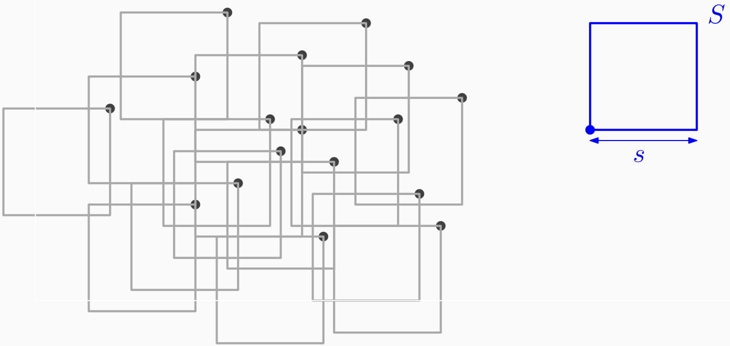


Placement space of a square

Given a set of n points in the plane, and a side length s, compute an axis-parallel placement of a square S with side length s such that it contains the maximum number of points.

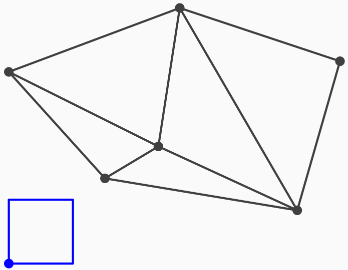
Placement space of a square

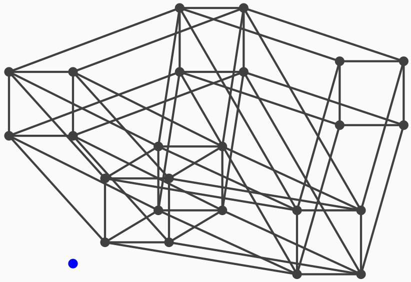
Given a set of n points in the plane, and a side length s, compute an axis-parallel placement of a square S with side length s such that it contains the maximum number of points.



Placement space of a square

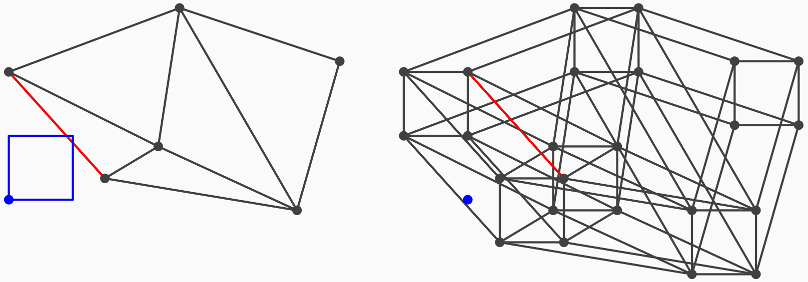
The parts of a subdivision intersecting a fixed-size square when that square can be placed anywhere is the overlay of four copies of the subdivision and n copies of the square





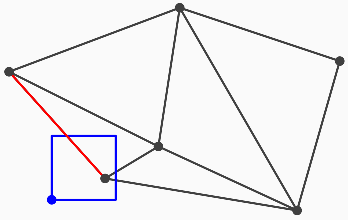
Placement space of a square

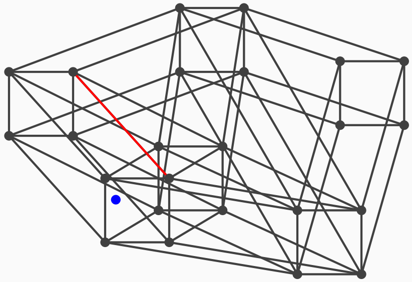
The parts of a subdivision intersecting a fixed-size square when that square can be placed anywhere is the overlay of four copies of the subdivision and n copies of the square



Placement space of a square

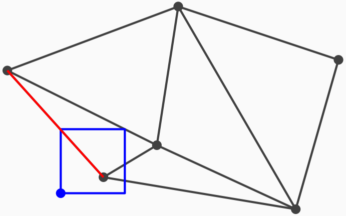
The parts of a subdivision intersecting a fixed-size square when that square can be placed anywhere is the overlay of four copies of the subdivision and n copies of the square

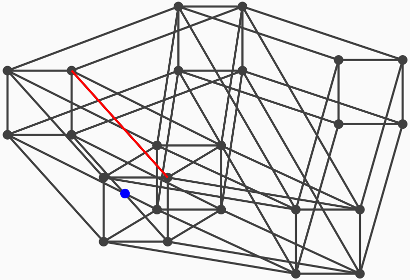


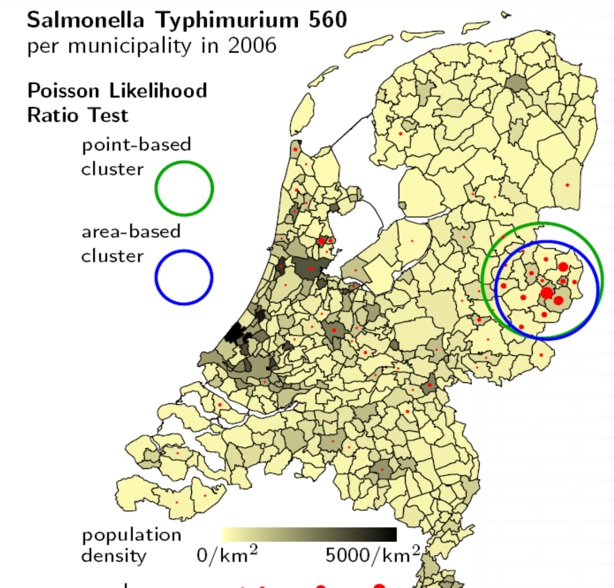


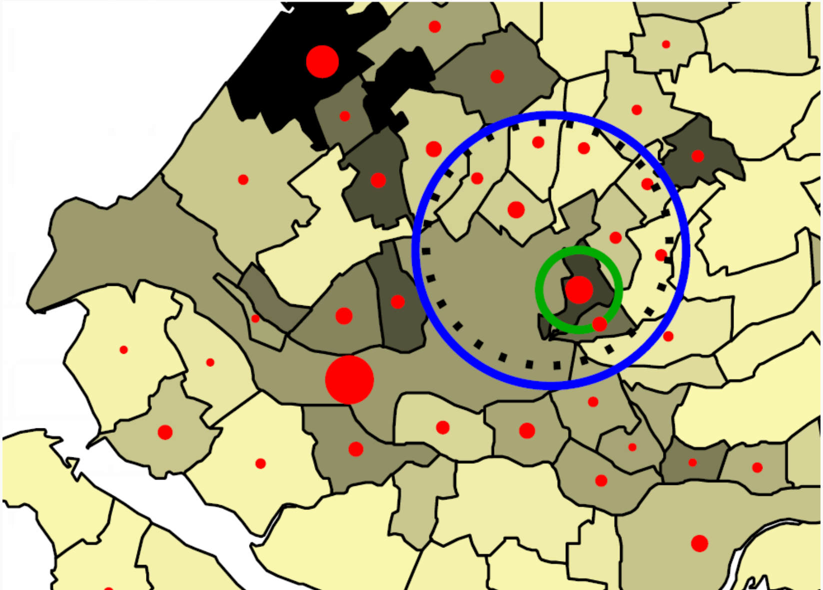
Placement space of a square

The parts of a subdivision intersecting a fixed-size square when that square can be placed anywhere is the overlay of four copies of the subdivision and n copies of the square






Salmonella Typhimurium 560
per municipality in 2006
Poisson Likelihood
Ratio Test
point-based
cluster
area-based
cluster
population
density
0/km2
5000/km2



Summary

Computing the overlay of two subdivisions, or the placement space of a shape, is a basic operation needed in GIS

To represent a planar subdivision, a doubly-connected edge list is a convenient data structure

To design efficient geometric algorithms, the plane sweep technique is often a good choice