Line Segment Intersection

1. Output Sensitive Algorithms
   - map overlay and input specification
   - using the `cgal-swig-bindings`
2. A Plane Sweep Algorithm
   - handling events caused by sweeping a line
   - adjacent line segments
   - classification of events
3. Data Structures
   - the event queue
   - the global algorithm
   - an example with CGAL in C++

MCS 481 Lecture 4
Computational Geometry
Jan Verschelde, 22 January 2025

Line Segment Intersection

1. Output Sensitive Algorithms
   - map overlay and input specification
   - using the `cgal-swig-bindings`
2. A Plane Sweep Algorithm
   - handling events caused by sweeping a line
   - adjacent line segments
   - classification of events
3. Data Structures
   - the event queue
   - the global algorithm
   - an example with CGAL in C++

map overlay

Geographic information systems store maps in layers,
each layer is a thematic map, stores only one type of information

- rainfall,
- altitude,
- vegitation,
- population density, etc...

In overlaying maps, we are interested in intersections.

In its simplest form we view a layer is a set of line segments
and to keep it simple, we focus on one set.

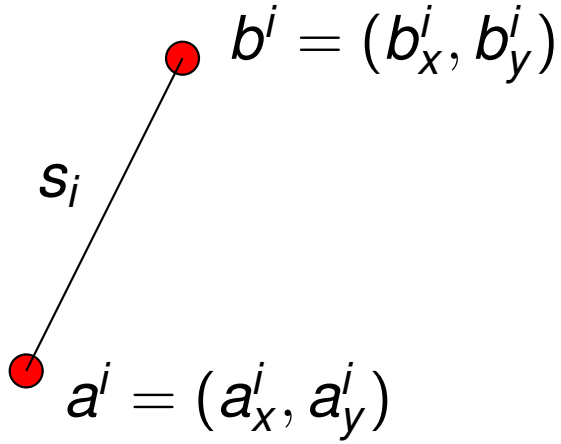
specification of input and output

Input: n line segments $S = \{s_1, s_2, \ldots, s_n\}$, $\#S = n < \infty$,

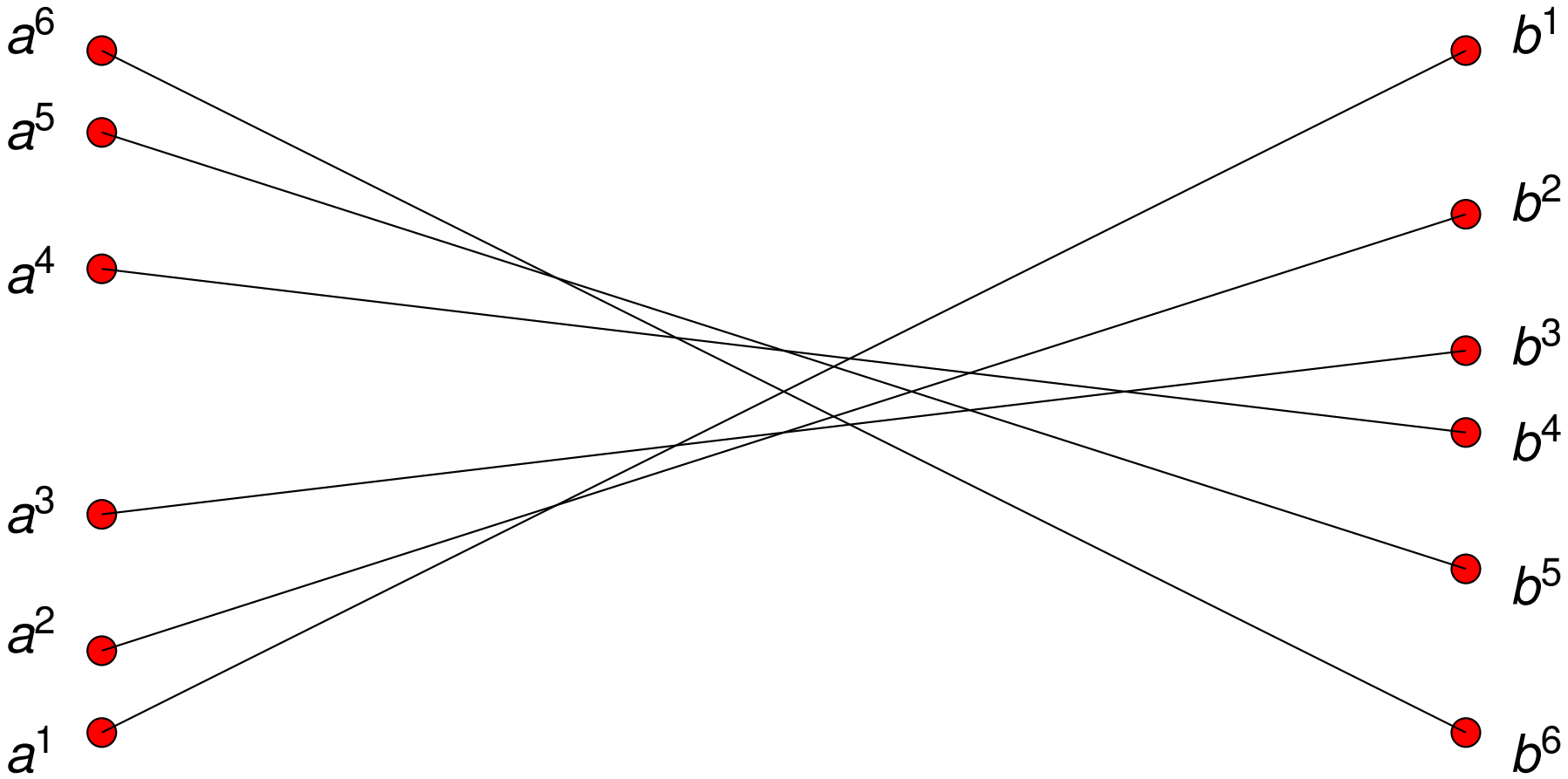
$$s_i = ((a_x^i, a_y^i), (b_x^i, b_y^i)), \; i = 1, 2, \ldots, n.$$

Output: intersection points $P = \{ ((i,j),(x,y)) \mid (x,y) \in s_i \cap s_j \}$.

Each line segment s_i is a tuple of end points:



an easy input



an easier input

output sensitive algorithms

Given n line segments,
the number of intersection points ranges between 0 and $n(n-1)/2$.

The direct algorithm intersects every line segment
with every other line segment.
It works well in case all line segments intersect each other.
This case is the worst case and the running time of any algorithm for this problem will therefore be $\Omega(n^2)$.

But, on average, we may expect to do better.

Definition (output sensitive algorithm)

An algorithm is *output sensitive* if its running time
is proportional to the size of its output.

big $O(\cdot)$, $\Omega(\cdot)$, $\Theta(\cdot)$, asymptotic bounds

Let $T(n)$ be the worst case running time for input size n.

Definition (big O asymptotic upper bound)

For functions $T(n)$ and $f(n)$, $T(n)$ *is* $O(f(n))$ if there exists a constant $c > 0$, independent of n: $T(n) \leq c\,f(n)$, as $n \to \infty$, for all $n \geq n_0$, for a fixed constant value n_0.

Definition (big $\Omega(\cdot)$ asymptotic lower bound)

For functions $T(n)$ and $f(n)$, $T(n)$ *is* $\Omega(f(n))$ if there exists a constant $c > 0$, independent of n: $T(n) \geq cf(n)$, as $n \to \infty$, for all $n \geq n_0$, for a fixed constant value n_0.

Definition (big $\Theta(\cdot)$ asymptotic sharp bound)

$T(n)$ *is* $\Theta(f(n))$ if $T(n)$ *is* $\Omega(f(n))$ and $T(n)$ *is* $O(f(n))$.

Line Segment Intersection

1. Output Sensitive Algorithms
   - map overlay and input specification
   - using the `cgal-swig-bindings`
2. A Plane Sweep Algorithm
   - handling events caused by sweeping a line
   - adjacent line segments
   - classification of events
3. Data Structures
   - the event queue
   - the global algorithm
   - an example with CGAL in C++

line segments in CGAL

Consider three line segments defined by the tuples

$$((0, 0), (2, 2)), ((1, 0), (0, 2)), ((2, 0), (1, 1))$$

and compute their intersections.

Defining the input in Python, with `cgal-swig-bindings`:

```
from CGAL.CGAL_Kernel import Point_2
from CGAL.CGAL_Kernel import Segment_2
from CGAL.CGAL_Kernel import intersection

segments = []
segments.append(Segment_2(Point_2(0, 0), Point_2(2, 2)))
segments.append(Segment_2(Point_2(1, 0), Point_2(0, 2)))
segments.append(Segment_2(Point_2(2, 0), Point_2(1, 2)))
```

Consider also the posted Jupyter notebook.

computing all intersection points

```
for i in range(len(segments)):
    for j in range(i+1, len(segments)):
        intsisj = intersection(segments[i], segments[j])
        print('segments', i, 'and', j, end=' ')
        if intsisj.empty():
            print('do not intersect')
        elif not intsisj.is_Point_2():
            print('do not intersect in a point')
        else:
            iptsisj = intsisj.get_Point_2()
            print('intersect :', end=' ')
            print(iptsisj.x(), ',', iptsisj.y())
```

which prints (edited to fit the slide):

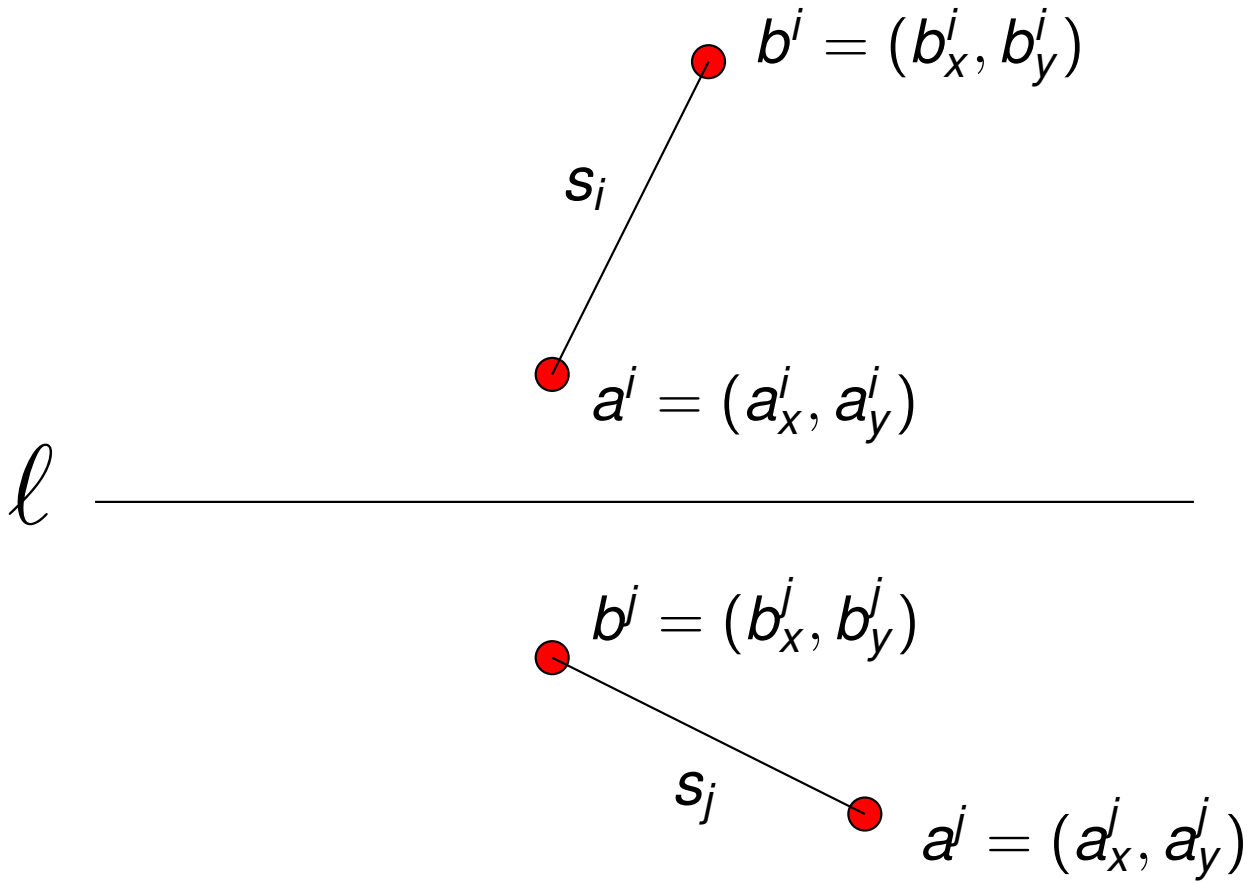
```
segments 0 and 1 intersect : 0.666..6 , 0.666..6
segments 0 and 2 intersect : 1.333..3 , 1.333..3
segments 1 and 2 do not intersect
```

Line Segment Intersection

1. Output Sensitive Algorithms
   - map overlay and input specification
   - using the `cgal-swig-bindings`
2. A Plane Sweep Algorithm
   - handling events caused by sweeping a line
   - adjacent line segments
   - classification of events
3. Data Structures
   - the event queue
   - the global algorithm
   - an example with CGAL in C++

ruling out intersections

Observe that $s_i = ((a_x^i, a_y^i), (b_x^i, b_y^i))$ and $s_j = ((a_x^j, a_y^j), (b_x^j, b_y^j))$ do not intersect if their y-coordinates do not overlap.



the idea for an algorithm

Preprocessing: sort the segments on their y-coordinate.
The first segment has an end point with the highest y-coordinate.

Imagine a line: ℓ

The line ℓ is *a sweep line*:

1. ℓ starts above all line segments,
2. ℓ slides gradually downwards,
3. ℓ encounters an end point of a segment: *an event*.

Two types of events:

1. meet highest end point: consider new segment, or
2. meet lowest end point: no longer consider segment.

Invariant: all intersections with segments above ℓ are computed.

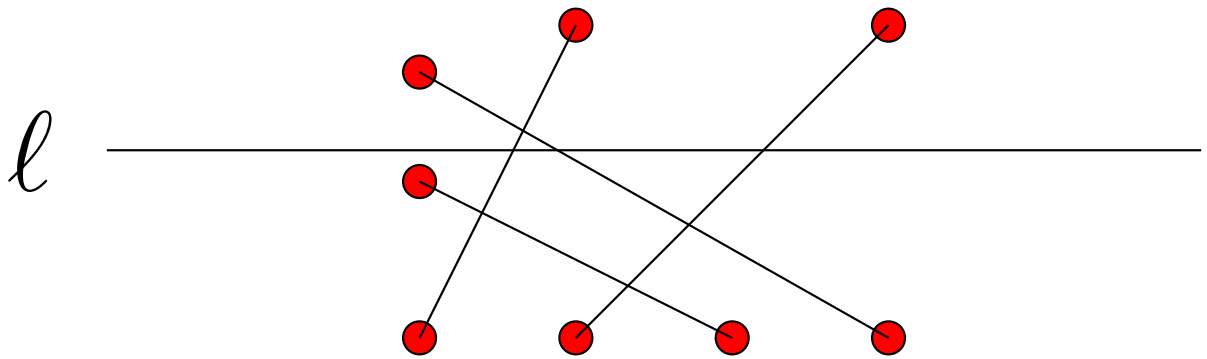
the status of the sweep line

The sweep line ℓ is characterized by its height.

Definition (status of the sweep line)

Given a set S of line segments and a sweep line ℓ,
the *status of the sweep line* ℓ is the set of segments meeting ℓ:

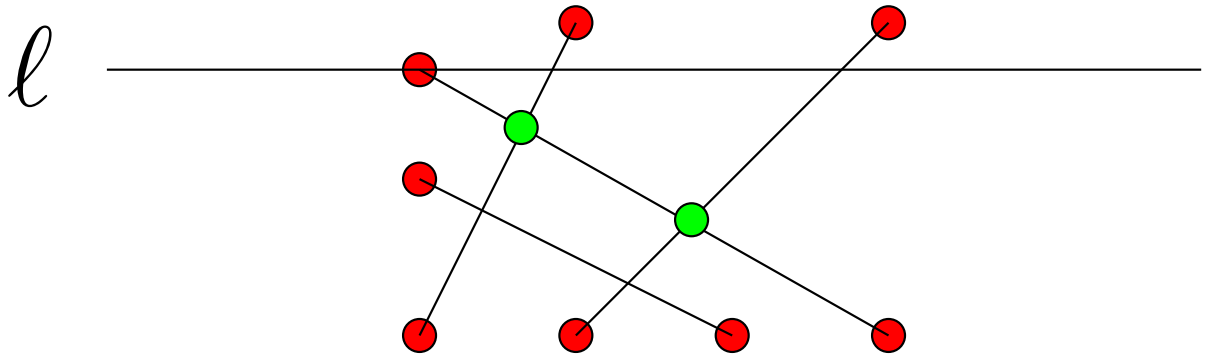
$$\text{status}(\ell) = \{ s \in S \mid s \cap \ell \neq \emptyset \}.$$



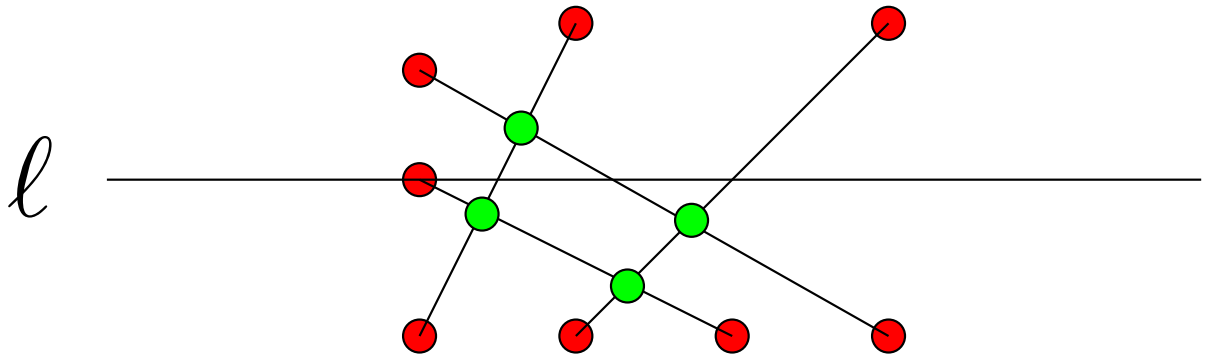
Exercise 1: What is the status in the case when the y-coordinates of the line segments in S do not overlap?

handling events

Invariant: all intersections with segments above ℓ are computed.



At the next event:

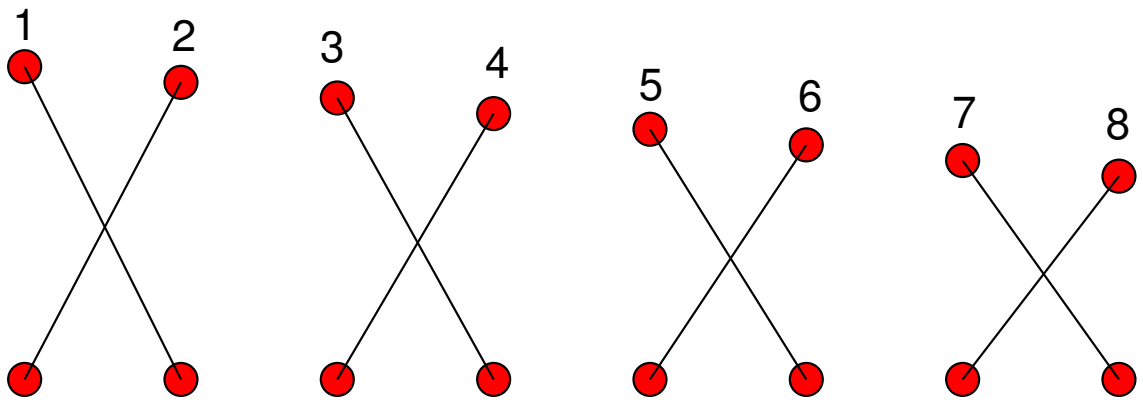


a first sweep algorithm

We can now formulate a first sweep algorithm:

for every new segment s_i about to enter $\text{status}(\ell)$ do
test whether s_i intersects $s \in \text{status}(\ell)$.

Consider the following input:



Exercise 2: Run the first sweep algorithm on the above input and describe the evolution of $\text{status}(\ell)$ at each event. Relate the number of intersection tests to the number of intersection points.

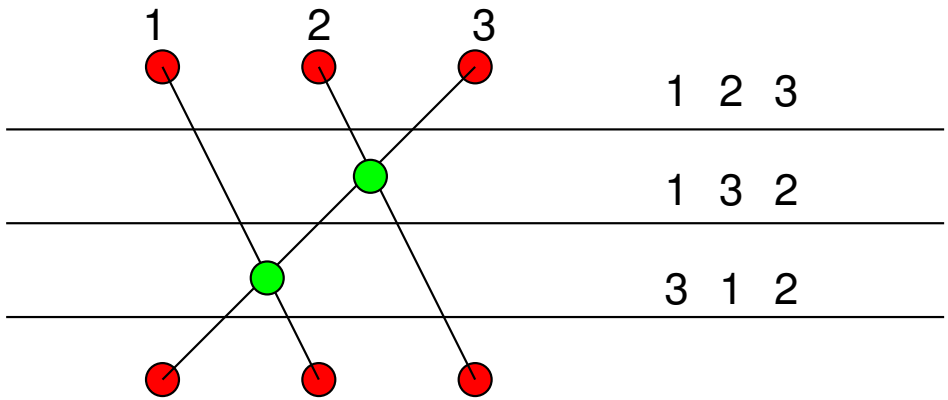
Line Segment Intersection

1. Output Sensitive Algorithms
   - map overlay and input specification
   - using the `cgal-swig-bindings`
2. A Plane Sweep Algorithm
   - handling events caused by sweeping a line
   - adjacent line segments
   - classification of events
3. Data Structures
   - the event queue
   - the global algorithm
   - an example with CGAL in C++

adjacent line segments

For an output sensitive algorithm, intersect only adjacent segments.

Consider three line segments:



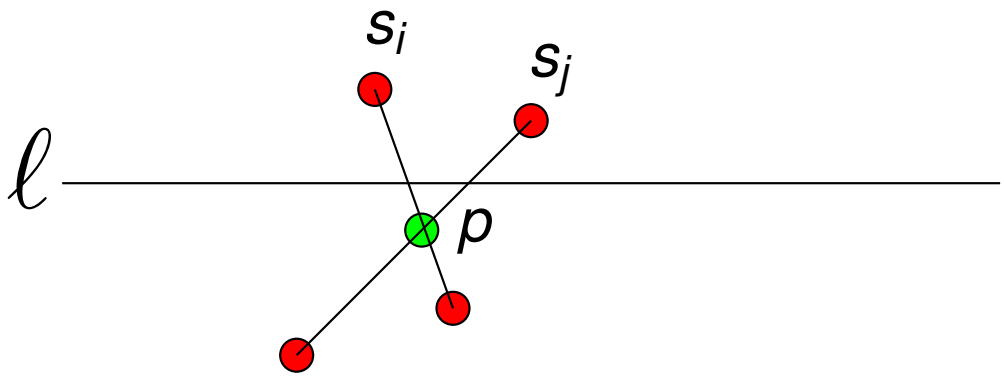
Observe: after an intersection, the adjacency changes.

Third type of event: change of adjacency.

status(ℓ) is ordered: two consecutive segments are adjacent.

Are intersections always detected?

As adjacencies change, will all intersections be detected?



Lemma (detection of intersection points)

Let s_i and s_j be two line segments, both are not horizontal.
If $s_i \cap s_j = \{p\} \notin s_k$, $k \neq i$, $k \neq j$,
then there is an event point above ℓ where s_i and s_j are adjacent.

Exercise 3: What is k in the Lemma?
Draw an example of a segment s_k so s_i and s_j are not adjacent.

intersections are detected

Lemma (detection of intersection points)

Let s_i and s_j be two line segments, both are not horizontal.
If $s_i \cap s_j = \{p\} \notin s_k$, $k \neq i$, $k \neq j$,
then there is an event point above ℓ where s_i and s_j are adjacent.

Proof. Two observations:

1. If ℓ is positioned just above p, then s_i and s_j are adjacent.
2. For high enough ℓ, $\text{status}(\ell) = \emptyset$.

Therefore, there must be an event point where s_i and s_j become adjacent are are tested for intersection.

Q.E.D.

Line Segment Intersection

1. Output Sensitive Algorithms
   - map overlay and input specification
   - using the `cgal-swig-bindings`
2. A Plane Sweep Algorithm
   - handling events caused by sweeping a line
   - adjacent line segments
   - classification of events
3. Data Structures
   - the event queue
   - the global algorithm
   - an example with CGAL in C++

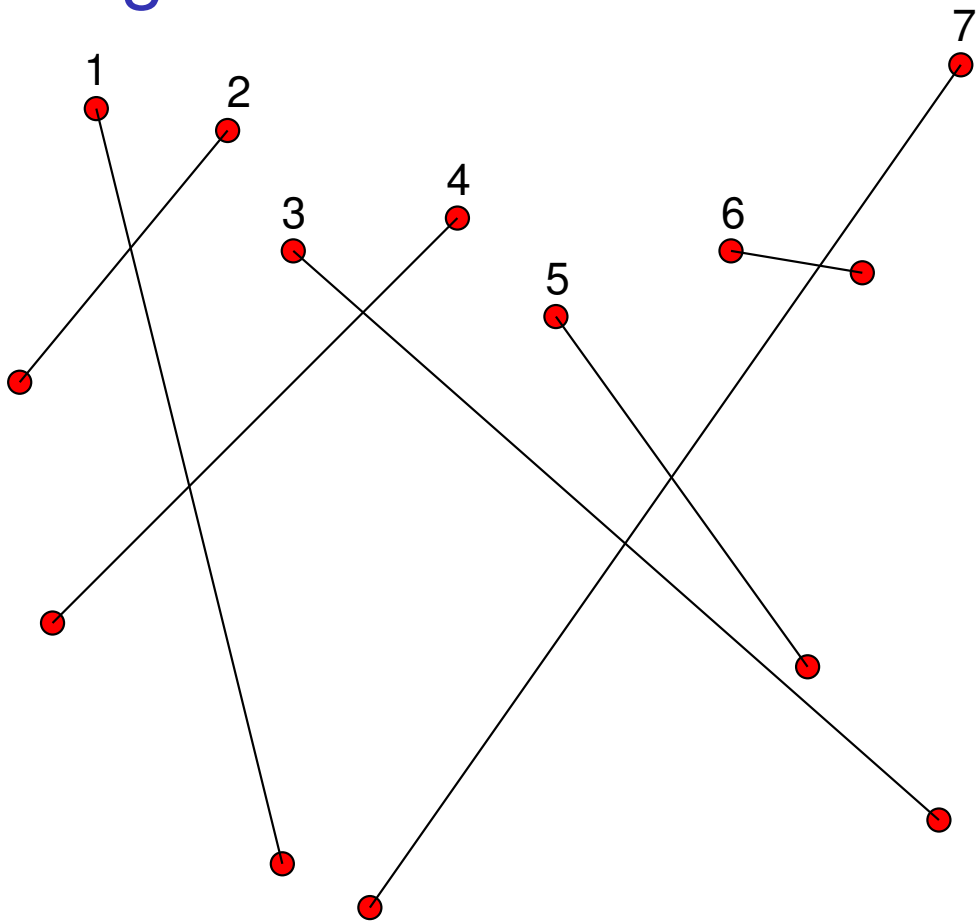
classification of event points

We have three type of event points p:

1. p is the upper end of a line segment:
   1. Find the neighbors of this line segment, test for intersection.
   2. Add the intersection points as event points.
2. p is an intersection point:
   1. Find new neighbors for intersecting segments.
   2. Test for intersection, for all new neighbors.
   3. Add the intersection points as event points.
3. p is the lower end of a line segment:
   1. Remove the segment from the status.
      The neighbors of the removed segment become adjacent.
   2. Test for intersection, for all new neighbors.
   3. Add the intersection points as event points.

This provides a sketch of a plane sweep algorithm.

a general configuration



Exercise 4: Run the plane sweep algorithm on the above input and define $\text{status}(\ell)$ at each event point.

Line Segment Intersection

1. Output Sensitive Algorithms
   - map overlay and input specification
   - using the `cgal-swig-bindings`
2. A Plane Sweep Algorithm
   - handling events caused by sweeping a line
   - adjacent line segments
   - classification of events
3. Data Structures
   - the event queue
   - the global algorithm
   - an example with CGAL in C++

the event queue

Events are stored in a queue, represented as a balanced binary tree.

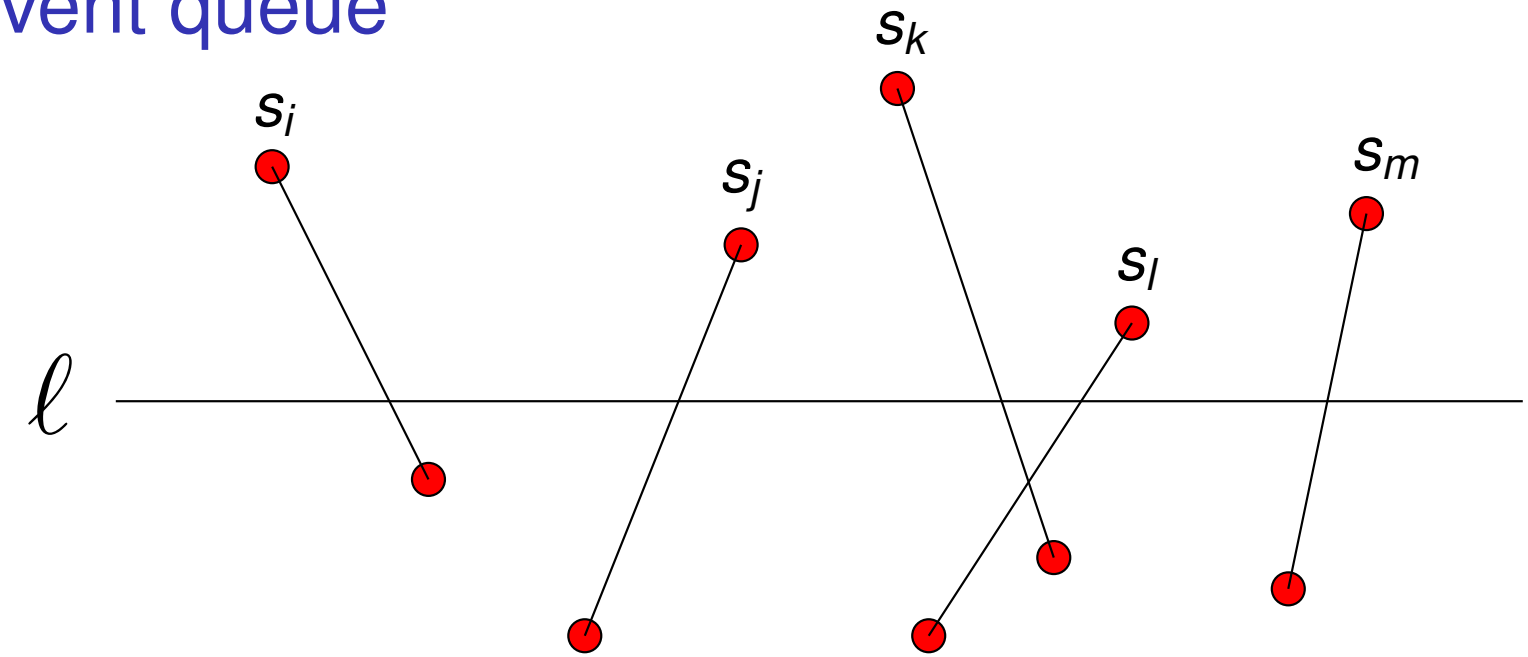
The order between event points p and q is defined as follows:

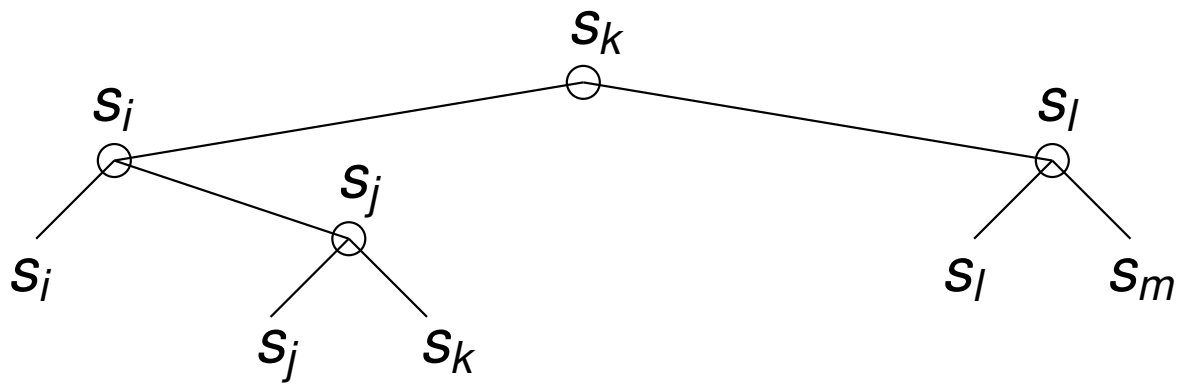
$$p(p_x, p_y) \prec q(q_x, q_y) \quad \Leftrightarrow \quad p_y > q_y \text{ or } p_y = q_y \text{ and } p_x < q_x.$$

In breaking a tie between two points with same height,
we prefer the leftmost point.

Segments are stored following $\prec$, starting with the upper end point.

an event queue





An update to the tree takes $O(\log(n))$ time.

Line Segment Intersection

1. Output Sensitive Algorithms
   - map overlay and input specification
   - using the `cgal-swig-bindings`
2. A Plane Sweep Algorithm
   - handling events caused by sweeping a line
   - adjacent line segments
   - classification of events
3. Data Structures
   - the event queue
   - the global algorithm
   - an example with CGAL in C++

the global algorithm

Input: $S = \{ s_i((a_x^i, a_y^i), (b_x^i, b_y^i)), i = 1, 2, \ldots, n \}$.
Output: $P = \{ ((i, j), (x, y)) \mid (x, y) \in s_i \cap s_j \}$.

1. Initialize event queue Q with upper end points of the segments.
2. Initialize status $T = \emptyset$.
3. While $Q \neq \emptyset$ do
4. $\quad p =$ pop off next event point from Q,
5. $\quad$ HANDLEEVENTPOINT(p, Q, T).

The subroutine HANDLEEVENTPOINT does two tasks:

1. adjust the adjacency information in the status T,
2. compute the new intersection points.

Line Segment Intersection

1. Output Sensitive Algorithms
   - map overlay and input specification
   - using the `cgal-swig-bindings`
2. A Plane Sweep Algorithm
   - handling events caused by sweeping a line
   - adjacent line segments
   - classification of events
3. Data Structures
   - the event queue
   - the global algorithm
   - an example with CGAL in C++

an example with CGAL in C++

```
#include <CGAL/Exact_predicates_exact_constructions_kernel.h>
#include <CGAL/Arr_segment_traits_2.h>
#include <CGAL/Surface_sweep_2_algorithms.h>

typedef CGAL::Exact_predicates_exact_constructions_kernel Kernel;
typedef Kernel::Point_2                                   Point_2;
typedef CGAL::Arr_segment_traits_2<Kernel>                Traits_2;
typedef Traits_2::Curve_2                                 Segment_2;

int main()
{
  // construct the input segments
  Segment_2 segments[] = {Segment_2(Point_2(0, 0), Point_2(2, 2)),
                          Segment_2(Point_2(1, 0), Point_2(0, 2)),
                          Segment_2(Point_2(2, 0), Point_2(1, 2))};

  // compute all intersection points
  std::list<Point_2> pts;

  CGAL::compute_intersection_points(segments, segments + 3,
                                    std::back_inserter(pts));

  // print the result
  std::cout << "Found " << pts.size() << " intersection points: " << std::endl;
  std::copy(pts.begin(), pts.end(),
            std::ostream_iterator<Point_2>(std::cout, "\n"));

  return 0;
}
```

compilation and running the code

Writing the executable to `sweep`, compile with the command

```
g++ sweep_segments.cpp -lgmp -lmpfr -o sweep
```

The output of `sweep` is

```
Found 2 intersection points:
0.666667 0.666667
1.33333 1.33333
```

On MacOS, when installed with `brew`, insert the `-I` after `g++` as

```
g++ -I/opt/homebrew/include/
```

and add to your .zshrc file the following lines:

```
export C_INCLUDE_PATH=/opt/homebrew/include
export LIBRARY_PATH=/opt/homebrew/lib
```

suggested activities

We started the second chapter in the textbook.

Consider the following activities, listed below.

0. Run the posted Python script and Jupyter notebook.
   Install CGAL on your computer (if not already done so).
   Browse the documentation and run examples.
1. Write the solutions to Exercises 1 through 4.
2. Consider the exercises 1,2,3 in the textbook.