2D Arrangements

Efi Fogel

Tel Aviv University 

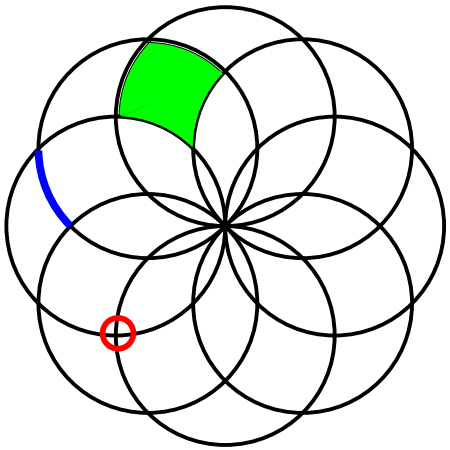
Computational Geometry
Apr. 7^{th}, 2014

Outline

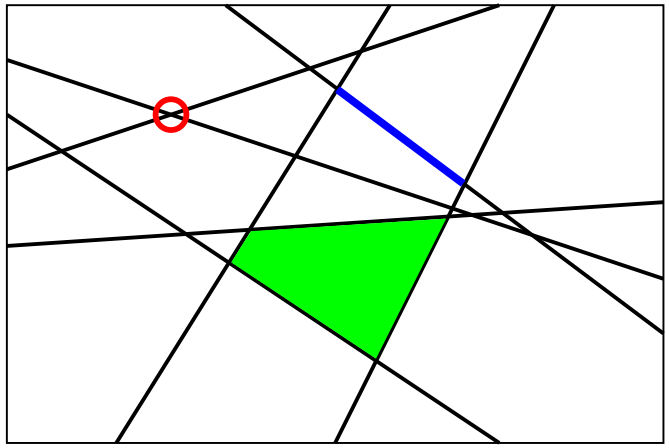
Two Dimensional Arrangements

Definition (Arrangement)

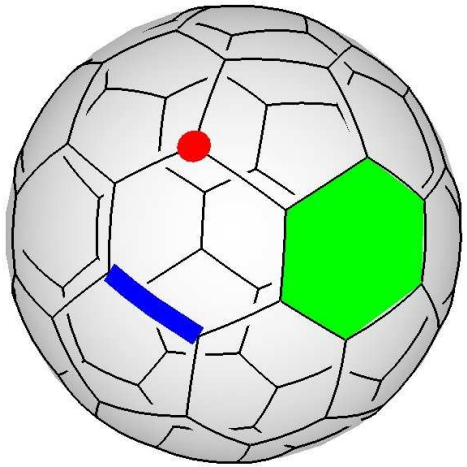
Given a collection $\mathcal{C}$ of curves on a surface, the arrangement $\mathcal{A}(\mathcal{C})$ is the partition of the surface into vertices, edges and faces induced by the curves of $\mathcal{C}$.



An arrangement of circles in the plane.



An arrangement of lines in the plane.



An arrangement of great-circle arcs on a sphere.

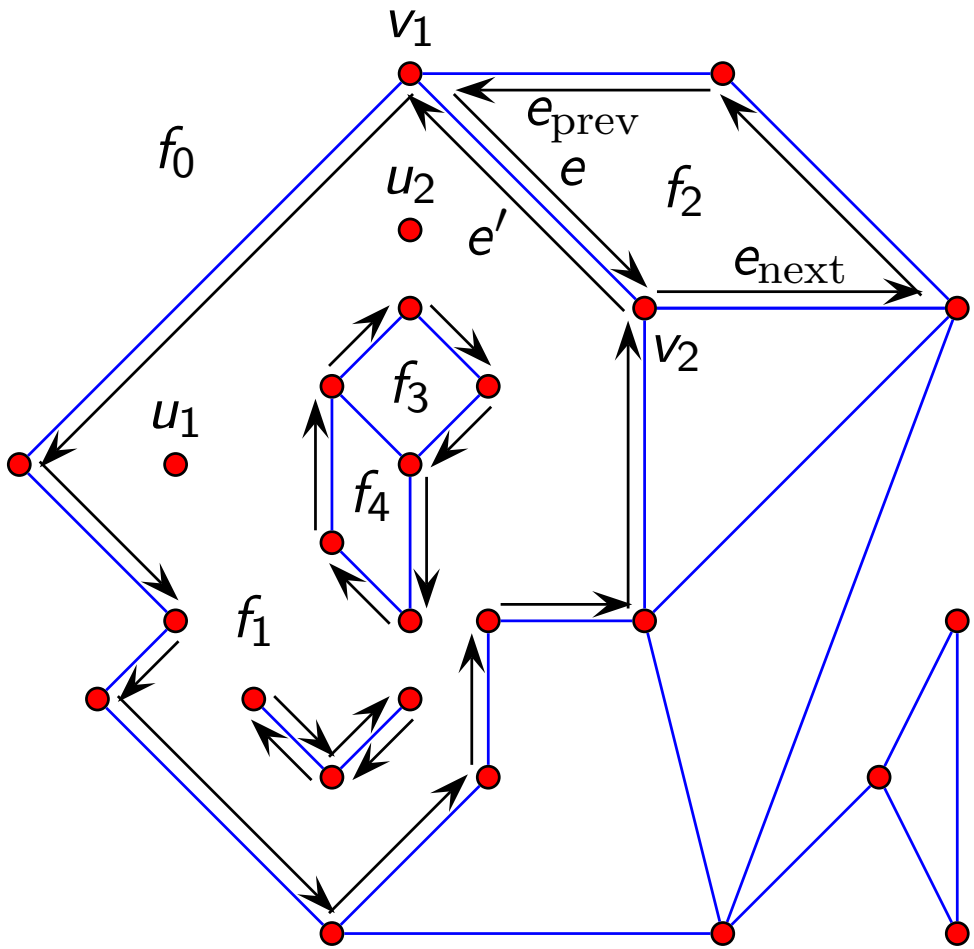
The CGAL Arrangement_on_surface_2 Package

- Constructs, maintains, modifies, traverses, queries, and presents arrangements on two-dimensional parametric surfaces.
- Complete and Robust
  - All inputs are handled correctly (including degenerate input).
  - Exact number types are used to achieve robustness.
- Generic – easy to interface, extend, and adapt
- Modular – geometric and topological aspects are separated
- Supports among the others:
  - various point location strategies
  - zone-construction paradigm
  - sweep-line paradigm
  - vertical decomposition
  - overlay computation
- Part of the CGAL basic library

Outline

The Doubly-Connected Edge List

- One of a family of combinatorial data-structures called the *halfedge data-structures*.
- Represents each edge using a pair of directed *halfedges*.
- Maintains incidence relations among cells of 0 (vertex), 1 (edge), and 2 (face) dimensions.



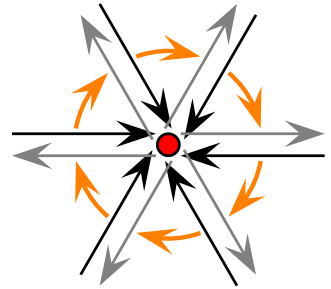
- The target vertex of a halfedge and the halefedge are incident to each other.
- The source and target vertices of a halfedge are adjacent.

The Doubly-Connected Edge List Components

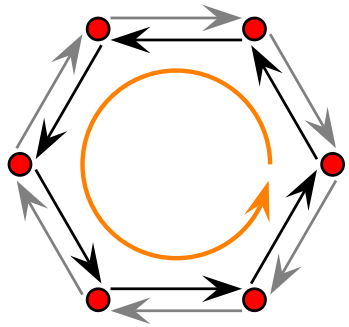
- Vertex
  - An incident halfedge pointing at the vertex.
- Halfedge
  - The opposite halfedge.
  - The previous halfedge in the component boundary.
  - The next halfedge in the component boundary.
  - The target vertex of the halfedge.
  - The incident face.
- Face
  - An incident halfedge on the outer Ccb.
  - An incident halfedge on each inner Ccb.
- Connected component of the boundary (Ccb)
  - The circular chains of halfedges around faces.

Arrangement Representation

- The halfedges incident to a vertex form a circular list.
- The halfedges are sorted in clockwise order around the vertex.



- The halfedges around faces form circular chains.
- All halfedges of a chain are incident to the same face.
- The halfedges are sorted in counterclockwise order along the boundary.



- Geometric interpretation is added by classes built on top of the halfedge data-structure.

Arrangement_2<Traits, Dcel>

- Is the main component in the *2D Arrangements* package.
- An instance of this class template represents 2D arrangements.
- The representation of the arrangements and the various geometric algorithms that operate on them are separated.
- The topological and geometric aspects are separated.
  - The Traits template-parameter must be substituted by a model of a geometry-traits concept, e.g., *ArrangementBasicTraits_2*.
    - ★ Defines the type X_monotone_curve_2 that represents x-monotone curves.
    - ★ Defines the type Point_2 that represents two-dimensional points.
    - ★ Supports basic geometric predicates on these types.
  - The Dcel template-parameter must be substituted by a model of the *ArrangementDcel* concept, e.g., Arr_default_dcel<Traits>.

Traversing the Halfedges Incident to an Arrangement Vertex

Print all the halfedges incident to a vertex.

```
template <typename Arrangement>
void print_incident_halfedges(typename Arrangement::Vertex_const_handle v)
{
  if (v->is_isolated()) {
    std::cout << "The vertex (" << v->point() << ") is isolated" << std::endl;
    return;
  }
  std::cout << "The neighbors of the vertex (" << v->point() << ") are:";
  typename Arrangement::Halfedge_around_vertex_const_circulator first, curr;
  first = curr = v->incident_halfedges();
  do std::cout << " (" << curr->source()->point() << ")";
  while (++curr != first);
  std::cout << std::endl;
}
```

Traversing the Halfedges of an Arrangement CCB

Print all x-monotone curves along a given CCB

```
template <typename Arrangement>
void print_ccb(typename Arrangement::Ccb_halfedge_const_circulator circ)
{
  std::cout << "(" << circ->source()->point() << ")";
  typename Arrangement::Ccb_halfedge_const_circulator  curr = circ;
  do {
    typename Arrangement::Halfedge_const_handle he = curr;
    std::cout << "␣␣␣[" << he->curve() << "]␣␣␣"
              << "(" << he->target()->point() << ")";
  } while (++curr != circ);
  std::cout << std::endl;
}
```

- he->curve() is equivalent to he->twin()->curve(),
- he->source() is equivalent to he->twin()->target(), and
- he->target() is equivalent to he->twin()->source().

Traversing the CCBs of an Arrangement Face

Print the outer and inner boundaries of a face.

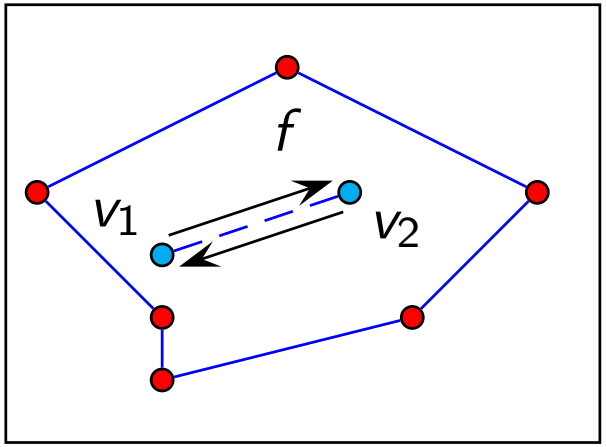
```
template <typename Arrangement>
void print_face(typename Arrangement::Face_const_handle f)
{
  // Print the outer boundary.
  if (f->is_unbounded()) std::cout << "Unbounded face. " << std::endl;
  else {
    std::cout << "Outer boundary: ";
    print_ccb<Arrangement>(f->outer_ccb());
  }

  // Print the boundary of each of the holes.
  int                                           index = 1;
  typename Arrangement::Hole_const_iterator  hole;
  for (hole = f->holes_begin(); hole != f->holes_end(); ++hole, ++index) {
    std::cout << "    Hole #" << index << ": ";
    print_ccb<Arrangement>(*hole);
  }

  // Print the isolated vertices.
  typename Arrangement::Isolated_vertex_const_iterator  iv;
  for (iv = f->isolated_vertices_begin(), index = 1;
       iv != f->isolated_vertices_end(); ++iv, ++index)
    std::cout << "    Isolated vertex #" << index << ": "
              << "(" << iv->point() << ")" << std::endl;
}
```

Traversing an Arrangement

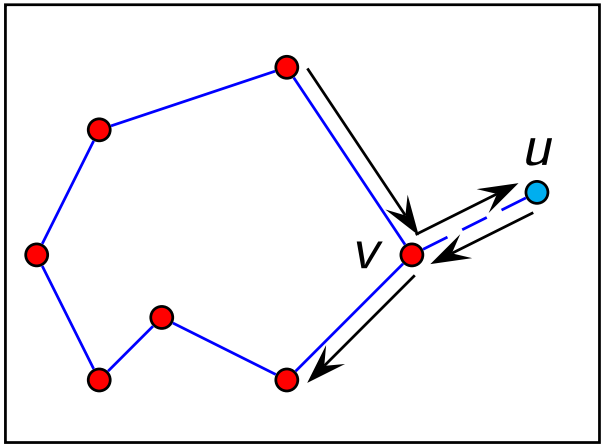
Print all the cells of an arrangement.

```
template <typename Arrangement>
void print_arrangement(const Arrangement& arr)
{
  CGAL_precondition(arr.is_valid());

  // Print the arrangement vertices.
  typename Arrangement::Vertex_const_iterator  vit;
  std::cout << arr.number_of_vertices() << " vertices:" << std::endl;
  for (vit = arr.vertices_begin(); vit != arr.vertices_end(); ++vit) {
    std::cout << "(" << vit->point() << ")";
    if (vit->is_isolated()) std::cout << " - Isolated." << std::endl;
    else std::cout << " - degree " << vit->degree() << std::endl;
  }

  // Print the arrangement edges.
  typename Arrangement::Edge_const_iterator    eit;
  std::cout << arr.number_of_edges() << " edges:" << std::endl;
  for (eit = arr.edges_begin(); eit != arr.edges_end(); ++eit)
    std::cout << "[" << eit->curve() << "]" << std::endl;

  // Print the arrangement faces.
  typename Arrangement::Face_const_iterator    fit;
  std::cout << arr.number_of_faces() << " faces:" << std::endl;
  for (fit = arr.faces_begin(); fit != arr.faces_end(); ++fit)
    print_face<Arrangement>(fit);
}
```

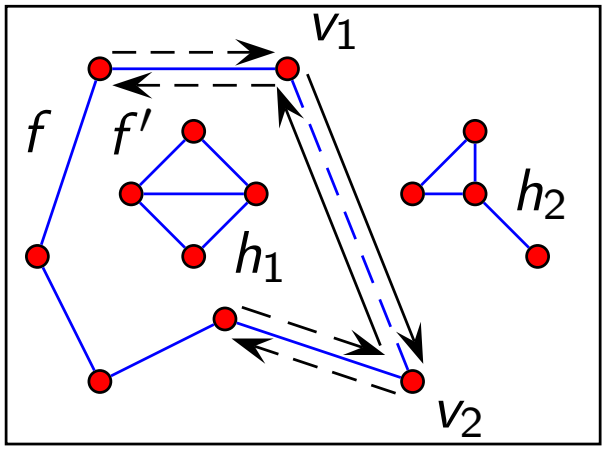
Modifying the Arrangement



Inserting a curve that induces a new hole inside the face f, `arr.insert_in_face_interior(c,f)`.



Inserting a curve from an existing vertex u that corresponds to one of its endpoints, `insert_from_left_vertex(c,v)`, `insert_from_right_vertex(c,v)`.



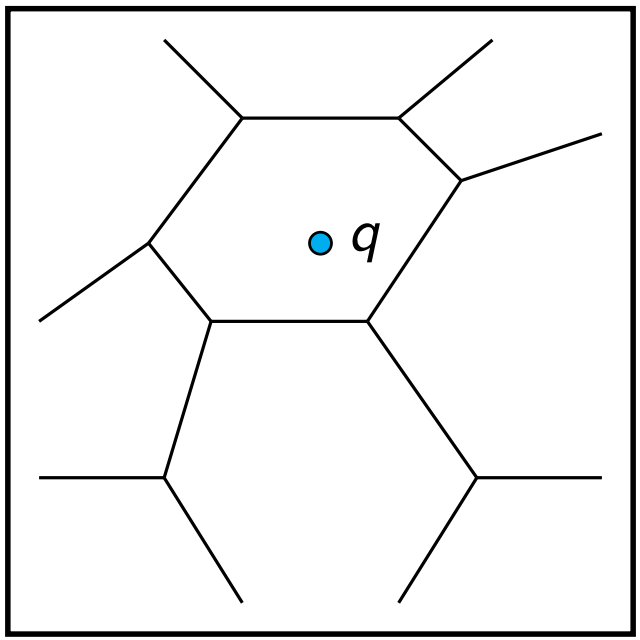
Inserting an x-monotone curve, the endpoints of which correspond to existing vertices v_1 and v_2, `insert_at_vertices(c,v1,v2)`.

- The new pair of halfedges close a new face f'.
- The hole h_1, which belonged to f before the insertion, becomes a hole in this new face.

Outline

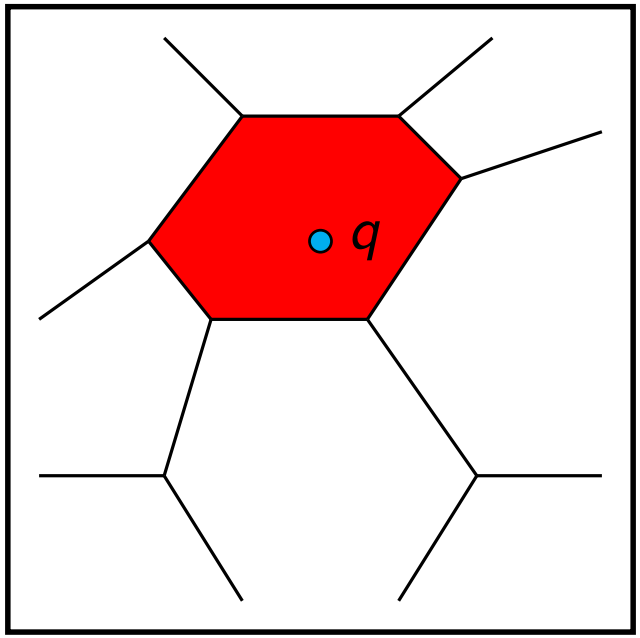
Arrangement Point Location

Given a subdivision A of the space into cells and a query point q, find the cell of A containing q.



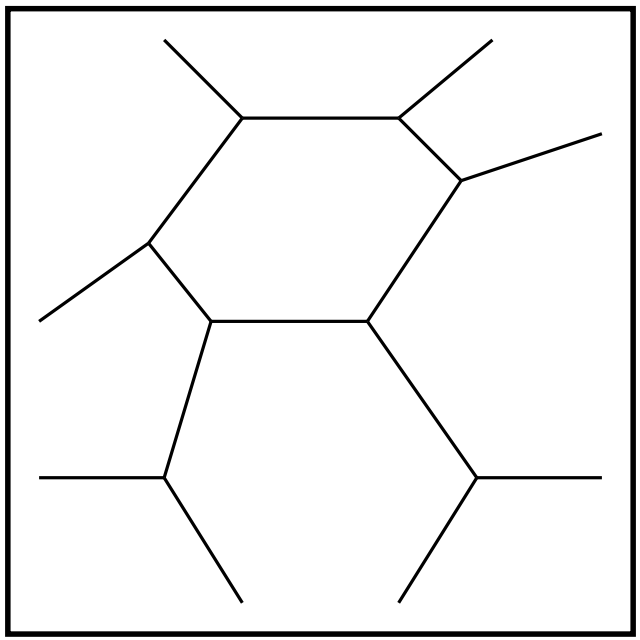
Arrangement Point Location

Given a subdivision A of the space into cells and a query point q, find the cell of A containing q.



Arrangement Point Location

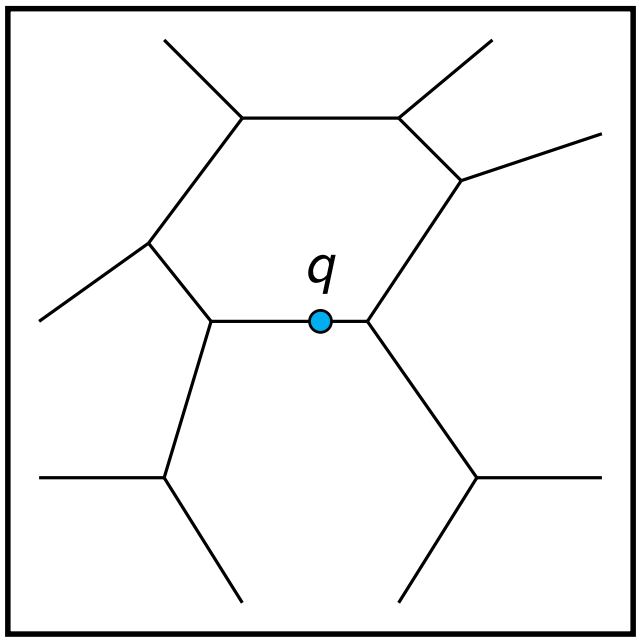
Given a subdivision A of the space into cells and a query point q, find the cell of A containing q.



- In degenerate situations the query point can

Arrangement Point Location

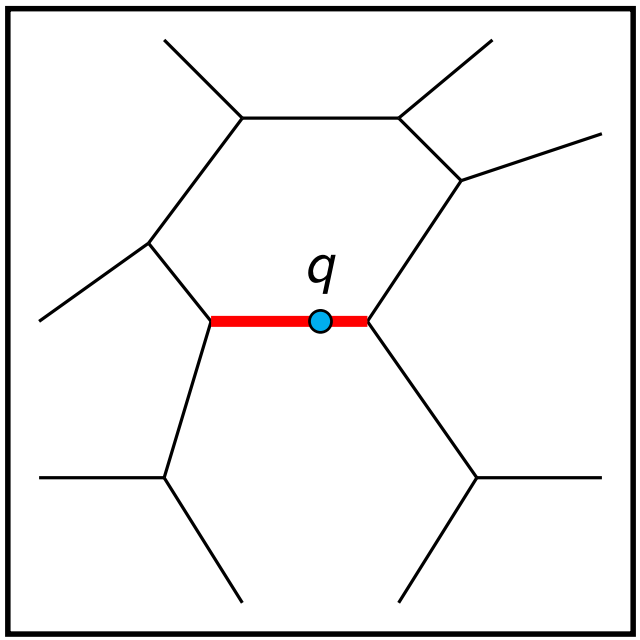
Given a subdivision A of the space into cells and a query point q, find the cell of A containing q.



- In degenerate situations the query point can
  - lie on an edge, or

Arrangement Point Location

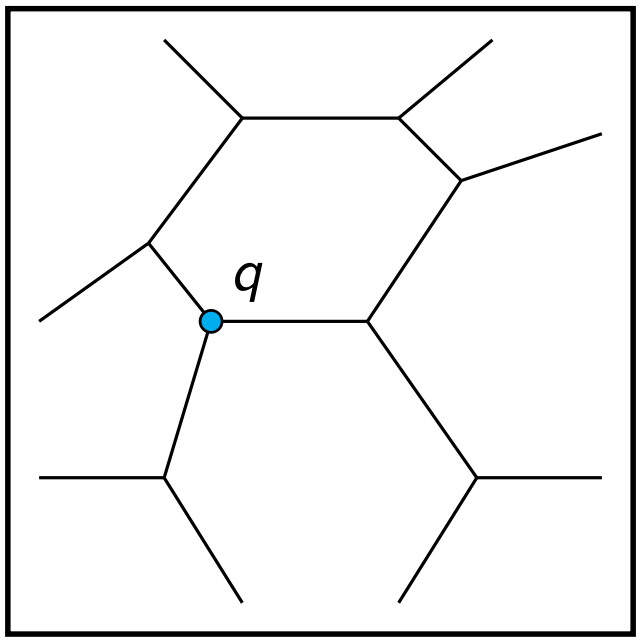
Given a subdivision A of the space into cells and a query point q, find the cell of A containing q.



- In degenerate situations the query point can
  - lie on an edge, or

Arrangement Point Location

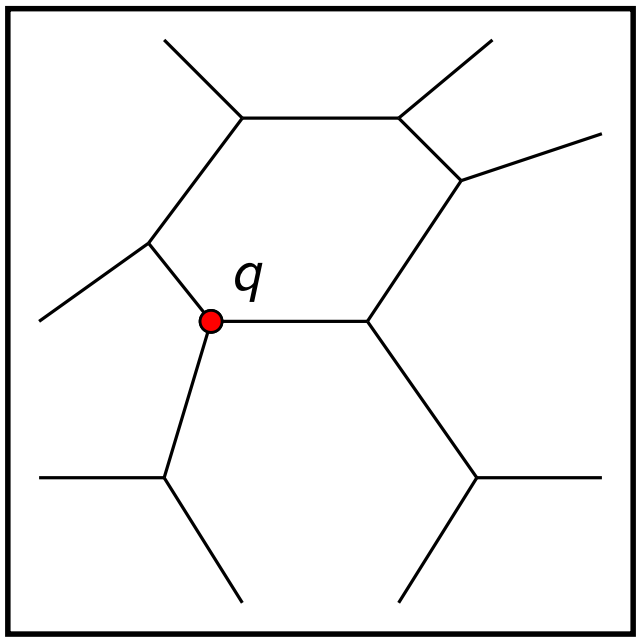
Given a subdivision A of the space into cells and a query point q, find the cell of A containing q.



- In degenerate situations the query point can
  - lie on an edge, or
  - coincide with a vertex.

Arrangement Point Location

Given a subdivision A of the space into cells and a query point q, find the cell of A containing q.



- In degenerate situations the query point can
  - lie on an edge, or
  - coincide with a vertex.

CGAL Point Location Strategies

- Naive
  - Traverse all edges of the arrangement to find the closes.
- Walk along line
  - Walk along a vertical line from infinity.
- Trapezoidal map **R**andomized **I**ncremental-**C**onstruction (RIC)
- Landmark

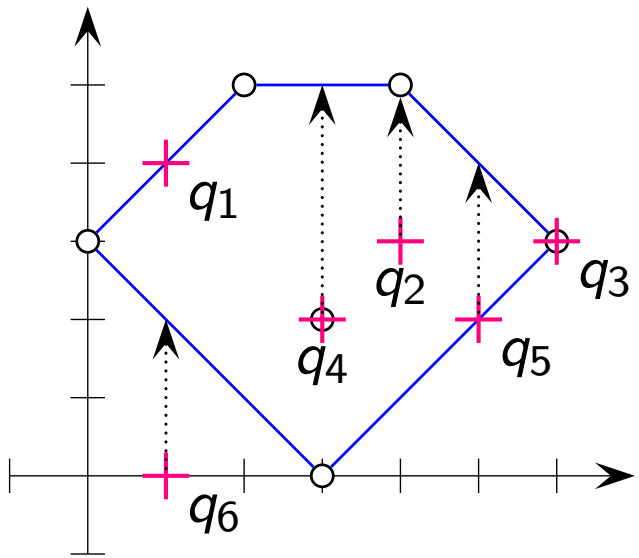
Point Location: Print

Print a polymorphic object.

```
template <typename Arrangement_>
void print_point_location(const typename Arrangement_::Point_2& q,
                          CGAL::Arr_point_location_result<Arrangement_>::Type& obj)
{
  typedef Arrangement_                                    Arrangement
  typedef typename Arrangement::Vertex_const_handle    Vertex_const_handle;
  typedef typename Arrangement::Halfedge_const_handle  Halfedge_const_handle;
  typedef typename Arrangement::Face_const_handle      Face_const_handle;

  const Vertex_const_handle*   v;
  const Halfedge_const_handle* e;
  const Face_const_handle*     f;

  std::cout << "The point (" << q << ") is located ";
  if ((f = boost::get<Face_const_handle>(&obj)))          // located inside a face
    std::cout << "inside "
              << (((*f)->is_unbounded()) ? "the unbounded" : "a bounded")
              << " face." << std::endl;
  else if ((e = boost::get<Halfedge_const_handle>(&obj))) // located on an edge
    std::cout << "on an edge: " << (*e)->curve() << std::endl;
  else if ((v = boost::get<Vertex_const_handle>(&obj)))   // located on a vertex
    std::cout << "on " << (((*v)->is_isolated()) ? "an isolated" : "a")
              << " vertex: " << (*v)->point() << std::endl;
  else CGAL_error_msg("Invalid object.");                 // this should never happen
}
```

Point Location: Locate

```
template <typename PointLocation>
void locate_point(const PointLocation& pl,
                  const typename Point_location::Arrangement_2::Point_2& q)
{
  typedef PointLocation                                    Point_location;
  typedef typename Point_location::Arrangement_2           Arrangement_2;
  typename CGAL::Arr_point_location_result<Arrangement_2>::Type obj = pl.locate(q);

  // Print the result.
  print_point_location<Arrangement_2>(q, obj);
}
```



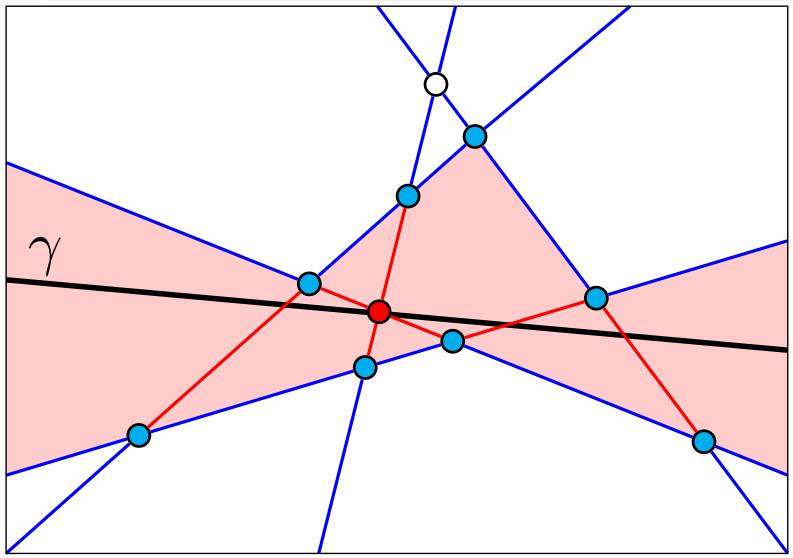
Point Location: Example

```
// File: ex_point_location.cpp

#include <CGAL/basic.h>
#include <CGAL/Arr_naive_point_location.h>
#include <CGAL/Arr_landmarks_point_location.h>

#include "arr_inexact_construction_segments.h"
#include "point_location_utils.h"

typedef CGAL::Arr_naive_point_location<Arrangement_2>       Naive_pl;
typedef CGAL::Arr_landmarks_point_location<Arrangement_2>   Landmarks_pl;

int main()
{
  // Construct the arrangement.
  Arrangement_2  arr;
  construct_segments_arr(arr);

  // Perform some point-location queries using the naive strategy.
  Naive_pl       naive_pl(arr);
  locate_point(naive_pl, Point_2(1, 4));          // q1

  // Attach the landmarks object to the arrangement and perform queries.
  Landmarks_pl landmarks_pl;
  landmarks_pl.attach(arr);
  locate_point(landmarks_pl, Point_2(3, 2));      // q4

  return 0;
}
```

The Zone of Curves in Arrangements

Definition (Zone)

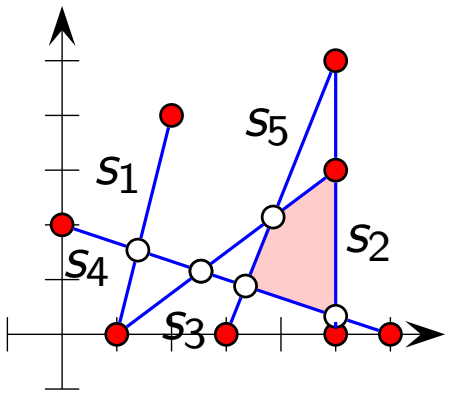
Given an arrangement of curves $\mathcal{A} = \mathcal{A}(\mathcal{C})$ in the plane, the zone of an additional curve $\gamma \notin \mathcal{C}$ in $\mathcal{A}$ is the union of the features of $\mathcal{A}$, whose closure is intersected by γ.



The zone of a line γ in an arrangement of lines.

Incremental Insertion

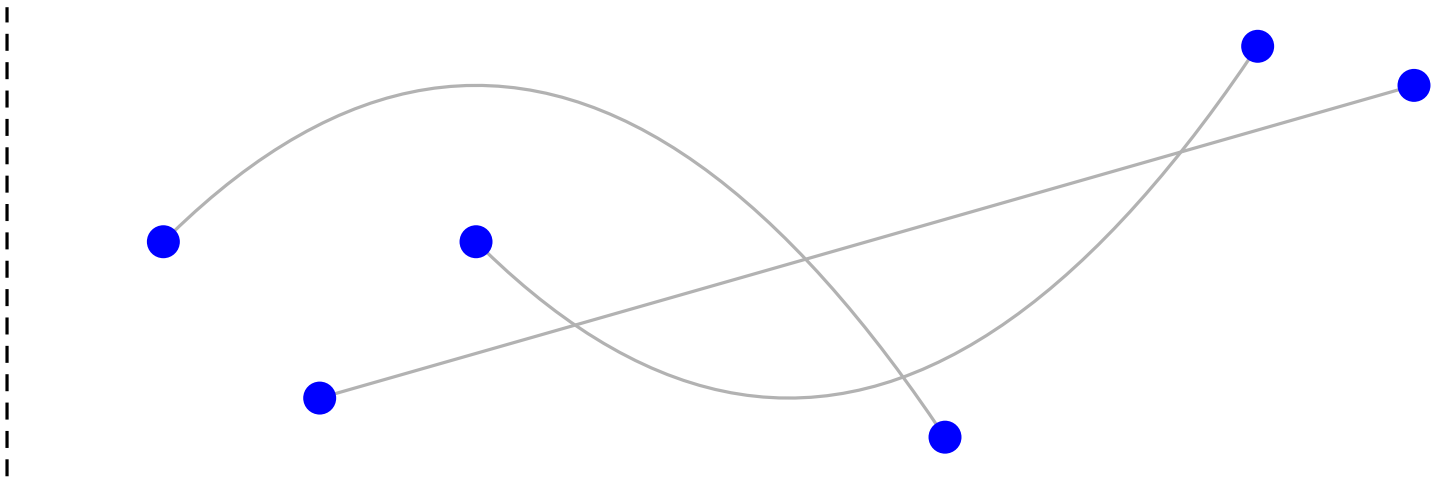
```
// File: ex_incremental_insertion.cpp

#include <CGAL/basic.h>
#include <CGAL/Arr_naive_point_location.h>

#include "arr_exact_construction_segments.h"
#include "arr_print.h"

int main()
{
  // Construct the arrangement of five line segments.
  Arrangement_2 arr;
  Naive_pl pl(arr);
  CGAL::insert_non_intersecting_curve(arr, Segment_2(Point_2(1, 0), Point_2(2, 4)), pl);
  CGAL::insert_non_intersecting_curve(arr, Segment_2(Point_2(5, 0), Point_2(5, 5)));
  CGAL::insert(arr, Segment_2(Point_2(1, 0), Point_2(5, 3)), pl);
  CGAL::insert(arr, Segment_2(Point_2(0, 2), Point_2(6, 0)));
  CGAL::insert(arr, Segment_2(Point_2(3, 0), Point_2(5, 5)), pl);
  print_arrangement_size(arr);
  return 0;
}
```



The Plane Sweep Algorithmic Framework

[BO79]

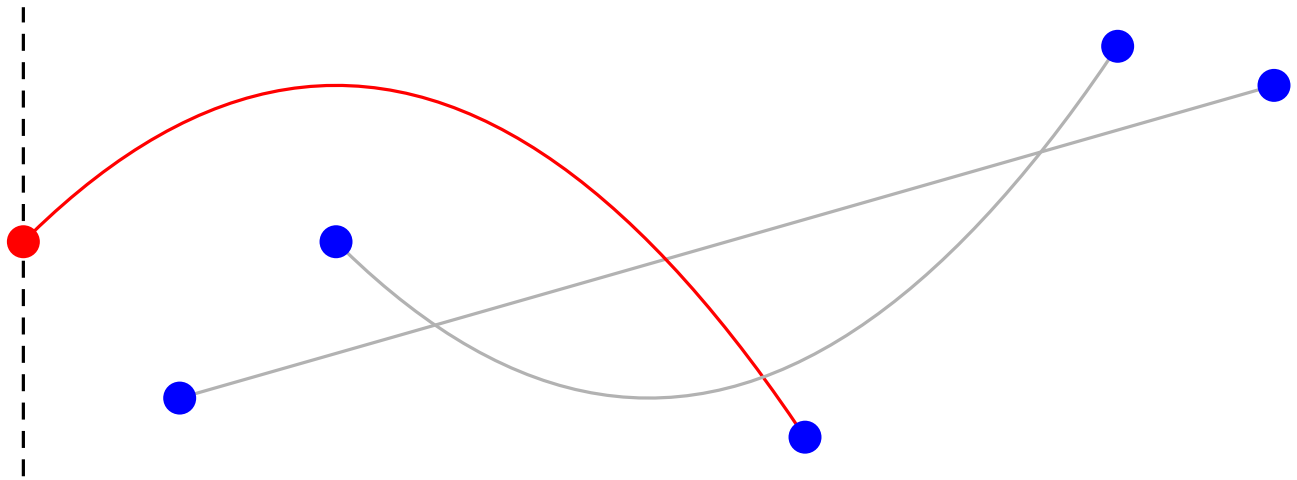
- Initialize an event queue with all endpoints sorted lexicographically
- While the queue is not empty, extract and process an event
  - Remove all x-monotone curves to the left of the current event point from a sorted container of curves
  - Insert all x-monotone curves to the right of the current event point into the curve container
  - Compute intersections between existing curves and newly inserted curves, and insert them into the event queue



The Plane Sweep Algorithmic Framework

[BO79]

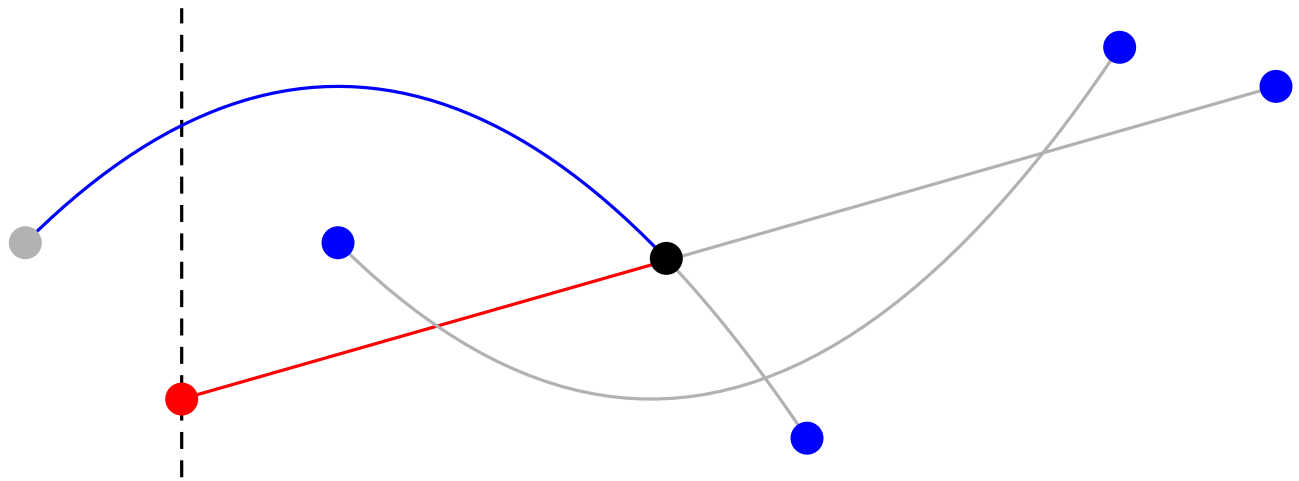
- Initialize an event queue with all endpoints sorted lexicographically
- While the queue is not empty, extract and process an event
  - Remove all x-monotone curves to the left of the current event point from a sorted container of curves
  - Insert all x-monotone curves to the right of the current event point into the curve container
  - Compute intersections between existing curves and newly inserted curves, and insert them into the event queue



The Plane Sweep Algorithmic Framework

[BO79]

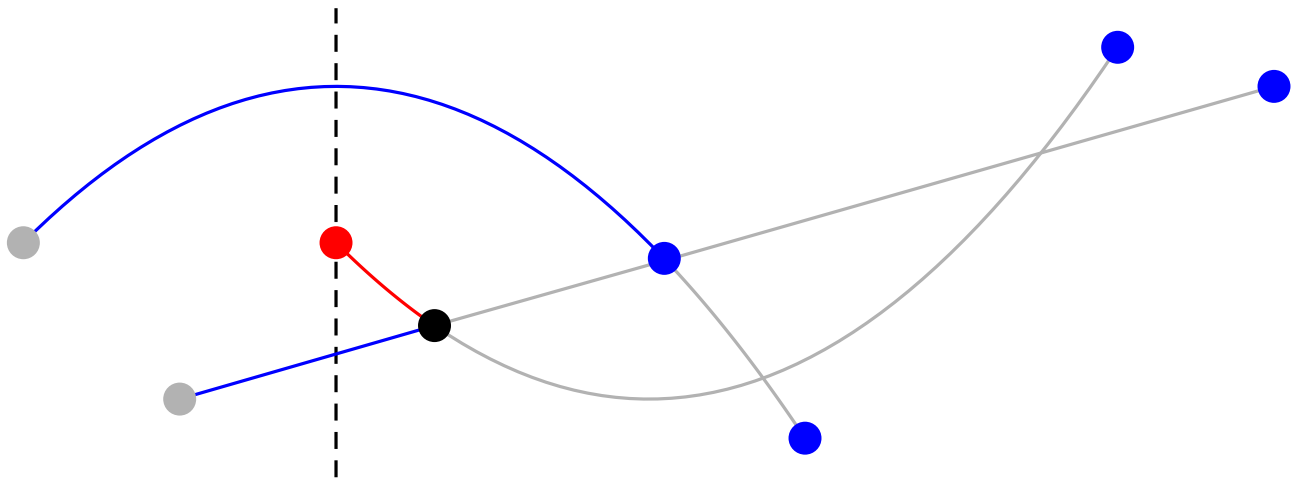
- Initialize an event queue with all endpoints sorted lexicographically
- While the queue is not empty, extract and process an event
  - Remove all x-monotone curves to the left of the current event point from a sorted container of curves
  - Insert all x-monotone curves to the right of the current event point into the curve container
  - Compute intersections between existing curves and newly inserted curves, and insert them into the event queue



The Plane Sweep Algorithmic Framework

[BO79]

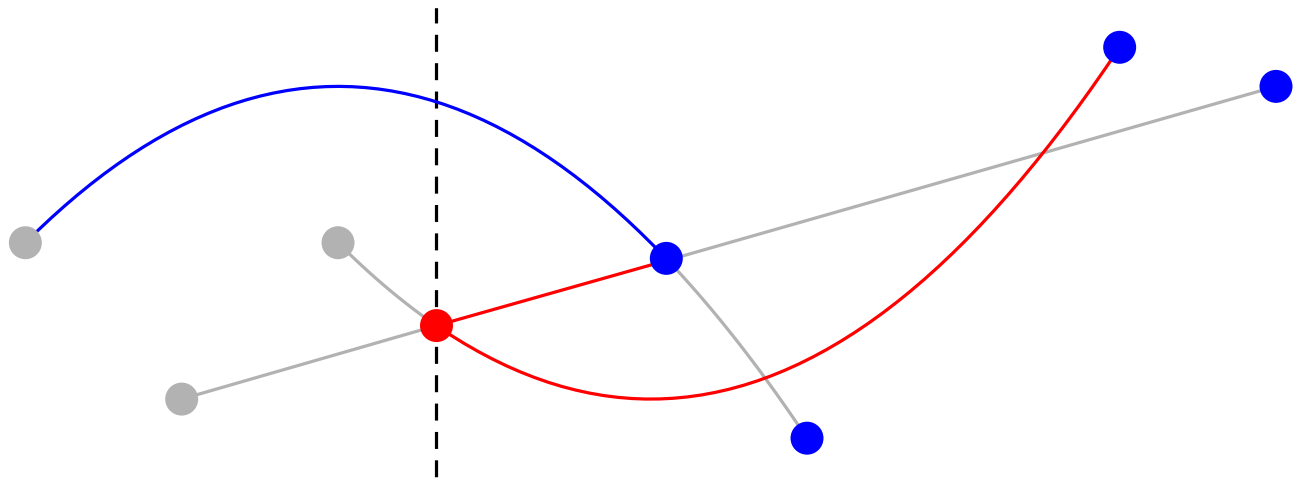
- Initialize an event queue with all endpoints sorted lexicographically
- While the queue is not empty, extract and process an event
  - Remove all x-monotone curves to the left of the current event point from a sorted container of curves
  - Insert all x-monotone curves to the right of the current event point into the curve container
  - Compute intersections between existing curves and newly inserted curves, and insert them into the event queue



The Plane Sweep Algorithmic Framework

[BO79]

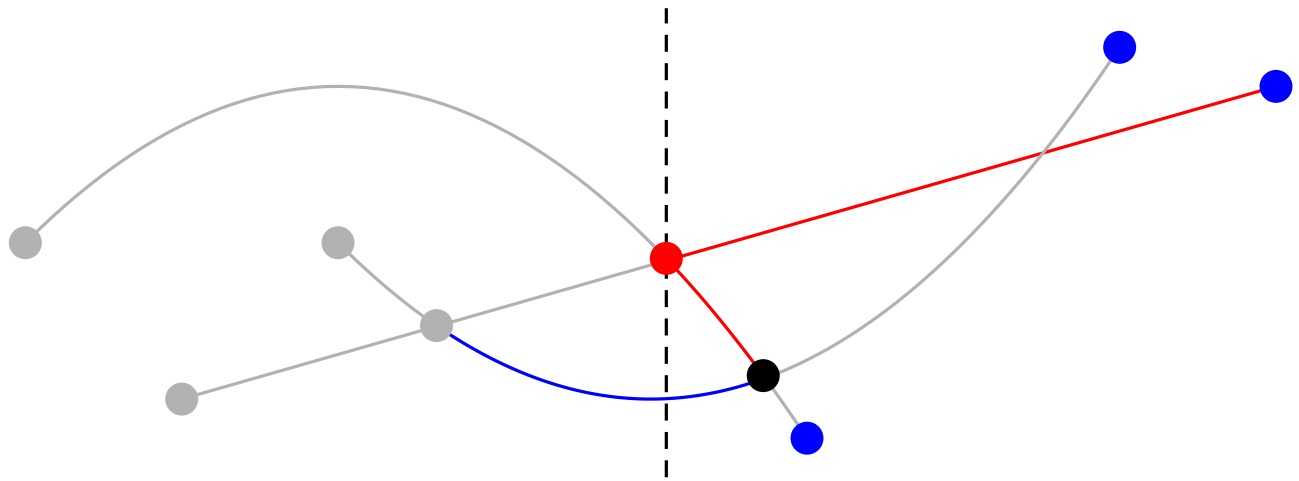
- Initialize an event queue with all endpoints sorted lexicographically
- While the queue is not empty, extract and process an event
  - Remove all x-monotone curves to the left of the current event point from a sorted container of curves
  - Insert all x-monotone curves to the right of the current event point into the curve container
  - Compute intersections between existing curves and newly inserted curves, and insert them into the event queue



The Plane Sweep Algorithmic Framework

[BO79]

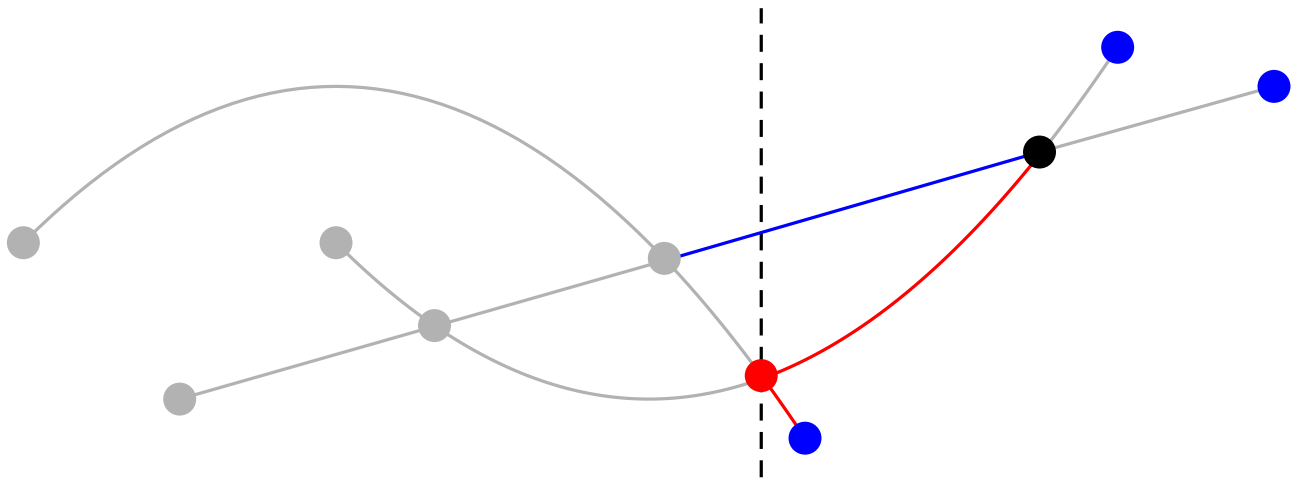
- Initialize an event queue with all endpoints sorted lexicographically
- While the queue is not empty, extract and process an event
  - Remove all x-monotone curves to the left of the current event point from a sorted container of curves
  - Insert all x-monotone curves to the right of the current event point into the curve container
  - Compute intersections between existing curves and newly inserted curves, and insert them into the event queue



The Plane Sweep Algorithmic Framework

[BO79]

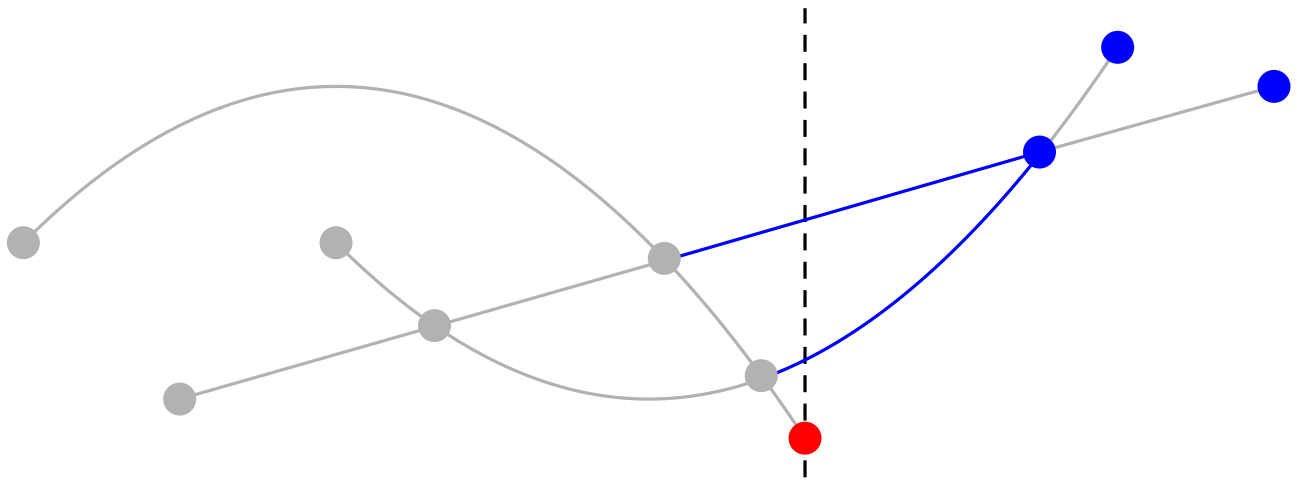
- Initialize an event queue with all endpoints sorted lexicographically
- While the queue is not empty, extract and process an event
  - Remove all x-monotone curves to the left of the current event point from a sorted container of curves
  - Insert all x-monotone curves to the right of the current event point into the curve container
  - Compute intersections between existing curves and newly inserted curves, and insert them into the event queue



The Plane Sweep Algorithmic Framework

[BO79]

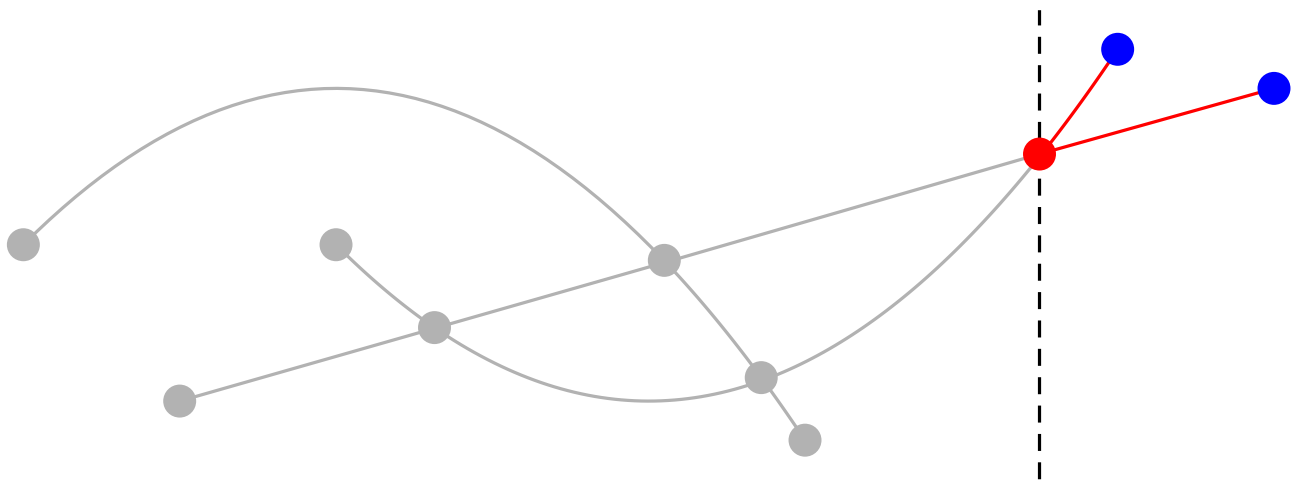
- Initialize an event queue with all endpoints sorted lexicographically
- While the queue is not empty, extract and process an event
  - Remove all x-monotone curves to the left of the current event point from a sorted container of curves
  - Insert all x-monotone curves to the right of the current event point into the curve container
  - Compute intersections between existing curves and newly inserted curves, and insert them into the event queue



The Plane Sweep Algorithmic Framework

[BO79]

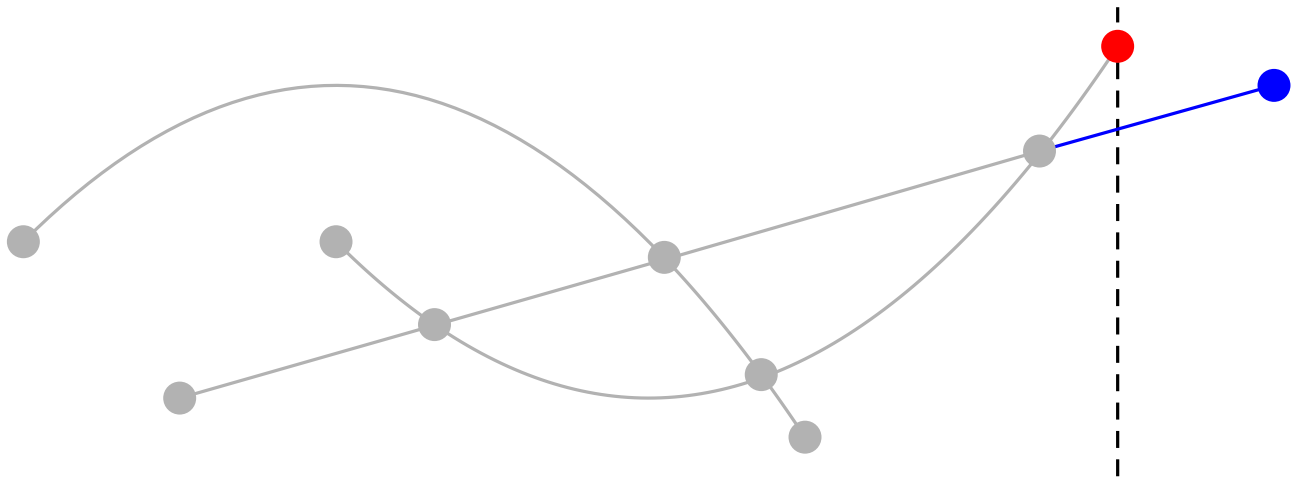
- Initialize an event queue with all endpoints sorted lexicographically
- While the queue is not empty, extract and process an event
  - Remove all x-monotone curves to the left of the current event point from a sorted container of curves
  - Insert all x-monotone curves to the right of the current event point into the curve container
  - Compute intersections between existing curves and newly inserted curves, and insert them into the event queue



The Plane Sweep Algorithmic Framework

[BO79]

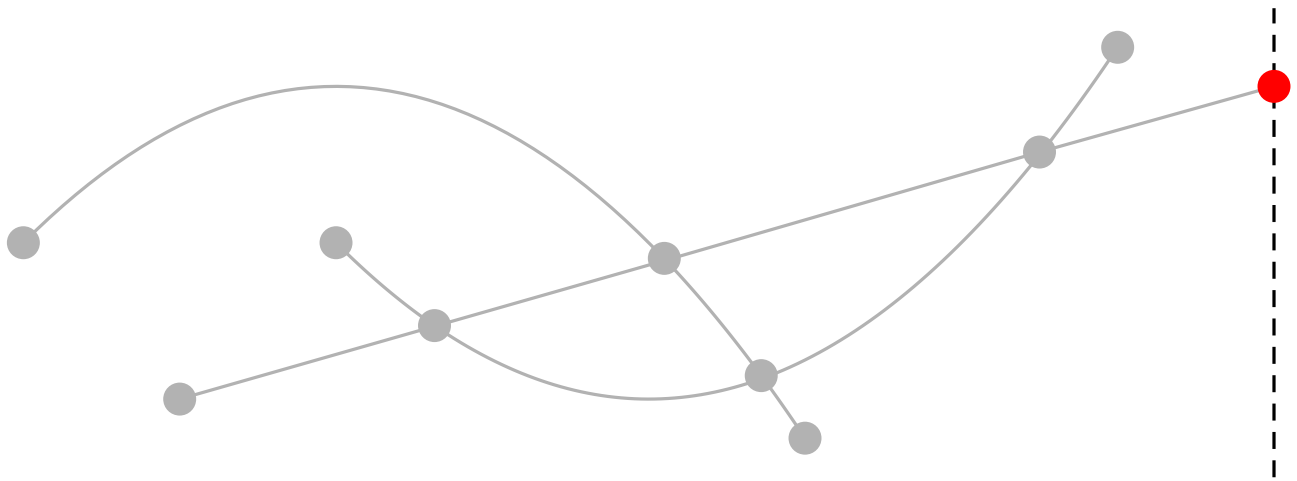
- Initialize an event queue with all endpoints sorted lexicographically
- While the queue is not empty, extract and process an event
  - Remove all x-monotone curves to the left of the current event point from a sorted container of curves
  - Insert all x-monotone curves to the right of the current event point into the curve container
  - Compute intersections between existing curves and newly inserted curves, and insert them into the event queue



The Plane Sweep Algorithmic Framework

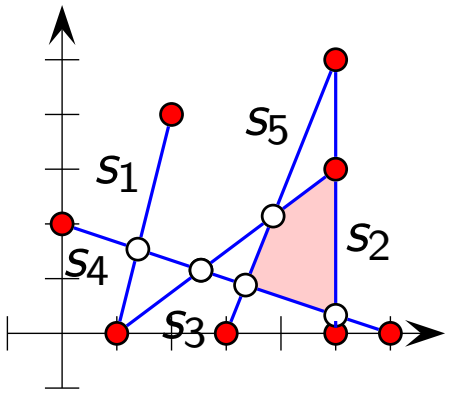
[BO79]

- Initialize an event queue with all endpoints sorted lexicographically
- While the queue is not empty, extract and process an event
  - Remove all x-monotone curves to the left of the current event point from a sorted container of curves
  - Insert all x-monotone curves to the right of the current event point into the curve container
  - Compute intersections between existing curves and newly inserted curves, and insert them into the event queue



Aggregate Insertion

```
// File: ex_aggregated_insertion.cpp

#include "arr_exact_construction_segments.h"
#include "arr_print.h"

int main()
{
  // Aggregately construct the arrangement of five line segments.
  Segment_2 segments[] = {Segment_2(Point_2(1, 0), Point_2(2, 4)),
                          Segment_2(Point_2(5, 0), Point_2(5, 5)),
                          Segment_2(Point_2(1, 0), Point_2(5, 3)),
                          Segment_2(Point_2(0, 2), Point_2(6, 0)),
                          Segment_2(Point_2(3, 0), Point_2(5, 5))};
  Arrangement_2 arr;
  CGAL::insert(arr, segments, segments + sizeof(segments)/sizeof(Segment_2));
  print_arrangement_size(arr);
  return 0;
}
```



Map Overlay

Definition (map overlay)

The map overlay of two planar subdivisions $\mathcal{S}_1$ and $\mathcal{S}_2$, denoted as overlay$(\mathcal{S}_1, \mathcal{S}_2)$, is a planar subdivision $\mathcal{S}$, such that there is a face f in $\mathcal{S}$ if and only if there are faces f_1 and f_2 in $\mathcal{S}_1$ and $\mathcal{S}_2$ respectively, such that f is a maximal connected subset of $f_1 \cap f_2$.

The overlay of two subdivisions embedded on a surface in $\mathbb{R}^3$ is defined similarly.

n_1, n_2, n — number of vertices in $\mathcal{S}_1$, $\mathcal{S}_2$, overlay$(\mathcal{S}_1, \mathcal{S}_2)$.

- Time complexities of the computation of the overlay of 2 subdivisions embedded on surfaces in $\mathbb{R}^3$:
  - Using sweep-line: $O((n)\log(n_1 + n_2))$. [BO79]
  - Using trapezoidal decomposition: $O(n)$. [FH95]
    - ★ Precondition: $\mathcal{S}_1$ and $\mathcal{S}_2$ are simply connected.

Map Overlay of CGAL

```
template <typename TraitsRed, typename TraitsBlue, typename TraitsRes,
          typename DcelRed, typename DcelBlue, typename DcelRes,
          typename OverlayTraits>
void overlay (const Arrangement_2<TraitsRed, DcelRed> & arr1,
              const Arrangement_2<TraitsBlue, DcelBlue> & arr2,
              Arrangement_2<TraitsRes, DcelRes> & arr_res, OverlayTraits & ovl_tr)
```

The concept *OverlayTraits* requires the provision of ten functions that handle all possible cases as follows:

1. A new vertex v is induced by coinciding vertices v_r and v_b.
2. A new vertex v is induced by a vertex v_r that lies on an edge e_b.
3. An analogous case of a vertex v_b that lies on an edge e_r.
4. A new vertex v is induced by a vertex v_r that is contained in a face f_b.
5. An analogous case of a vertex v_b contained in a face f_r.
6. A new vertex v is induced by the intersection of two edges e_r and e_b.
7. A new edge e is induced by the overlap of two edges e_r and e_b.
8. A new edge e is induced by the an edge e_r that is contained in a face f_b.
9. An analogous case of an edge e_b contained in a face f_r.
10. A new face f is induced by the overlap of two faces f_r and f_b.

Outline

Arrangement Geometry Traits

- Separates geometric asspects from topological asspects
  - `Arrangement_2<Traits, Dcel>` — main component.
- Is a parameter of the data structures and algorithms.
  - Defines the family of curves that induce the arrangement.
  - A parameterized data structure or algorithm can be used with any family of curves for which a traits class is supplied.
- Aggregates
  - Geometric types (point, curve).
  - Operations over types (accessors, predicates, constructons).

Arrangement Geometry Traits

- Separates geometric asspects from topological asspects
  - Arrangement_2<Traits , Dcel> — main component.
- Is a parameter of the data structures and algorithms.
  - Defines the family of curves that induce the arrangement.
  - A parameterized data structure or algorithm can be used with any family of curves for which a traits class is supplied.
- Aggregates
  - Geometric types (point, curve).
  - Operations over types (accessors, predicates, constructons).
- Each input curve is subdivided into x-monotone subcurves.
  - Most operations involve points and x-monotone curves.

Arrangement Traits Models

- Line segments:
  1. Uses the kernel point and segment types.
  2. Caches the underlying line.
- Linear curves, i.e., line segments, rays, and lines.
- Circular arcs and line segments.
- Conic curves
- Arcs of rational functions.
- Bézier curves.
- Algebraic curves of arbitrary degrees.

Arrangement Geometry Traits Models

① — *ArrangementLandmarkTraits_2* ② — *ArrangementTraits_2*
③ — *ArrangementDirectionalXMonotoneTraits_2* ④ — *ArrangementOpenTraits_2*.

Model Name	Curve Family	Degree	Concepts	
Arr_non_caching_segment_basic_traits_2	line segments	1	①	
Arr_non_caching_segment_traits_2	line segments	1	①,②,③	
Arr_segment_traits_2	line segments	1	①,②,③	
Arr_linear_traits_2	line segments, rays, and lines	1	①,②,③,④	
Arr_circle_segment_traits_2	line segments and circular arcs	≤ 2	②,③	
Arr_circular_line_arc_traits_2	line segments and circular arcs	≤ 2	②	
Arr_conic_traits_2	circles, ellipses, and conic arcs,	≤ 2	①,②,③	
Arr_rational_function_traits_2	curves of rational functions	≤ 2	①,②,③,④	
Arr_Bezier_curve_traits_2	Bézier curves	$\leq n$	②,③	
Arr_algebraic_segment_traits_2	algebraic curves	$\leq n$	②,③,④	
Arr_polyline_traits_2	polylines	∞	①,②,③	

Outline

The Notification Mechanism

Definition (Observer)

An observer defines a one-to-many dependency between objects, so that when one object changes state, all its dependents are notified and updated automatically.

- The *2D Arrangements* package offers a mechanism that uses *observers*.
- The observed type is derived from an instance of `Arr_observer<Arrangement>`.
- The observed object does not know anything about the observers.
- Each arrangement object stores a list of pointers to `Arr_observer` objects.
- The trapezoidal-RIC and the landmark point-location strategies use observers to keep their auxiliary data-structures up-to-date.

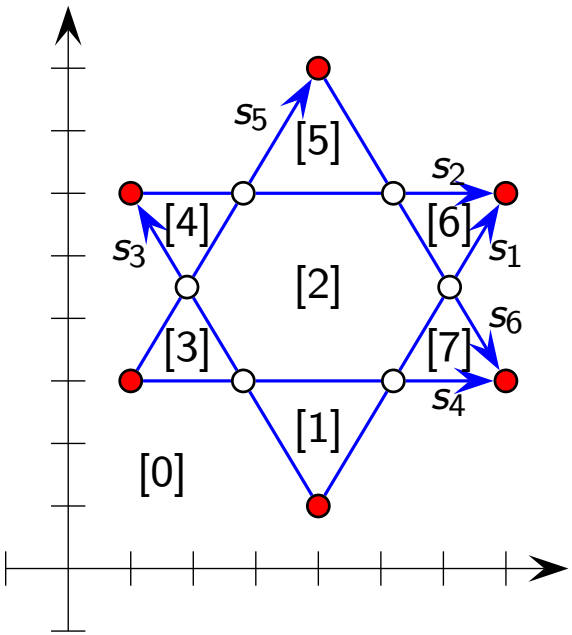
Observer Notification Functions

The set of functions can be divided into 3 categories:

1. Notifiers on changes that affect the topological structure of the arrangement. There are 2 pairs (*before* and *after*) that notify when
   - the arrangement is cleared or
   - the arrangement is assigned with the contents of another one.
2. Pairs of notifiers before and after of a local change that occurs in the topological structure.
   - A new vertex is constructed or deleted.
   - An new edge is constructed or deleted.
   - 1 edge is split into 2 edges, or 2 are merged into 1.
   - 1 face is split into 2 faces, or 2 are merged into 1.
   - 1 hole is created in the interior of a face or removed from it.
   - 2 holes are merged into 1, or 1 is split into 2.
   - A hole is moved from one face to another.
3. Notifiers on a structural change caused by a free function. A single pair `before_global_change()` and `after_global_change()`.

Extending the DCEL Faces

- An instance of `Arr_face_extended_dcel<Traits, FaceData>` is a DCEL that extends the face record with the `FaceData` type.
- Data-fields must be maintained by the user application.
  - You can construct an arrangement, go over the faces, and store data in the appropriate face data-fields.
  - You can use an observer that receives updates whenever a face is modified and sets its data fields accordingly.



- `ex_face_extension.cpp`
- Assigns indeces to all faces in the order of creation.

Extending the DCEL Faces (Cont.)

```
#include <CGAL/basic.h>
#include <CGAL/Arr_extended_dcel.h>
#include <CGAL/Arr_observer.h>

#include "arr_exact_construction_segments.h"

typedef CGAL::Arr_face_extended_dcel<Traits_2, unsigned int>  Dcel;
typedef CGAL::Arrangement_2<Traits_2, Dcel>                   Ex_arrangement_2;

// An arrangement observer used to receive notifications of face splits and
// to update the indices of the newly created faces.
class Face_index_observer : public CGAL::Arr_observer<Ex_arrangement_2> {
private:
  unsigned int  n_faces;                  // the current number of faces

public:
  Face_index_observer(Ex_arrangement_2& arr) :
    CGAL::Arr_observer<Ex_arrangement_2>(arr), n_faces(0)
  {
    CGAL_precondition(arr.is_empty());
    arr.unbounded_face()->set_data(0);
  }

  virtual void after_split_face(Face_handle old_face, Face_handle new_face, bool)
  {
    new_face->set_data(++n_faces);          // assign index to the new face
  }
};
```

Extending all the DCEL Records

- An instance of

  `Arr_extended_dcel<Traits, VertexData, HalfedgeData, FaceData>`

  is a DCEL that extends the vertex, halfedge, and face records with the corresponding types.

```
enum Color {BLUE, RED, WHITE};

typedef CGAL::Arr_extended_dcel<Traits_2, Color, bool, unsigned int>    Dcel;
typedef CGAL::Arrangement_2<Traits_2, Dcel>                             Ex_arrangement_2;
```

```
  Ex_arrangement_2::Vertex_iterator           vit;
  for (vit = arr.vertices_begin(); vit != arr.vertices_end(); ++vit) {
    unsigned int degree = vit->degree();
    vit->set_data((degree == 0) ? BLUE : ((degree <= 2) ? RED : WHITE));
  }
```

```
  std::cout << "The arrangement vertices:" << std::endl;
  for (vit = arr.vertices_begin(); vit != arr.vertices_end(); ++vit) {
    std::cout << '(' << vit->point() << ") - ";
    switch (vit->data()) {
      case BLUE  : std::cout << "BLUE."  << std::endl; break;
      case RED   : std::cout << "RED."   << std::endl; break;
      case WHITE : std::cout << "WHITE." << std::endl; break;
    }
  }
```

Extending all the DCEL Records (Cont.)

```
#include <string>

#include <CGAL/basic.h>
#include <CGAL/Arr_dcel_base.h>

/*! The map extended dcel vertex */
template <typename Point_2>
class Arr_map_vertex : public CGAL::Arr_vertex_base<Point_2> {
public:
  std::string name, type;
};

/*! The map extended dcel halfedge */
template <typename X_monotone_curve_2>
class Arr_map_halfedge : public CGAL::Arr_halfedge_base<X_monotone_curve_2> {
public:
  std::string name, type;
};

/*! The map extended dcel face */
class Arr_map_face : public CGAL::Arr_face_base {
public:
  std::string name, type;
};

/*! The map extended dcel */
template <typename Traits>
class Arr_map_dcel : public
  CGAL::Arr_dcel_base<Arr_map_vertex<typename Traits::Point_2>,
                      Arr_map_halfedge<typename Traits::X_monotone_curve_2>,
                      Arr_map_face>
{};
```

Outline

Arrangement Bibliography I


Jon Louis Bentley and Thomas Ottmann.
Algorithms for Reporting and Counting Geometric Intersections.
IEEE Transactions on Computers, 28(9): 643–647, 1979.


Eric Berberich, Efi Fogel, Dan Halperin, Michael Kerber, and Ophir Setter.
Arrangements on parametric surfaces ii: Concretizations and applications, 2009.
Mathematics in Computer Science, 4(1):67–91,2010.


Ulrich Finke and Klaus H. Hinrichs.
Overlaying simply connected planar subdivisions in linear time.
In *Proceedings of* 11^{th} *Annual ACM Symposium on Computational Geometry (SoCG)*, pages 119–126. Association for Computing Machinery (ACM) Press, 1995.


Ron Wein, Efi Fogel, Baruch Zukerman, Dan Halperin, and Eric Berberich.
2D Arrangements.
In CGAL Editorial Board, editor, *CGAL User and Reference Manual*. 4.4 edition, 2014. `http://www.cgal.org/Manual/latest/doc_html/cgal_manual/packages.html#Pkg:Arrangement2`.


David G. Kirkpatrick.
Optimal search in planar subdivisions.
SIAM Journal on Computing. 12(1):28–35,1983.


N. Sarnak and Robert E. Tarjan.
Planar point location using persistent search trees.
Communications of the ACM. 29(7):669–679, 1986.


Kentan Mulmuley.
A fast planar partition algorithm, I.
Journal of Symbolic Computation. 10(3-4):253–280,1990.

Arrangement Bibliography II


Raimund Seidel
A Simple and Fast Incremental Randomized Algorithm for Computing Trapezoidal Decompositions and for Triangulating Polygons.
Computational Geometry: Theory and Applications. 1(1):51–64, 1991.


Olivier Devillers, Sylvain Pion, and Monique Teillaud.
Walking in a triangulation.
International Journal of Foundations of Computer Science. 13:181–199,2002.


Luc Devroye Christophe, Christophe Lemaire, and Jean-Michel Moreau.
Fast Delaunay Point-Location with Search Structures.
In *Proceedings of* 11^{th} *Canadian Conference on Computational Geometry*. Pages 136–141, 1999.


Luc Devroye, Ernst Peter Mücke, and Binhai Zhu.
A Note on Point Location in Delaunay Triangulations of Random Points.
Algorithmica. 22:477–482, 1998.


Olivier Devillers.
The Delaunay hierarchy.
International Journal of Foundations of Computer Science. 13:163-180, 2002.


Sunil Arya
A Simple Entropy-Based Algorithm for Planar Point Location.
ACM Transactions on Graphics. 3(2), 2007


Masato Edahiro, Iwao Kokubo, And Takao Asano
A new Point-Location Algorithm and its Practical Efficiency: comparison with existing algorithms
ACM Transactions on Graphics. 3(2):86–109, 1984.


Micha Sharir and Pankaj Kumar Agarwal
Davenport-Schinzel Sequences and Their Geometric Applications.
Cambridge University Press, New York, NY, 1995.

Arrangement Bibliography III


Bernard Chazelle, Leonidas J. Guibas, and Der-Tsai Le.
The Power of Geometric Duality.
BIT, 25:76–90, 1985.


Herbert Edelsbrunner,
Algorithms in Combinatorial Geometry,
Springer, Heidelberg, 1987.


Mark de Berg, Mark van Kreveld, Mark H. Overmars, and Otfried Cheong.
Computational Geometry: Algorithms and Applications.
Springer, 3^{rd} edition, 2008.


Herbert Edelsbrunner, Raimund Seidel, and Micha Sharir.
On the Zone Theorem for Hyperplane Arrangements.
SIAM Journal on Computing. 22(2):418–429,1993.


Silvio Micali and Vijay V. Vazirani.
An $O(\sqrt{|V||E|})$ Algorithm for Finding Maximum Matching in General Graphs.
Proceedings of 21^{st} *Annual IEEE Symposium on the Foundations of Computer Science*, pages 17–27, 1980.


Jack Edmonds.
Paths, Trees, and Flowers.
Canadian Journal of Mathematics, 17:449–467,1965.

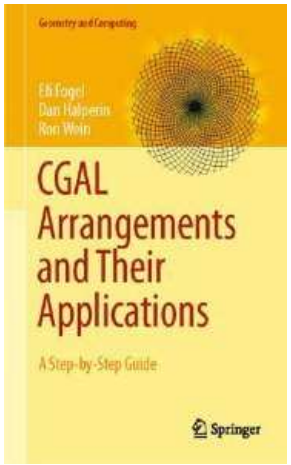

Robert Endre Tarjan.
Data structures and network algorithms, Society for Industrial and Applied Mathematics (SIAM), 1983.


Marcin Mucha and Piotr Sankowski.
Maximum Matchings via Gaussian Elimination
Proceedings of 45^{th} *Annual IEEE Symposium on the Foundations of Computer Science*, pages 248–255, 2004.

Arrangement Bibliography IV

Jeremy G. Siek, Lie-Quan Lee, and Andrew Lumsdaine.
The Boost Graph Library.
Addison-Wesley, 2002

Efi Fogel, Ron Wein, and Dan Halperin.
Cgal Arrangements and Their Applications, A Step-by-Step Guide.
Springer, 2012.



Movies

Eyal Flato, Efi Fogel, Dan Halperin, and Eyal Leiserowitz.

Movie: Exact Minkowski Sums and Applications.

In *Proceedings of* 18^{th} *Annual ACM Symposium on Computational Geometry (SoCG)*, pages 273–274. Association for Computing Machinery (ACM) Press, 2005.

Efi Fogel and Dan Halperin.

Movie: Exact Minkowski sums of convex polyhedra.

In *Proceedings of* 21^{st} *Annual ACM Symposium on Computational Geometry (SoCG)*, pages 382–383. Association for Computing Machinery (ACM) Press, 2005.

Efi Fogel, Ophir Setter, and Dan Halperin.

Movie: Arrangements of Geodesic Arcs on the Sphere.

In *Proceedings of* 24^{th} *Annual ACM Symposium on Computational Geometry (SoCG)*, pages 218–219. Association for Computing Machinery (ACM) Press, 2008.