Two Dimensional Arrangements

Definition (Arrangement)

Given a collection $\mathcal{C}$ of curves on a surface, the arrangement $A(\mathcal{C})$ 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

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 halfedge 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.

2D Arrangements 7

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.

2D Arrangements 8

Traversing the Halfedges Incident to an Arrangement Vertex

Print all the halfedges incident to a vertex.

```cpp
template<typename Arrangement>
void print_incident_halfedges(Arrangement& arr)
{
    std::cout << "The neighbors of the vertex (" << iv−>point() << ") are: ";
    print_ccb(arr); std::cout << std::endl;
}
```

2D Arrangements 9

Traversing the Halfedges of an Arrangement CCB

Print all x-monotone curves along a given CCB.

```cpp
// 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().
```

2D Arrangements 10

Traversing the Ccbs of an Arrangement Face

Print the outer and inner boundaries of a face.

```cpp
template<typename Arrangement>
void print_face(Arrangement& arr)
{
    // Print the outer boundary.
    if (fhs.is_unbounded())
        std::cout << "Unbounded face ";
    else
        std::cout << "Outer boundary (";
    print_ccb(arr); std::cout << std::endl;
}
```

2D Arrangements 11
Traversing an Arrangement

Print all the cells of an arrangement.

```cpp
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->is_isolated() ? "- Isolated. " : "- 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)
    CGAL::print_face<Arrangement>(fit);
}
```

Modifying the Arrangement

Inserting a curve that induces a new hole inside the face $f$,

\[ \text{arr.insert_in_face_interior}(c, f) \]

Inserting a curve from an existing vertex $u$ that corresponds to one of its endpoints,

\[ \text{arr.insert_from_vertex}(c, v) \]

or

\[ \text{arr.insert_from_left_vertex}(c, v) \]

or

\[ \text{arr.insert_from_right_vertex}(c, v) \]

Inserting an $x$-monotone curve, the endpoints of which correspond to existing vertices $v_1$ and $v_2$,

\[ \text{arr.insert_at_vertices}(c, v_1, v_2) \]

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

1. Definitions
2. Representation
3. Queries & Free Functions
   - Point Location Queries
   - The Zone Computation Algorithm
   - The Plane Sweep Algorithm
   - Vertical Decomposition
   - Map Overlay
4. Arrangement-Traits Classes
5. Extending the Arrangement
   - Extending Cell Records
6. Literature

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
  - 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
  - Randomized Incremental-Construction (RIC)
- Landmark

Point Location: Print

Print a polymorphic object.

```cpp
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," << std::endl;
    else if (v = boost::get<Vertex_const_handle>(&obj))
        // located on a vertex
        std::cout << "on point (" << v->point() << std::endl;
    else
        CGAL_error_msg("Invalid object.");
    // this should never happen
}
```
The Zone of Curves in Arrangements

Definition (Zone)

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

The Plane Sweep Algorithmic Framework

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

- 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
**Map Overlay**

**Definition (map overlay)**

The map overlay of two planar subdivisions $S_1$ and $S_2$, denoted as $\text{overlay}(S_1, S_2)$, is a planar subdivision $S$, such that there is a face $f$ in $S$ if and only if there are faces $f_1$ and $f_2$ in $S_1$ and $S_2$ 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 $S_1$, $S_2$, $\text{overlay}(S_1, S_2)$.

- Time complexities of the computation of the overlay of 2 subdivisions embedded on surfaces in $\mathbb{R}^3$:
  - Using sweep-line: $O(n \log(n_1 + n_2))$.
  - Using trapezoidal decomposition: $O(n)$.

**Precondition**: $S_1$ and $S_2$ are simply connected.

---

**Aggregate Insertion**

```cpp
// 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(0, 2), Point_2(6, 0)),
                           Segment_2(Point_2(3, 0), Point_2(5, 5)),
                           Segment_2(Point_2(2, 2), Point_2(4, 4)),
                           Segment_2(Point_2(2, 8), Point_2(6, 2))};
  Arrangement_2 arr;
  CGAL::insert(arr, segments);
  CGAL::insert(arr, segments + sizedof(segments) / sizedof(Segment_2));
  return 0;
}
```

---

**The Plane Sweep Algorithmic Framework**

- 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

**Map Overlay of CGAL**

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 $v_1$ and $v_2$.
2. A new vertex $v$ is induced by a vertex $v_1$ that lies on an edge $e_2$.
3. An analogous case of a vertex $v$ that lies on an edge $e_1$.
4. A new vertex $v$ is induced by a vertex $v_1$ that is contained in a face $f_2$.
5. An analogous case of a vertex $v_1$ contained in a face $f_2$.
6. A new vertex $v$ is induced by the intersection of two edges $e_1$ and $e_2$.
7. A new edge $e$ is induced by the overlap of two edges $e_1$ and $e_2$.
8. A new edge $e$ is induced by the an edge $e_1$ that is contained in a face $f_2$.
9. An analogous case of an edge $e_1$ contained in a face $f_2$.
10. A new face $f$ is induced by the overlap of two faces $f_1$ and $f_2$.
Outline

1 Definitions
2 Representation
3 Queries & Free Functions
   - Point Location Queries
   - The Zone Computation Algorithm
   - The Plane Sweep Algorithm
   - Vertical Decomposition
   - Map Overlay
4 Arrangement-Traits Classes
5 Extending the Arrangement
   - Extending Cell Records
6 Literature

Arrangement Geometry Traits

- Separates geometric aspects from topological aspects
  - 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, constructors).
- Each input curve is subdivided into x-monotone subcurves.
- Most operations involve points and x-monotone curves.

Arrangement Traits Models

- Line segments:
  - Uses the kernel point and segment types.
  - 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.

Model Name | Curve Family | Degree | Concepts
--- | --- | --- | ---
Arr_non_caching_segment_basic_traits_2 | line segments | 1 | ArrangementLandmarkTraits_2
Arr_non_caching_segment_traits_2 | line segments | 1 | ArrangementTraits_2
Arr_segment_traits_2 | line segments | 1 | ArrangementDirectionalXMonotoneTraits_2
Arr_circular_traits_2 | line segments, rays and lines | 1 | ArrangementOpenTraits_2
Arr_circular_segment_traits_2 | line segments and circular arcs | 1 | ArrangementTraits_2
Arr_rational_function_traits_2 | curves of rational functions | 1 | ArrangementOpenTraits_2
Arr_Rational_curve_traits_2 | Bézier curves | 1 | ArrangementTraits_2
Arr_algebraic_segment_traits_2 | polylines | ∞ | ArrangementTraits_2
Arr_algebraic_curve_traits_2 | algebraic curves | ≤ n | ArrangementTraits_2
Arr_algebraic_rational_traits_2 | algebraic curves | ≤ n | ArrangementTraits_2
Arr_triangulation_delaunay_traits_2 | triangulation | ≤ n+2/3 | ArrangementTraits_2
Arr_planar_map_traits_2 | planar map | ≤ n | ArrangementTraits_2
Arr_triangulation_circular_points_traits_2 | circular points | ≤ n | ArrangementTraits_2
Arr_triangulation_circular_lines_traits_2 | circular lines | ≤ n | ArrangementTraits_2
Arr_triangulation_circular_halfedges_traits_2 | circular halfedges | ≤ n | ArrangementTraits_2
Arr_triangulation_circular_refines_traits_2 | circular refinements | ≤ n | ArrangementTraits_2
Arr_triangulation_circular_points_2 | circular points | ≤ n | ArrangementTraits_2
Arr_triangulation_circular_lines_2 | circular lines | ≤ n | ArrangementTraits_2
Arr_triangulation_circular_faces_2 | circular faces | ≤ n | ArrangementTraits_2
Arr_triangulation_circular_halfedges_2 | circular halfedges | ≤ n | ArrangementTraits_2
Arr_triangulation_circular_refines_2 | circular refinements | ≤ n | ArrangementTraits_2
Arr_triangulation_hierarchy_traits_2 | triangulation hierarchy | ≤ n | ArrangementTraits_2
Arr_triangulation_delaunay_traits_2 | delaunay triangulation | ≤ n | ArrangementTraits_2
Arr_triangulation_circular_triangulation_traits_2 | circular triangulation | ≤ n | ArrangementTraits_2
Arr_triangulation_2 | triangulation | ≤ n | ArrangementTraits_2
Arr_triangulation_circular_triangulation_2 | circular triangulation | ≤ n | ArrangementTraits_2
Arr_triangulation_hierarchy_2 | triangulation hierarchy | ≤ n | ArrangementTraits_2

2D Arrangements 49
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 Records

An instance of Arr_extended_dcel<Traits, VertexData, HalfedgeData, 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.

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.

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.
Outline

1 Definitions
2 Representation
3 Queries & Free Functions
  - Point Location Queries
  - The Zone Computation Algorithm
  - The Plane Sweep Algorithm
  - Vertical Decomposition
  - Map Overlay
4 Arrangement-Traits Classes
5 Extending the Arrangement
6 Literature

Arrangement Bibliography I


Arrangement Bibliography II

Raimund Seidel


Arrangement Bibliography III


Arrangement Bibliography IV


Movies

