Applied Geometric Computing and CGAL

The Design of the Arrangement Package:
Observers, Visitors and Adaptors

Ron Wein

wein@tau.ac.il

Tel Aviv University

The Current Arrangement Hierarchy

- `Topological_map` —
  Representing and maintaining a DCEL.
- `Planar_map_2` —
  For non-intersecting x-monotone curves.
- `Planar_map_with_intersections_2` —
  For arbitrary curves, allows intersections.
- `Arrangement_2` —
  Storing a curve hierarchy, tracing the curve history.

The Planar-Map Class-Template

- Inherits from `Topological_map` and adds geometric data: vertices are associated with *points*, edges are associated with *x-monotone curves*.
- Has two template parameters:
  `Planar_map_2<Dcel,Traits>`.
- Separates the topological representation (the DCEL and the topological map) from the geometry (the *traits class*).
- Enables *point-location* (and *vertical ray-shooting*) queries, where a single point-location object is attached to the planar map at compile time.

Insertion of Curves

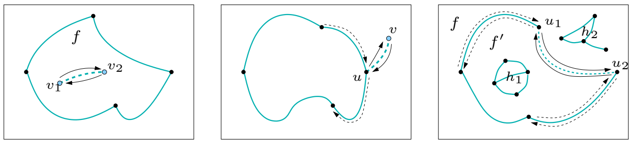
- `Planar_map_2::insert()`:
  - For a single x-monotone curve (incremental).
  - For a range of x-monotone curves (aggregated).

  The inserted curves must by interior disjoint from all existing x-monotone curves in the map.
- `Pmwx_2::insert()` (incremental and aggregated): The inserted curves may intersect one another and the existing map.
- `Pmwx_2::non_intersecting_insert()`: Same as `Planar_map_2::insert()` (faster than the general `insert()` above).

Specialized Insertion Functions

Atomic operations, used by all insertion procedures:

- `insert_in_face_interior()`,
- `insert_from_vertex()`,
- `insert_at_vertices()`.



Detecting Changes in the Planar Map (I)

- Detecting changes for maintaining auxiliary point-location structures:
  - The various point-location strategies are implemented as classes that all inherit from the same base class.
  - The base class defines some virtual functions, such as `add_edge()`, `split_edge()`, etc. and gives them empty implementations.
  - Some point-location strategies override these functions.

Detecting Changes in the Planar Map (II)

- The insertion functions have an additional parameter: pointer to a *change-notification* object (`NULL` by default).
  One can use this parameter to receive notifications about changes that are caused by the insertion operation.
- The `Arrangement_2` class uses a special change-notification class in order to maintain the curve hierarchy.

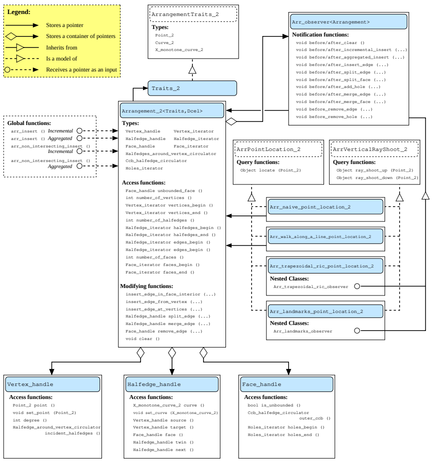
Drawbacks of the Current Package

- The DCEL template parameter is confusing.
- $1:1$ relation between planar maps and point-location objects.
- Counter-intuitive interface for the insertion functions.
- Duplicated notification mechanism.
- Cumbersome arrangement class.

Guidelines for the New Design

- Separate the arrangement representation from the various algorithms that relate to it:
  Insertion, point location, overlay, etc.
- Supply a default DCEL parameter, so the DCEL representation is not exposed if not necessary.
- Collapse the class hierarchy
  (a single `Arrangement_2` class).
- $1:n$ relation between arrangements and point-location objects (n may be 0).
- Unified notification mechanism.
- Use meta-traits classes to maintain extra (non-geometric) data.

The New Arrangement Package



The Notification Mechanism

- The notification mechanism enables external classes to be informed on changes that occur in the arrangement:
  - Before and after *global operations*: clear, insertion of a single curve, aggregated insertion of a range of curves.
  - Before and after *local topological changes*: add edge, split edge, split face, merge face, add hole, remove edge, etc.
- It is possible to attach several observers to each arrangement object.

Point-Location Strategies

- Separate the two concepts: *point location* and *vertical ray shooting*. Actually all classes we supply are models of both concepts.
- Simple strategies (naïve, walk along a line) just store a pointer to an arrangement object.
- Strategies that need extra data structures (trapezoidal RIC, landmarks) use the notification mechanism to keep their structures updated.

Insertion Functions

- Insertion is performed via global functions that receive and arrangement object and a curve (or a range of curves):
  - `arr_non_intersecting_insert()` for x-monotone non-intersecting curves, and `arr_insert()` for arbitrary curves.
  - We support the incremental insertion operation (requires a point-location object) and the aggregated insertion operation (uses sweep-line).
- It is now possible to construct an arrangement with no point-location object attached to it.

Meta-Traits Classes

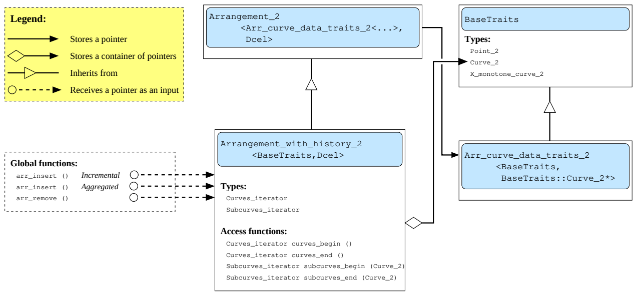
- The traits class is responsible to extra (non-geometric) data attached to points and to curves.
- Meta-traits classes automatically handle extra data attached to arrangement vertices and edges.
- Maintaining data with arrangement faces is the user's responsibility. This can be done by extending the DCEL.

The Curve-Data Traits

The `Arr_curve_data_traits_2<BaseTraits,Data>` class-template is already included in CGAL 3.1:

- The `Curve_2` and `X_monotone_curve_2` types are inherited from the base-traits class types, with a `Data` object attached to them.
- The class relies on the base traits for all the geometric operations.
- The functions that construct x-monotone curve (namely `curve_make_x_monotone()`, `curve_split()` and the intersection functions) are overridden.

Maintaining the Curve History

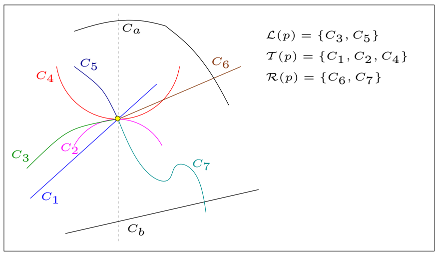


The Sweep-Line Paradigm

- Introduced by Bentley and Ottmann (1979) for reporting all intersections among a set of line segments.
- Can deal with degeneracies.
- Can be extended to arbitrary x-monotone curves.
- Useful for many other tasks:
  - Computing the subdivision to sub-curves,
  - constructing an arrangement,
  - computing overlays, etc.

Sketch of the Sweep-Line Algorithm

- Initialize the *X-structure* by the endpoints of all curves.
- Initialize an empty *Y-structure*.
- While the X-structure is not empty, pop the next event and handle it — update the X and Y-structures accordingly.



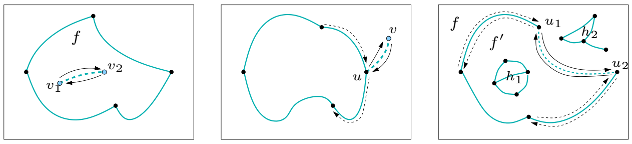
The New Sweep-Line Class-Template

We introduce the class template
`Sweep_line_2<Traits,Event,Subcurve,Visitor>`:

- It uses two auxiliary data types:
  `Event_base` (stored in the X-structure) and
  `Subcurve_base` (stored in the Y-structure).
- Users may extend these type by inheriting types from them and instantiating the respective template parameters.
- The `Visitor` class should supply the two following functions:
  - `void before_handle_event` $(E_p, \mathcal{L}(p), \mathcal{T}(p), \mathcal{R}(p));$
  - `bool after_handle_event` $(E_p, \mathcal{L}'(p), \mathcal{R}'(p), C_a, C_b);$

Sweep-Line Visitors

- Trivial visitors:
  reporting intersection points, reporting sub-curves.
- A visitor class that uses the specialized insertion functions to construct a DCEL representing the arrangement of the input set of curves.



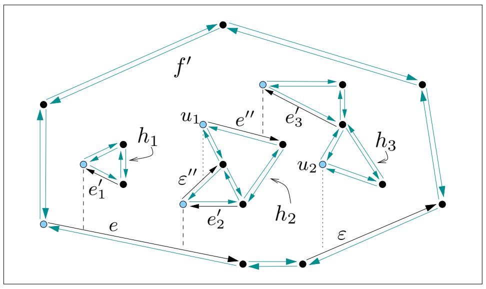
in face interior *from vertex* *at vertices*

Visitor for DCEL Construction

- Extends the `Event` class to contain:
  - **v** a pointer to a corresponding DCEL vertex (may be null).
  - **n** number of unhandled subcurves, with a list **L** of pointers to them, sorted by their increasing y order immediately to the right of the event point.
- Extends the `Subcurve` class to store a pointer to an event object, associated with the left endpoint of the Subcurve.
- In `before_handle_event()` the extra data fields of the events are initialized.
- In `after_handle_event()` it is possible to insert all subcurves of $\mathcal{L}'(p)$.

Relocating Holes inside New Faces

Yet another task our visitor has to deal with ...



Additional Sweep-Line Visitors

- Simultaneous point location of multiple queries.
- Inserting a range of curves into an existing arrangement.
- Constructing the *vertical decomposition* of an arrangement.
- Computing the *overlay* of two arrangements, represented as two DCELs.