Advanced Programming Techniques
Applied to:

\textsc{Cgal}'s Arrangement Package

Ron Wein, Efi Fogel, Baruch Zukerman, and Dan Halperin
Tel Aviv Univ.

LCSD, San Diego
October 16, 2005
Overview

- Related Work
- CGAL
- Arrangements
- Adapters
- Decorators
- Observers
- Visitors
- Video
Overview

- Related Work
- CGAL
- Arrangements
- Adapters
- Decorators
- Observers
- Visitors
- Video
Related Work

- Keyser et al. implemented a module that constructs arrangements of algebraic curves constrained to some general-position assumptions.
- The LEDA library contains a module that constructs and maintains arrangements of line segments.
- CGAL’s arrangement package constructs and maintains arrangements of arbitrary curves and supports operations and queries on such arrangements:
  - All inputs are handled correctly (including degenerate).
  - Exact number types are used to achieve exact results.
Overview

- Related Work
- **Cgal**
- Arrangements
- Adapters
- Decorators
- Observers
- Visitors
- Video
CGAL

Written in C++
Follows the *generic programming* paradigm
Development started in 1995
Consortium of 6 active European sites:

1. Utrecht University
2. INRIA Sophia Antipolis
3. ETH Zürich
4. MPII Saarbrücken
5. Tel Aviv University
6. Freie Universität Berlin
7. RISC Linz
8. Martin-Luther-Universität Halle
**CGAL Structure**

- **Basic Library**
  - Algorithms and Data Structures

- **Kernel**
  - Geometric Objects of constant size
  - Geometric Operations on objects of constant size

- **Support Library**
  - configurations, assertions, ...

- **Visualization**
  - File
  - I/O
  - Number Types
  - Generators
  - ...

**Advanced Programming Techniques**
Overview

- Related Work
- **CGAL**
- **Arrangements**
- Adapters
- Decorators
- Observers
- Visitors
- Video
Arrangements

Given a collection $\Gamma$ of planar curves, the arrangement $A(\Gamma)$ is the partition of the plane into vertices, edges and faces induced by the curves of $\Gamma$.

An arrangement of lines  
An arrangement of circles
Arrangement Applications

Arrangements have numerous applications

robot motion planning, computer vision, GIS, optimization, molecular biology, etc.

Path Verification for NC-Machining

Hybrid Motion Planning

2D Minkowski Sums

3D Minkowski Sums

A planar map of the Boston area

Maximizing the Area of an Axis-Symmetric Polygon

Inner-cover of Non-convex Shapes
Arrangement Goals

- Construct, maintain, modify, traverse, query and present subdivisions of the plane
- Robust, handles all degeneracies, exact
- Efficient
- **Generic**, easy to interface, extend, and adapt
- Part of the Cgal basic library
Arrangement Traits

- Separates geometric aspects from topological aspects
  - Arrangement_2<Traits,Dcel> — main component
- Parameter of package
  - Defines the family of curves of interest
  - Package 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)
Arrangement Traits Requirements

- Types: Curve_2, X_monotone_curve_2, Point_2
- Methods:
  1. Compare two points
Arrangement Traits Requirements

- **Types:** Curve_2, X_monotone_curve_2, Point_2

- **Methods:**
  1. Compare two points
  2. Determine the relative position of a point and an \(x\)-monotone curve
  3. Determine the relative position of two \(x\)-monotone curves to the left (or right) of a point
  4. Subdivide a curve into \(x\)-monotone curves
  5. Find all intersections of two \(x\)-monotone curves
Arrangement Architecture

Arrangement_2<Traits,Dcel>

Arr_default_dcel

ArrangementBasicTraits_2

ArrangementTraits_2

Arr_non_caching_segment_traits_2

Arr_segment_traits_2

Arr_polyline_traits_2

Arr_conic_traits_2

Inheritance

refinement

A model of

ArrangementDcel

pointers
Overview

- Related Work
- C\textsc{g}\textsc{a}l
- Arrangements
- Adapters
- Decorators
- Observers
- Visitors
- Video
Adapters

The **adapter** design-pattern “*converts the interface of a class into another interface clients expect. Adapters let classes work together that could not otherwise, because of incompatible interfaces.*” (Gamma *et al.*)
Adapters

The adapter design-pattern “converts the interface of a class into another interface clients expect. Adapters let classes work together that could not otherwise, because of incompatible interfaces.” (Gamma et al.)

Adapters that appear in the arrangement package:

- The traits adapter
- The DCEL face extender
- Boost graph adapters
Boost Graph Adapters

- Boost Graph Library (BGL) supports graph algorithms
- Arrangements are embedded in the plane as planar graphs
  - Extend arrangements with the interface BGL expects

Primal adapter

Dual adapter
Boost Graph Adapters

- Boost Graph Library (BGL) supports graph algorithms
- Arrangements are embedded in the plane as planar graphs

\[\text{Extend arrangements with the interface BGL expects}\]

![Primal adapter](image1)
![Dual adapter](image2)

Primal adapter
Dual adapter
Boost Graph Adapters

- Boost Graph Library (BGL) supports graph algorithms
- Arrangements are embedded in the plane as planar graphs
  - Extend arrangements with the interface BGL expects

Primal adapter

Dual adapter
Overview

- Related Work
- CGAL
- Arrangements
- Adapters
- Decorators
- Observers
- Visitors
- Video
Decorators

The *decorator* design-pattern “attaches additional responsibilities to an object dynamically. Decorators provide a flexible alternative to sub-classing for extending functionality.” (Gamma *et al.*)
Decorators

The decorator design-pattern “attaches additional responsibilities to an object dynamically. Decorators provide a flexible alternative to sub-classing for extending functionality.” (Gamma et al.)

Decorators that appear in the arrangement package:

- Model the ArrangementTraits_2 concept
  - Merged curve-data traits — extends the \(x\)-monotone curve-type with a single data field
  - Consolidated curve-data traits — extends the \(x\)-monotone curve-type with a container of data fields
  - Used to implement the Arrangement_with_history_2
    Maintains its construction history
Arrangement with History: the Netherlands

Roads, railroads, rivers and water canals on the map of the Netherlands. Bridges are marked by circles.

- border — 13 polylines, 373 segments
- roads — 877 polylines, 2360 segments
- water routes — 88 polylines, 1397 segments
Overview

- Related Work
- CGAL
- Arrangements
- Adapters
- Decorators
- Observers
- Visitors
- Video
Observers

The observer design-pattern “defines a one-to-many dependency between objects, so that when one object changes state, all its dependents are notified and updated automatically.” (Gamma et al.)
Observers

The observer design-pattern “defines a one-to-many dependency between objects, so that when one object changes state, all its dependents are notified and updated automatically.” (Gamma et al.)

Observers used by the Arrangement package:

- Point-location strategies that maintain auxiliary data
  - Landmark point-location
  - Trapezoidal decomposition based point-location
- Dynamic maintenance of attributes stored at the arrangement features
Notification Mechanism

- Arrangement_2<Traits,Dcel>
- Arr_naive_point_location
- Arr_walk_along_line_point_location
- Arr_trapezoidal_ric_point_location
- Arr_landmarks_point_location
- Arr_trapezoidal_ric_observer
- Arr_landmarks_observer
- Arr_observer<Arrangement>
- ArrangementPointLocation_2
Overview

- Related Work
- CGAL
- Arrangements
- Adapters
- Decorators
- Observers
- Visitors
- Video
Visitors

The **visitor** design-pattern “represents an operation to be performed on an object or on the elements of an object structure. Visitors allow the definition of new operations without changing the classes of the elements on which they operate.” (Gamma *et al.*)
Visitors

The *visitor* design-pattern “*represents an operation to be performed on an object or on the elements of an object structure. Visitors allow the definition of new operations without changing the classes of the elements on which they operate.*” (Gamma *et al.*)

Implementing an algorithm based on a framework below reduces to implementing an appropriate visitor class

- (Vertical) line sweep of the plane — given a set of curves, identify all endpoints and intersection points
- Zone computation of a curve — given an arrangement and an $x$-monotone curve, identify all arrangement cells that the curve crosses
Sweep Line
Sweep Line
Sweep Line
Sweep Line
Sweep Line
Free Functions

- `insert (arr, begin, end);`
- `overlay (arr1, arr2, res_arr);`
- `Arrangement_zone_2<Arrangement,Visitor>`
- `Arrangement_zone_2<Traits,Event,Arr_insertion_visitor>`
- `Arrangement_zone_2<Traits,Dcel>`
- `ArrangementZoneVisitor_2`
- `Arr_inc_insert_zone_visitor`
- `Sweep_line_2<Traits,Event,Subcurve,Visitor>`
- `SweepLineVisitor_2`
- `Arr_insertion_visitor`
- `Arr_overlay_visitor`
- `insert (arr, cv);` (incremental insertion)
- `insert (arr, begin, end);` (aggregated insertion)
- `overlay (arr1, arr2, res_arr);`
Overview

- Related Work
- CGAL
- Arrangements
- Adapters
- Decorators
- Observers
- Visitors
- Video
Video

http://www.cs.tau.ac.il/~efif/CD/movies/Mink3d.mov
TOC

Overview ✤
Overview ✤
Related Work ✤
Overview ✤
CGAL ✤
CGAL Structure ✤
Overview ✤
Arrangements ✤
Arrangement Applications ✤
Arrangement Goals ✤
Arrangement Traits ✤
Arrangement Traits Requirements ✤
Arrangement Architecture ✤
Overview ✤

Adapters ✤
Boost Graph Adapters ✤
Boost Graph Adapters ✤
Overview ✤
Decorators ✤
Arrangement with History: the Netherlands ✤
Overview ✤
Observers ✤
Notification Mechanism ✤
Overview ✤
Visitors ✤
Sweep Line ✤
Sweep Line ✤