Specification of the traits classes for CGAL arrangements of curves

Efi Fogel, Dan Halperin, Ron Wein                Monique Teillaud
Tel Aviv University                             INRIA Sophia-Antipolis

Eric Berberich, Arno Eigenwillig, Susan Hert, Lutz Kettner
Max-Planck Institut für Informatik, Saarbrücken

May 4, 2003

Abstract

This document describes the specification of the new and improved traits classes for CGAL arrangements of curves. It is the product of collaboration between TAU in Israel, INRIA in France, and MPI in Germany. Together we were able to tighten the set of requirements of the arrangement traits, that is the interface between arrangements and the primitives they use. The dialog also emphasized the need for a redesign and a new implementation of the sweep-line module. This has been successfully completed recently.

1 Introduction

During the last year the Computational Geometry Lab at the Tel-Aviv University was involved in several mini-projects where different parties form various institutions across Europe collaborated to achieve a few mutual goals. The Computational Geometry Lab is the author and the maintainer of, among others, the arrangement packages of CGAL, the Computational Geometry Algorithms Library. The development of these packages is an ongoing activity. Different parts of these packages are constantly improved. Aside of daily maintenance that consists of bug fixing, documentation enhancement, etc., entire modules are rewritten and replaced and new modules are introduced. The process of detecting weak spots, inefficiencies, and missing functionalities followed by the rectification of the problems, purification of the code, and the introduction of new and innovative techniques that in some cases requires redesign of internal parts and establishment of new user interfaces was boosted due to the successful cooperation.

2 Preliminaries

CGAL arrangement packages consist of a hierarchy of three objects. At the bottom, the Planar_map_2 object represents the subdivision of the plane induced by a collection of \(x\)-monotone curves that
are mutually disjoint in their interiors, but may have common endpoints. Higher in the hierarchy, the Planar_map_with_intersections_2 object, derived from the former, handles curves that are not necessarily $x$-monotone and may intersect each other. Such curves are split into several $x$-monotone subcurves if necessary, before they are inserted into the planar map. At the top, the Arrangement_2 object maintains the hierarchical history for each subcurve in the planar map.

The arrangement classes are parameterized with corresponding traits-interface classes that define the abstract interface between the arrangement object and the primitives it uses. The concept of traits for Arrangement_2 is a refinement of the concept of traits for Planar_map_with_intersections_2. The latter refines the concept of traits for Planar_map_2 in turn.

The arrangement packages are charged with default traits classes implemented in exact arithmetic for various curve-type primitives such as segments, polylines, and circle and conic arcs. The CGAL kernel encapsulates representations of constant-size non-modifiable geometric primitive object such as points and segments, and operations (predicates and constructors) on these objects. It is adaptable and extensible, it is efficient, and is suitable to be used as a traits class for algorithms. In fact, the traits classes are parameterized by a kernel representation.

Any arrangement supports incremental and aggregate constructions. The latter is the most efficient method to construct a planar map, as it is based on an efficient and optimized sweep-line algorithm. It is suitable, however, only in cases where all the curves are known in advance. The nature of the sweep-line algorithm, sweeping a line in the plane in a given direction, relaxes the requirements of the traits classes. This induces a mutual exclusive refinement relation between the traits used for aggregate construction and the traits used for incremental construction. Notice, however, that this subtle observation has little practical affect, as any point-location query applied to a constructed arrangement implies the full set of requirements.

3 Specification

The following three subsections specify the three new and improved interface traits-classes and their interrelations in details. The set of requirements was made sound and complete. A couple of requirements were eliminated, few others were redefined, and some were renamed. The hierarchy of three traits classes was established to include only the necessary requirements at each level. It was determined that for the aggregate insertion-operation based on a sweep-line algorithm only a subset of the requirements is needed. Preconditions were added where appropriate to tighten the requirements further.

Concept PlanarMapTraits_2

Definition

A model of the PlanarMapTraits_2 concept aggregates the geometric types and primitive operations used by the data structure Planar_map_2<Deel,Traits>. It must provide the types and operations listed
Types

The geometric types defined below must have a default constructor, copy constructor, and assignment operator.

PlanarMapTraits_2:: X_monotone_curve_2

A type that holds an \( x \)-monotone curve in the plane.

PlanarMapTraits_2:: Point_2

A type that holds the position of a vertex in the plane. The type of the end points of \( X_{\text{monotone\_curve\_2}} \) curves.

The following methods that have a curve parameter of type \( X_{\text{monotone\_curve\_2}} \) have the implicit precondition that requires the curve to be \( x \)-monotone.

Enumerations

enum Comparison_result \{ SMALLER, EQUAL, LARGER \}:

\[
\text{a constant describing the relative position between objects.}
\]

Creation

Only a default constructor is required. Note that further constructors can be provided.

PlanarMapTraits_2::

A default constructor.

Operations

Comparison_result

\[
\text{pm\_traits\_compare\_x( Point\_2 p0, Point\_2 p1)}
\]

\[
\text{compares the } x\text{-coordinates of } p0 \text{ and } p1. \text{ Returns } LARGER \text{ if } x(p0) > x(p1); \text{ SMALLER if } x(p0) < x(p1); \text{ EQUAL otherwise.}
\]
Comparison_result

\texttt{pm\_traits.compare\_xy( Point\_2 p0, Point\_2 p1)}

compares lexicographically the two points \( p0 \) and \( p1 \). First the \( x \)-coordinates are compared. In case of a tie, the \( y \)-coordinates are compared. Returns \textsc{LARGER} if \( x(p1) > x(p2) \), or if \( x(p1) = x(p2) \) and \( y(p1) > y(p2) \); \textsc{smaller} if \( x(p1) < x(p2) \), or if \( x(p1) = x(p2) \) and \( y(p1) < y(p2) \); \textsc{equal} otherwise.

\textbf{bool} \quad \texttt{pm\_traits.curve\_is\_vertical( X\_monotone\_curve\_2 cv)}

returns \texttt{true} if \( cv \) is a vertical segment, \texttt{false} otherwise.

\textbf{bool} \quad \texttt{pm\_traits.point\_is\_in\_x\_range( X\_monotone\_curve\_2 cv, Point\_2 pnt)}

returns \texttt{true} if \( pnt \) is in the \( x \)-range of \( cv \), \texttt{false} otherwise.

Comparison_result

\texttt{pm\_traits.curves\_compare\_y\_at\_x( X\_monotone\_curve\_2 cv1,}
\texttt{ X\_monotone\_curve\_2 cv2,}
\texttt{ Point\_2 pnt)}

compares the \( y \)-coordinate of \( cv1 \) and \( cv2 \) at the \( x \)-coordinate of \( pnt \). Returns \textsc{LARGER} if \( cv1(x(q)) > cv2(x(q)) \); \textsc{smaller} if \( cv1(x(q)) < cv2(x(q)) \); \textsc{equal} otherwise. 
\textbf{Precondition}: \( cv1 \) and \( cv2 \) are defined at \( pnt \)'s \( x \)-coordinate.

Comparison_result

\texttt{pm\_traits.curves\_compare\_y\_at\_x\_to\_right( X\_monotone\_curve\_2 cv1,}
\texttt{ X\_monotone\_curve\_2 cv2,}
\texttt{ Point\_2 pnt)}

compares the \( y \)-coordinate of \( cv1 \) and \( cv2 \) immediately to the right of the \( x \)-coordinate of \( pnt \).
\textbf{Precondition}: \( cv1 \) and \( cv2 \) meet at \( pnt \) \( x \)-coordinate.
\textbf{Precondition}: \( cv1 \) and \( cv2 \) are defined to the right of \( pnt \)'s \( x \)-coordinate.
Comparison_result

\texttt{pm_traits.curves\_compare\_y\_at\_x\_to\_left( X\_monotone\_curve\_2 \ cv1, X\_monotone\_curve\_2 \ cv2, Point\_2 \ pnt)}

compares the \( y \)-coordinate of \( \cv1 \) and \( \cv2 \) immediately to the left of the \( x \)-coordinate of \( \pnt \). This predicate is not required for the aggregate insertion of curves into the planar map. Recall, that the aggregate insertion is based on a sweep-line algorithm.

\textbf{Precondition}: \( \cv1 \) and \( \cv2 \) meet at \( \pnt \) \( x \)-coordinate.

\textbf{Precondition}: \( \cv1 \) and \( \cv2 \) are defined to the left of \( \pnt \)'s \( x \)-coordinate.

Comparison_result

\texttt{pm_traits.curve\_compare\_y\_at\_x( X\_monotone\_curve\_2 \ cv, Point\_2 \ pnt)}

compares the \( y \)-coordinates of \( \pnt \) and the vertical projection of \( \pnt \) on \( \cv \). Returns \texttt{SMALLER} if \( \cv(x(p)) < y(p) \); \texttt{LARGER} if \( \cv(x(p)) > y(p) \); \texttt{EQUAL} otherwise (\( p \) is on the curve).

\textbf{Precondition}: \( \cv \) is defined at \( \pnt \)'s \( x \)-coordinate.

\textbf{bool} \texttt{pm_traits.curve\_is\_equal( X\_monotone\_curve\_2 \ cv1, X\_monotone\_curve\_2 \ cv2)}

returns \texttt{true} if \( \cv1 \) and \( \cv2 \) have the same graph, \texttt{false} otherwise.

\textbf{bool} \texttt{pm_traits.point\_is\_equal( Point\_2 \ p1, Point\_2 \ p2)}

returns \texttt{true} if \( \p1 \) is the same as \( \p2 \), \texttt{false} otherwise.

\textbf{Point\_2} \texttt{pm_traits.curve\_source( X\_monotone\_curve\_2 \ cv)}

returns the source of \( \cv \).

\textbf{Point\_2} \texttt{pm_traits.curve\_target( X\_monotone\_curve\_2 \ cv)}

returns the target of \( \cv \).
Has Models

\texttt{Pm\_segment\_traits\_2<Kernel>}

Concept \texttt{PlanarMapWithIntersectionsTraits\_2}

Definition

A model of the \texttt{PlanarMapWithIntersectionsTraits\_2} concept aggregates the geometric types and primitive operations used by the data structure \texttt{Planar\_map\_with\_intersections\_2<Dcel,Traits>}. Note that the concept \texttt{PlanarMapWithIntersectionsTraits\_2} refines the concept \texttt{PlanarMapTraits\_2} and inherits all its types and operations.

In addition to the requirements of the \texttt{PlanarMapTraits\_2} concept, it must provide the types and operations listed below.

Refines

\texttt{PlanarMapTraits\_2}

Types

The geometric types defined below must have a default constructor, copy constructor, and assignment operator.

\texttt{PlanarMapWithIntersectionsTraits\_2:: Curve\_2}

A type that holds a general curve in the plane. Its end points must be of type \texttt{Point\_2}. Curves of type either \texttt{X\_monotone\_curve\_2} or \texttt{Curve\_2} can be inserted into a \texttt{Planar\_map\_with\_intersections\_2<Dcel,Traits>} object.

Operations

\begin{verbatim}
bool pmwx_traits.curves_do_overlap( X_monotone_curve_2 cv1,
                                    X_monotone_curve_2 cv2)

    returns true if cv1 and cv2 overlap in a one-dimensional subcurve (i.e., not in a finite number of points), false otherwise.
\end{verbatim}
template<class OutputIterator>
OutputIterator pmwx_traits.make_x_monotone(Curve_2 cv, OutputIterator res)
cuts cv into x-monotone subcurves and stores them in a sequence starting at res. The order in which they are stored defines their order in the hierarchy tree. Returns past-the-end iterator of the sequence.

void pmwx_traits.curve_split(X_monotone_curve_2 cv, X_monotone_curve_2& c1, X_monotone_curve_2& c2, Point_2 split_pt)
splits cv at split_pt into two curves, and assigns them to c1 and c2 respectively.
Precondition: split_pt is on cv but is not one of its endpoints.

bool pmwx_traits.nearest_intersection_to_right(X_monotone_curve c1, X_monotone_curve c2, Point_2 pt, Point_2& p1, Point_2& p2)
finds the nearest intersection point (or points) of c1 and c2 lexicographically to the right of pt not including pt itself, (with one exception explained below). If the intersection of c1 and c2 is an X_monotone_curve_2, that is, they overlap at infinitely many points, then if the right endpoint and the left endpoint of the overlapping subcurve are strictly to the right of pt, they are returned through the two point references p1 and p2 respectively. If pt is between the overlapping-subcurve endpoints, or pt is its left endpoint, pt and the right endpoint of the subcurve are returned through p1 and p2 respectively. If the intersection of the two curves is a point to the right of pt, it is returned through p1 and p2. Returns true if c1 and c2 do intersect to the right of pt, false otherwise.

bool pmwx_traits.nearest_intersection_to_left(X_monotone_curve c1,
Monotone curve $c_2$, 
Point_2 $pt$, 
Point_2 $p_1$, 
Point_2 $p_2$)
finds the nearest intersection point (or points) of $c_1$ and $c_2$ lexicographically to the left of $pt$ not including $pt$ itself, (with one exception explained below). If the intersection of $c_1$ and $c_2$ is an $X_{\text{monotone\_curve}}_2$, that is, they overlap at infinitely many points, then if the left endpoint and the left endpoint of the overlapping subcurve are strictly to the left of $pt$, they are returned through the two point references $p_1$ and $p_2$ respectively. If $pt$ is between the overlapping-subcurve endpoints, or $pt$ is its left endpoint, $pt$ and the left endpoint of the subcurve are returned through $p_1$ and $p_2$ respectively. If the intersection of the two curves is a point to the left of $pt$, it is returned through $p_1$ and $p_2$. Returns true if $c_1$ and $c_2$ do intersect to the left of $pt$, false otherwise. This constructor is not required for the aggregate insertion of curves into the planar map. Recall, that the aggregate insertion is based on a sweep-line algorithm.

Has Models

$\text{Arr\_segment\_traits\_2}\langle\text{Kernel}\rangle$
$\text{Arr\_segment\_cached\_traits\_2}\langle\text{Kernel}\rangle$
$\text{Arr\_conic\_traits\_2}\langle\text{Kernel}\rangle$
$\text{Arr\_polyline\_traits\_2}\langle\text{Kernel,Container}\rangle$
$\text{Arr\_segment\_circle\_traits\_2}\langle\text{NT}\rangle$

Concept ArrangementTraits_2

Definition

A model of the ArrangementTraits_2 concept aggregates the geometric types and primitive operations used by the data structure $\text{Arrangement\_2}\langle\text{Dcel,Traits}\rangle$.

Note that the concept ArrangementTraits_2 refines the concept $\text{PlanarMapWithIntersectionTraits\_2}$ and inherits all its types and operations.
In addition to the requirements of the `PlanarMapWithIntersectionTraits_2` concept, it must provide the operation listed below.

**Refines**

`PlanarMapWithIntersectionTraits_2`

**Creation**

`ArrangementTraits_2`;

A default constructor.

**Operations**

`X_monotone_curve_2`

\[
\text{arr.traits.curve.flip( } X_{\text{monotone\_curve\_2}} \text{ cv)}
\]

flip the curve \( cv \). This constructor is not required for the aggregate insertion of curves into the planar map. Recall, that the aggregate insertion is based on a sweep-line algorithm.