# Reference Manual for Object-Oriented Data Structure for the Rectangle Intersection Problem

Vikas Kapoor [*] Alexander Wolff [*]

Revision: 1.1, Date: 1998/01/22

[*]Institut für Informatik, Fachbereich Mathematik und Informatik, Freie Universität Berlin, Takustraße 9, 14195 Berlin, Germany, {kapoor | awolff}@inf.fu-berlin.de

# Introduction

An algorithm for the rectangle intersection problem can be sketched as follows: initialize the interval tree with the intervals (on the y-axis) to which the left sides of the rectangles correspond. Then sweep horizontally across the plane. When the sweep line hits a rectangle, query with the corresponding interval and insert it into the interval tree. An interval is deleted when the sweep line leaves the corresponding rectangle.

As usual, the sweep line is supported by two data structures, the event-point schedule and the sweep-line status.

Event points are the x-values where the sweep line must stop because either its status changes or intersections have to be reported. In our case all event points are known before the sweep begins: they are the x-values of left and right edges of rectangles. It suffices to use a simple list of event points sorted in ascending x-order to implement our event-point schedule. An event point must be stored in such a way that it is clear whether it refers to a left or right edge of a rectangle.

The sweep-line status stores intervals corresponding to the intersections of the sweep line with rectangles. Initially the sweep-line status is empty. When a left edge of a rectangle is encountered during the sweep it is inserted into the sweep-line status. All rectangles whose edges are present in the sweep-line status and intersect the interval represented by this edge are reported. When a right edge of a rectangle is encountered, the left edge is deleted from the sweep-line status.

This reduces the rectangle intersection problem to the problem of maintaining a set of intervals such that intervals can be efficiently inserted and deleted, and queries asking for all intervals that intersect a given interval can be answered quickly. To achieve this, we implement our sweep-line status with the interval tree data structure. This data structure allows inserting and deleting in $\mathcal{O}(\log n)$ time and reporting all intervals that intersect a given interval in $\mathcal{O}(k + \log n)$ time, where $k$ is the number of intersections. The space requirement for the interval tree is linear.

# Chapter 1

# Rectangle Intersection

### Definition

Class *OORecInters<CoordType>* reports intersections among input rectangles.

*#include <oo_rec_inters.h>*

### Template Parameters

The class *OORecInters<CoordType>* is templated with the type of the cornerpoints of the input rectangles.

### Types

*struct Rectangle<T> { T left, right;};*

Type of the input rectangles.

*OORecInters<CoordType>:: CoordType value_type;*

Type of the coordinates of the cornerpoints of input rectangles.

*OORecInters<CoordType>:: list<Rectangle<CoordType> > Rectangle_ls;*

Type of the list used to store the input rectangles.

*OORecInters<CoordType>:: Rectangle_ls::Rectangle_ls_it;*

Type of the iterator used to traverse the input rectangles.

*OORecInters<CoordType>:: Ugraph<Rectangle_ls_it, int> Graph;*

Type of the graph used to report intersections among the input rectangles.

#### Creation

*OORecInters<CoordType> rec_inters( Rectangle_ls& rectangles);*

creates the object *rec_inters* of type *OORecInters<CoordType>*.

#### Access Functions

| | | |
|---|---|---|
| *Graph** | *rec_inters.run()* | returns the conflict graph for the input rectangles. |
| *int* | *rec_inters.num_of_conflicts()* | returns the total number of intersections. |

# Chapter 2

# Example

The following example illustrates the use of interval tree.

```
#if (__SUNPRO_CC)
#define typename
#endif

#include <iostream.h>
#include <fstream.h>
#include <vector.h>
#include <oo_rec_inters.h>

/* input operator for a rectangle: */
istream &operator>> (istream &in, Rectangle<double>& rec)
{
    in >> rec.llx >> rec.lly >> rec.urx >> rec.ury;
    return in;
}

/* output operator for a rectangle: */
ostream &operator<< (ostream &out, Rectangle<double> const &rec)
{
    out << "(" << rec.llx << ", " << rec.lly << ") === (";
    out << rec.urx << ", " << rec.ury << ")";
    return out;
}

main(int argc, char *argv[])
{
    /* type definitions */
    typedef OORecInters<double> Rec_algo;
    typedef typename Rec_algo::Rectangle_ls Rec_ls;
    typedef typename Rec_algo::Rectangle_ls_it Rec_it;
    typedef typename Rec_algo::Graph My_graph;
    typedef typename My_graph::Node_it Node_it;
    typedef typename My_graph::Adj_edge_it Adj_edge_it;

    cout << "This program detects pairs of properly intersecting rectangles."
```

```
   << endl << endl;

  /* read the input rectangles */
  Rec_ls rectangles;
  cout << "Please enter the coordinates of lower left and upper right"
       << endl << "corner of the rectangles (ll x, ll y, ur x, ur y)."
       << endl << endl;
  Rectangle<double> rectangle;
  while (cin >> rectangle)
      rectangles.push_back(rectangle);

  /* initialize the algorithm */
  cout << endl << "initializing..." << endl;
  Rec_algo my_recalgo(rectangles);

  /* the sweep */
  cout << "sweeping..." << endl;
  My_graph* graph = my_recalgo.run();
  cout <<"No. of nodes: " << graph->num_of_nodes() << endl
   <<"No. of edges: " << graph->num_of_edges() << endl;
  Node_it node_current = graph->nodes_begin();
  Node_it node_end = graph->nodes_end();
  for (int i = 0; node_current != node_end; ++node_current, i++)
  {
  cout << "[" << i << ", " << (**node_current) << "]:";
  Adj_edge_it edge_current = graph->adj_edges_begin(node_current);
  Adj_edge_it edge_end = graph->adj_edges_end(node_current);
  for ( ; edge_current != edge_end; ++edge_current)
    cout << "[" << **graph->opposite(node_current,
                    graph->get_edge_it(edge_current))
        << "]";
  cout << endl;
 }
 cout << "done." << endl;
}
```