6th International Conference on Applied Informatics
Eger, Hungary, January 27–31, 2004.

Planar union of rectangles with sides parallel to the coordinate axis

Daniel Schmid, Hanspeter Bopp

Department of Geomatics, Computer Science and Mathematics
Stuttgart University of Applied Sciences
e-mail: daniel.schmid@hft-stuttgart.de, hanspeter.bopp@hft-stuttgart.de

Abstract

Computational Geometry is a relatively new subject of Computer Science (since c. 1975), which is concerned with the development of efficient algorithms and data structures for geometric problems. For example, the calculation of Convex Hulls or Voronoi-Diagrams for given sets of points are typical problems of Computational Geometry.

The topic of this paper is the examination of the so called *Rectangle Union*, that means given is a set of rectangles with sides parallel to the coordinate axis and searched are the measure and the boundary lines of the union of the inside areas of the given rectangles.

First, this problem will be defined and some examples will be illustrated. After that, algorithms and data structures for the calculation of the searched values will be discussed and evaluated. In doing so, the *plane-sweep technique* as a basic principle of Computational Geometry will be applied.

Categories and Subject Descriptors: I.3.5 [Computer Graphics]: Computational Geometry and Object Modelling

Key Words and Phrases: O-Notation, plane-sweep technique

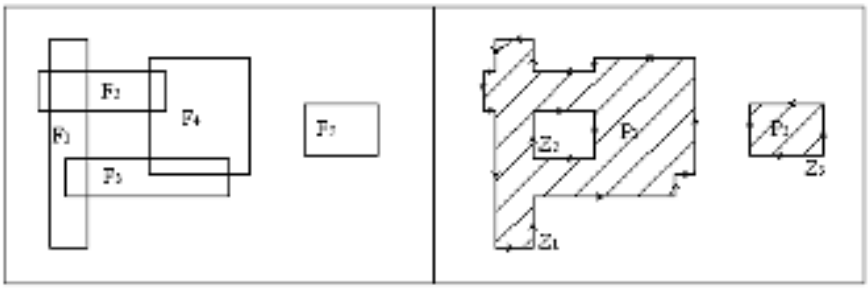
1. Introduction

Given is a finite set of n not degenerated rectangles with sides parallel to the coordinate axis in the plane.

Searched are the measure and the boundary lines of the *Rectangle Union* related to the given n rectangles.

Definition 1.1. The **Rectangle Union** F is an area which is defined by the union of the inside areas F_i of the given rectangles, that means $F = \bigcup_{i=1}^{n} F_i$.

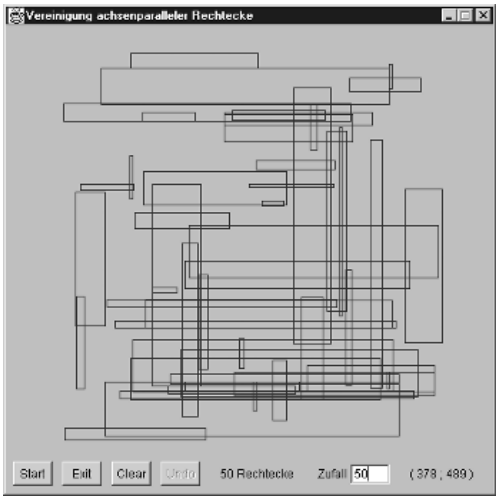
Example 1.1. Here is $n = 5$. The Rectangle Union consists of two disjoint parts, that means $F = P_1 \cup P_2$. There are two outer boundary lines (Z_1 and Z_3) and one inner boundary line (Z_2), altogether 24 boundary edges. The inner contour line Z_2 encloses a whole inside P_1 and is orientated in a mathematically negative manner.



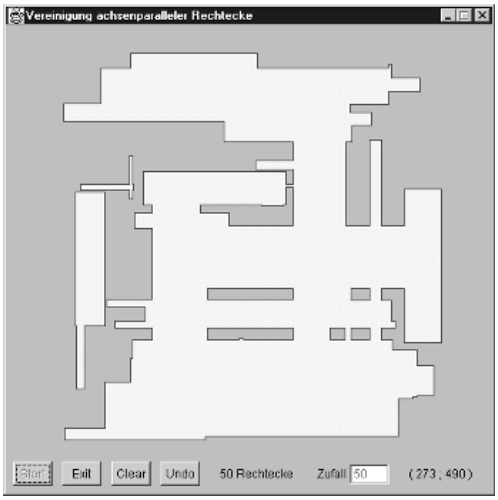
Picture 1.1: Rectangle Union example 1.1

For this example, the problem seems to be easy - the three boundary lines can be recognized by looking.

Example 1.2. This second example with n = 50 illustrates that the Rectangle Union is a relatively complex problem which has to be calculated by a computer for big values of n. In this case the result consists of 3 outer boundary lines and 6 inner boundary lines with 146 boundary edges altogether.



Picture 1.2: input rectangles example 1.2



Picture 1.3: Rectangle Union example 1.2

There are practical applications of the Rectangle Union - for example in the field of design of digital circuits ("VLSI-design") or in the field of concurrency controls of data bases (see [2]).

2. Calculation of the measure of the Rectangle Union

This problem can be solved by two different methods: "divide and conquer" and "plane-sweeping". Both methods are basic principles of Computational Geometry (see [3]). In the following, the plane-sweep algorithm will be explained :

For example, the input data can be provided in the form of coordinates of end points of the rectangle diagonals. The strictly monotonic increasing sequence of the rectangle abscissae $\{x_i\}$ with $i \in \{1, 2, \ldots n_x\}$ can be determined directly by scanning the input coordinates. It is essential : $n_x \leq 2n$.

In the same way the strictly monotonic increasing sequence of the rectangle ordinates $\{y_i\}$ with $i \in \{1, 2, \ldots n_y\}$ is found. Here it is essential again : $n_y \leq 2n$.

Now, the sorted sequence of the vertical rectangle sides $\{s_k\}$ can be determined by a scan of the input coordinates: the sequence has $2n$ elements and must be sorted by ascending abscissae. In this context, a vertical rectangle side $\{s_k\}$ is defined by $s_k := (a_k; l_k, u_k, t_k)$ (abscissa; lower ordinate, upper ordinate, type) with $a_k \in \{x_i\}$ and $l_k, u_k \in \{y_i\}$ and $t_k =$ "left" or $t_k =$ "right".

The following iterative method for the calculation of the measure Ω is an example of "plane-sweeping", that means the plane will be "swept" with a line $x = x_i$ (so-called "sweep line") from left to right. (So the number of iterations is n_x).

Necessary for this method is a data structure for the Ordinate Interval $[y_1, y_{n_y}]$. Any time a left rectangle side s_L is found during the "plane-sweeping", the interval

$[l_L, u_L] \subseteq [y_1, y_{n_y}]$ must be "allocated" until the corresponding right rectangle side s_R is found. (Then the interval $[l_R, u_R] = [l_L, u_L]$ must be "deallocated" again).

Let $A(j)$ be the number of allocations of an interval of form $[y_j, y_{j+1}]$, where $1 \leq j < n_y$. (At the beginning of the method the Ordinate Interval is "not allocated": $A(j) = 0 \; \forall \; j$).

The **allocation** of a (left) vertical rectangle side s_L means:

$$A(j) = A(j) + 1 \; \forall \; j \text{ with } [y_j, y_{j+1}] \subseteq [l_L, u_L].$$

Analogous the **deallocation** of a (right) vertical rectangle side means:

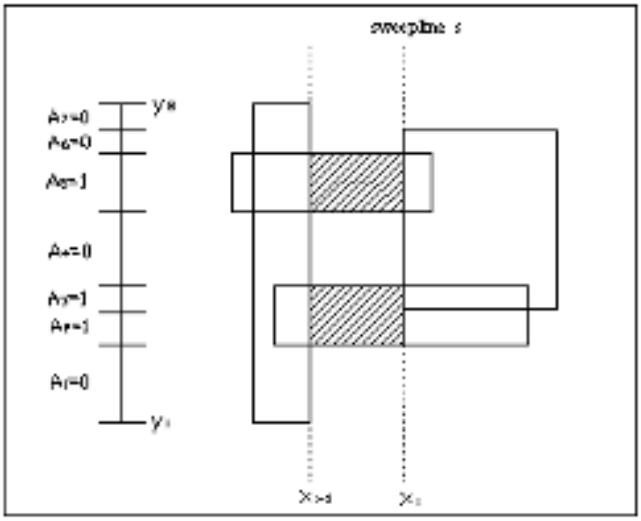
$$A(j) = A(j) - 1 \; \forall \; j \text{ with } [y_j, y_{j+1}] \subseteq [l_R, u_R].$$

Furthermore let L be the total length of all disjoint and allocated intervals $\subseteq [y_1, y_{n_y}]$. With this definition, L can be calculated as the total length of all allocated intervals of form

$$[y_j, y_{j+1}] : \; L = \sum_j (y_{j+1} - y_j) \; \forall \; j \text{ with } A(j) > 0.$$

The formulation of the method for the calculation of the measure Ω can now be done like this : If the "sweep line" is currently at place x_i, the increase of measure Ω is the product of $(x_i - x_{i-1})$ and the total length L of all disjoint and allocated intervals $\subseteq [y_1, y_{n_y}]$ (see picture 2.1).

Before the next iteration step, all left vertical rectangle sides s_k with abscissa x_i must be allocated . All right vertical rectangle sides s_k with abscissa x_i must be deallocated.



Picture 2.1: example of a "plane-sweeping" with "sweep line" at place $x = x_i$

So the "sweep" of the plane will be done by a scan of the strictly growing sequence of the rectangle abscissae $\{x_i\}$. It's now possible to produce a formulation in meta language:

```
Ω = 0;  k = 1;  x_0 := x_1
for ( int i = 1;  i ≤ n_x ;  i = i + 1 )
{
        Ω = Ω + ( x_i - x_i-1 ) · L
        while ( a_k = x_i AND k ≤ 2n )
        {
                if ( t_k = "left" )
                {
                        allocate( s_k )
                }
                else
                {
                        deallocate( s_k )
                }
                k = k + 1
        }
}
```

Theorem 2.1. *The calculation of the measure of a Rectangle Union with n rectangles requires a time complexity $O(n \log_2 n)$ and a storage complexity $O(n)$.*

Proof. Storage complexity: The cardinality of both sequences $\{x_i\}$ and $\{y_i\}$ is not bigger than $2n$. The number of allocations to store is not bigger than $2n$, either. The sequence $\{s_k\}$ has $2n$ elements. For the total number M of elementary storage cells needed, it follows that $M = M(n) \leq (c_1 + c_2 + c_3 + c_4)2n \leq cn \Rightarrow M = O(n)$.

Time complexity: For obtaining the abscissae, the ordinates and the vertical rectangle sides by scanning the input data (coordinates of rectangle diagonals), only a linear complexity $O(n)$ is required. The sorting of these data can be achieved with a complexity $O(n \log_2 n)$ (see[1]).

For example the sorted sequences $\{x_i\}$, $\{y_i\}$ and $\{s_k\}$ can be stored in arrays, so that the value of an element for a given index can be found with constant complexity $O(c)$. Finding an index for a given value of an element can be done with logarithmic complexity $O(\log_2 n)$ (binary search).

For the Ordinate Interval $[y_1, y_{n_y}]$, it's possible to use a data structure called Segment Tree (see [2] and [3]). This special binary tree allows us to allocate and deallocate rectangle sides (as partial intervals of $[y_1, y_{n_y}]$) with logarithmic complexity $O(\log_2 n)$. Furthermore it is possible to obtain the total length L of all disjoint and allocated intervals in the Segment Tree with constant complexity $O(c)$.

Complexity of "plane-sweeping": Each of the $2n$ vertical rectangle sides will be allocated or deallocated exactly once during the "plane-sweeping". Furthermore for

each iteration step it is necessary to actualize some loop parameters and to check the orientation of the rectangle side - this always requires constant complexity $O(c)$. The measure has to be actualized exactly -times (each time $O(c)$).

For the number of needed elementary operations for the "plane-sweeping" it follows that $A_p(n) \leq 2n(c_1 \log_2 n + c_2) + c_3 n_x \Rightarrow A_p(n) = O(n \log_2 n)$.

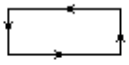
It follows that a total time complexity $O(n \log_2 n)$ is necessary. □

3. Boundary lines of the Rectangle Union

3.1. Properties of the boundary lines

Theorem 3.1. *Number of boundary edges For the number p of boundary edges of a Rectangle Union of n rectangles it is essential:* $p \leq n^2 + 4n$ *(proof: see [3]).*

Definition 3.1. Orientation of a boundary edge
The orientation of a boundary edge is defined as the orientation of its rectangle sides (every boundary edge is a part of a rectangle side). The orientation of a rectangle side is defined like this: if you "walk" along the rectangle, the inside area of the rectangle always lies to the left.



Definition 3.2. Orientation of a boundary line
The orientation of a boundary line is defined as the orientation of its vertical boundary edge with minimal abscissa - if this boundary edge is orientated "down", then the boundary line is orientated in mathematical positive manner and we call it an outer boundary line. Otherwise the boundary line is orientated in a mathematically negative manner and we call it an inner boundary line (see Example 1.1).

Consequence 3.1. The boundary lines of a Rectangle Union are closed and orientated polygons. Each boundary line consists of alternating vertical and horizontal boundary edges.

3.2. Calculation of the boundary lines

The algorithm consists of three phases :
3.2.1 Sorting the input data
3.2.2 Calculation of the boundary edges
3.2.3 Calculation of the boundary lines

3.2.1. Sorting the input data

Like in chapter 2, the sorted abscissae $\{x_i\}$ and the sorted ordinates $\{y_i\}$ can be obtained by scanning the input coordinates. Depending on the way we solve the main problem in phase 3.2.2, we also have to create the sorted sequences of the vertical rectangle sides $\{s_k\}$ and the horizontal rectangle sides $\{s_k^*\}$. If we use a "plane-sweep" algorithm in phase 3.2.2, the sorting of the rectangle sides has to be done by using several sorting attributes : for example the vertical rectangle sides must be sorted primarily for ascending abscissae, secondly for type (left before right rectangle sides) and thirdly for ascending lower and upper ordinates.

3.2.2. Calculation of the boundary edges

The calculation of the boundary edges is the main problem for the calculation of the boundary lines. For doing that, there are several "plane-sweep" algorithms which use a modified form of the Segment Tree as data structure (see [2], [3]). Furthermore there is one "divide and conquer" algorithm (see [1]).

For the solution of this problem we also need additional data structures like linear lists and binary search trees.

Because of the complexity of the algorithms, turn to the references for details.

3.2.3. Calculation of the boundary lines

If all vertical and horizontal boundary lines are found, the last step is to identify the closed and oriented boundary lines which consist of alternating horizontal and vertical boundary edges.

Given is (after phase 3.2.2) the sorted sequence of vertical boundary edges $\{v_i\}$. $v_i = (a_i; l_i, u_i, d_i)$ (abscissa; lower ordinate, upper ordinate, orientation $\downarrow$ or $\uparrow$) $\{v_i\}$ is sorted primarily for ascending abscissae, secondly for orientation ($\downarrow$ before $\uparrow$) and thirdly for ascending lower ordinates.

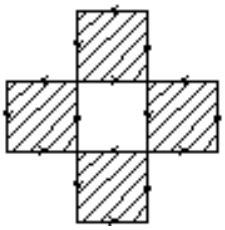
Moreover, given is (after phase 3.2.2) the sorted sequence of horizontal boundary edges $\{h_i\}$. $h_j = (o_j; l_j, r_j, d_j)$ (ordinate; left abscissa, right abscissa, orientation $\rightarrow$ or $\leftarrow$) $\{h_i\}$ is primarily sorted for ascending abscissae, secondly for orientation ($\rightarrow$ before $\leftarrow$) and thirdly for ascending left abscissae.

For the calculation of the closed and oriented boundary lines, the method starts with the first vertical boundary edge v_1: a horizontal following boundary edge must be found for a given vertical boundary edge. For this horizontal edge, the next vertical following boundary edge must be found, and so on.

Definition 3.3. Rule for finding a following boundary edge
If there are two possible following boundary edges for a given boundary edge, then the following boundary edge to one's right must be chosen.

If this rule is applied, all the outer boundary lines will be pairwise disjoint. That means the boundary lines will be created in a way that produces the maximum number of inner boundary lines. With this rule, the information about all "holes" in the Rectangle Union are included in the inner boundary lines.

The following example demonstrates the importance of the definition 3.3: Given are 4 congruent rectangles whereby all 16 rectangle sides are boundary edges of the Rectangle Union. Applying the rule of Definition 3.3, one outer boundary line (12 boundary edges) and one inner boundary line (4 boundary edges) is obtained.



Each vertical boundary edge which is found by applying Definition 3.3 must be deleted. If there is no more vertical following boundary edge for a boundary line, then this boundary line is closed.

The next boundary line starts with the "smallest" vertical boundary edge related to the still existing sorted boundary edges $\{v_i\}$.

In this way the first vertical boundary edge of a boundary line always has the smallest abscissa. With the orientation of this first edge the orientation of the boundary line is defined (inner or outer boundary line).

All boundary lines are found if there are no more elements in $\{v_i\}$.

Theorem 3.2. *The calculation of the boundary lines of a Rectangle Union with n rectangles requires a time complexity* $O(n^2 \log_2 n)$ *and a storage complexity* $O(n^2)$ *(proof: see [3]).*

4. Conclusion

The Rectangle Union is a geometric problem which can be solved efficiently by applying basic principles of Computational Geometry ("plane-sweeping", "divide and conquer") in combination with adequate data structures (special binary trees, Lists, Arrays).

In the scope of this project, a computer program for the calculation and visualization of Rectangle Unions was realized with the programming language Java.

An interesting extension to this problem would be the spatial union of cuboids with sides parallel to the coordinate axis.

References

[1] Güting , Ralf Hartmut : Datenstrukturen und Algorithmen, Verlag B.G. Teubner Stuttgart, 1992

[2] Preparata , Franco P. ; Shamos , Michael Ian : Computational Geometry, Springer Verlag , 1985

[3] Schmid, Daniel : Prinzipien der Algorithmischen Geometrie, dargestellt an Aufgaben mit Rechtecken, Diplomarbeit im Fachbereich C, Fachhochschule für Technik Stuttgart WS 2002/2003

Postal address

Daniel Schmid, Hanspeter Bopp
Department of Geomatics, Computer Science and Mathematics
Stuttgart University of Applied Sciences
Schellingstraße 24, 70174 Stuttgart
Germany