Boolean Operations on Intervals and Axis-Aligned Rectangles

David Eberly
Geometric Tools, LLC
http://www.geometrictools.com/

Created: July 28, 2008

Contents

The motivating problem is to compute Boolean operations on sets of axis-aligned rectangles. These operations include *union*, *intersection*, *difference*, and *exclusive-or*. The algorithm for computing a Boolean operation is simplified if we represent each set of rectangles as a disjoint union of rectangles. Boolean operations between two disjoint unions of rectangles reduces in dimension to Boolean operations between disjoint unions of intervals.

1 Boolean Operations for Intervals

In the context of real numbers, a *closed interval* is the set

$$[a, b] = \{z : a \leq z \leq b\}$$

where it is assumed that $a < b$. The set includes all real numbers between a and b inclusive. If $a = b$, the interval is considered to be *degenerate*, a single point. An *open interval* is the set

$$(a, b) = \{z : a < z < b\}$$

In this case, the values a and b are excluded from the set. Two more variations are *half-open* intervals

$$[a, b) = \{z : a \leq z < b\}$$

In this case, a is included but b is excluded. A similar definition applies to the half-open interval $(a, b]$. The *empty set* $\emptyset$ represents an interval that contains no values.

When applying Boolean operations to intervals it is convenient to use input intervals all the same type (all closed, all open, all half-open). Moreover, we would like the resulting output intervals to be the same type as the inputs. Table 1.1 shows examples where the input intervals are closed.

Table 1.1 *Boolean operations for some closed intervals.*

$[0, 2] \cap [1, 3]$	$=$	$[1, 2]$	*intersection*
$[0, 2] \cup [1, 3]$	$=$	$[0, 3]$	*union*
$[0, 2] \setminus [1, 3]$	$=$	$[0, 1)$	*difference*
$[0, 2] \otimes [1, 3]$	$=$	$[0, 1) \cup (2, 3]$	*exclusive or*
$[0, 1] \cap [1, 2]$	$=$	$\{1\}$	*degenerate intersection*

In the last three examples, the result of the Boolean operation is not a closed interval or a set of closed intervals, which is undesirable. We instead use half-open intervals of the type $[a, b)$ where $a < b$. The intersection of two half-open intervals is either a half-open interval or the empty set. Any Boolean operation between two disjoint sets of half-open intervals always produces a disjoint set of half-open intervals, as illustrated in Table 1.2.

Table 1.2 *Boolean operations on half-open intervals produce half-open intervals.*

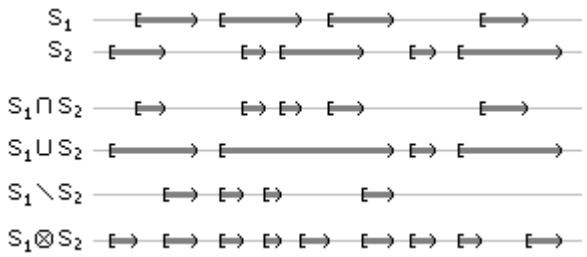
$[0,2) \cap [1,3)$	$=$	$[1,2)$	*intersection*
$[0,2) \cup [1,3)$	$=$	$[0,3)$	*union*
$[0,2) \setminus [1,3)$	$=$	$[0,1)$	*difference*
$[0,2) \otimes [1,3)$	$=$	$[0,1) \cup [2,3)$	*exclusive or*
$[0,1) \cap [1,2)$	$=$	$\emptyset$	*intersection*
$[0,1) \cup [1,2)$	$=$	$[0,2)$	*union*

The empty set is considered to be a degenerate half-open interval and is therefore a valid result.

2 Boolean Operations for Disjoint Unions of Intervals

Given two sets S_1 and S_2, each a disjoint union of half-open intervals, we want to compute the intersection $S_1 \cap S_2$, the union $S_1 \cup S_2$, the difference $S_1 \setminus S_2$, and the exclusive-or $S_1 \otimes S_2$. Figure 2.1 is an illustration of these operations.

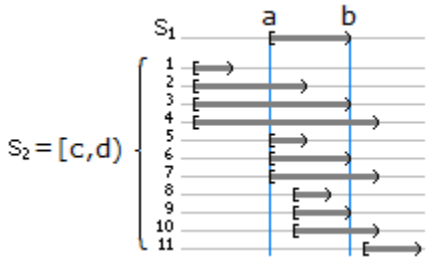
Figure 2.1 Boolean operations in one dimension.



Observe that the endpoints of the half-open intervals produced by the Boolean operation must be endpoints of half-open intervals in the original sets. The implementation of each Boolean operation involves identifying which of those endpoints are part of the result.

The algorithm is iterative. The first interval of S_2 is partitioned by the intervals of S_1. Each subinterval in the partition is either kept or discarded according to the Boolean operation that is applied. The partitioning is illustrated by Figure 2.2.

Figure 2.2 An interval of S_2 is partitioned by an interval of S_1. The partitioning has 11 cases.



Let the interval of S_1 be $[a, b)$. Let the interval of S_2 be $[c, d)$. Let R be the disjoint union of intervals produced by the Boolean operation applied to S_1 and S_2. The cases are listed below.

1. The interval $[c, d)$ needs no partitioning. It is added to R for union or exclusive-or but not for intersection or difference.

2. The interval $[c, d)$ is partitioned into $[c, a)$ and $[a, d)$. $[c, a)$ is added to R for union or exclusive-or but not for intersection or difference. $[a, d)$ is added to R for union or intersection but not for exclusive-or or difference.

3. The interval $[c, d)$ is partitioned into $[c, a)$ and $[a, d)$. $[c, a)$ is added to R for union or exclusive-or but not for intersection or difference. $[a, d)$ is added to R for union or intersection but not for exclusive-or or difference. Distinguishing case 3 from case 2 is useful for Boolean operations on axis-aligned rectangles.

4. The interval $[c, d)$ is partitioned into $[c, a)$, $[a, b)$, and $[b, d)$. $[c, a)$ is added to R for union or exclusive-or but not for intersection or difference. $[a, b)$ is added to R for union or intersection but not for exclusive-or or difference. $[b, d)$ becomes the next S_2 interval that is processed against the next interval of S_1.

5. The interval $[c, d)$ needs no partitioning. It is added to R for union or intersection but not for exclusive-or or difference.

6. The interval $[c, d)$ needs no partitioning. It is added to R for union or intersection but not for exclusive-or or difference. Keeping case 6 separate from case 5 is useful for 2D Boolean operations.

7. The interval $[c, d)$ is partitioned into $[c, b)$ and $[b, d)$. $[c, b)$ is added to R for union or intersection but not for exclusive-or or difference. $[b, d)$ becomes the next S_2 interval that is processed against the next interval of S_1.

8. The interval $[c, d)$ needs no partitioning. It is added to R for union or intersection but not for exclusive-or or difference. Distinguishing case 8 from cases 5 and 6 is useful for Boolean operations on axis-aligned rectangles.

9. The interval $[c, d)$ needs no partitioning. It is added to R for union or intersection but not for exclusive-or or difference. Distinguishing case 9 from cases 5, 6, and 8 is useful for Boolean operations on axis-aligned rectangles.

10. The interval $[c, d)$ is partitioned into $[c, b)$ and $[b, d)$. $[c, b)$ is added to R for union or intersection but not for exclusive-or or difference. $[b, d)$ becomes the next S_2 interval that is processed against the next interval of S_1.

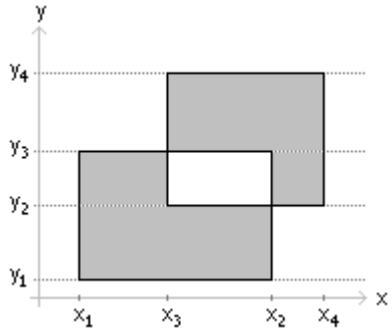
11. The interval $[c, d)$ is processed against the next interval of S_1.

The methods for intersection, union, difference, and exclusive-or use an algorithm for identifying the endpoints for the resulting set in a manner identical to *merge sort* of two sorted lists. The two lists are iterated simulataneously, keeping track of the ordering of the two current values from each list. Whenever a change in ordering occurs, a subinterval is generated that must be included in, or omitted from, the Boolean result.

3 Boolean Operations for Rectangles

In two dimensions, Boolean operations are applied to axis-aligned rectangles by decomposing the problem into one-dimensional operations. Figure 3.1 illustrates with the exclusive-or of two rectangles.

Figure 3.1 Exclusive-or of two rectangles. The use of row strips is for horizontal buffers. Column strips are used for vertical buffers.



Two rectangles are $[x_1, x_2) \times [y_1, y_2)$ and $[x_3, x_4) \times [y_3, y_4)$. The first rectangle may be viewed as a *strip* with a *minimum value* y_1 and a *maximum value* y_2. The strip represents a set of one-dimensional half-open intervals, in this case the set consisting of a single interval $\{[x_1, x_2)\}$. More complex objects can be built in this manner as an array of disjoint strips, each strip representing a set of one-dimensional half-open intervals.

The second rectangle is represented by a strip with minimum value y_3 and maximum value y_4. Its set of half-open intervals has one element, namely $\{[x_3, x_4)\}$.

First, the strips are partition in the y-direction. The partitioning is according to what was described previously with the 11 possible cases. In the current example, the two strips for $[y_1, y_2)$ and $[y_3, y_4)$ generate a partitioning into three strips $[y_1, y_3)$, $[y_3, y_2)$, and $[y_2, y_4)$.

Second, within each of the new strips, a 1-dimensional Boolean operation is applied between the sets of intervals represented by the strips. In the current example, consider the $[y_1, y_3)$ strip. The set of intervals

for this strip are just those of the strip $[y_1, y_2)$, namely $\{[x_1, x_2)\}$. There is no set of intervals in this strip due to the other rectangle, so we may choose $\emptyset$ (the empty set) for that rectangle. The exclusive-or of $\{[x_1, x_2)\}$ and $\emptyset$ is $\{[x_1, x_2)\}$.

Each of the two rectangles has a set of intervals for the strip $[y_3, y_2)$. The first rectangle provides $\{[x_1, x_2)\}$ and the second rectangle provides $\{[x_3, x_4)\}$. The resulting set of intervals for this strip is the exclusive-or of the initial two sets, namely $\{[x_1, x_3), [x_2, x_4)\}$.

The strip $[y_2, y_4)$ has no intervals from the first rectangle, so that set is $\emptyset$. The second rectangle provides the set $\{[x_3, x_4)\}$. The exclusive-or of these two sets is $\{[x_3, x_4)\}$ and is the resulting set for the new strip.

The final representation is shown abstractly in Table 3.1

Table 3.1 *The abstract representation of the exclusive-or of the rectangles as shown in Figure 3.1.*

strip	*set of intervals*
$[y_1, y_3)$	$\{[x_1, x_2)\}$
$[y_3, y_2)$	$\{[x_1, x_3), [x_2, x_4)\}$
$[y_2, y_4)$	$\{[x_3, x_4)\}$

The source code uses a simple loop iteration with loop counters `i0` and `i1`. The advancement of the loop counters is similar to the 1-dimensional case of merge sort mentioned earlier. The loop body has comments with case numbers that correspond to the 11 cases mentioned previously in this document. Cases 1 and 11 are tested first because they require no partitioning. If neither case is encountered, the intervals of the two operands must intersect. Cases 2, 3, or 4 are reduced to cases 5, 6, or 7 by processing any subinterval of the second operand that is to the left of the interval for the first operand. Similarly, cases 8, 9, or 10 are reduced to to cases 5, 6, or 7 by processing any subinterval of the first operand that is to the left of the interval for the second operand. If case 10 is encountered, the subinterval of the second operand to the right of the interval for the first operand is saved for later processing (the next pass through the loop). After these reductions, case 5 requires partitioning one strip into two, case 6 requires no partitioning, and case 7 requires partitioning one strip into two. The second strip is processed later, similar to case 10.

Within each new strip for which both original strips provide sets of half-open intervals, the 1-dimensional Boolean operation is applied to those sets.