Pairwise Triangles Intersections in a Query Rectangle

Waseem Akram[1] **and Sanjeev Saxena** [2]

1 **Indian Institute of Technology, Kanpur(India)**
`akram@iitk.ac.in`

2 **Indian Institute of Technology, Kanpur(India)**
`ssax@iitk.ac.in`

Abstract

The range searching problem is one of the most studied problems in computational geometry. In this paper, we study the following range-searching problem. Given a set of n homothetic right-angled triangles in the plane, we want to compute the pairs of triangles intersecting inside an axis-aligned query rectangle. A triangle T is said to be homothetic to another triangle T' if T can be obtained from T' using scaling and translation operations. We show that after preprocessing the given set in $O(n \log n)$-space and time, each subsequent query can be answered in $O(\log n + k)$-time, where k is the number of reported pairs.

1 Introduction

Range searching problems are fundamental in computational geometry and find applications in numerous domains, including motion planning, robotics, and spatial databases [2, 4, 5]. The range searching problems involving orthogonal objects (points, line segments, rectangles) have been well explored [3, 4, 7]. In this paper, we study a range-searching problem that considers a class of non-orthogonal objects (homothetic triangles). A triangle T is said to be homothetic to another triangle T' if T can be obtained from T' using scaling and translation operations. The problem is defined as follows.

> Given a set of n homothetic right-angled triangles with perpendicular sides parallel to the coordinate axes, we want to preprocess the set so that, given an axis-aligned query rectangle Q, all the pair of triangles (T_i, T_j) intersecting inside the rectangle Q (i.e. $T_i \cap T_j \cap Q \neq \emptyset$) can be reported efficiently.

This problem is a generalization of the problem studied in [4, 5], which asks to compute all pairs of rectangles intersecting inside a query rectangle. Mark de Berg et al. [4] solved the problem using $O(n \log n)$ space and $O(\log n \log^* n + k \log n)$ query time, where k is the output size. Oh and Ahn [5] improved the query time to $O(\log n + k)$ without aggravating the space bound. The problem finds application in motion planning. A common application scenario for such problems is described next. We can think that geometric objects are the trajectories of moving objects (drones/airplanes in the sky or robots in a factory), and we would like to know parts of trajectories where two entities may collide so that one can take appropriate measures to avoid collisions. Furthermore, we likely want to ask this question for a particular small region only.

Our Result: We show that after preprocessing in $O(n \log n)$-space and time, each query can be answered in $O(\log n + k)$-time, where k is the number of reported pairs of triangles.

40th European Workshop on Computational Geometry, Ioannina, Greece, March 13–15, 2024.

Definitions and Notations

Let $S = \{T_1, T_2, \ldots, T_n\}$ be a set of n homothetic right-angled triangles in the plane such that their orthogonal sides are parallel to the coordinate axes. Without loss of generality, we assume that all the triangles are present in the first quadrant and their right-angled vertices are at the bottom-left position (i.e., minimum x and minimum y coordinates). Let m be the slope of the hypotenuses of the triangles in S. The horizontal, vertical, and hypotenuse sides of a triangle T_i are denoted by h_i, v_i, and hy_i respectively.

Let l_i be the line containing the segment hy_i, and let l_0 be the line passing through the origin $(0, 0)$ with slope m. For each $i \in \{1, 2, \ldots, n\}$, we define the *distance* of the segment hy_i from the line l_0, denoted by d_i, as the Euclidean distance between the parallel lines l_0 and l_i. We say that $T_i \preceq T_j$ if $d_i \leq d_j$.

Let ab be a side of a triangle $T \in S$ (with a and b as its endpoints). The *stretch* of ab, denoted by $a'b'$, is the smallest segment of ab which contains all the intersection points of the side ab with sides of other triangles of S. If ab has no intersection point, then *stretch* of ab is undefined. Thus, a triangle in S can have at most three stretches, one for each side. Let P' be the set of their endpoints. Note that $|P'| \leq 6n$. We say that a line segment s crosses a rectangle Q if $s \cap Q \neq \emptyset$ and each endpoint lies outside Q (see Figure 1).

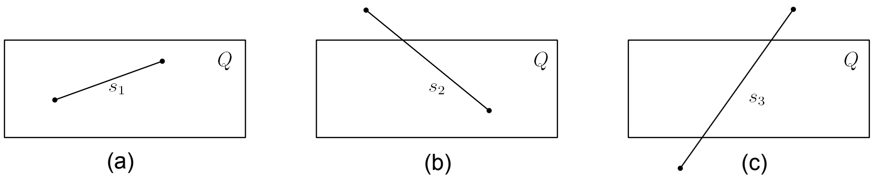

Figure 1 Segments s_1 and s_2 do not cross Q, while the segment s_3 does.

2 Proposed Solution

We design a space-efficient data structure that supports queries in optimal time. Our approach is similar to the one used in [5]. For a given query rectangle Q, we need to compute all the pairs (T_i, T_j) of S such that T_i and T_j intersect each other inside Q. Observe that the intersection of T_i and T_j, denoted by $I(i, j)$, is also a triangle homothetic to the triangles in S. We identify a few configurations for a given Q, which will be used to compute the output set $U(Q)$. The size of the set $U(Q)$ is denoted by $k(Q)$. For any pair (T_i, T_j) of triangles of S with $I(i, j) \cap Q \neq \emptyset$, it can be seen that at least one of the following conditions holds. The configurations have been depicted in Figure 2.

- C1: T_i contains Q and T_j intersects Q or vice-versa.
- C2: An endpoint of a stretch l of T_i (or T_j) lies in Q, and T_j (or T_i) intersects $l \cap Q$.
- C3: A stretch l_i of T_i and a stretch l_j of T_j cross Q and intersect each other inside Q.
- C4: $I(i, j)$ contains a vertex of Q.
- C5: The vertical sides of Q intersect the horizontal and hypotenuse sides of $I(i, j)$, or the horizontal sides of Q intersect the vertical and hypotenuse sides of $I(i, j)$.

2.1 Data structures

We now build a set of data structures that will be used to compute the pairs of the configurations mentioned above.

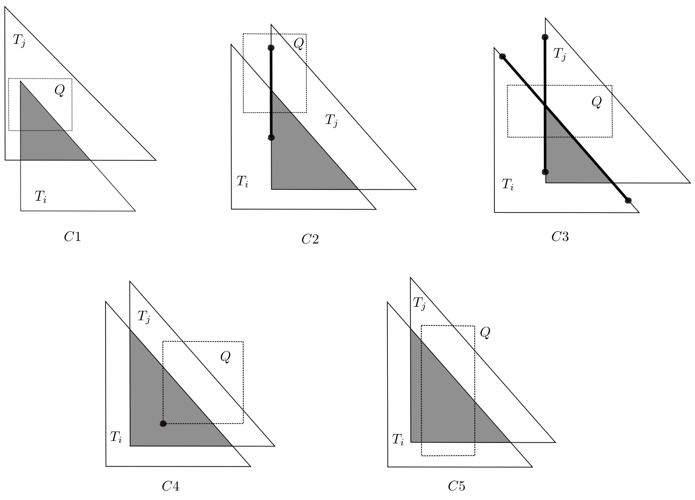

Figure 2 The gray region denotes the intersection $I(i, j)$ of T_i and T_j in each configuration. The bold segment of a side denotes its stretch.

- *Enclosure Searching Problem*: Given a set of geometric objects, the problem is to find the objects o enclosing a query object q (i.e., $q \in o$). We preprocess the set S of triangles for enclosure searching queries as described in [1]. Let D_{enc} denotes the built data structure. The structure D_{enc} supports enclosure searching queries for points, line segments, and trapezoids in $O(\log n + t)$-time, where t is the number of reported triangles.
- *The Orthogonal Range Searching Problem*: Given a set of points in the d-dimensional space and an orthogonal rectangle, the problem is to find all the points lying inside the rectangle. Chazelle [3] gave an optimal solution to the problem in the 2-dimensional space, which takes $O(\log n + \#output)$-query time and $O(n \log n / \log \log n)$-space. We build an orthogonal range reporting data structure for the set P' of stretch endpoints using Chazelle's method [3] and denote it by D'_r.
- *The Segment Intersection Problem*: Given a set of line segments and a line segment q, the segment intersection problem is to find the segments s intersected by the segment q (i.e., $s \cap q \neq \emptyset$). The following result is due to Chazelle [3].

  Theorem 1 in [3]: It is possible to preprocess n segments in $O(n \log n)$ time and $O(n)$-space so that computing their intersections with a query segment that either has a fixed slope or has its supporting line passing through a fixed point can be done in $O(k + \log n)$ time, where k is the number of intersections to be reported. It is assumed that the interior of the segments are pairwise disjoint.

  We now describe an overview of the Chazelle's solution. Given a set of pairwise (interior) disjoint line segments, a planar subdivision is built. The subdivision is then preprocessed for *point location queries*, which, given a point in the plane, finds the face of the subdivision containing the point. Given a query segment, the face containing an endpoint of the segment is located and one moves towards the other endpoint along the query segment. The segments encountered on the way are exactly those which are intersected by the query segment. The query takes $O(\log n + t)$-time, where t is the output size: $O(\log n)$ time for locating the face and $O(t)$ time for reporting the segments. Note that if we know the face containing an endpoint of the query segment, then query time would be $O(t)$. For more details, please see [3].

  Let H' be the set of all horizontal stretches. We build two data structures D_h^v and D_h^{hy}

over the set H' using Chazelle's method [3]: D_h^v supports queries with vertical segments, and D_h^{hy} supports queries with segments along the hypotenuse-sides. Similarly, we build the structures D_v^h and D_v^{hy} over the set V' of all vertical stretches, and D_{hy}^v and D_{hy}^h over the set H_y' of all stretches with slope m. We keep two pointers with each stretch-endpoint $a' \in P'$ to save time while answering a query. If a' is an endpoint of a vertical stretch, we store two pointers pointing to the faces of D_h^v and D_{hy}^v containing the endpoint a'. Analogously, we store pointers for the endpoints of the other types of stretches.

2.2 Query Algorithms

Let Q be the query rectangle. A pair (T_i, T_j) with $I(i,j) \cap Q \neq \emptyset$ may belong to more than one configurations. We denote by k_i the number of C_i-pairs not belonging to any other configuration C_j with $j < i$.

Reporting $C1$-pairs: We find the set E of the triangles enclosing Q by querying the data structure D_{enc} in $O(\log n + |E|)$-time. For each triangle $T_i \in E$, a triangle $T_j \in S$ intersecting Q would form a $C1$-pair with T_i. If the set E is empty, then no $C1$-pair exists. The triangles $T_i \in S$ intersecting Q can be computed in $O(\log n + \#output)$-time due to the Lemma 2.1.

▶ **Lemma 2.1.** *We can preprocess the set S in $O(n \log n)$-time and space so that given a query rectangle Q, the triangles $T \in S$ with $T \cap Q \neq \emptyset$ can be computed in $O(\log n + \#output)$-time.*

▶ **Corollary 2.2.** *Given a query rectangle Q, we can compute the $C1$-pairs for Q in $O(\log n + k_1)$-time and $O(n \log n)$-space.*

Reporting $C2$-pairs: We find the stretches with an endpoint inside Q by querying the structure D_r'. For each reported stretch l, we compute the triangles intersecting $l \cap Q$ using the segment intersection structures. If l is the stretch of the vertical side of a triangle T_i, we can compute the horizontal and hypotenuse sides intersecting $l \cap Q$ using D_h^v and D_{hy}^v, respectively. The details are omitted due to the space limitations.

▶ **Lemma 2.3.** *We can compute all $C2$-pairs for Q in $O(\log n + k_2)$-time.*

Reporting $C3$-pairs: We first consider the simpler case of orthogonal stretches: a vertical stretch l_i and a horizontal stretch l_j crossing Q. We can compute such pairs of stretches in $O(\log n + t)$-time using 3-d range reporting queries [4, 6], where t is the number of reported pairs. The space used is $O(n \log n / \log \log n)$.

We now consider the case of computing all pairs $(s_1, s_2), s_1 \in H'$ and $s_2 \in H_y'$, such that both stretches cross Q and intersect each other inside Q; the symmetric case of vertical and hypotenuse stretches is analogous. We design a segment tree-based data structure and give an algorithm that takes $O(\log n + \#output)$-time to find all such pairs. The space used by the structure is $O(n \log n)$.

▶ **Lemma 2.4.** *All the $C3$-pairs for Q can be computed in $O(\log n + k_3)$-time. The space used is $O(n \log n)$.*

Reporting $C4$-pairs: For each vertex of Q, we find the set of the triangles enclosing the vertex using the structure D_{enc}. If the set size is less than two, no pair of triangles enclosing the vertex exists. Otherwise, every possible pair of triangles would be a $C4$-pair. Of course, we make sure that each pair is reported once.

▶ **Lemma 2.5.** *One can compute all $C4$-pairs for Q in $O(\log n + k_4)$-time.*

Reporting $C5$-pairs: We show how to find the $C5$-pairs (T_i, T_j) not belonging to any other configuration such that the non-vertical sides of $I(i,j)$ are intersected by the vertical sides of Q. The $C5$-pairs of the other type can be computed analogously.

We build a segment tree $\mathcal{T}$ over S along the x-axis. Each node $v \in \mathcal{T}$ stores a vertical slab $SL(v) = [x, x'] \times \mathbb{R}^2$ such that the vertical slabs corresponding to the nodes of a particular level form a partition of $\mathbb{R}^2$. We say that a triangle $T \in S$ spans a slab $SL(v)$ if $T \cap SL(v) \neq \emptyset$ and none of its vertex lies in $SL(v)$. We store two sets $S_c(v)$ and $S_b(v)$ at node v. The set $S_c(v)$ consists of the triangles $T \in S$ such that T spans the slab $SL(v)$ but not the slab of v's parent node. The set $S_b(v)$ consists of the triangles $T \in S$ such that one or both endpoints of its horizontal side lies in $SL(v)$. We denote by $S(v)$ the union of the sets $S_c(v)$ and $S_b(v)$. The segment tree $\mathcal{T}$ can be built in $O(n \log n)$-time and space.

Consider a pair (T_i, T_j) of triangles of S with $T_i \cap T_j \cap Q \neq \emptyset$. A node v in the tree $\mathcal{T}$ is said to be a canonical node of (i, j, Q) if the left side l_Q of the rectangle Q lies in the interior or on the left side of $SL(v)$, and $T_i, T_j \in S(v)$ such that $T_i \in S_c(v)$ or $T_j \in S_c(v)$. We have the following results related to the canonical nodes.

▶ **Lemma 2.6.** *For each $C5$-pair (T_i, T_j) of Q such that the non-vertical sides of $I(i,j)$ intersect the vertical sides of Q, there is a canonical node of (i, j, Q) in $\mathcal{T}$.*

Proof. The proof is analogous to that of Lemma 3 in [5]. ◀

▶ **Lemma 2.7.** *For any rectangle Q and any pair (T_i, T_j) of rectangles of S with $I(i,j) \cap Q \neq \emptyset$, there is at most one canonical node of (i, j, Q) in $\mathcal{T}$.*

Proof. The proof is analogous to that of Lemma 4 in [5]. ◀

We have the following corollary from Lemma 2.6 and Lemma 2.7.

▶ **Corollary 2.8.** *The total number of canonical nodes for a query rectangle Q is $O(k(Q))$.*

▶ Remark. A node in $\mathcal{T}$ could be the canonical node for more than one pair $(T_i, T_j) \in U(Q)$.

Given a rectangle Q, we compute a set of nodes of $\mathcal{T}$ that includes the unique canonical node of (i, j, Q) for each $C5$-pair (T_i, T_j) not belonging to any other configuration such that the non-vertical sides of $T_i \cap T_j$ intersect the vertical sides of Q. For each such node v, we find all $C5$-pairs such that v is the canonical node of (i, j, Q).

Finding all canonical nodes for $C5$-pairs: We compute a set $\mathcal{V}_Q$ of canonical nodes which contains the canonical node of (i, j, Q) for every $C5$-pair (T_i, T_j). For computing such a set, we define the *trimmed polygon* for a pair (T, v), $T \in S(v)$, as the smallest simple polygon containing the region $T \cap SL(v) \cap U(v)$. Here, $U(v)$ is the union of all the triangles in $S_c(v)$ except T if $T \in S_c(v)$. See Figure 3 for examples. The trimmed polygon for (T, v) has at most two sides with slope m; the number of horizontal and vertical sides are bounded above by 2 and 3, respectively. We compute the required set $\mathcal{V}_Q$ by finding the sides of all trimmed polygons intersected by the left side Q (the details are omitted).

▶ **Lemma 2.9.** *Given a query rectangle Q, we can find a set of at most k nodes containing all canonical nodes of $C5$-pairs not belonging to any other configuration in $O(\log n + k)$ time.*

Handling each canonical node to find all $C5$-pairs: Let $\mathcal{V}_Q$ be the set of canonical nodes for Q computed due to Lemma 2.9. We now compute all $C5$-pairs present at each node $v \in \mathcal{V}_Q$. For this, we have the following lemma.

▶ **Lemma 2.10.** *For each $v \in \mathcal{V}_Q$, we can compute all $C5$-pairs (T_i, T_j) such that v is the canonical node of (i, j, Q) in $O(k_5)$-time. The preprocessing takes $O(n \log n)$-time and space.*

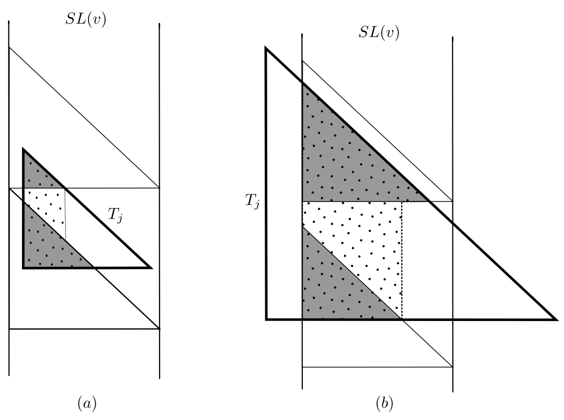

Figure 3 The trimmed polygon (dotted region) for (T_j, v) when (a) $T_j \in S_b(v)$ and (b) $T_j \in S_c(v)$.

We finally put together the results for all configurations and got the following theorem.

▶ **Theorem 2.11.** *Given a set of n right-angled homothetic triangles, one can preprocess so that all pairs of triangles intersecting inside a query rectangle can be computed in $O(\log n + \#output)$-time. The space complexity is $O(n \log n)$.*

References

1 Waseem Akram and Sanjeev Saxena. Dominance for containment problems. *arXiv preprint arXiv:2212.10247*, 2022.

2 Kreveld Overmars Berg, Cheong. *More Geometric Data Structures*, pages 219–241. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. doi:10.1007/978-3-540-77974-2_10.

3 Bernard Chazelle. Filtering search: A new approach to query-answering. *SIAM Journal on Computing*, 15(3):703–724, 1986. doi:10.1137/0215051.

4 Mark de Berg, Joachim Gudmundsson, and Ali D. Mehrabi. Finding pairwise intersections inside a query range. *Algorithmica*, 80(11):3253–3269, Nov 2018. doi:10.1007/s00453-017-0384-3.

5 Eunjin Oh and Hee-Kap Ahn. Finding pairwise intersections of rectangles in a query rectangle. *Computational Geometry*, 85:101576, 2019. doi:10.1016/j.comgeo.2019.101576.

6 Kalina Petrova and Robert Tarjan. A dynamic data structure for segment in-tersection queries. URL: http://www.moi.math.bas.bg/moiuser/~ACCT2016/a41.pdf.

7 Saladi Rahul, Ananda Swarup Das, K. S. Rajan, and Kannan Srinathan. Range-aggregate queries involving geometric aggregation operations. In *WALCOM: Algorithms and Computation*, pages 122–133, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.