HW 1—Exact motion planning

Nir Manor 305229627, Ortal Cohen 308524875

15/02/2024

1 Properties of Minkowski Sums and Euler's theorem

1.1

Given sets A, B and C, formally prove that: $A \oplus (B \cup C) = (A \oplus B) \cup (A \oplus C)$

Reminder:
Given 2 sets $S_1 \subset \mathbb{R}^2$ and $S_2 \subset \mathbb{R}^2$ the Minkowski sum is: $S1 \oplus S2 = \{p + q : p \in S1, q \in S2\}$. Where $p + q := (p_x + q_x, p_y + q_y)$.

Proof:
1) $A \oplus (B \cup C) \subseteq (A \oplus B) \cup (A \oplus C)$

Let there exist $a + b \in A \oplus (B \cup C)$, such that $a \in A$ and $b \in (B \cup C)$.
$a + b \in A \oplus (B \cup C) \implies a \in A$ and ($b \in B$ or $b \in C$) $\implies a + b \in (A \oplus B)$ or $a + b \in (A \oplus C) \implies a + b \in (A \oplus B) \cup (A \oplus C)$

We showed that $a + b \in A \oplus (B \cup C)$ implies that $a + b \in (A \oplus B) \cup (A \oplus C)$ and therefore $A \oplus (B \cup C) \subseteq (A \oplus B) \cup (A \oplus C)$

2) $(A \oplus B) \cup (A \oplus C) \subseteq A \oplus (B \cup C)$

Let there exist $a + b \in (A \oplus B) \cup (A \oplus C)$.
$a + b \in (A \oplus B) \cup (A \oplus C) \implies a + b \in (A \oplus B)$ or $a + b \in (A \oplus C) \implies (a \in A$ and $b \in B)$ or $(a \in A$ and $b \in C) \implies a \in A$ and ($b \in B$ or $b \in C$) $\implies a \in A$ and $b \in (B \cup C) \implies a + b \in A \oplus (B \cup C)$

We showed that $a + b \in (A \oplus B) \cup (A \oplus C)$ implies that $a + b \in A \oplus (B \cup C)$ and therefore $(A \oplus B) \cup (A \oplus C) \subseteq A \oplus (B \cup C)$

Finally, we have showed that $A \oplus (B \cup C) \subseteq (A \oplus B) \cup (A \oplus C)$ and $(A \oplus B) \cup (A \oplus C) \subseteq A \oplus (B \cup C)$ and therefore:

$$A \oplus (B \cup C) = (A \oplus B) \cup (A \oplus C)$$

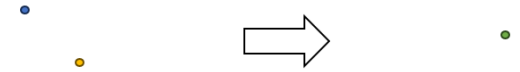
1.2

What is the Minkowski sum (what geometric object and what can you say about it) of:

(i) **Two points?**

Answer:

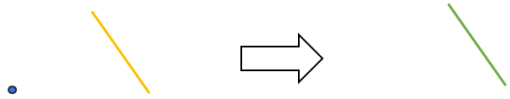
Given 2 sets $S_1 \subset \mathbb{R}^2$ and $S_2 \subset \mathbb{R}^2$ the Minkowski sum is: $S1 \oplus S2 = \{p + q : p \in S1, q \in S2\}$. Where $p + q := (p_x + q_x; p_y + q_y)$. So, given 2 points p and q, s.t. $p \neq q$ the Minkowski sum $p \oplus q$ will return a point.



(ii) **A point and a line?**

Answer:

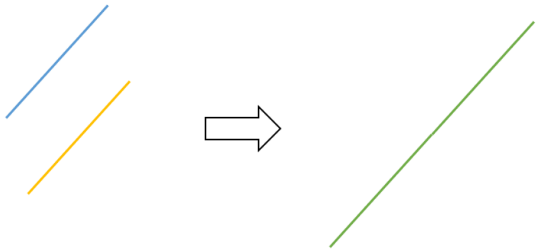
Given a point p and a line l the Minkowski sum $p \oplus l$ will return the line.



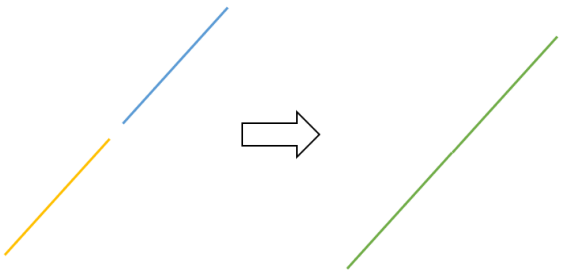
(iii) **Two lines segments?**

Answer:

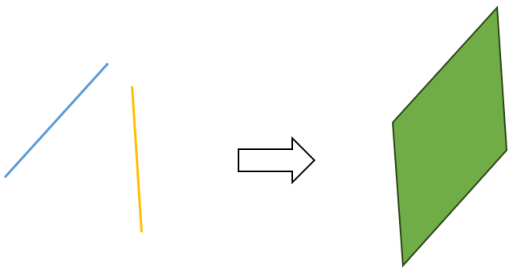
- Given 2 parallel lines l_1 and l_2 the Minkowski sum $l_1 \oplus l$ will return another line which is also parallel to the 2 given lines and which its size is the sum of both of them.



- Given 2 collinear lines l_1 and l_2 the Minkowski sum $l_1 \oplus l$ will return another line which is also collinear to the 2 given lines and which its size is the sum of both of them.



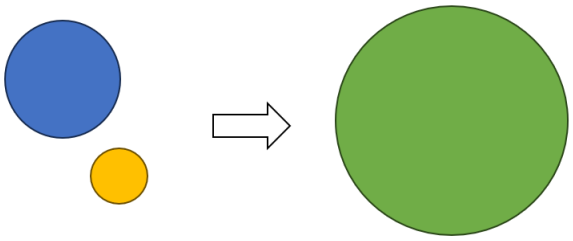
- Given 2 intersecting lines l_1 and l_2 the Minkowski sum $l_1 \oplus l$ will return a parallelogram which is constructed from one line "sliding" over the other line.



(iv) **Two Disks?**

Answer:

Given 2 discs d_1 and d_2 the Minkowski sum $d_1 \oplus d_2$ will return a disc which its diameter is the sum of the diameter of the given discs.



1.3

Recall that for the proof of Lemma. 6.2 (complexity of a trapezoidal map) we used the property that in a planar graph we have that $E <= 3V - 6$. Here E and V are the number of edges and vertices in a planar graph, respectively. Prove that $E \leq 3V - 6$ while assuming that $V \geq 3$.

Reminder - Planar graph:

A planar graph is a graph that can be embedded in the plane. In other words, it can be drawn in such a way that no edges cross each other.

Proof:

Let G be a connected planar graph with $|V|$ vertices, $|E|$ edges and $|F|$ faces, such that $|V| \geq 3$. We will prove that $E <= 3V - 6$ in steps:

1. First we will prove that $3|F| \leq 2|E|$:

1.1. Claim: The sum of the number of edges in the boundary of all faces is equal to 2E.

Proof:

Let define a function for counting the number of edges in the boundary of each face: $X(F_i) \longrightarrow \mathbb{N}$ where n is the number of edges in F_i, and $i \in \{1, ..., |F|\}$.

- Let consider an edge e_i that bounds with two faces F_i and F_j. So both $X(F_i)$ and $X(F_j)$ will count this edge.

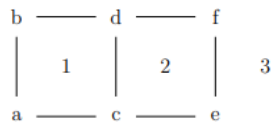


Figure 1: $X(F_1)$ will count c-d and also $X(F_2)$ will count c-d

- Let consider an edge e_i that bounds only with one face F_i. So, $X(F_i)$ will count this edge twice (back and forth along the edge).

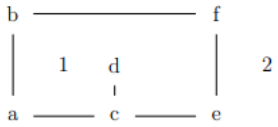


Figure 2: $X(F_1)$ will count the edge c-d twice

Therefore: $\Sigma_{i=1}^{|F|} X(F_i) = 2E$

1.2. Claim: In a connected planar graph G with $|V| \geq 3$, the number of edges in the boundary of each face is at least 3.

Proof:
We will prove it by induction on $|V|$:
Base case: $|V| = 3$
case 1: G is not a tree
G is connected and therefore G is in the shape of a triangle with 3 edges, and thus there are 2 faces: the inner one and the outside. The number of edges in the boundary of the inner face is 3 and the number of edges in the boundary of the outer face is also 3, and it satisfies the claim.

case 2: G is a tree
Since G is connected $E = 2$, e_1, e_2. If G is a tree it implies that $F = 1$ because the unbounded area outside the whole graph is considered as one face. Therefore, the number of edges in the boundary of this face is 4 because we count e_1, e_2 and than in reverse e_2, e_1 and it satisfies the claim.

The induction hypothesis: In a connected planar graph G with $n \geq 3$ vertices, the number of edges in the boundary of each face is at least 3.

Step:
Let consider a G' with $n + 1$ vertices:
case 1: The vertex_{n+1} creates new face
In order to this to happen it means that at least 2 new edges are connected to it, Either by:
- 2 new edges from the ends of an existing edge and thus creating a triangle and then we are in the base case.
- One new edge from an end of an existing edge and the second new edge from an end of another existing edge in the graph and thus creating a polygon with at least 3 vertices and then by the induction hypothesis it holds.

case 2: The vertex$_{n+1}$ does not create new face (for example: inner/outsider point in a face connected by one edge, splitting an existing edge to 2), So by the induction hypothesis it holds.

So in any case, in a connected planar graph with $|V| \geq 3$ the number of edges in the boundary of each face is indeed at least three. And by the notation in 1.1. it means that $X(F_i) \geq 3 \quad \forall i \in \{i, ..., |F|\}$. Finally, from 1.1 we get:

$$\Sigma_{i=1}^{|F|} X(F_i) = 2|E|$$

From 1.2 we get:

$$X(F_i) \geq 3 \quad \forall i \in \{i, ..., |F|\}$$

And thus,

$$\Sigma_{i=1}^{|F|} X(F_i) \geq 3|F|$$

And thus,

$$2|E| = \Sigma_{i=1}^{|F|} X(F_i) \geq 3|F|$$

And thus we conclude that

$$2|E| \geq 3|F|$$

2. Now, by using Euler's formula for connected planar graph:

$$|V| - |E| + |F| = 2 \quad / \cdot 3$$

$$3|V| - 3|E| + 3|F| = 6$$

$$3|F| = 6 - 3|V| + 3|E|$$

$$\xrightarrow{3|F| \leq 2|E|} 6 - 3|V| + 3|E| \leq 2|E|$$

$$|E| \leq 3|V| - 6$$

Extension to Disconnected Planar Graphs:

We note that disconnected planar graphs can be constructed from connected planar graphs by adding edges between the components. We can prove it using induction on the number of connected components in a graph:

Base Case (n = 1):

For a single connected component, the inequality $|E| \leq 3|V| - 6$ holds as previously shown.

Inductive Hypothesis:

Assume that for any connected planar graph with n components, the inequality $|E| \leq 3|V| - 6$ holds.

Inductive Step:

Consider a disconnected planar graph with $n + 1$ components. We can merge two components into a single connected component by adding an edge between them and thus get n components. By the inductive hypothesis, the inequality holds for this new connected component.

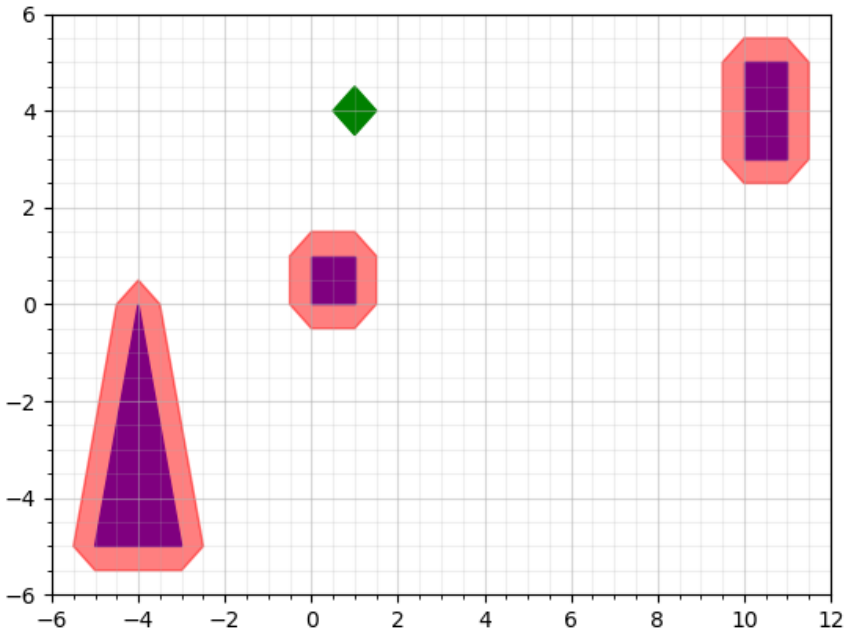
Thus, by induction, the inequality $|E| \leq 3|V| - 6$ holds for any disconnected planar graph.

2 Exact Motion Planning for a Diamond-Shaped Robot

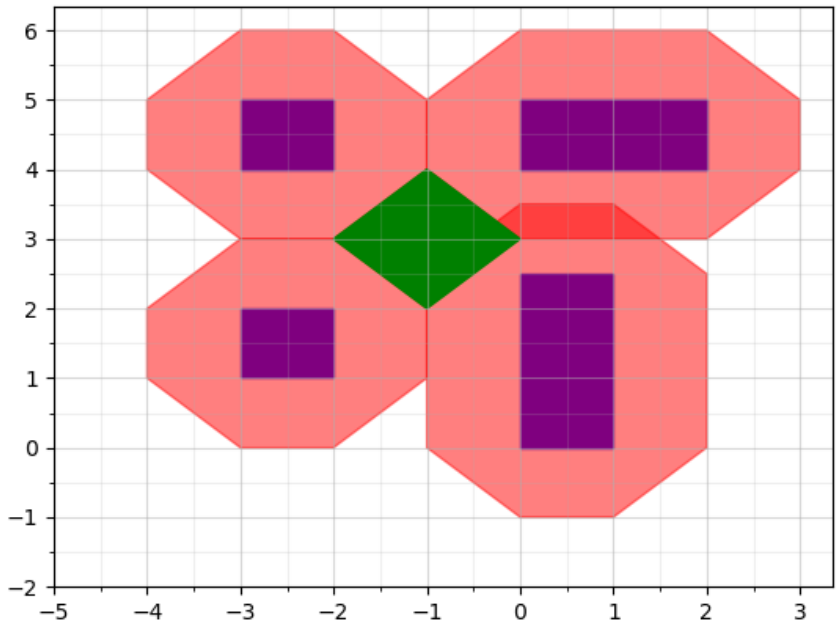
2.1 Preprocessing phase (1) - constructing the C-space

Visualization of the C-space:

i) Visualization for the input files:



ii) Visualization for our input:



The computational complexity of our implementation:

According to Thm[13.11], if P and R are convex polygons with n and m vertices, respectively, the complexity of the Minkowski Sum $P \oplus R$ is $O(n+m)$.

Our robot (R) has 4 vertices and it is convex. Our obstacles has 3-4 vertices and they are convex. We implemented the Minkowski Sum Algorithom and therefore the complexity of computing the Minkowski Sum of $obs_i \oplus R$ for each obstacle is $O(4+4) = O(8)$. Let notate that the number of obstacles is k and than the complexity of our implementation is $O(k \cdot 1) = O(8k) = O(k)$.

How can non-convex obstacles affect the results? Why? When? Support your claim with examples.

- The calculation of the Minkowski Sum of non-convex obstacles, as we implemented it, will not calculate it right, as shown in the example below:

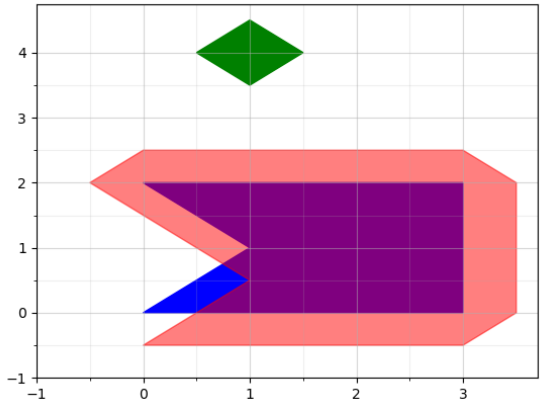


Figure 3: The Minkowski Sum of our convex robot with non-convex obstacle

And therefore we won't be able to construct the visibility graph and thus finding the shortest path from goal to destination.

In order to compute the Minkowski Sum of non-convex obstacles we will need to:

1. Triangulate each obstacle P to get a collection of $n-2$ triangles $t_1, ..., t_{n-2}$.
According to lemma: Any non-convex polygon P with n vertices can be triangulated into $n-2$ triangles $t_1, ..., t_{n-2}$.
2. Compute Minkowski sum of all pairs of convex regions: $t_1 \oplus R, ..., t_{n-2} \oplus R$
3. $P \oplus R$ is the union of $n-2$ constant-complexity polygons

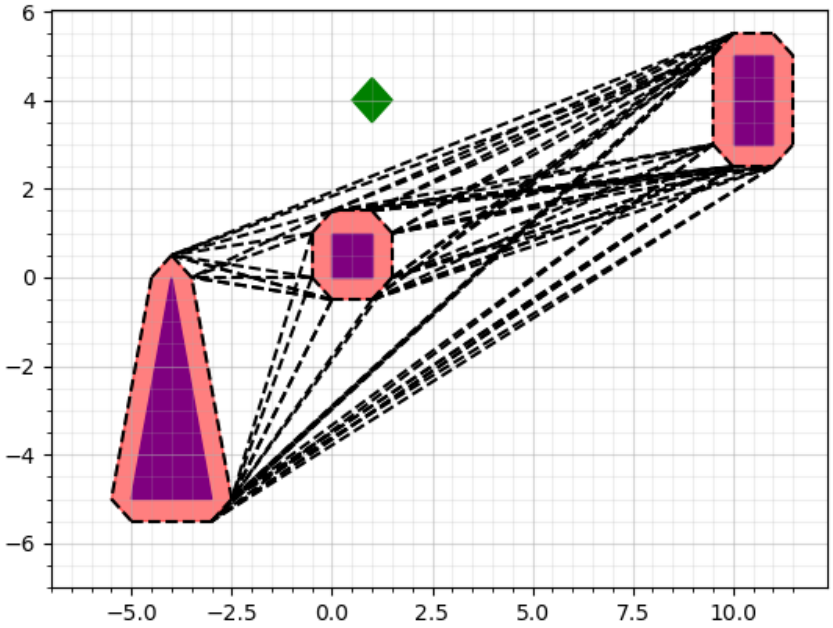
$$P \oplus R = \bigcup_{i=1}^{n-2} t_i \oplus R$$

Therefore, if R is convex polygon and P is non-convex polygon with n and m vertices, respectively, the complexity of the Minkowski Sum $P \oplus R$ is $O(nm)$. Thus, the complexity time of Minkowski Sum of non-convex obstacles will be bigger.

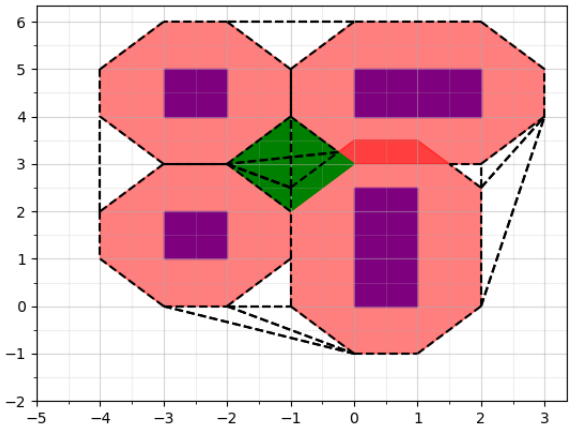
2.2 Preprocessing phase (2)—building the visibility diagram

Visualization of the C-space:

i) Visualization for the input files:



ii) Visualization for our input:



The computational complexity of our implementation:

Creating the visibility graph involves verifying the collision-free nature of line segments connecting the vertices of the C-Space obstacles and the start and destination positions of the robot. Let denote there are n obstacles which are triangles or squares, thus each obstacle has 3-4 edges. Therefore there are at most $(4n) = O(n)$ edges.

Our implementation involves:

1. Checking if the boundaries of the obstacles intersect with each other. There is an outer loop where we iterates over each obstacle in the obstacles list. Within each iteration of the outer loop, we iterate over all the rest of the obstacles and check. This takes $O(n^2)$.

2. Creating all possible edges between the obstacles, the start and the destination and checking if they intersect with each other. There is the outer loop where we iterates over each obstacle ("current-obstacle") in the obstacles list. Within each iteration of the outer loop, we iterate over all the rest

of the obstacles ("other obstacle") that we haven't iterate yet in the outer loop (we maintain a list of the obstacles from which we remove the current obstacle in the outer loop). For each vertex of "current-obstacle" we generate an edge with each vertex of the "other obstacle" for all obstacle in All-Other-Obstacles list. Therefore, the time complexity for generating the edges in the i iteration with $n - i$ obstacles in All-Other-Obstacles is $O(4 \cdot 4 \cdot n - i)$. For each potential edge, we check for intersections with all-obstacles. Therefore, the time complexity is $O(4 \cdot 4 \cdot n - i \cdot n)$.

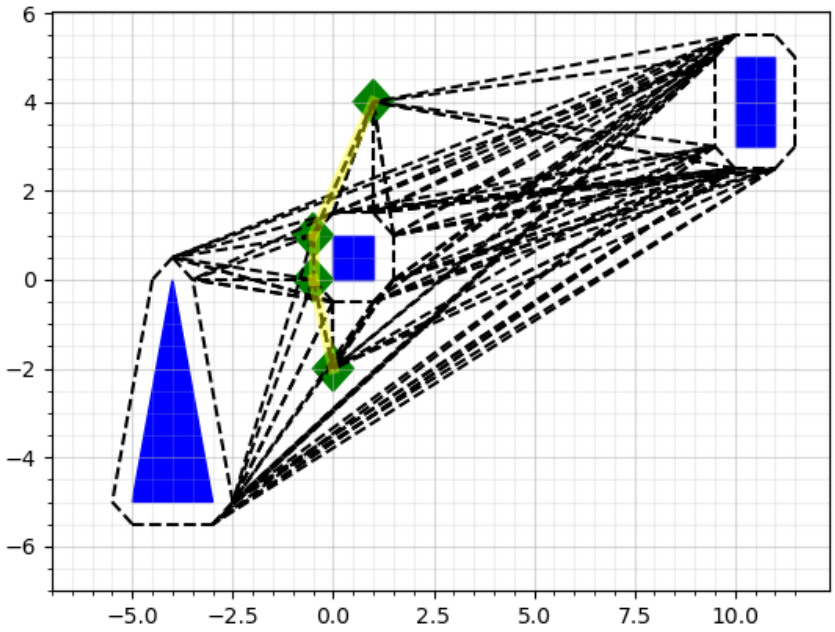
Overall Complexity: Considering the complexities of the outer and inner loops, the overall time complexity is $O(4 \cdot 4 \cdot \Sigma_{i=1}^{n} S_i \cdot n) = O(4 \cdot 4 \cdot n^2 \cdot n) = O(n^3)$. Where i is the i_{th} outer iteration with $n - i$ obstacles in All-Other-Obstacles list.

We know there is an implementation where the complexity of computing the visibility graph is $O(n^2 logn)$ but we haven't done it :(

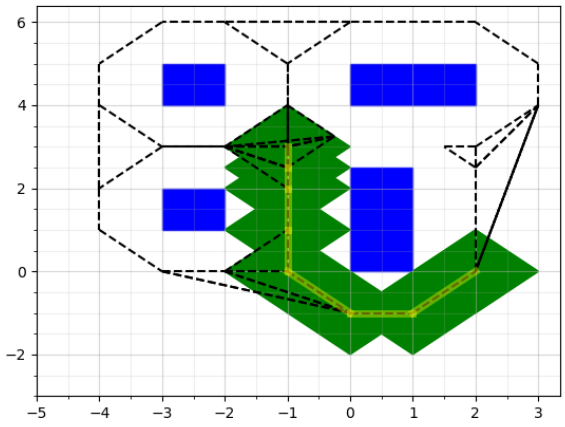
2.3 Query phase—computing shortest paths

Visualization of the C-space:

i) Visualization for the input files:



ii) Visualization for our input:



The computational complexity of our implementation:

The computational complexity of our implementation can be described as follows: In the query phase, we update the visibility graph to include the query and implement Dijkstra's algorithm to compute a shortest path between the start and the goal.
Let V be the number of vertices and E be the number of edges in the graph.

- Creating the visibility graph with source and destination - $O(n^3)$.
- Dijkstra's algorithm:
  - As a preprocess for the Dijkstra's algorithm we created an adjacency list from the edges which takes $O(E)$.
  - The time complexity of Dijkstra's algorithm is typically expressed as $O((V + E) \log V)$ due to the priority queue operations.

Therefore, the overall computational complexity of updating the visibility graph and computing shortest paths in the query phase is $O(n^3 + (V + E) \log V)$.