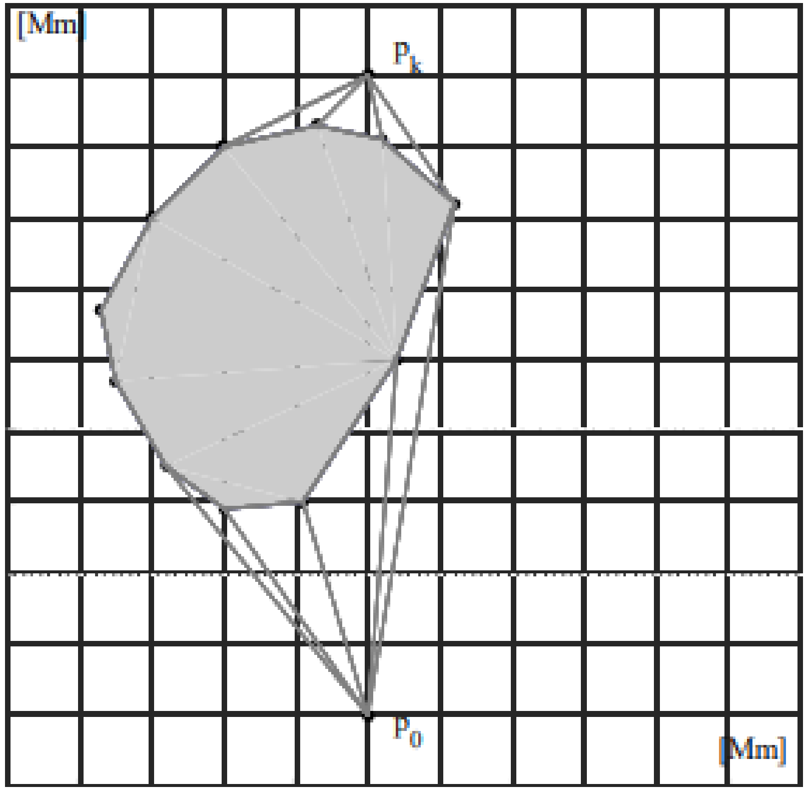
Obliczanie grafu widoczności

Bartosz Nowak
Iwo Szczepaniak

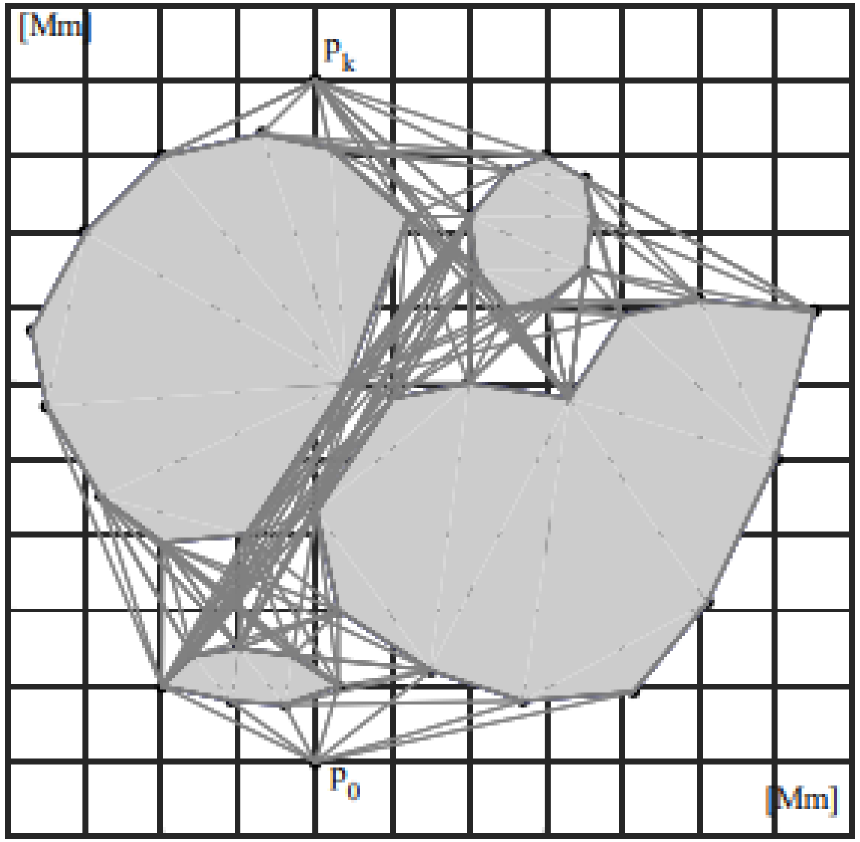
Graf widoczności - definicja



Graf widoczności to graf złożony z pewnej liczby wierzchołków oraz krawędzi łączących wierzchołki „widzące się wzajemnie”. Wierzchołki „widzą się wzajemnie”, jeśli krawędź łącząca te wierzchołki nie przekracza żadnej z figur zadanych przez użytkownika. Wierzchołkami grafu widoczności są wierzchołki figur oraz punkt początkowy i punkt końcowy. Krawędziami grafu widoczności są także odcinki, stanowiące boki poszczególnych wielokątów. Definiowanie grafu widoczności polega na znalezieniu wszystkich par wierzchołków, które widzą się wzajemnie.

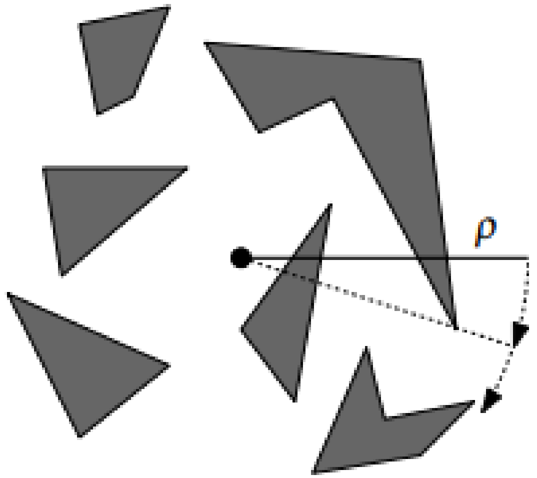
Naiwne podejście?

Naiwne podejście zatem dawałoby rozwiązanie n^3(dla każdego wierzchołka sprawdzenie n^2 możliwości).



Sortowanie obrotowe

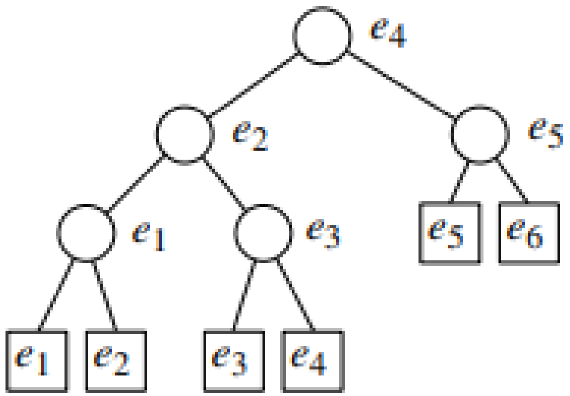
Istnieje jednak sposób na poprawienie tej złożoności poprzez zastosowanie obrotowego zamiatania. Stanem jest w tym przypadku uporządkowany ciąg krawędzi przecinanych przez półprostą, a zdarzeniami są wierzchołki figur(oraz punkty start i end).



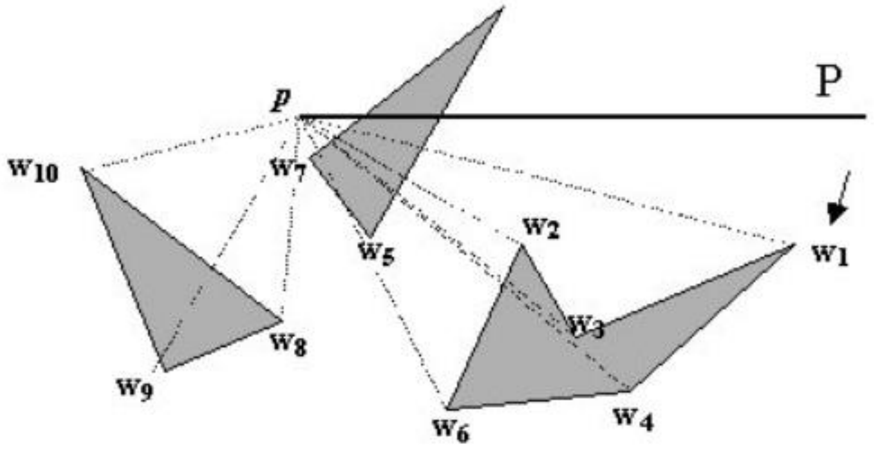
Struktura umożliwiająca poprawienie złożoności

```python
class EdgeSet:
    def __init__(self):
        self._open_edges = []

    def insert(self, v1, v2, edge):
        self._open_edges.insert(self._index(v1, v2, edge), edge)

    def delete(self, v1, v2, edge):
        index = self._index(v1, v2, edge) - 1
        if self._open_edges[index] == edge:
            del self._open_edges[index]

    def smallest(self):
        return self._open_edges[0]

    def _index(self, v1, v2, edge):
        lo = 0
        hi = len(self._open_edges)
        while lo < hi:
            mid = (lo + hi) // 2
            if cmp_edges(v1, v2, edge, self._open_edges[mid]):
                hi = mid
            else:
                lo = mid + 1
        return lo

    def __len__(self):
        return len(self._open_edges)

    def __getitem__(self, index):
        return self._open_edges[index]
```

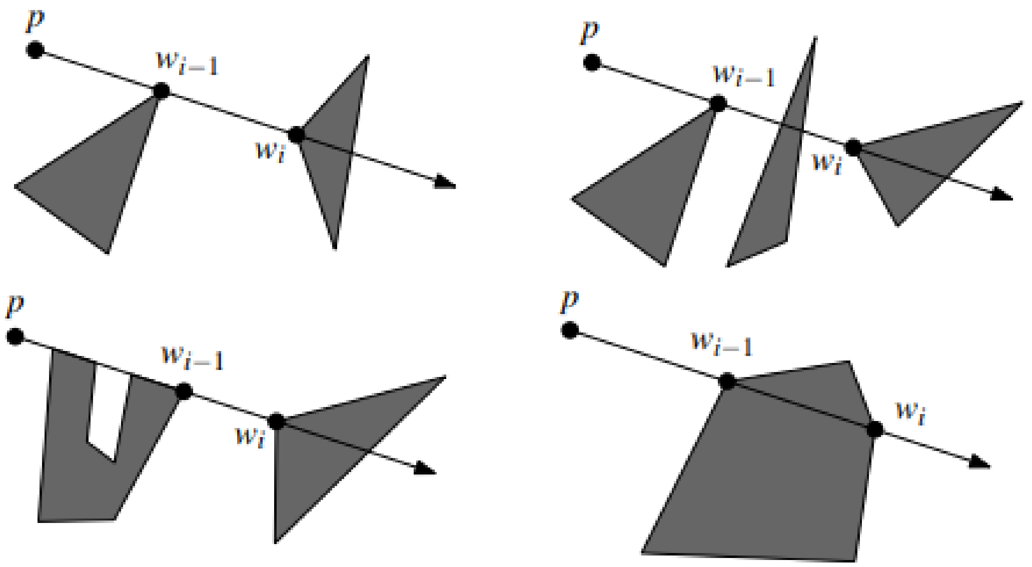
Stan jest reprezentowany przez drzewiastą strukturę EdgeSet zaimplementowaną w trig.py. Daje nam to porządek wzdłuż półprostej zamiatającej.



Zamiatanie



Zamiatanie rozpoczyna dla półprostej OX skierowanej dodatnio i przebiega w kierunku przeciwnym do ruchu wskazówek zegara. Jeśli wierzchołek widoczny – dodajemy nową krawędź do listy widocznych krawędzi. Następnie przechodzimy do kolejnego wierzchołka i usuwamy krawędzie których półprosta już nie przecina oraz dodajemy nowe przecięcia.



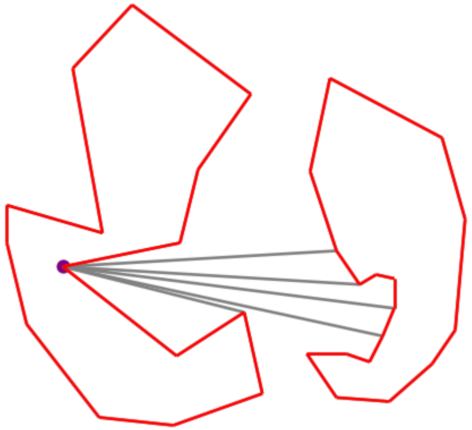
Co gdy wierzchołki są w linii?

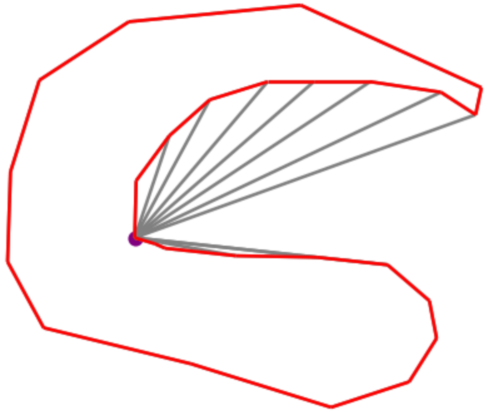
Na szczęście sortowanie wierzchołków, gdy są na linii zwraca te bliższe jako pierwsze, więc rozważając odcinek wi możemy wykorzystać wiedzę o wierzchołku wi-1.

Zauważmy, że jeśli wi-1 nie jest widzialny, to wi nie może być widzialny. Żeby wi był widzialny, wi-1 musi być widzialny, ale nie daje to gwarancji widoczności wi. Gdy wi-1 jest widoczny, to wi może być niewidoczny na dwa sposoby – albo odcinek wi-1 wi jest wnętrzem figury do której należą oba te wierzchołki, albo między nimi znajduje się figura, która odcinek wi-1 wi przecina

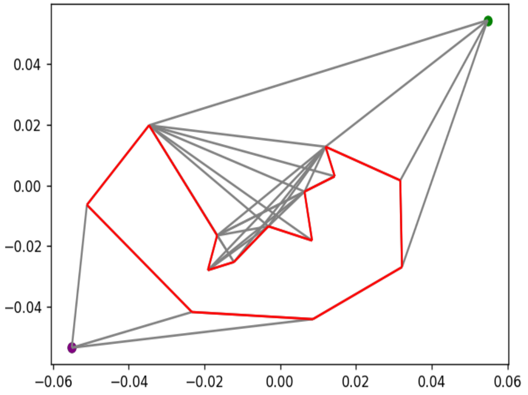
Widoczne krawędzie

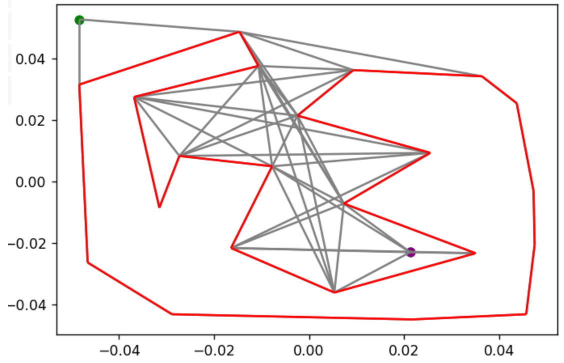
Dzięki wykorzystaniu zamiatania i drzewiastej struktury przetrzymującej wierzchołki oraz sortowaniu obrotowemu jesteśmy w stanie uzyskać krawędzie widoczne z dowolnego punktu.

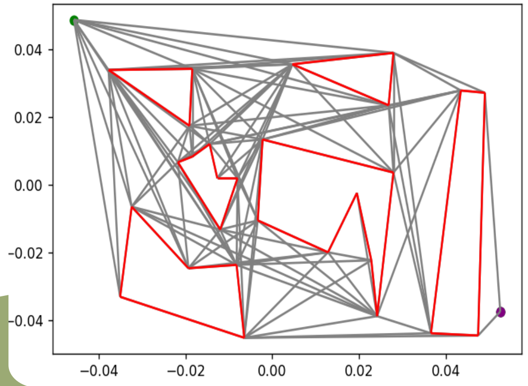


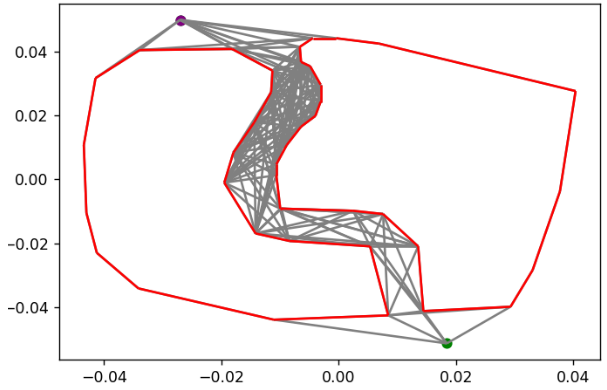


Jedna pętla





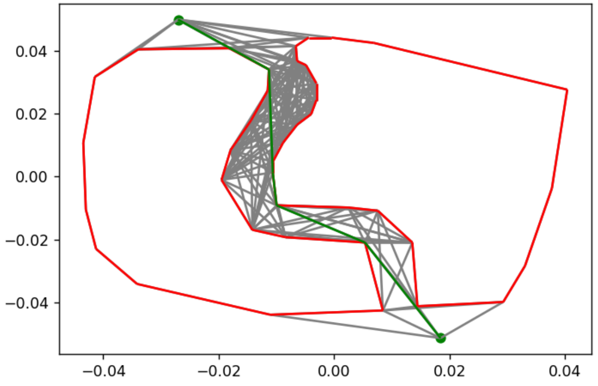


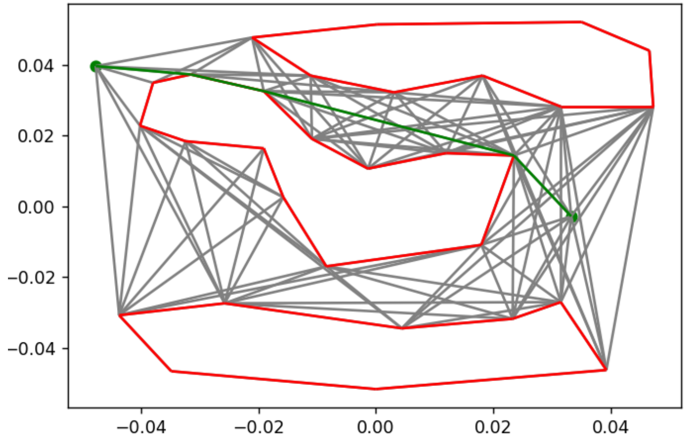


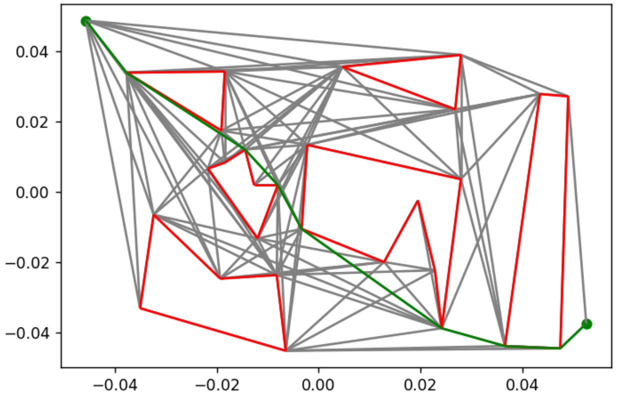
Skoro potrafimy sprawdzić widoczność z dowolnego wierzchołka, to przechodząc pętlą po wszystkich wierzchołkach otrzymamy graf widoczności.

Ostatni krok – najkrótsza ścieżka

Do ostatniego kroku wykorzystano algorytm Dijkstry zwracający najkrótszą ścieżkę w zadanym grafie.







Bibliografia

- Geometria obliczeniowa. Algorytmy i zastosowania - M. Berg, M. Kreveld, M. Overmars, O.Schwarzkopf
- https://www.science.smith.edu/~istreinu/Teaching/Courses/274/Spring98/Projects/Philip/fp/algVisibility.htm
- https://sj.umg.edu.pl/sites/default/files/ZN501.pdf
- https://www.geeksforgeeks.org/check-if-two-given-line-segments-intersect/