**Przetwarzanie i przechowywanie opisu siatki trójkątnej na płaszczyźnie**

1. **Dane techniczne:**

W eksperymencie skorzystano z następujących narzędzi:

- Język programowania: Python 3.10.12

- System operacyjny: Linux Mint 21 x86 64 (kernel: 5.15.0-91-generic)

- CPU: 8-rdzeniowy Intel i7-6700HQ 3.500GHz

- GPU: Intel HD Graphics 530 + NVIDIA GeForce GTX 960M

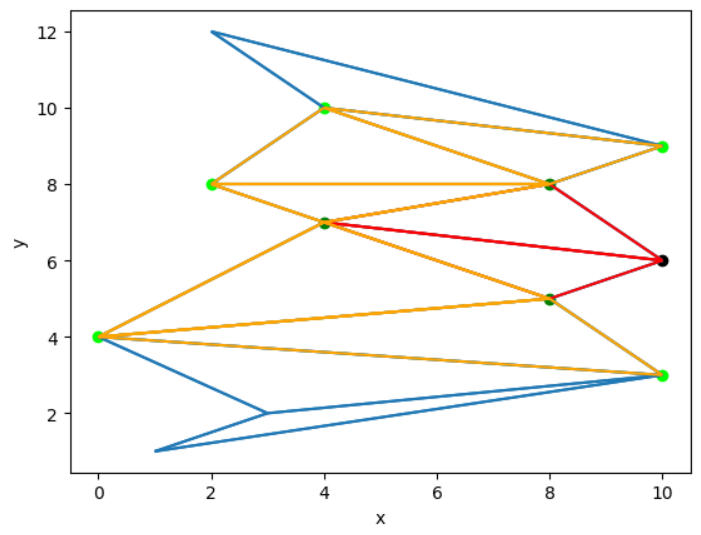
- RAM: 16 GB

1. **Opis programu:**

Program zawiera funkcje do wizualizacji początkowej triangulacji oraz funkcje wykonujące operacje na danej triangulacji, a następnie wizualizujące ich wyniki. Na początku operacje są wykonywane na nieprzetworzonej triangulacji. Wykorzystane algorytmy to:

**Do wyznaczania otoczenia dla wybranego wierzchołka**

Funkcja find\_neighbours rozpoczyna się od inicjalizacji zbiorów neighbors\_one\_layer i neighbors\_two\_layers do przechowywania incydentnych wierzchołków na pierwszej warstwie i wierzchołków na drugiej warstwie. Następnie algorytm iteruje przez wszystkie trójkąty w triangulacji. Jeśli wierzchołek vertex\_index należy do trójkąta, to dodaje wszystkie wierzchołki tego trójkąta do neighbors\_one\_layer (z wyłączeniem vertex\_index). Równocześnie, algorytm wizualizuje krawędzie trójkąta na czerwono. Dla każdego wierzchołka z neighbors\_one\_layer iteruje ponownie przez wszystkie trójkąty w triangulacji. Jeśli sąsiedni wierzchołek nie należy do trójkąta i vertex\_index nie należy do trójkąta, to dodaje wierzchołki tego trójkąta do neighbors\_two\_layers. Równocześnie, algorytm wizualizuje krawędzie trójkąta na pomarańczowo. Usuwa także powtarzające się wierzchołki w neighbours\_two\_layers, które są już zawarte w pierwszym zbiorze. Oznaczamy wybrany punkt na czarno, wierzchołki w pierwszej warstwie na zielono a wierzchołki w drugiej warstwie na jasno-zielono.

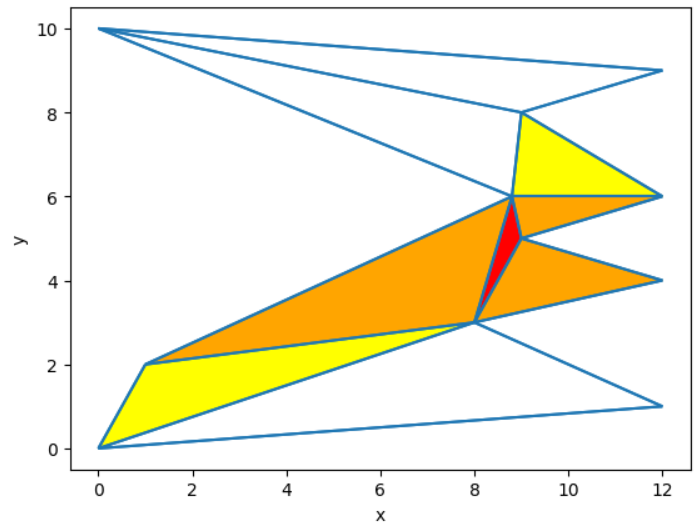


Rysunek 1: Przedstawienie obu warstw incydentnych wierzchołków.

Obie pętle iterują poprzez wszystkie trójkąty, a liczba trójkątów jest proporcjonalna do ilości wierzchołków. Druga pętla iteruje także po wszystkich wierzchołkach w pierwszej warstwie, których ilość w najgorszym przypadku też może być proporcjonalna do ilości wierzchołków. Zatem złożoność tego algorytmu to O(n^2) gdzie n to ilość wierzchołków.

**Do wyznaczania otoczenia dla wybranego trójkąta**

Funkcja find\_triangle\_neighbors służy do znajdowania sąsiednich trójkątów dla danego trójkąta w triangulacji. Algorytm inicjalizuje dwa zbiory: neighbors\_one\_layer dla jednowarstwowych sąsiadów i neighbors\_two\_layers dla dwuwarstwowych sąsiadów. Algorytm Iteruje przez wszystkie trójkąty w triangulacji. Jeśli trójkąt nie jest równy selected\_triangle i sąsiaduje z nim (co jest sprawdzane za pomocą funkcji are\_triangles\_incident), to dodaje go do neighbors\_one\_layer. Wizualizuje te trójkąty na ekranie kolorem pomarańczowym. Dla każdego trójkąta z neighbors\_one\_layer, iteruje ponownie przez wszystkie trójkąty w triangulacji. Jeśli trójkąt nie jest równy ani selected\_triangle, ani trójkątowi z neighbors\_one\_layer, i sąsiaduje z tym trójkątem, to dodaje go do neighbors\_two\_layers. Wizualizuje te trójkąty na ekranie kolorem żółtym. Kolor wybranego trójkąta jest zaznaczony kolorem czerwonym.

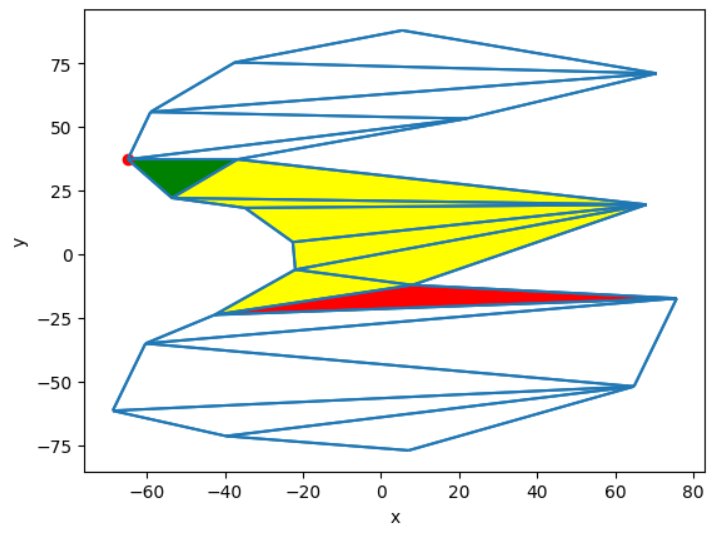


Rysunek 2: Przedstawienie obu warstw incydentnych trójkątów.

Algorytm analogicznie jak w poprzednim przykładzie jest ograniczony poprzez ilość trójkątów oraz ilość trójkątów w pierwszej warstwie. Tym razem jednak ilość trójkątów w pierwszej incydentnej warstwie to maksymalnie 3, więc złożoność algorytmu to O(n).

**Do przeglądania incydentnych trójkątów od wybranego trójkąta w kierunku wybranego punktu**

Algorytm zaczyna od trójkąta start\_triangle oraz punktu docelowego target\_point. Inicjalizuje także zmienną pomocniczą current\_triangle na wartość start\_triangle. Następnie wchodzi do pętli, która będzie powtarzana, dopóki current\_triangle nie zostanie znalezione, lub zostanie odwiedzony już wcześniej. Jeśli punkt target\_point znajduje się w trójkącie current\_triangle, algorytm zwraca ten trójkąt jako wynik. Dla trójkąta current\_triangle znajduje sąsiadujące trójkąty (takie, które mają przynajmniej 2 wspólne wierzchołki z current\_triangle). Spośród sąsiadujących trójkątów wybiera te, które nie zostały jeszcze odwiedzone, i ustawia current\_triangle na pierwszy z nich. Wizualizuje aktualną ścieżkę na ekranie, malując trójkąty na ścieżce na kolor żółty, trójkąt zawierający punkt na kolor zielony oraz trójkąt i punkt początkowy na kolor czerwony.



Rysunek 3: Przedstawienie drogi algorytmu do znalezienia punktu.

Algorytm przechowuje informacje o tym które trójkąty zostały już odwiedzone więc w najgorszym przypadku jego złożoność będzie zależeć od ilości trójkątów czyli O(n).