

# Przetwarzanie i przechowywanie opisu siatki trójkątnej na płaszczyźnie

## Mateusz Zając, Błażej Kapkowski

### Opis problemu

Zadanie polegało na porównaniu wydajności dwóch struktur służących do przechowywania opisu siatki trójkątnej na płaszczyźnie. Te struktury to:

- lista współrzędnych wierzchołków i lista trójkątów – czyli trójek indeksów wierzchołków, które tworzą dany trójkąt,
- Half Edge data structure.

Na obu strukturach wykonano trzy operacje:

- wyznaczenie otoczenia dla wybranego wierzchołka,
- wyznaczenie otoczenia dla wybranego trójkąta,
- przeglądanie sąsiednich trójkątów w celu znalezienia trójkąta zawierającego dany wierzchołek.

Porównano ich wydajność obliczeniową i pamięciową.

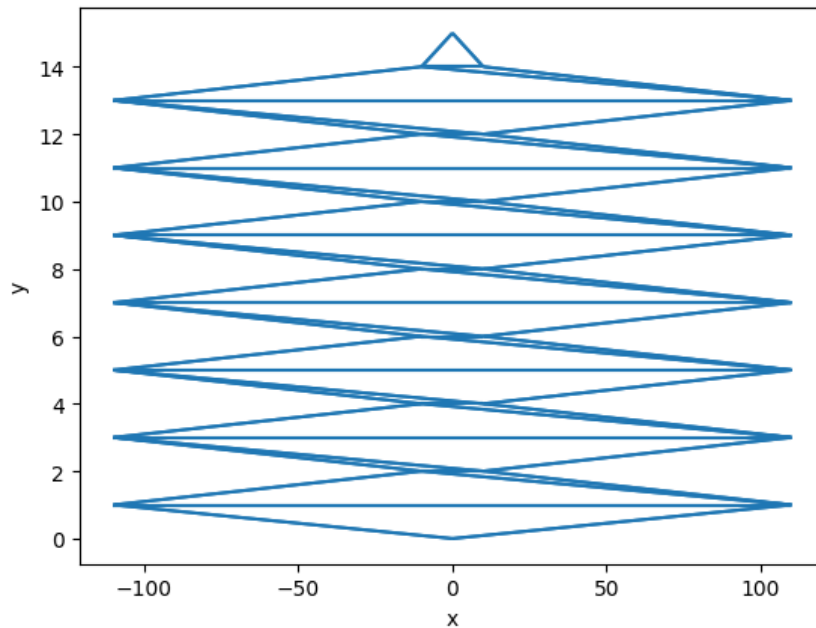
### Dane techniczne

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
- do wizualizacji wykorzystano program napisany przez KN BIT
- zewnętrzne biblioteki Pythona: memory-profiler, numpy, pandas, matplotlib

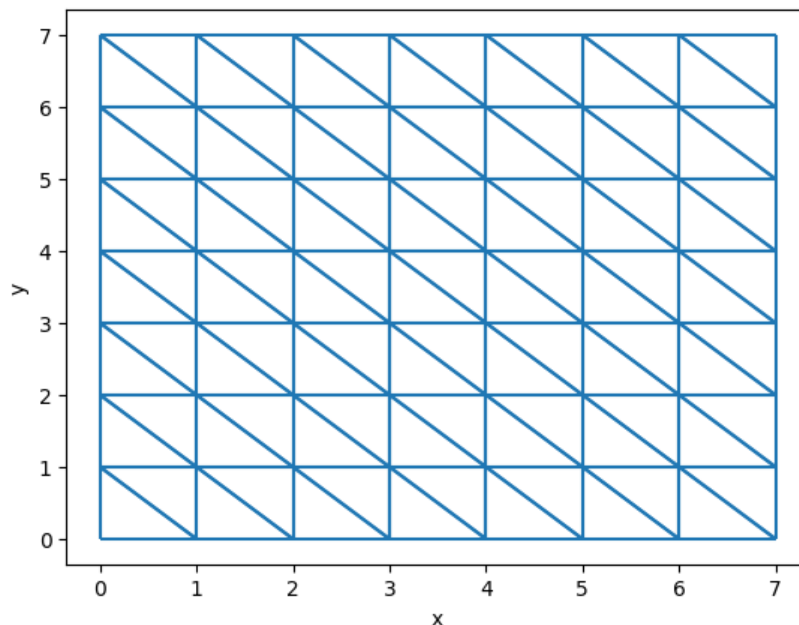
## Rodzaje testów

Testy wykonano na dwóch rodzajach siatek. Wybrano je tak, by można było łatwo generować siatki o zadanej wielkości. Na potrzeby sprawozdania, nazwano je Siatkami typu 1 i Siatkami typu 2. Siatki typu 1 zbudowane są z symetrycznych lub prawie symetrycznych wielokątów, które mają po  $(2n - 2) / 2$  wierzchołków na prawo i na lewo od osi symetrii, a także po 1 punkcie na szczycie i na dole. Wielokąty te poddano triangulacji za pomocą funkcji napisanej na potrzeby wcześniejszych laboratoriów.



*Ilustracja 1: wizualizacja pierwszego typu siatek*

Siatki typu 2 powstały z podzielenia każdego pola siatki kwadratowej (o polach z bokami o długości 1) na dwa trójkąty. Ten typ siatek zastosowano, bo średnia ilość wierzchołków i trójkątów w otoczeniu, dla wyższych warstw, będzie większa, tak więc obliczenia powinny trwać dłużej. Ponadto, wygenerowanie takich siatek zajmuje mniej czasu, nawet gdy są duże – bo niepotrzebne jest obliczanie triangulacji.



*Ilustracja 2: wizualizacja drugiego typu siatek*

# Wydajność obliczeniowa – siatki typu 1

Porównano wydajność obliczeniową obu struktur na pierwszym typie siatek. W ramach przekroju, przedstawiono wyniki testów dla każdej z trzech operacji. Dla pierwszej i drugiej operacji, przedstawiono wyniki testów dla szukanej 1 warstwy wierzchołków/trójkątów, a także dla 2 warstw. Wartości w tabelach podane są w sekundach.

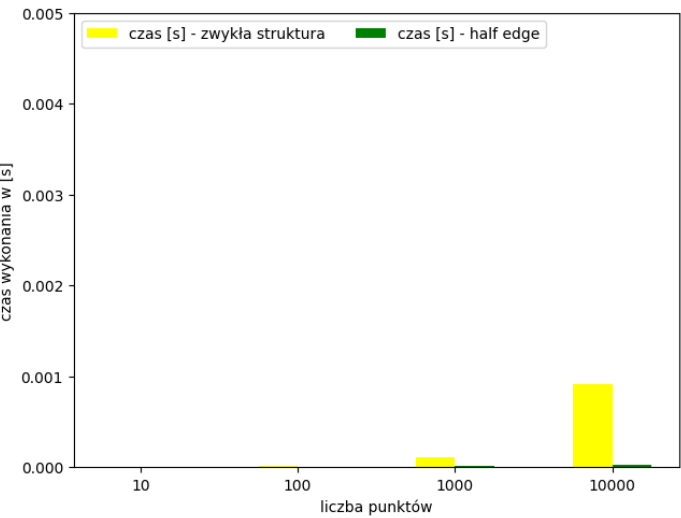


Diagram 1: pomiar czasu wykonania 1. operacji,  
1 warstwa sąsiednich wierzchołków

	czas [s] - zwykła struktura	czas [s] - half edge	liczba wierzchołków
0	0.00001	0.00001	10
1	0.00002	0.00001	100
2	0.00011	0.00001	1000
3	0.00092	0.00002	10000

Tabela 1: pomiar czasu wykonania 1. operacji,  
1 warstwa sąsiednich wierzchołków

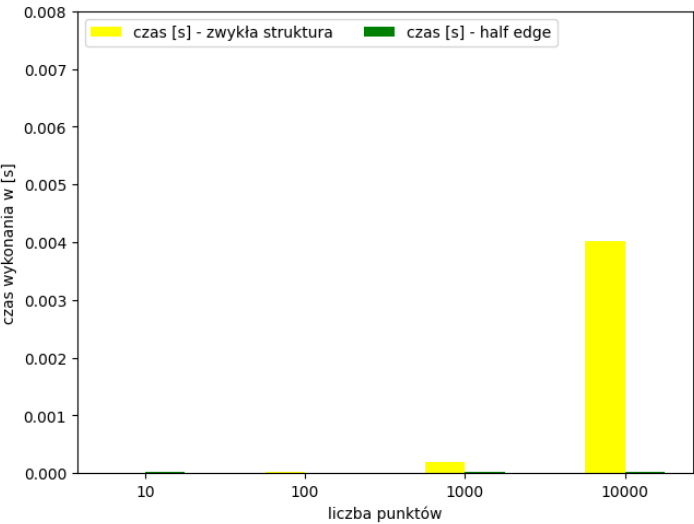


Diagram 2: pomiar czasu wykonania 1. operacji,  
2 warstwy sąsiednich wierzchołków

	czas [s] - zwykła struktura	czas [s] - half edge	liczba wierzchołków
0	0.00001	0.00002	10
1	0.00003	0.00001	100
2	0.00019	0.00003	1000
3	0.00402	0.00002	10000

Tabela 2: pomiar czasu wykonania 1. operacji,  
2 warstwy sąsiednich wierzchołków

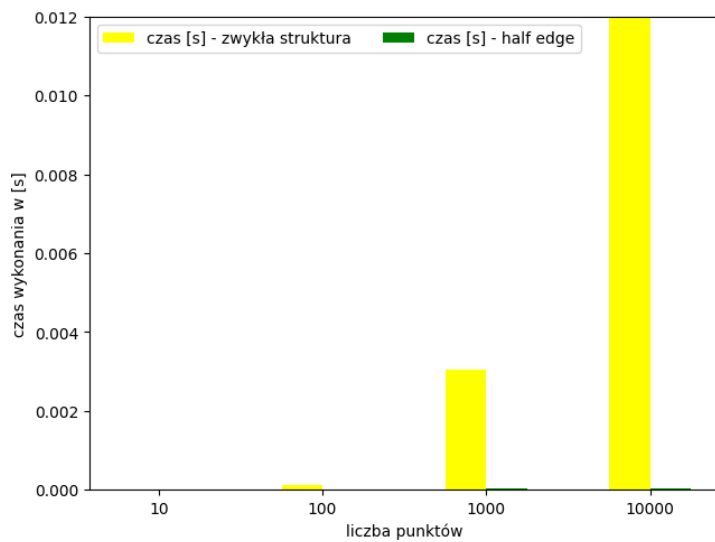


Diagram 3: pomiar czasu wykonania 2. operacji,  
1 warstwa sąsiednich trójkątów

	czas [s] - zwykła struktura	czas [s] - half edge	liczba wierzchołków
0	0.00002	0.00001	10
1	0.00012	0.00001	100
2	0.00304	0.00003	1000
3	0.01880	0.00002	10000

Tabela 3: pomiar czasu wykonania 2. operacji,  
1 warstwa sąsiednich trójkątów

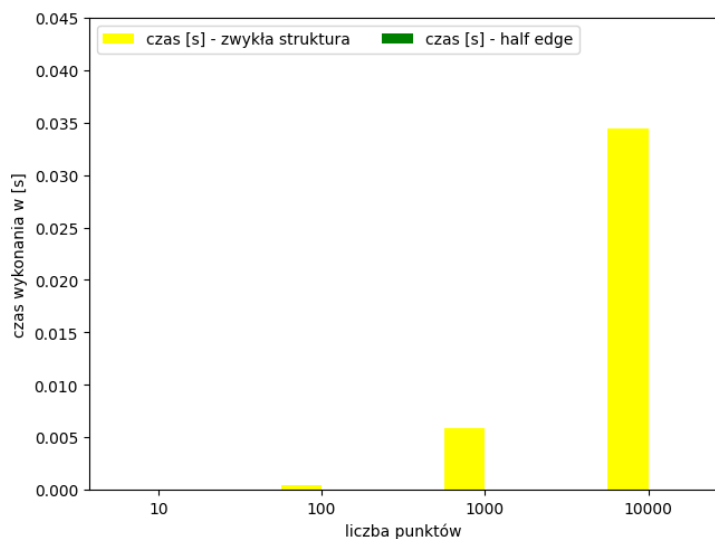


Diagram 4: pomiar czasu wykonania 2. operacji,  
2 warstwy sąsiednich trójkątów

	czas [s] - zwykła struktura	czas [s] - half edge	liczba wierzchołków
0	0.00004	0.00005	10
1	0.00046	0.00004	100
2	0.00592	0.00004	1000
3	0.03446	0.00002	10000

Tabela 4: pomiar czasu wykonania 2. operacji,  
2 warstwy sąsiednich trójkątów

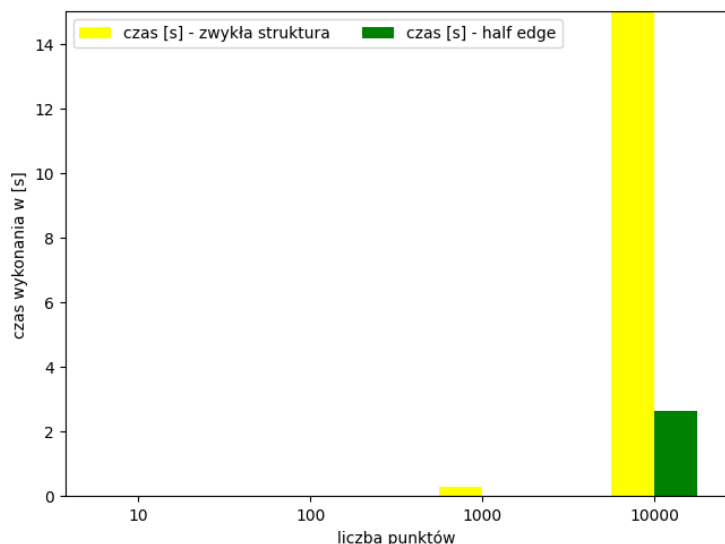


Diagram 5: pomiar czasu wykonania 3. operacji,  
przeszukiwanie trójkątów w poszukiwaniu wierzchołka

	czas [s] - zwykła struktura	czas [s] - half edge	liczba wierzchołków
0	0.00002	0.00001	10
1	0.00636	0.00008	100
2	0.29234	0.01549	1000
3	16.55994	2.63428	10000

Tabela 5: pomiar czasu wykonania 3. operacji,  
przeszukiwanie trójkątów w poszukiwaniu wierzchołka

Wyniki pomiarów przedstawione na powyższych ilustracjach (Diagramy i Tabele 1-5) wskazały, że jeśli chodzi o szybkość obliczeń, struktura Half Edge sprawdza się dużo lepiej. Najbardziej uwypuklają to Diagramy 2-4, gdzie praktycznie nie widać słupków odpowiadających pomiarom czasu dla Half Edge. Jak pokazuje Diagram 5, różnica zaczyna mieć szczególne znaczenie przy siatkach złożonych w większej liczby punktów – różnica w czasach obliczeń potrafiła osiągnąć prawie 9 sekund.

## Wydajność obliczeniowa – siatki typu 2

Przeprowadzono testy dla siatek o większej liczbie sąsiadujących ze sobą trójkątów. Po to, by uwypuklić różnicę między wydajnościami obu struktur. Wybrano te zbiory, dla których różnice były największe.

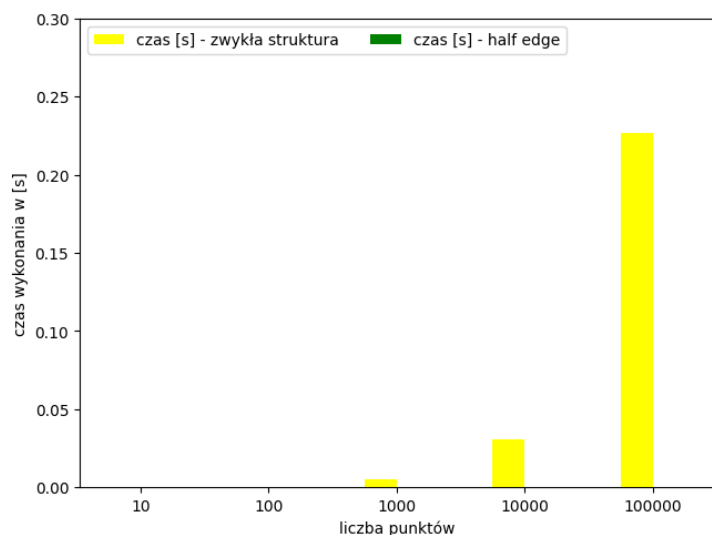


Diagram 6: pomiar czasu wykonania 2. operacji,  
2 warstwy sąsiednich trójkątów

	czas [s] - zwykła struktura	czas [s] - half edge	liczba wierzchołków
0	0.00003	0.00001	10
1	0.00016	0.00002	100
2	0.00527	0.00001	1000
3	0.03080	0.00001	10000
4	0.22667	0.00003	100000

Tabela 6: pomiar czasu wykonania 2. operacji,  
2 warstwy sąsiednich trójkątów

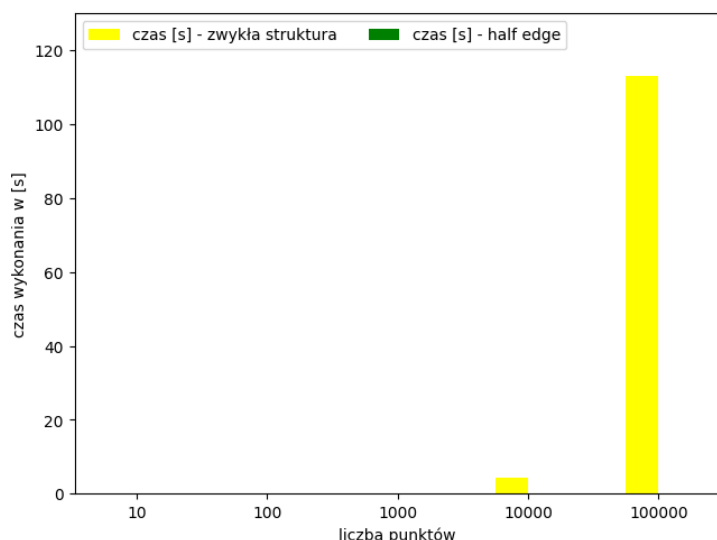


Diagram 7: pomiar czasu wykonania 3. operacji,  
przeszukiwanie trójkąta w poszukiwaniu wierzchołka

	czas [s] - zwykła struktura	czas [s] - half edge	liczba wierzchołków
0	0.00006	0.00001	10
1	0.00428	0.00002	100
2	0.13998	0.00002	1000
3	4.39678	0.00003	10000
4	113.10611	0.00004	100000

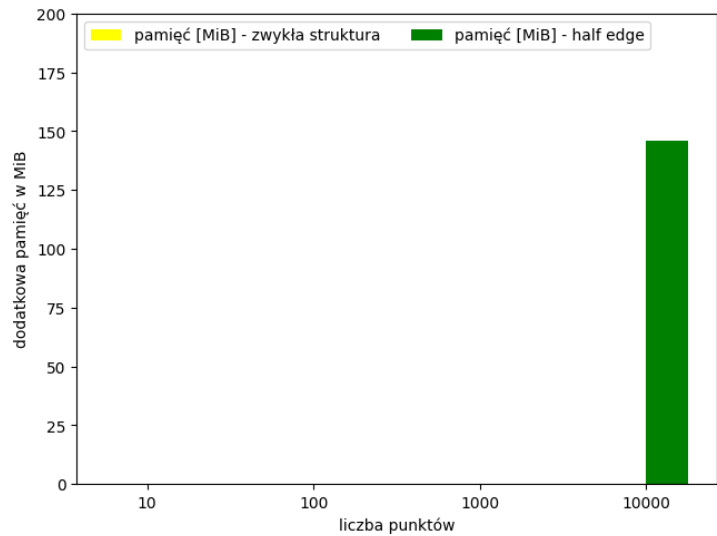
Tabela 7: pomiar czasu wykonania 3. operacji,  
przeszukiwanie trójkąta w poszukiwaniu wierzchołka

Różnice między wydajnościami operacji wzrosły tak, jak oczekiwano. W przypadku drugiej operacji (Tabela 6), dla największej liczby wierzchołków, czas obliczeń dla prostej struktury danych był około 7555 razy większy, niż dla Half Edge. Stosunek był jeszcze większy dla trzeciej operacji (Tabela 7). Dla 100 tysięcy wierzchołków, znalezienie trójkąta

zawierającego pewien wierzchołek zajęło ponad 113 sekund, a wykorzystanie Half Edge skróciło ten czas do ułamku sekundy.

### Wydajność pamięciowa – siatki typu 1

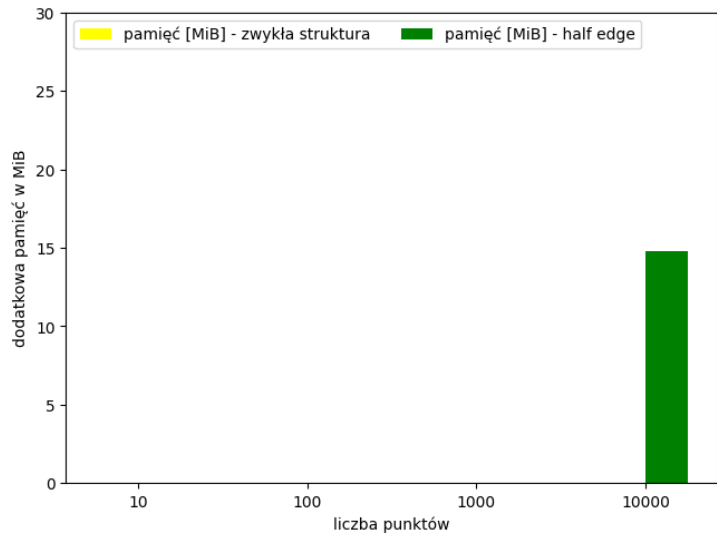
Testy wykonano za pomocą biblioteki memory-profiler. Funkcja memory\_usage() z tejże biblioteki czyta co pewien ułamek sekundy rozmiar pamięci wykorzystywanej w tej chwili. W każdej takiej próbkę znajduje się pewna wartość maksymalna i minimalna. W wynikach przedstawiono ich różnice. Dla przekroju, przedstawiono wyniki wszystkich pomiarów.



	pamięć [MiB] - zwykła struktura	pamięć [MiB] - half edge	liczba wierzchołków
0	0.00000	0.00000	10
1	0.00000	0.00000	100
2	0.00000	0.00000	1000
3	0.00000	145.84375	10000

Diagram 8: pomiar pamięci używanej przez 1. operację, 1 warstwa sąsiednich wierzchołków

Tabela 8: pomiar pamięci używanej przez 1. operację, 1 warstwa sąsiednich wierzchołków



	pamięć [MiB] - zwykła struktura	pamięć [MiB] - half edge	liczba wierzchołków
0	0.00000	0.00000	10
1	0.00000	0.00000	100
2	0.00000	0.00000	1000
3	0.00000	14.78125	10000

Diagram 9: pomiar pamięci używanej przez 1. operację, 2 warstwy sąsiednich wierzchołków

Tabela 9: pomiar pamięci używanej przez 1. operację, 2 warstwy sąsiednich wierzchołków

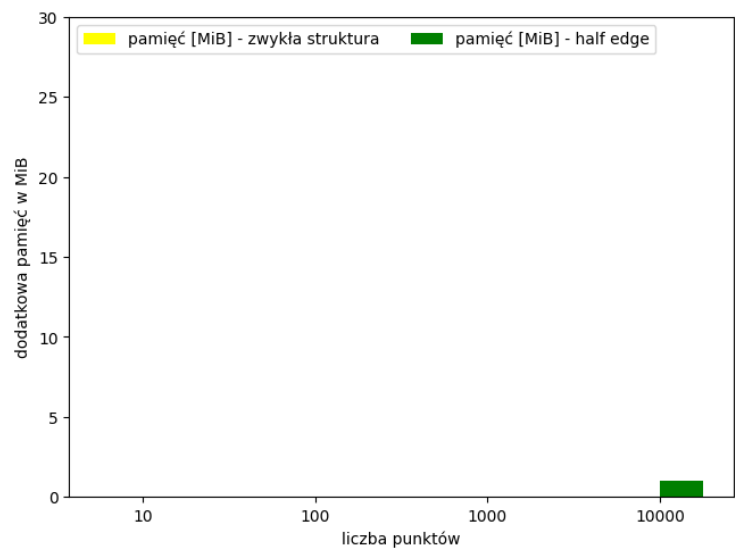


Diagram 10: pomiar pamięci używanej przez 2. operację,  
1 warstwa sąsiednich trójkątów

	pamięć [MiB] - zwykła struktura	pamięć [MiB] - half edge	liczba wierzchołków
0	0.00000	0.00000	10
1	0.00000	0.00000	100
2	0.00000	0.01562	1000
3	0.00000	0.98438	10000

Tabela 10: pomiar pamięci używanej przez 2. operację,  
1 warstwa sąsiednich trójkątów

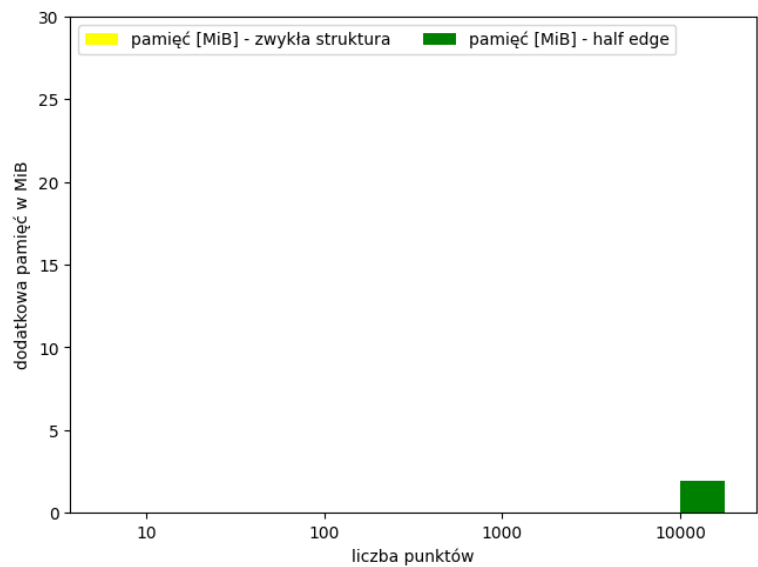


Diagram 11: pomiar pamięci używanej przez 2. operację,  
2 warstwy sąsiednich trójkątów

	pamięć [MiB] - zwykła struktura	pamięć [MiB] - half edge	liczba wierzchołków
0	0.00000	0.00000	10
1	0.00000	0.00000	100
2	0.00000	0.00000	1000
3	0.00000	1.96875	10000

Tabela 11: pomiar pamięci używanej przez 2. operację,  
2 warstwy sąsiednich trójkątów

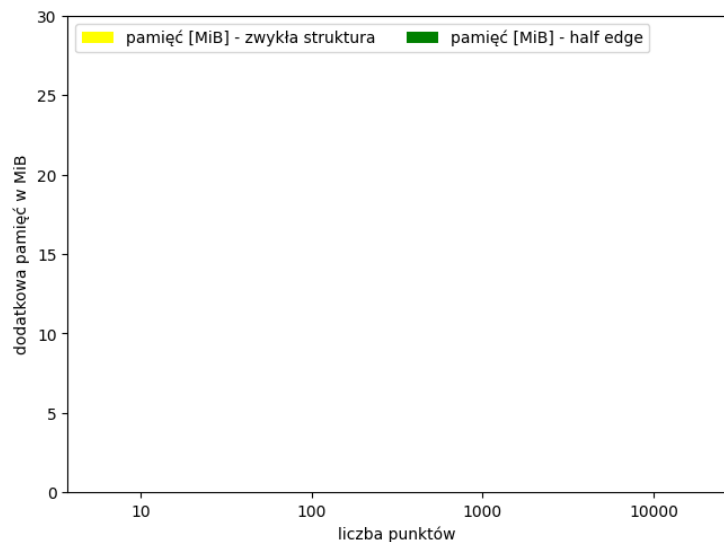


Diagram 12: pomiar pamięci używanej przez 3. operację, przeszukiwanie sąsiednich trójkątów w poszukiwaniu wierzchołka

	pamięć [MiB] - zwykła struktura	pamięć [MiB] - half edge	liczba wierzchołków
0	0.00000	0.00000	10
1	0.00000	0.00000	100
2	0.00000	0.00000	1000
3	0.00000	0.00000	10000

Tabela 12: pomiar pamięci używanej przez 3. operację, przeszukiwanie sąsiednich trójkątów w poszukiwaniu wierzchołka

Średnio, przy większych siatkach, operacje na strukturze Half Edge zajmowały więcej pamięci. Pokazują to Diagramy i Tabele 1-4. W przypadku mniejszych siatek, różnica była zbyt mała, by zwrócić ją funkcja biblioteczna używana do mierzenia zajmowanej pamięci. Funkcja zwracała wartości w MiB.

Trudno powiedzieć, co mogło spowodować, że wartości w Tabeli 5 są równe zeru. Mogło to być spowodowane przeoczoną błędem, jednak w późniejszej części sprawozdania sprawdzono tą samą operację na większych siatkach. Wtedy ten artefakt się nie pojawił. Więc możliwe, że wpływ mogło mieć zarządzanie pamięci przez notatniki Jupytera. Mimo, że podjęto starań, by uniknąć takich sytuacji.

Podobnie dziwnym zjawiskiem jest, że pierwsza operacja, przy wyznaczaniu 1 warstwy, zajęła tak dużo pamięci (prawie 146 MiB).

## Wydajność pamięciowa – siatki typu 2

Żeby uwypuklić różnice w wykorzystywanej pamięci, operacje przeprowadzono także na większych siatkach. Poza tym, jako że testy wykonane zostały po testach dla poprzedniego typu, artefakty pochodzące z alokowania przez notatnik pamięci były mniej prawdopodobne. Choć trudno to stwierdzić bez głębszej znajomości implementacji bibliotek.

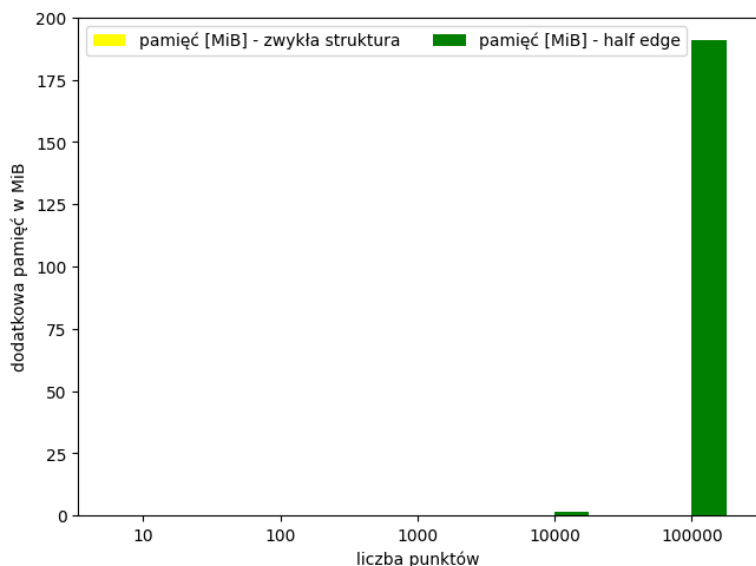


Diagram 13: pomiar pamięci używanej przez 1. operację, 1 warstwa sąsiednich wierzchołków

	pamięć [MiB] - zwykła struktura	pamięć [MiB] - half edge	liczba wierzchołków
0	0.00000	0.00000	10
1	0.00000	0.00000	100
2	0.00000	0.00000	1000
3	0.00000	1.31250	10000
4	0.00000	190.72266	100000

Tabela 13: pomiar pamięci używanej przez 1. operację, 1 warstwa sąsiednich wierzchołków



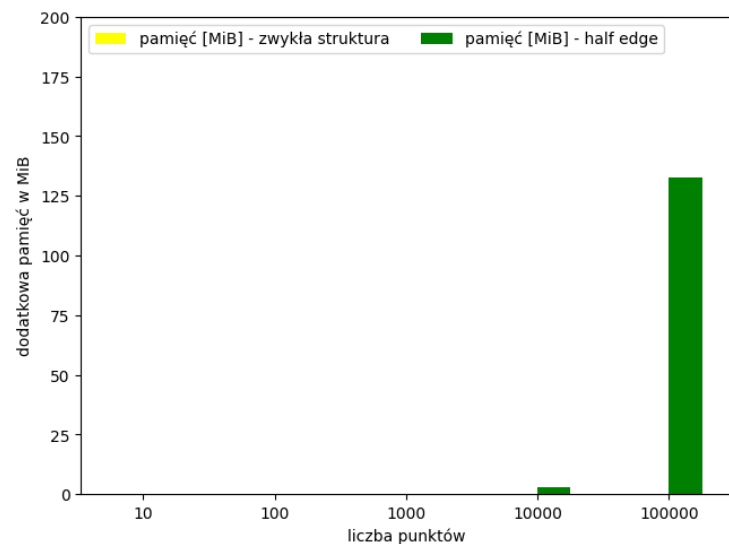


Diagram 14: pomiar pamięci używanej przez 2. operację, 1 warstwa sąsiednich trójkątów

	pamięć [MiB] - zwykła struktura	pamięć [MiB] - half edge	liczba wierzchołków
0	0.00000	0.00391	10
1	0.00000	0.00000	100
2	0.00000	0.00000	1000
3	0.00000	3.12500	10000
4	0.00000	132.49219	100000

Tabela 14: pomiar pamięci używanej przez 2. operację, 1 warstwa sąsiednich trójkątów

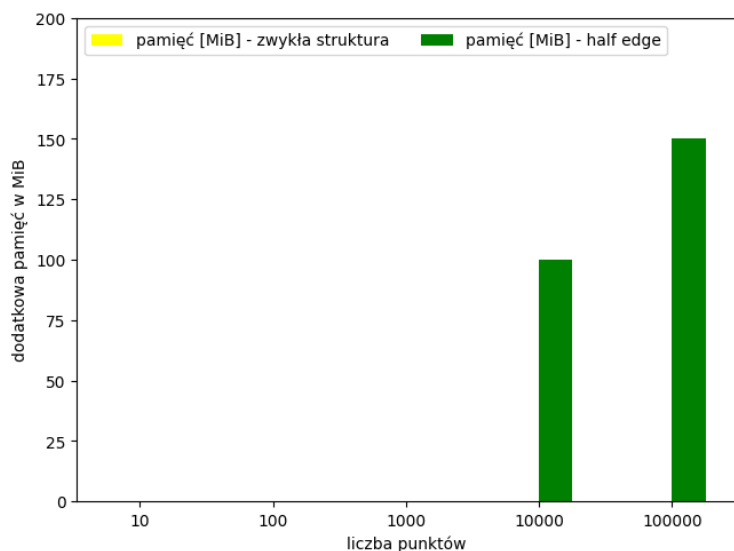


Diagram 15: pomiar pamięci używanej przez 3. operację, przeszukiwanie sąsiednich trójkątów w poszukiwaniu wierzchołka

	pamięć [MiB] - zwykła struktura	pamięć [MiB] - half edge	liczba wierzchołków
0	0.00000	0.00000	10
1	0.00000	0.00000	100
2	0.00000	0.00391	1000
3	0.00000	99.95312	10000
4	0.00000	150.03125	100000

Tabela 15: pomiar pamięci używanej przez 3. operację, przeszukiwanie sąsiednich trójkątów w poszukiwaniu wierzchołka

Każda operacja zachowała się bardziej stabilnie, to jest – nie pojawiły się większe artefakty w wynikach pomiarów. Zauważalny jest trend zużywania większej ilości pamięci dla coraz większych siatek, jedynie w przypadku struktury Half Edge. To jest akurat zrozumiałe – struktura wymaga alokowania pamięci dla obiektów odpowiadających krawędziom, oraz przypisanych im wskaźnikom. Ilość pamięci używanej przez każdą operację się zwiększyła – dla tej samej liczby wierzchołków, było więcej trójkątów, a tym samym krawędzi do zapisania. Dla każdej z operacji, przy 100 tysiącach wierzchołków, ilość zużywanej pamięci mieściła się w przedziale 130-200 MiB.

## Podsumowanie

Dla wybranych do przetestowania operacji, wykorzystanie struktury Half Edge znacząco poprawia prędkość obliczeń. Wynika to z tego, że szukając sąsiednich wierzchołków/trójkątów, iteruje się tylko po wybranych, najbliższych krawędziach. Podczas gdy przy prostej strukturze danych, konieczne jest iterowanie po wszystkich trójkątach. Half Edge wymaga jednak dodatkowej pamięci, co przy wyjątkowo dużych siatkach może okazać się problemem. Mimo wszystko, zazwyczaj łatwiej o dużą ilość RAM-u w komputerze, niż o lepszy procesor, więc strata nie jest zbyt duża.

Wybór narzędzi do pomiaru pamięci miał spore znaczenie. Jedną z opcji, by uniknąć artefaktów, mogłoby być przeprowadzenie testów nie korzystając z notatnika Jupytera. Wtedy sposób, w jakim notatnik zarządza pamięcią, nie mógłby powodować żadnych problemów. Inną z opcji jest, by zamiast używać biblioteki memory-profiler, użyć funkcji `getsizeof()` z biblioteki `sys`, będącej w standardzie Pythona. To wymagałoby jednak dostosowania funkcji odpowiedzialnych za operacje, by za każdym razem mierzyć, ile pamięci zajmują zmienne wykorzystywane przez algorytm. Byłoby to nieco bardziej żmudne. Moduł Guppy3, służący do profilowania programów napisanych w Pythonie, też byłby opcją, ale trudniej było go wykorzystać w notatniku.

Struktura Half Edge jest szczególnie przydatna, gdy trzeba przeszukiwać więcej warstw sąsiednich trójkątów lub wierzchołków, co pokazały skrajnie duże różnice w wynikach pomiarów dla trzeciej operacji.

Generowanie siatek, generując wpierw wielokąty i licząc ich triangulacje, było skrajnie nieefektywnym podejściem, gdyż to była jedna z najpowolniejszych części programu służącego do testów, zaraz obok tej odpowiedzialnej za testowanie trzeciej operacji dla obu rodzajów siatek.

## Bibliografia

- <https://jerryin.info/geometry-processing-algorithms/half-edge/>
- <https://cs184.eecs.berkeley.edu/sp19/article/15/the-half-edge-data-structure>
- <https://stackoverflow.com/questions/15365471/initializing-half-edge-data-structure-from-vertices>
- <https://kaba.hilvi.org/homepage/blog/halfedge/halfedge.htm>
- Mark de Berg – „Computational Geometry – Algorithms and Applications”