Obliczanie nałożenia dwóch podziałów 2D

Algorytmy geometryczne

Michał Szafarczyk i Piotr Czarnik

9 stycznia 2022

Rozdział 1

Część techniczna

1.1 Opis programu

Program, napisany w języku `Python 3` z wykorzystaniem bibliotek `matplotlib`, `numpy`, `random`, składa się z 6 modułów:

- `classes.py` - zawiera definicje klas używanych w programie.
- `map_overlay_algorithm.py` - zawiera główne algorytmy wykorzystywane w programie, w tym algorytm obliczający nałożenie dwóch podziałów płaszczyzny.
- `faces_fix.py` - zawiera algorytmy naprawiający ściany z wykorzystaniem odpowiednich struktur również zdefiniowanych w tym pliku.
- `tests.py` - zawiera przykładowe testy i procedury dbające o odpowiednie ich uruchamianie.
- `map_input.py` - zawiera obsługę wprowadzania podziału płaszczyzny za pomocą interfejsu graficznego.
- `visualizer.py` - zmodyfikowane narzędzie graficzne do prezentowania wyników działania algorytmu.

Dla drzewiastych struktur danych użyto własnych (odpowiednio dostosowanych do potrzeb algorytmów) implementacji drzewa AVL, dla którego operacje wyciągania elementu z drzewa, wkładanie elementu do drzewa, znajdywanie poprzednika i następnika działają w czasie $O(\log n)$, gdzie n to liczba elementów w drzewie.

Kod programu zawiera dosyć obszerne opisy działania w komentarzach.

1.2 Wymagania techniczne

Program był uruchamiany na komputerze z systemem Windows 10 z ośmiordzeniowym procesorem AMD 7 Ryzen 3700X i 16 GB pamięci RAM, `Python 3.9` i na komputerze z systemem macOS 12.1 z dwurdzeniowym procesorem Intel i5 i 8 GB pamięci RAM, `Python 3.9`.

Rozdział 2

Część użytkownika

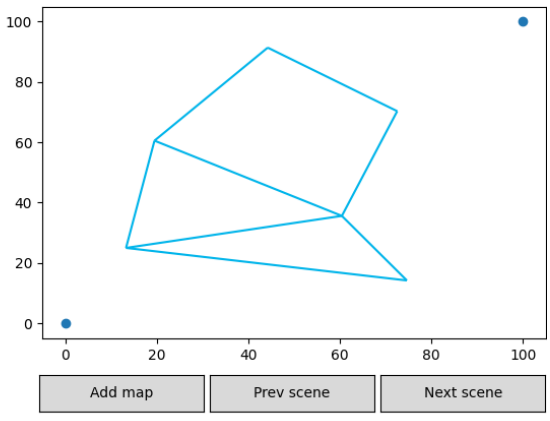
Program może być uruchomiony w dwóch różnych trybach:

- testowym, służył do testowania poprawności działania algorytmu. Można go uruchomić komendą:

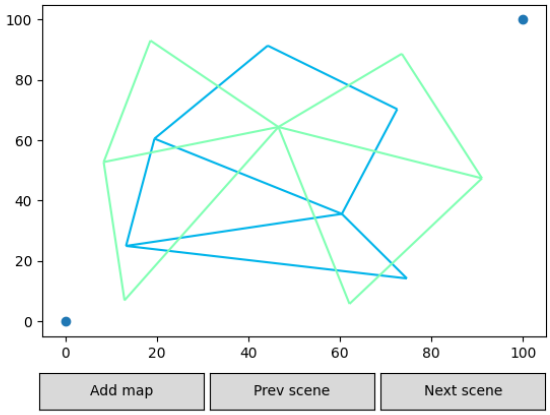
  `python3 test.py`
  lub w przypadku odmiennego sposobu zainstalowania `Pythona`
  `python test.py`

- użytkownika, w którym można wprowadzić własne dwa podziały płaszczyzny za pomocą intefejsu graficznego. Można go uruchomić komendą

  `python3 map_input.py`
  lub w przypadku odmiennego sposobu zainstalowania `Pythona`
  `python map_input.py`



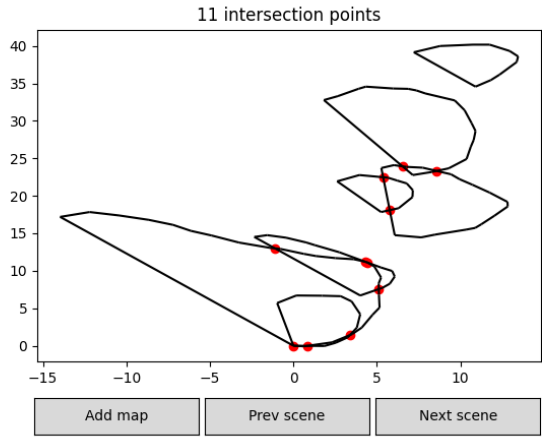
Rysunek 2.1: Podział 1. narysowany przy użyciu interfejsu graficznego.



Rysunek 2.2: Podział 2. narysowany przy użyciu interfejsu graficznego.

Po uruchomieniu wersji użytkownika, pojawi się puste okno z osiami współrzędnych. Aby narysować pierwszą mapę należy nacisnąć przycisk `Add map`. Za pomocą kliknięć lewym klawiszem myszki wyznacza się kolejne wierzchołki podziału

płaszczyzny. Kliknięcie w pobliżu innego wierzchołka spowoduje przyciągnięcie końca krawędzi do niego. Rysowania działa w trybie ciągłym - nowe krawędzie zaczynają się w ostatnio kliknięty punkcie. Prawym klawiszem myszki można wybrać inny punkt początkowy, klikając nim na wcześniej narysowany wierzchołek. Aby narysować drugą mapę, należy ponownie nacisnąć przycisk `Add map`. Po skończeniu rysowania należy zamknąć okno, w to miejsce pojawi się nowe okno z graficznymi wynikami obliczeń algorytmu. Aby zobaczyć inne przedstawienie wyników należy nacisnąć przycisk `Next scene`.



Rysunek 2.3: Naciśnięcie przycisku `Next scene` spowoduje przejście do kolejnego rodzaju przedstawienia wyników

Rysunek 2.4: Naciśnięcie przycisku `Prev scene` spowoduje przejście do poprzednio wyświetlanych wyników.

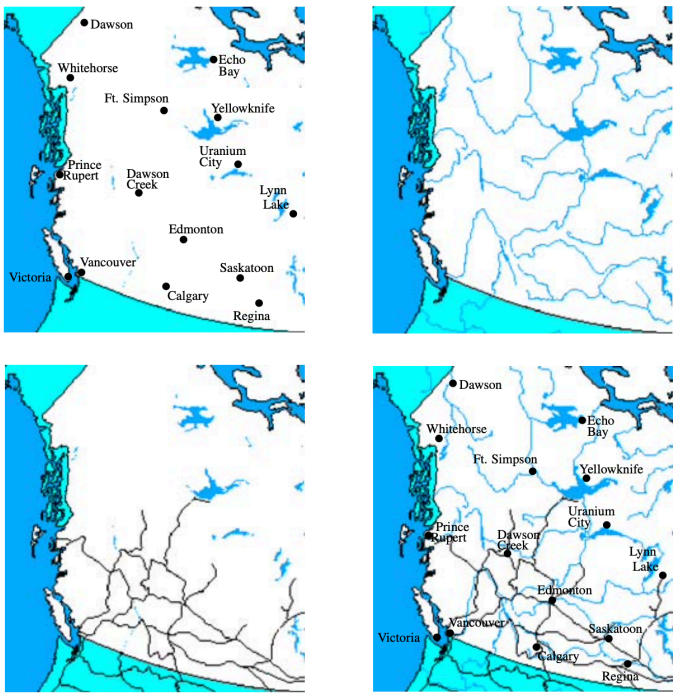
Tryb testowy działa podobnie do trybu użytkownika, oprócz faktu, że algorytm działa na wcześniej zadeklarowanych lub losowych danych. Aby zobaczyć kolejny zestaw danych należy zamknąć obecne okno - w to miejsce pojawi się nowe okno z nowymi danymi wejściowymi.

Rozdział 3

Sprawozdanie

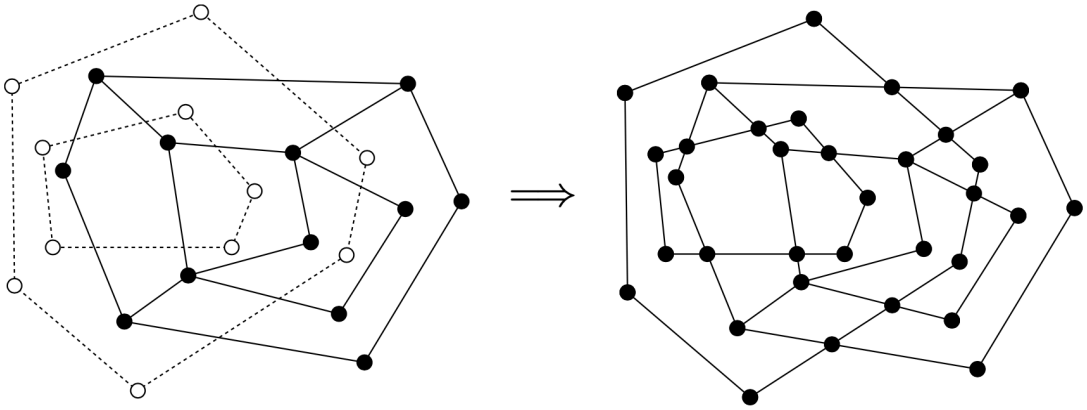
3.1 Opis problemu

W celu uniknięcia nadmiaru informacji na mapie, a jednocześnie z zamiarem maksymalizacji przekazywanej ilości informacji, mapy dzieli się na warstwy, z których każda przedstawia inny rodzaj informacji (np. położenie rzek, miast i torów kolejowych).



Rysunek 3.1: Mapy miast, rzek, kolei i ich nałożenie w zachodniej Kanadzie.

Mając takie różne podziały płaszczyzny (mapy) można chcieć uzyskać informację z wielu warstw równocześnie (np. gdzie występują przecięcia rzek i torów tam należy zbudować wiadukty). Aby taką informację uzyskać należy obliczyć nałożenie dwóch podziałów płaszczyzny, wynikiem czego uzyska się jeden podział, zawierający równocześnie informacje z obu podziałów.



Rysunek 3.2: Przykładowe przekształcenie dwóch podziałów płaszczyzny w jeden.

3.2 Struktury danych

3.2.1 Podwójnie łączona lista krawędzi $DCEL$

Podwójnie łączona lista krawędzi (doubly connected edge list, w skrócie $DCEL$) to struktura służąca do przechowywania podziału płaszczyzny za pomocą grafu planarnego. Charakteryzuje się tym, że z każdą krawędzią związane są dwie półkrawędzie. Na strukturę składają się trzy rodzaje rekordów:

- wierzchołek (*VertexRecord*): współrzędne x, y; wskaźnik do dowolnej incydentnej krawędzi *incidentalEdge*.
- ściana (*FaceRecord*): wskaźnik do dowolnej półkrawędzi otaczającej ścianę *outerEdge*; tablica wskaźników do półkrawędzi, które znajdują się wewnątrz ściany *innerEdges*.
- półkrawędź (*HalfEdgeRecord*): wskaźnik do wierzchołka, z którego półkrawędź wychodzi *beginning*; wskaźnik do bliźniaczej półkrawędzi *twinEdge*; wskaźnik do kolejnej półkrawędzi w cyklu *nextEdge*; wskaźnik do poprzedniej półkrawędzi w cyklu *previousEdge*; wskaźnik do incydentnej ściany (otaczanej przez dany cykl) *incidentalFace*.

3.2.2 Struktura zdarzeń

Struktura zdarzeń jest implementowana przez zrównoważone binarne drzewo wyszukiwań (w tym przypadku AVL). Zawiera zdarzenia, w których miotła się zatrzyma i zawiera jedynie zdarzenia (punkty) znajdujące się poniżej miotły. W węzłach przechowywane są obiekty typu *QueueRecord*:

- współrzędne punktu reprezentującego zdarzenie x, y.
- wskaźniki *left*, *right*, *parent*.
- rodzaj zdarzenia *type* (początek krawędzi, koniec krawędzi, przecięcie krawędzi).

- tablica krawędzi U, dla których punkt jest górnym końcem.
- tablica krawędzi L, dla których punkt jest dolnym końcem.
- tablica krawędzi C, które zawierają punkt w swoim wnętrzu.

W drzewie został wprowadzony następujący porządek:

$$p_1 \prec p_2 \Leftrightarrow p_{1y} > p_{2y} \vee (p_{1y} = p_{2y} \wedge p_{1x} < p_{2x})$$

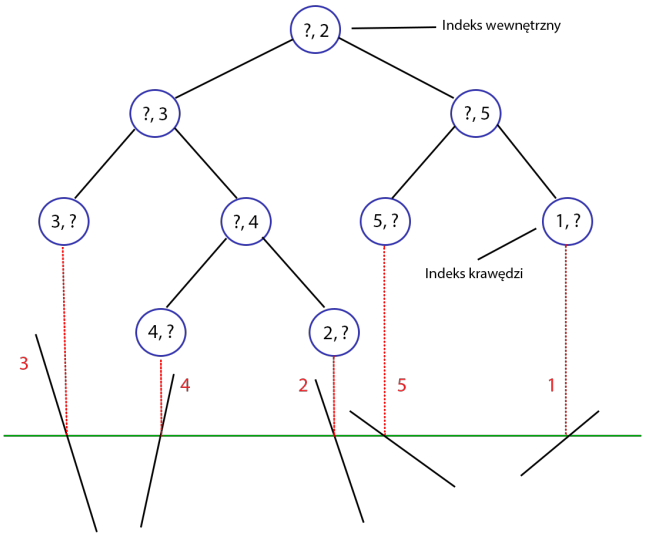
3.2.3 Struktura stanu miotły

Struktura stanu miotły implementowana jest przez zrównoważone binarne drzewo wyszukiwań (w tym przypadku AVL). Wykorzystywane jest do przechowywania kolejności krawędzi, w jakiej przecinają miotłę. W tym celu w liściach trzymane są indeksy krawędzi przecinających miotłę, a pozostałe wierzchołki służą do nawigowania po tym drzewie.

W każdym wierzchołku miotły trzymany jest obiekt typu *StateRecord*:

- indeks krawędzi *edgeIndex*, jeśli obiekt jest liściem.
- indeks krawędzi na lewo i maksymalnie na prawo *insideNodeIndex*, jeśli obiekt nie jest liściem.
- wskaźniki *left*, *right*, *parent*.

StateRecord umożliwia działanie zarówno jako liść, jak i wierzchołek wewnętrzny drzewa. Wierzchołki pośrednie przechowują indeks krawędzi w liściu na lewo i maksymalnie na prawo w drzewie.



Rysunek 3.3: Przykładowe drzewo stanu miotły. Na zielono zaznaczono aktualną pozycję miotły, na czerwono indeksy krawędzi.

3.3 Algorytmy

3.3.1 Obliczanie nałożenia podziałów płaszczyny

Pseudokod głównego algorytmu programu, który oblicza nałożenie dwóch podziałów płaszczyzny 2D. Wykorzystuje algorytmy MERGE-DCEL, CONVERT-TO-PLANAR-REPRESENTATION i FIX-FACES, które zostaną opisane w kolejnych podpunktach.

COMPUTE-MAP-OVERLAY(D_1, D_2)

Wejście: Dwie podwójnie łączone listy krawędzi (DCEL)
D_1 i D_2 reprezentujące podziały płaszczyzny.
Wyjście: DCEL D reprezentująca nałożenie podziałów
reprezentowanych przez dwie wejściowe DCEL D_1 i D_2.

1 D = MERGE-DCEL(D_1, D_2)
2 CONVERT-TO-PLANAR-REPRESENTATION(D)
3 FIX-FACES(D)
4 **return** D

3.3.2 Scalanie podwójnie łączonych list krawędzi

Pseudokod algorytmu scalającego dwie podwójnie łączone listy krawędzi w jedną.

MERGE-DCEL(D_1, D_2)

1 Utwórz nową pustą DCEL D.
2 Przepisz każdą krawędź z D_1 i D_2 do D, jeżeli się nie powtarza.
3 Przepisz każdy wierzchołek z D_1 i D_2 do D, jeżeli się nie powtarza.
Jeżeli istnieją dwa takie same wierzchołki w D_1 i D_2, przepisz jeden
z nich i przepnij wszystkie incydentne krawędzie do tego wierzchołka.
4 **return** D

3.3.3 Przekształcanie do reprezentacji planarnej

Pseudokod algorytmu przekształcającego podział płaszczyzny uzyskany po scaleniu dwóch $DCEL$ do poprawnej, planarnej postaci $DCEL$. Wykorzystuje algorytmy opisane w pozostałych podpunktach.

CONVERT-TO-PLANAR-REPRESENTATION(D)

Zainicjalizuj poniższe struktury:
1 Q - Kolejka zdarzeń. Początkowo zawiera zdarzenia, będące punktami znalezionymi podczas scalania dwóch DCEL.
2 T - Struktura stanu miotły. Początkowo jest puste.
3 ME - Tablica indeksów krawędzi przyporządkowanych w następujący sposób: po podziale krawędzi, jej indeks jest przypisywany tej nowo powstałej krawędzi, która jest położona niżej. Początkowo każdej krawędzi odpowiada jej własny indeks.
4 IP - Tablica z punktami przecięć. Początkowo zawiera wszystkie punkty przecięć znalezione podczas scalania dwóch DCEL.
5 **while** Q nie jest puste
6 Niech p będzie kolejnym zdarzeniem wyjętym z Q.
7 Przyporządkuj właściwe indeksy w ME wszystkim krawędziom zawierającym punkt p w swoim wnętrzu.
8 **if** jakaś krawędź przechodzi przez lub kończy się w punkcie p i jest przechowywana w zdarzeniu p
9 Usuń z T wszystkie krawędzie, które zawierały lub kończyły się w punkcie p i były przechowywane w zdarzeniu p. Dla ostatniej usuniętej krawędzi znajdź lewego i prawego sąsiada.
10 Dodaj do Q ewentualne przecięcie znalezionych sąsiadów.
11 **else** przez punkt p może przechodzić jedna krawędź, która nie została zapisana w zdarzeniu p. Może tak się stać, gdy w punkcie p zaczyna się wiele krawędzi.
12 Nawigując po drzewie T spróbuj usunąć krawędź, która przechodzi przez punkt p.
13 Jeżeli usunięto krawędź, to dodaj ją do zbioru krawędzi przechodzących przez punkt p oraz dodaj p do zbioru IP
14 **if** punkt p to przecięcie krawędzi e_1 i e_2
15 HANDLE-INTERSECTION-A(D, p, e_1, e_2)
16 **elseif** krawędź e przechodzi przez punkt p
17 HANDLE-INTERSECTION-B(D, p, e)
18 **else** punkt p jest początkiem krawędzi
19 **if** w p zaczyna się tylko jedna krawędź
20 Dodaj tę krawędź do drzewa T.
21 Znajdź lewego i prawego sąsiada tej krawędzi i dodaj ich ewentualne przecięcie do kolejki Q.
22 **elseif** w p zaczyna się lub p przecina więcej niż jedna krawędź
23 Dodaj w odpowiedniej kolejności (od lewej do prawej) nowe krawędzie do T.
24 Znajdź lewego sąsiada nowo dodanej krawędzi znajdującej się najbardziej na lewo i dodaj ich ewentualne przecięcie do Q. Zrób to samo dla prawego sąsiada krawędzi znajdującej się najbardziej na prawo.
25 **return** IP

3.3.4 Naprawienie *DCEL* dla przecięcia typu A

Pseudokod algorytmu naprawiającego D, gdy krawędzie e_1 i e_2 przecinają się w punkcie p.

HANDLE-INTERSECTION-A(D, p, e_1, e_2)

1 Podziel krawędzie e_1 i e_2 względem punktu przecięcia. Dla każdej części utwórz dwie nowe półkrawędzie (łącznie osiem nowych półkrawędzi) i połącz je odpowiednio w bliźniacze pary (pole *twinEdge*).
2 Dodaj po jednej półkrawędzi z bliźniaczej pary do D.
3 Napraw wskaźniki z nowymi półkrawędziami na końcach wejściowych krawędzi.
4 Jeśli któraś z wejściowych krawędzi była krawędzią incydentną dla pewnego wierzchołka, ustaw odpowiednie wskaźniki dla nowych półkrawędzi.
5 Połącz odpowiednie półkrawędzie wokół punktu przecięcia w kolejności wyznaczonej przez współczynnik nachylenia krawędzi wraz z informacją, w której ćwiartce układu współrzędnych znajduje się półkrawędź.

3.3.5 Naprawienie *DCEL* dla przecięcia typu B

Pseudokod algorytmu naprawiającego D, gdy krawędź e przechodzi przez punkt p.

HANDLE-INTERSECTION-B(D, p, e)

1 Podziel krawędź e na dwie części względem punktu p, utwórz dla każdej nowej krawędzi po dwie półkrawędzie i połącz je w bliźniacze pary (pole *twinEdge*).
2 Napraw wskaźniki z nowymi półkrawędziami na końcach wejściowej krawędzi.
3 Dodaj nowe półkrawędzie, mające początek w wierzchołku p, do D.
4 Jeżeli któraś półkrawędź krawędzi wejściowej była półkrawędzią incydentną dla któregoś z wierzchołków końcowych, to napraw ten wskaźnik.
5 Znajdź odpowiednie półkrawędzie incydentne do wierzchołka p, między które zostaną podpięte nowe półkrawędzie.

3.3.6 Naprawianie ścian

Pseudokod algorytmu naprawiającego ściany w D.

FIX-FACES(D)

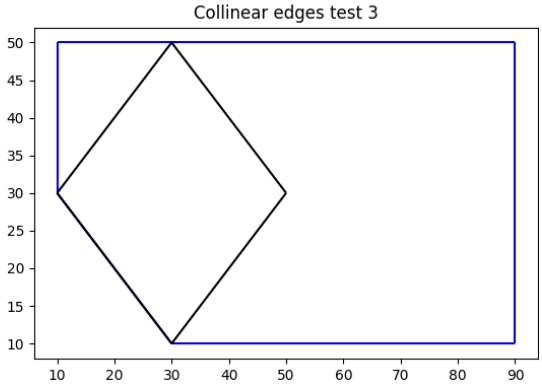
1 Przejdź algorytmem DFS po wszystkich cyklach i zaindeksuj każdy cykl.
2 Zainicjalizuj graf nieskierowany G, w którym wierzchołki to znalezione cykle, a krawędź u-v oznacza, że cykl zewnętrzny u, ograniczający dziurę w ścianie, lub cykl wewnętrzny u leży bezpośrednio na lewo od cyklu zewnętrznego v.
3 Uzupełnij krawędzie w grafie G, ponownie zamiatając wierzchołki z D.
4 Na podstawie grafu G utwórz rekordy ścian i połącz dziury z cyklami ograniczającymi ściany.
5 Ponownie przejdź algorytmem DFS po wszystkich cyklach i uzupełnij informacje o ścianach w przechadzanych cyklach.
6 Dla każdej ściany uzupełnij brakujące informacje na podstawie cykli zawierających tę ścianę.

3.4 Wykonane testy i uzyskane wyniki

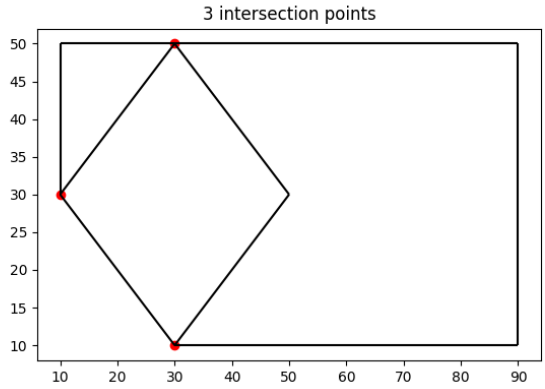
Na rysunkach 3.4 - 3.19 przedstawiono przykładowe podziały płaszczyzny i otrzymane wyniki obliczeń algorytmu.

3.4.1 Podział częściowo współliniowy

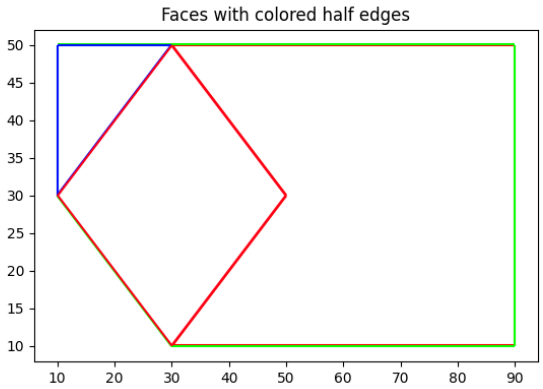
Testy poprawności działania algorytmu dla podziałów częściowo współliniowych (podział 1. zawieraja krawędzie współliniowe z pewnymi krawędziami z podziału 2.), na rys. 3.4 - 3.6.



Rysunek 3.4: Test dla dwóch podziałów częściowo współliniowych.



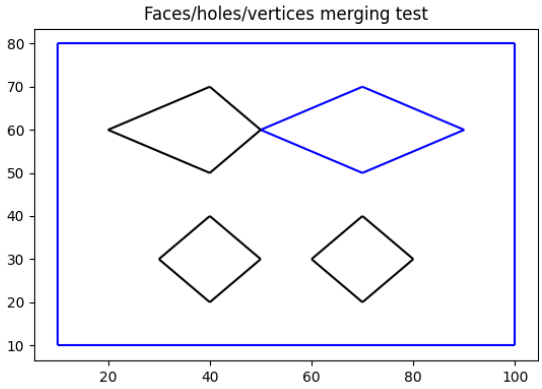
Rysunek 3.5: Znalezione przecięcia dla zbioru z rys. 3.4.



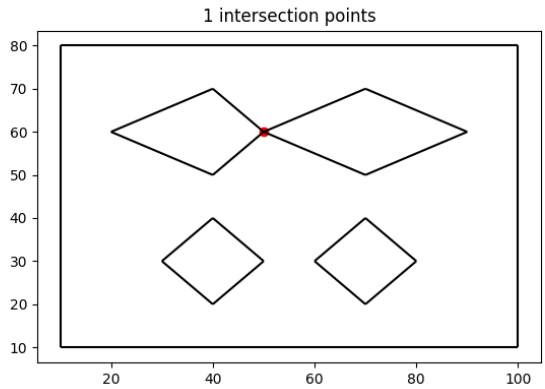
Rysunek 3.6: Znalezione ściany dla zbioru z rys. 3.4.

3.4.2 Podział z dziurami wewnątrz ścian

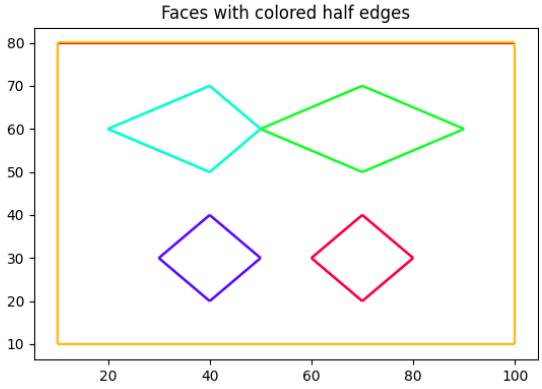
Testy poprawności działania algorytmu dla podziałów zawierających dziury w ścianach i wierzchołki trudne do scalenia, na rys. 3.7 - 3.10.



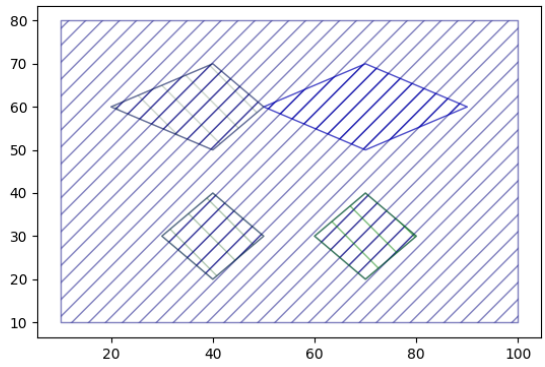
Rysunek 3.7: Test dwóch podziałów z dziurami w ścianach.



Rysunek 3.8: Znalezione przecięcia dla zbioru z rys. 3.7.



Rysunek 3.9: Znalezione ściany dla zbioru z rys. 3.4.



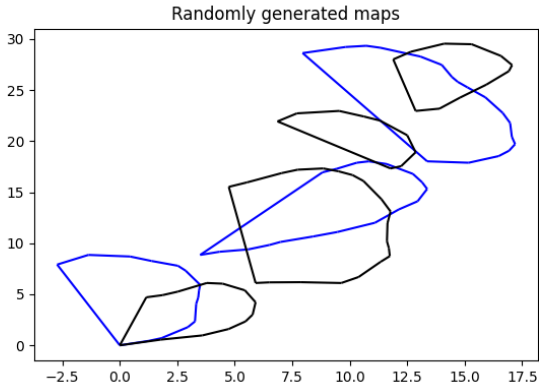
Rysunek 3.10: Przykładowe informacje przechowywane w ścianach z rys. 3.7.

3.4.3 Podziały wygenerowane losowo

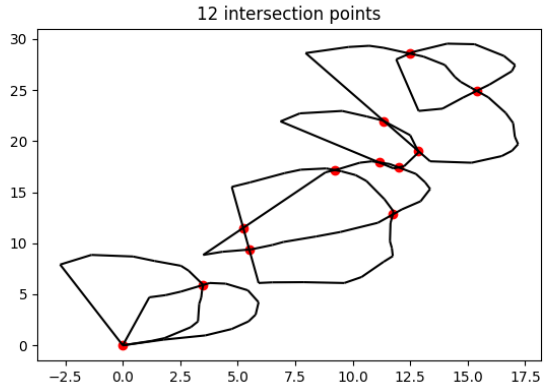
Na rys. 3.11 - 3.14 przedstawiono losowo wygenerowane podziały płaszczyzny i wyniki działania algorytmu na tych podziałach.

3.4.4 Podziały zadane ręcznie

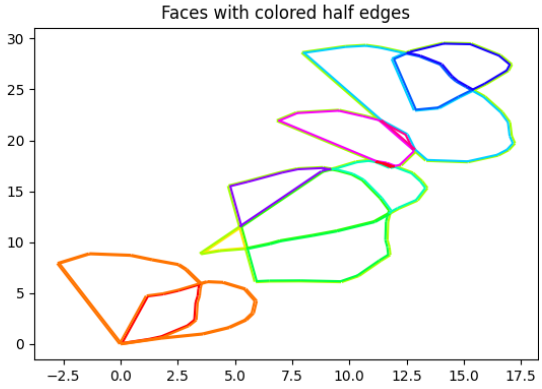
Na rys. 3.15 - 3.19 przedstawiono podziały zadane przy pomocy interfejsu graficznego i wyniki obliczeń algorytmu dla tak zadanych podziałów.



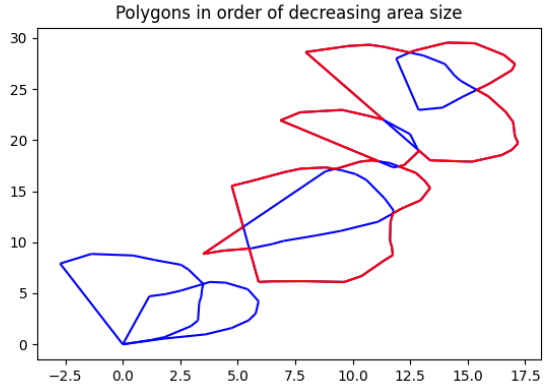
Rysunek 3.11: Losowo wygenerowane podziały płaszczyzny.



Rysunek 3.12: Znalezione przecięcia dla zbioru z rys. 3.11.



Rysunek 3.13: Pokolorowane półkrawędzie odpowiadające ścianom z rys. 3.11.

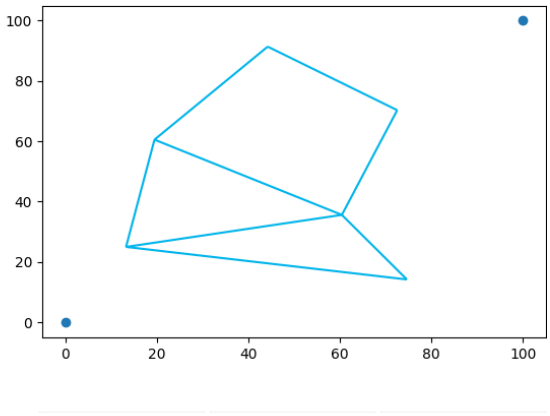


Rysunek 3.14: Test poprawności połączenia zewnętrznych półkrawędzi z rys. 3.11.

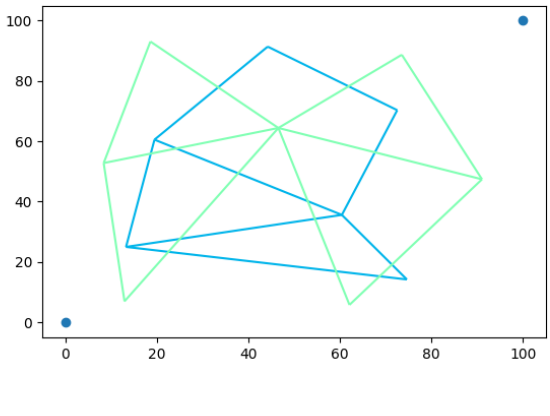
3.5 Wnioski

Do porównywania liczb zmiennoprzecinkowych wykorzystywano, znalezioną dzięki testom wartość $\varepsilon = 10^{-5}$. Dla ε mniejszych algorytm nie działał stabilnie, porządek w strukturach mógł stać się zaburzony.

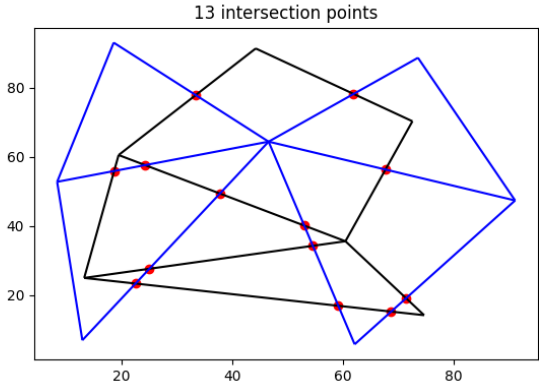
Algorytm ma wysoki poziom złożoności, rozwiązuje wiele przypadków skrajnych, wynikiem czego bez względu na rodzaj zadanych podziałów, poprawne ich nałożenie zostanie znalezione. Niewykluczone jednak, że istnieje jakiś przypadek, dla którego może nie zadziałać poprawnie.



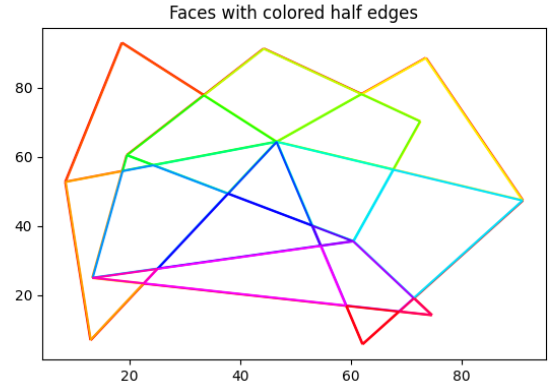
Rysunek 3.15: Podział 1. narysowany przy użyciu interfejsu graficznego.



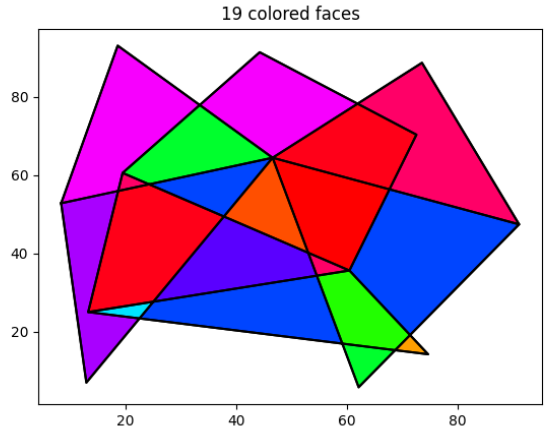
Rysunek 3.16: Podział 2. narysowany przy użyciu interfejsu graficznego.



Rysunek 3.17: Znalezione przecięcia dla podziałów z rys. 3.16.



Rysunek 3.18: Pokolorowane półkrawędzie odpowiadające ścianom z rys. 3.16



Rysunek 3.19: Pokolorowane ściany z podziałów z rys. 3.16.

Bibliografia

[1] Berg et al. (2000) *Computational Geometry: Algorithms and Applications*, Springer.