Obliczanie grafu widoczności

Algorytmy geometryczne 2022/2023
grupa nr 2

Dominik Adamczyk
Szymon Nowak-Trzos

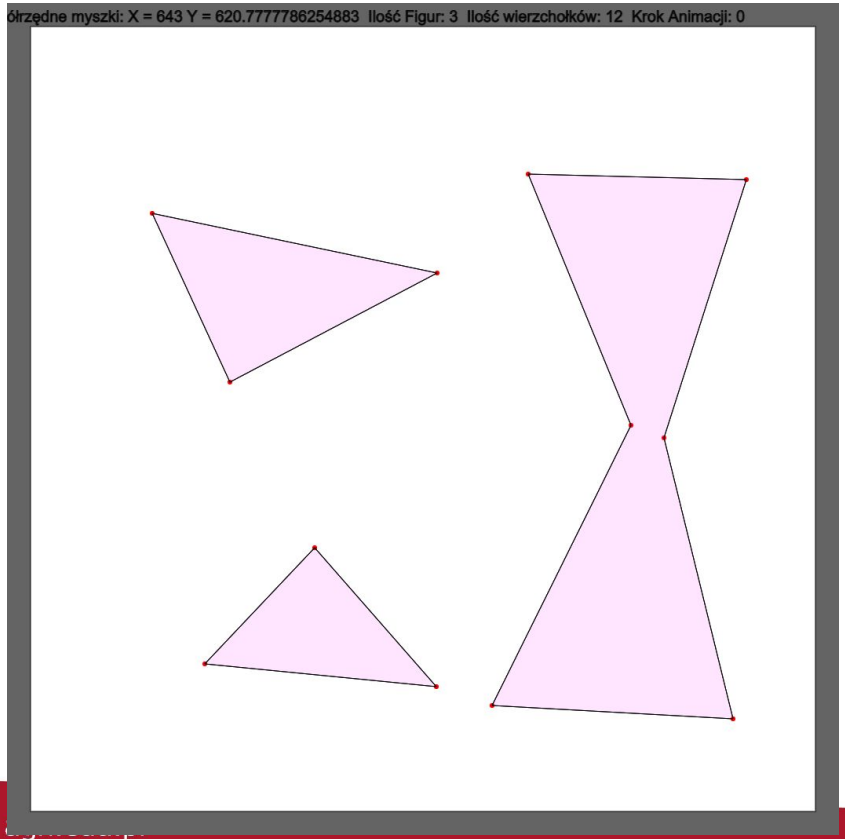
Opis problemu

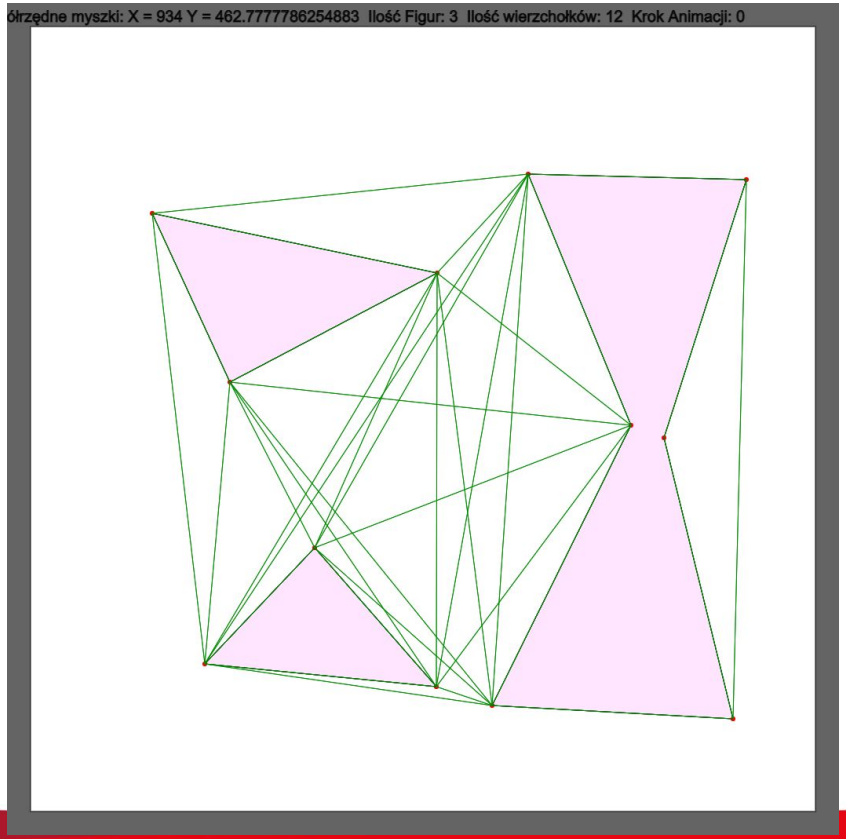
Dane - zbiór S rozłącznych wielokątów zadanych na płaszczyźnie.

Cel - dla powyższego zbioru danych wyznaczenie grafu widoczności.

Graf widoczności - graf złożony z wierzchołków i krawędzi łączących wierzchołki “widzące się wzajemnie”. Dwa wierzchołki “widzą się” kiedy odcinek przez nie wyznaczony nie przecina żadnej z przeszkód zadanych w danych wejściowych.

Przykładowe dane i odpowiadający im graf widoczności





Zastosowania grafów widoczności

- Znajdowanie najkrótszej ścieżki pomiędzy punktami na płaszczyźnie z przeszkodami
- Wyznaczanie położeń anten radiowych
- Planowanie miejskie
- Analiza danych

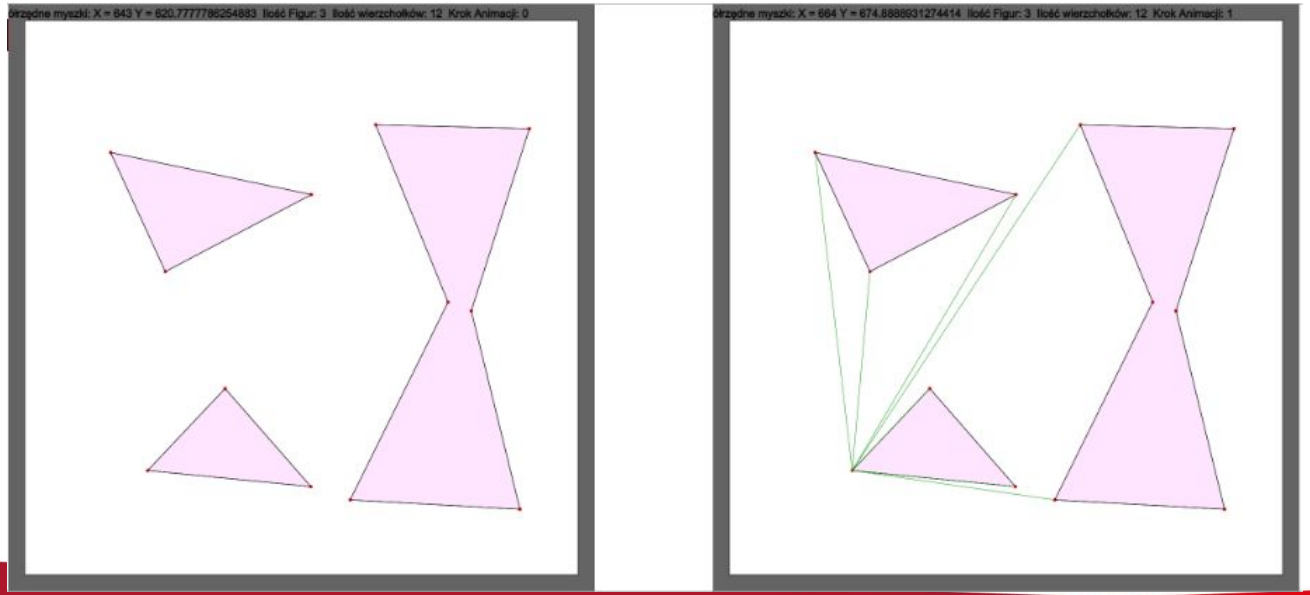
Naiwny algorytm

Prosty algorytm tworzący graf widoczności ma złożoność O(n^3), gdzie n to liczba krawędzi zbioru wejściowego. Jego działanie polega na sprawdzeniu, czy dla każdej pary wierzchołków istnieje krawędź przecinająca odcinek tworzony przez te wierzchołki.

Prezentowany w tej pracy algorytm osiąga złożoność O(n^2 * log(n)). Istnieją również algorytmy o lepszej złożoności obliczeniowej.

Zamysł działania algorytmu - główna pętla

Główna część algorytmu (**grafWidoczności**) polega na znalezieniu widocznych wierzchołków dla każdego z wierzchołków.



Algorytm grafWidoczności

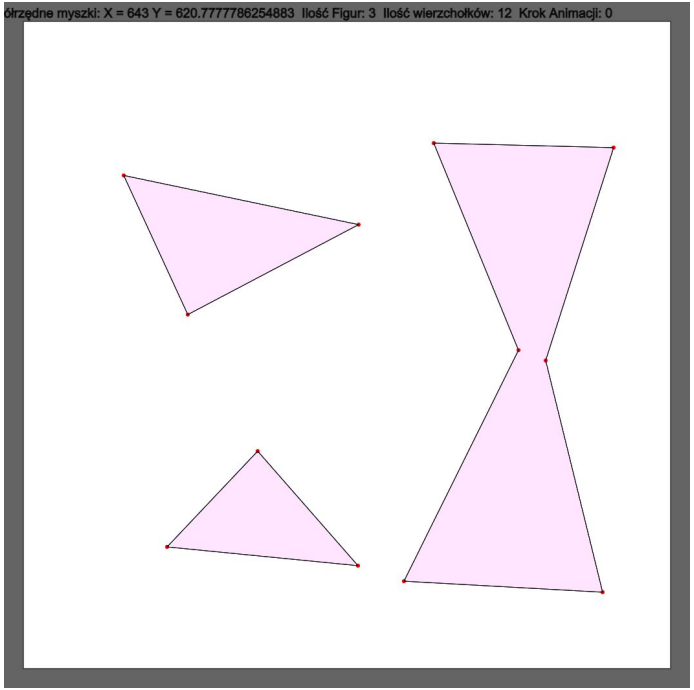
grafWidoczności(S):

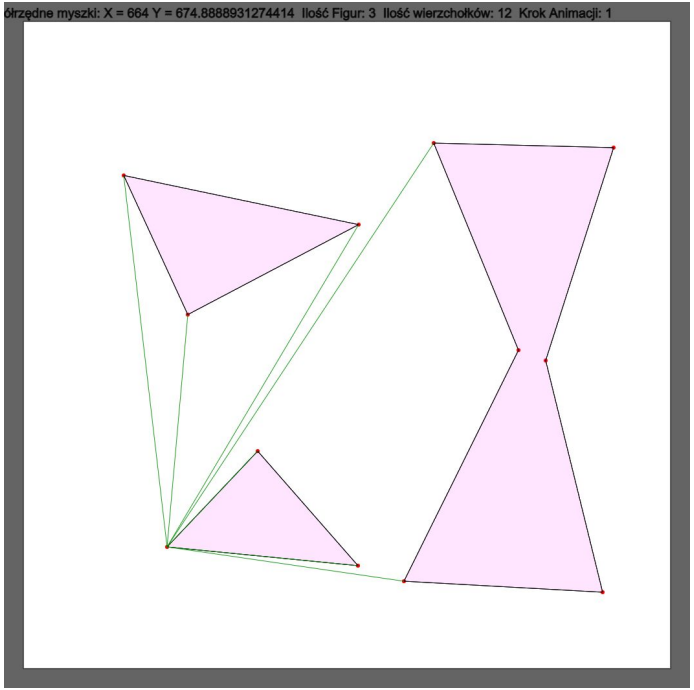
wejście: zbiór S rozłącznych wielokątów

wyjście: graf widoczności G(S)

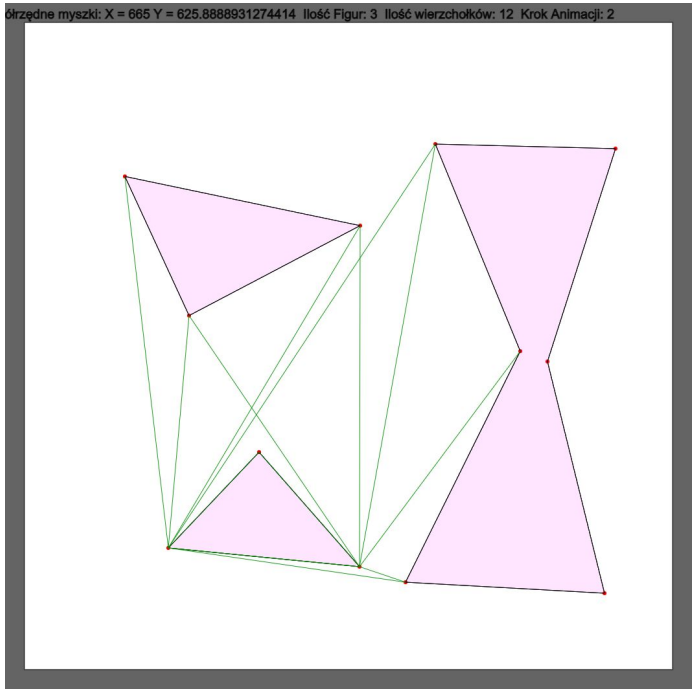
1. Zainicjuj graf G = (V, E), gdzie zbiór V to zbiór wierzchołków z S, a E = ∅
2. dla każdego wierzchołka v ∈ V
3. W = widoczneWierzchołki(v, S)
4. dla każdego w ∈ W dodaj (v,w) do E
5. zwróć G

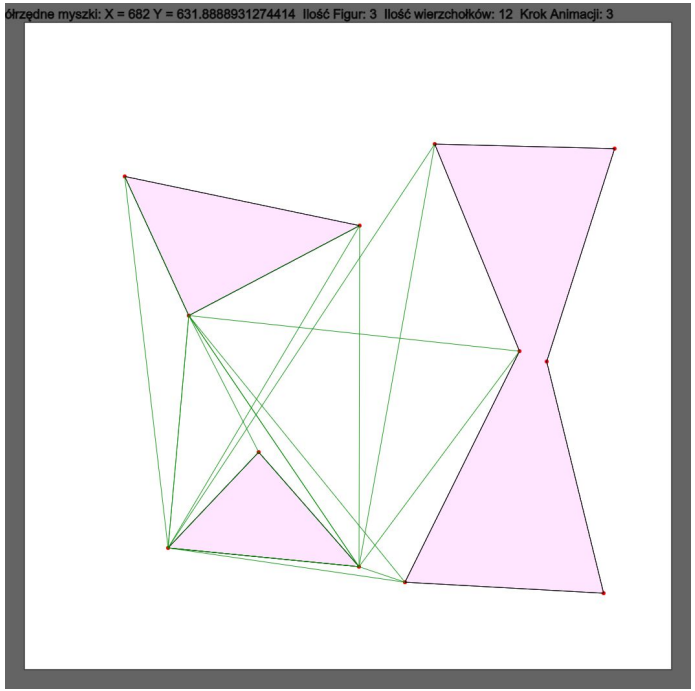
Algorytm grafWidoczności



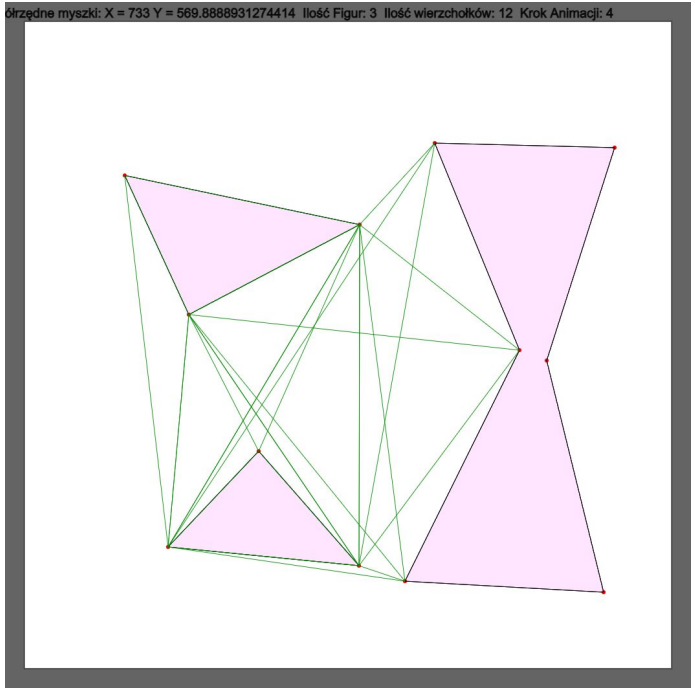


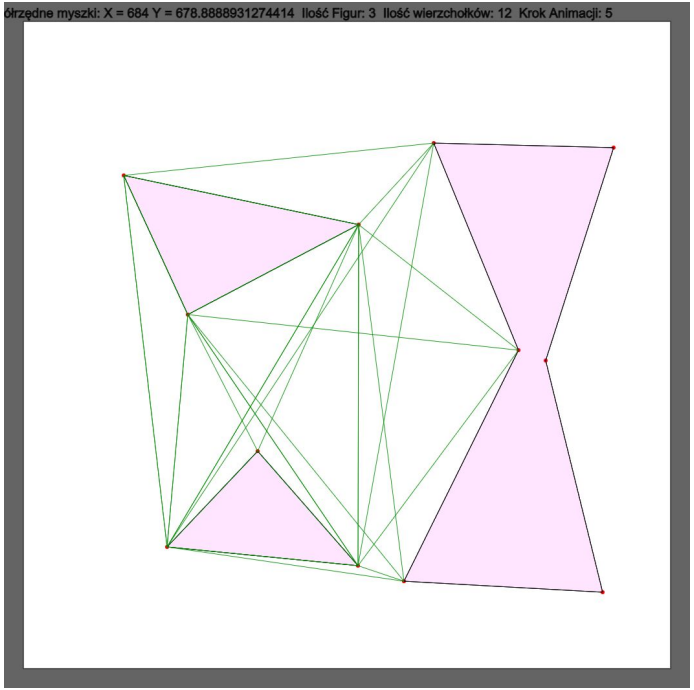
Algorytm grafWidoczności



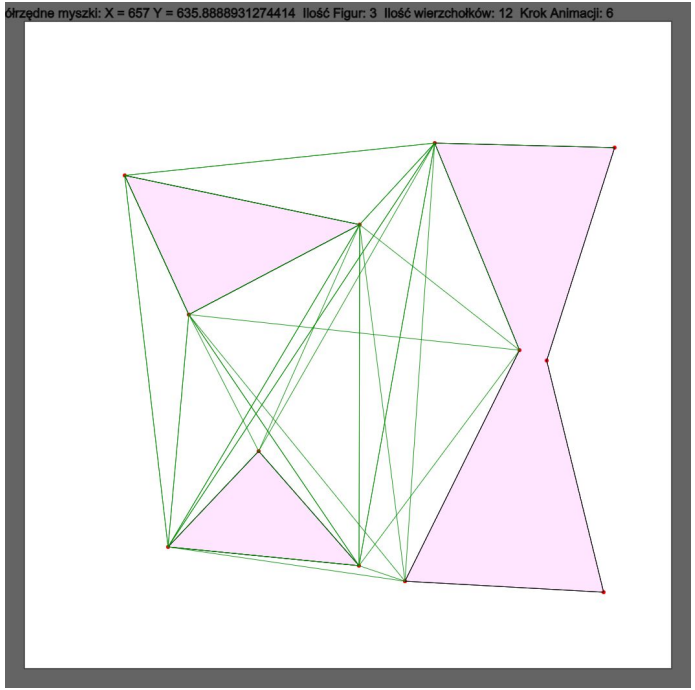


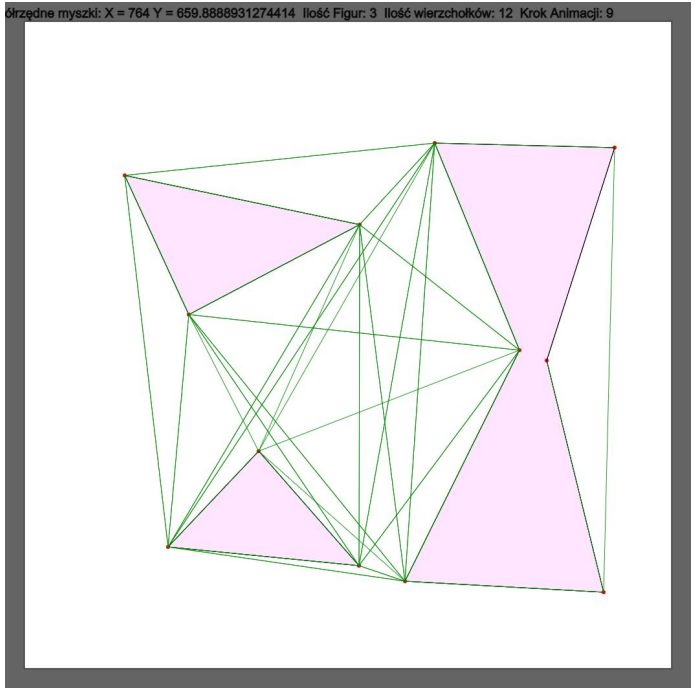
Algorytm grafWidoczności





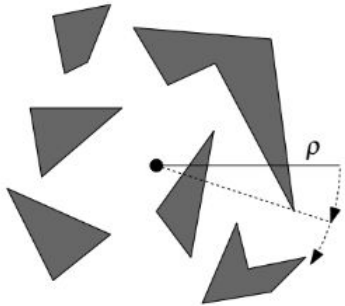
Algorytm grafWidoczności

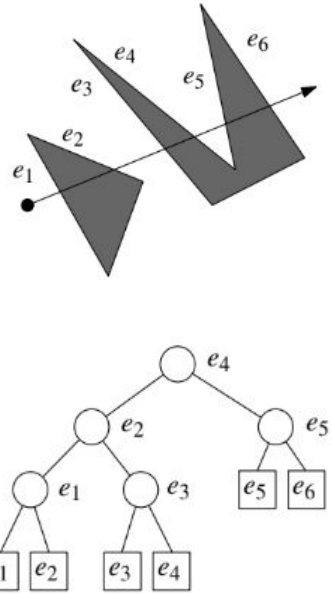




Zamysł działania algorytmu - określanie widocznych wierzchołków

Znajdowanie widocznych wierzchołków z wierzchołka P jest realizowane poprzez algorytm zamiatania (**widoczneWierzchołki**), w którym miotła jest półprostą wychodzącą z P i obraca się po kolejnych zdarzeniach, którymi są posortowane po kącie pozostałe punkty (więc struktura zdarzeń nie zmienia się w trakcie działania algorytmu). Miotła będąca strukturą stanu implementowana jest jako drzewo poszukiwań binarnych, na którym przechowywane są krawędzie aktualnie przecinające się z miotłą (posortowane po odległości pomiędzy punktem P, a punktem przecięcia) na danej pozycji. W każdym kroku oceniane jest czy punkt aktywny jest widoczny z punktu P.





Ilustracje zaczerpnięte z książki:
"Geometria obliczeniowa. Algorytmy i zastosowania" - de Berg M.

Algorytm widoczneWierzchołki

widoczneWierzchołki(p, S):

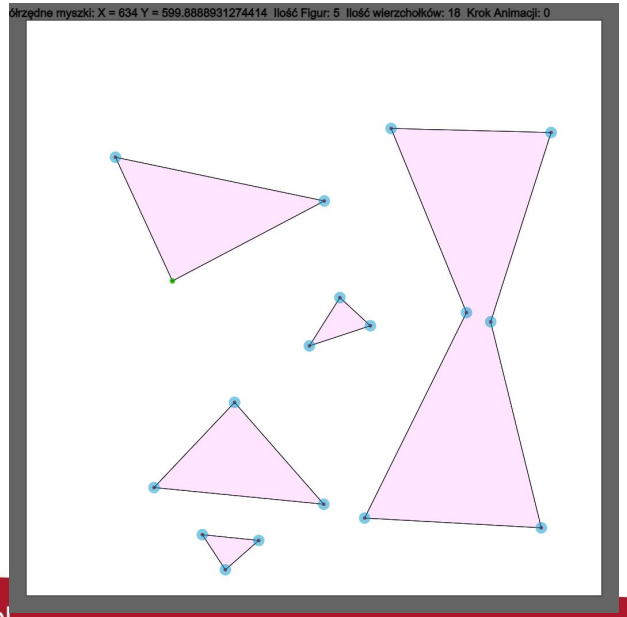
Wejście: Zbiór S rozłącznych wielokątów i punkt p

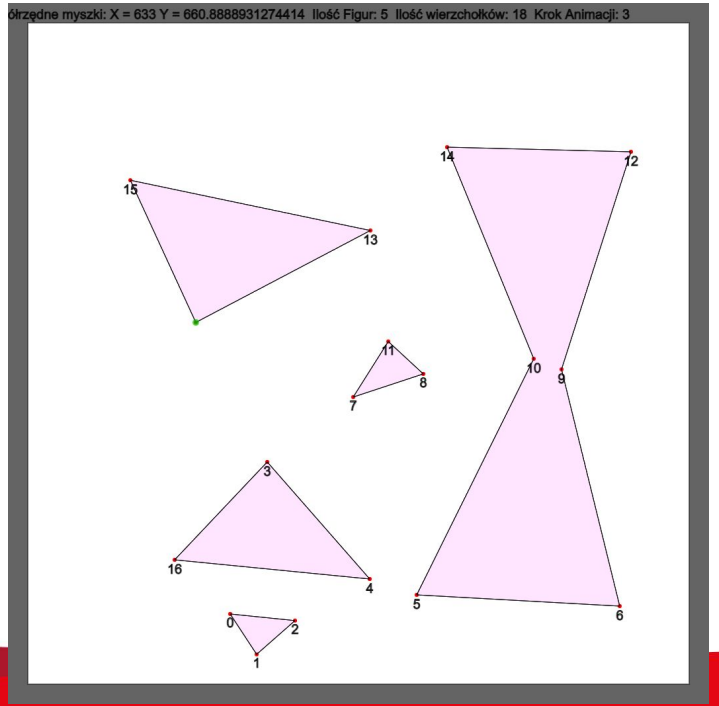
Wyjście: Wszystkie widoczne z punktu p wierzchołki należące do S

1. Posortuj wierzchołki ze zbioru S po ich kącie jaki tworzą z półprostą wychodzącą z punktu p, skierowaną pionowo w dół, przeciwnie do ruchu wskazówek zegara. Dla tych samych kątów posortuj rosnąco po odległości między wierzchołkiem a p. Niech W zawiera posortowane wierzchołki.
2. Niech B będzie półprostą mającą swój początek w p i przechodzącą przez pierwszy z posortowanych punktów. Znajdź wszystkie krawędzie przecinające się z B i umieść je w drzewie BST T w kolejności w jakiej przecinają B.
3. O = ∅
4. dla i ∈ {0..len(W)-1}
5. jeżeli czyWidoczny(W[i]): dodaj W[i] do O
6. dodaj do T krawędzie wychodzące z W[i] i leżące po lewej stronie półprostej B
7. usuń z T krawędzie wychodzące z W[i] i leżące po prawej stronie półprostej B
8. Zwróć O

Algorytm widoczneWierzchołki

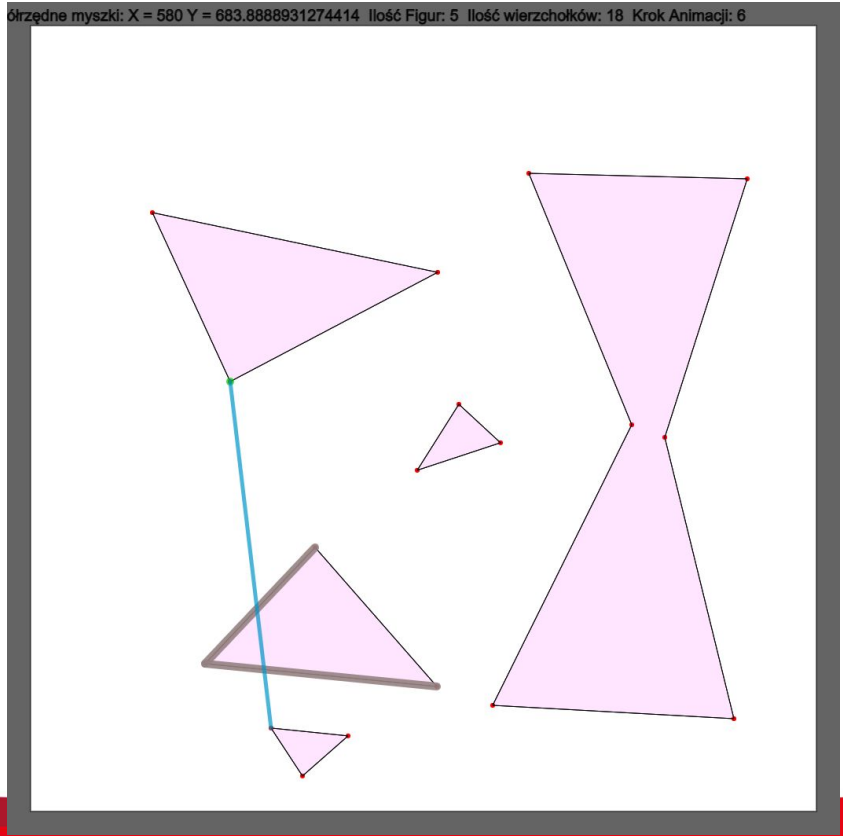
1. **Posortuj wierzchołki ze zbioru S po ich kącie jaki tworzą z półprostą wychodzącą z punktu p, skierowaną pionowo w dół, przeciwnie do ruchu wskazówek zegara. Dla tych samych kątów posortuj rosnąco po odległości między wierzchołkiem a p. Niech W zawiera posortowane wierzchołki.**





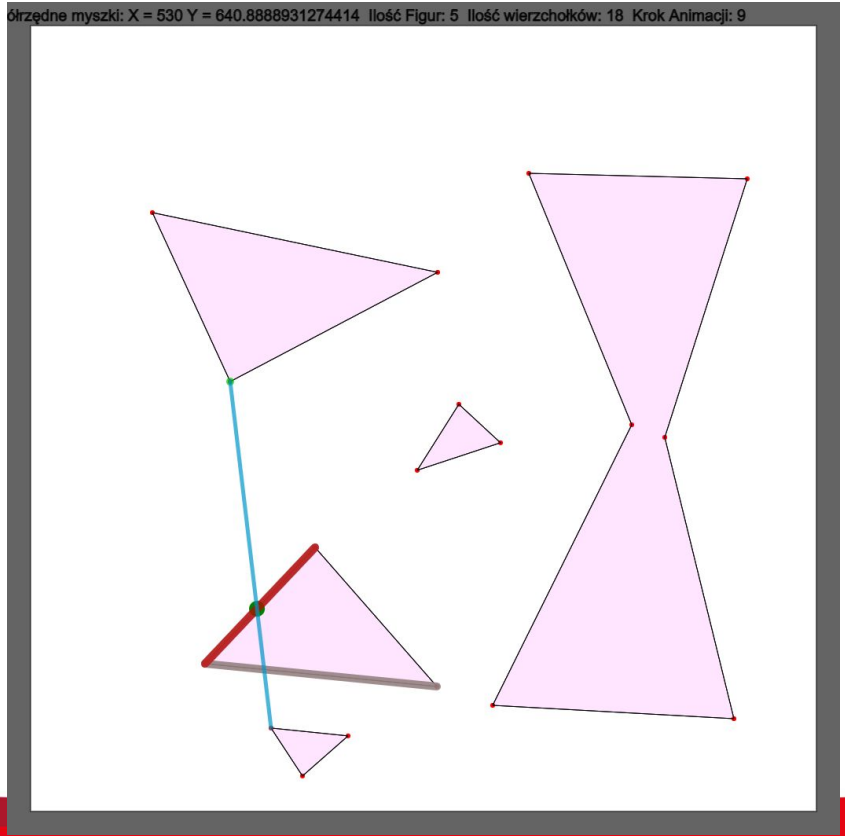
Algorytm widoczneWierzchołki

2. **Niech B będzie półprostą mającą swój początek w p i przechodzącą przez pierwszy z posortowanych punktów. Znajdź wszystkie krawędzie przecinające się z B i umieść je w drzewie BST T w kolejności w jakiej przecinają B.**



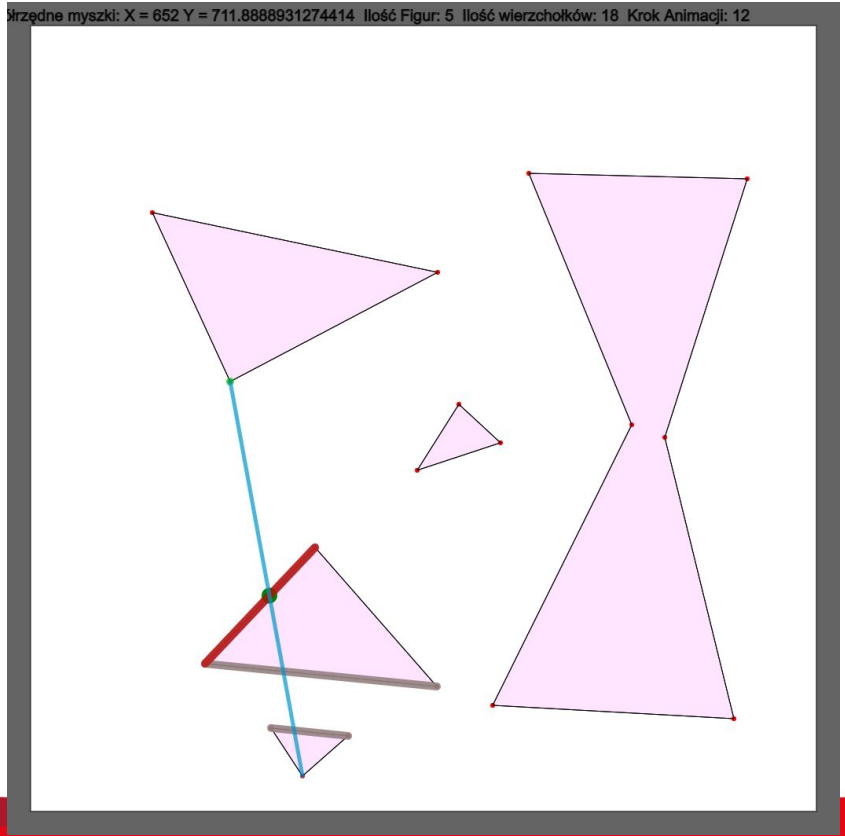
Algorytm widoczneWierzchołki

4. dla i ∈ {0..len(W)-1}
5. jeżeli czyWidoczny(W[i]): dodaj W[i] do O
6. dodaj do T krawędzie wychodzące z W[i] i leżące po lewej stronie półprostej B
7. usuń z T krawędzie wychodzące z W[i] i leżące po prawej stronie półprostej B



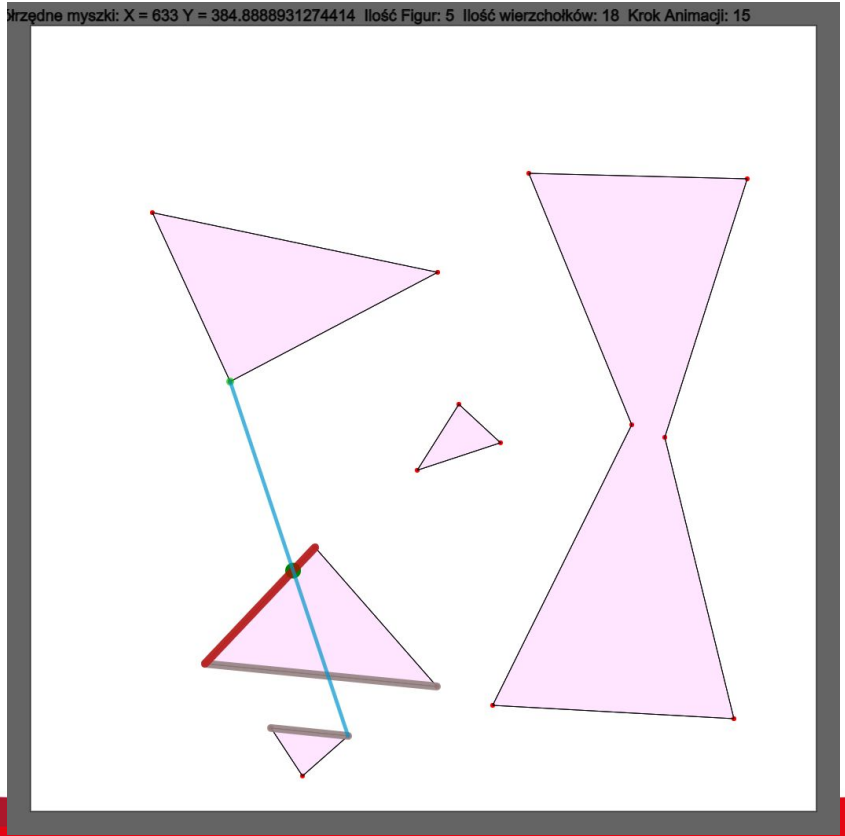
Algorytm widoczneWierzchołki

4. dla i ∈ {0..len(W)-1}
5. jeżeli czyWidoczny(W[i]): dodaj W[i] do O
6. dodaj do T krawędzie wychodzące z W[i] i leżące po lewej stronie półprostej B
7. usuń z T krawędzie wychodzące z W[i] i leżące po prawej stronie półprostej B



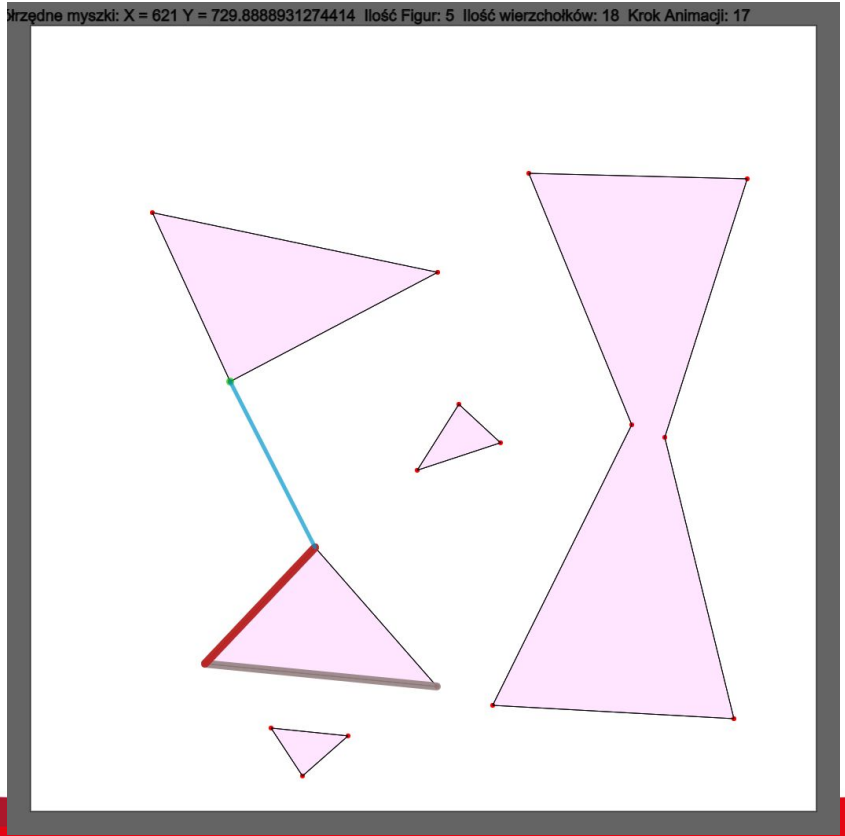
Algorytm widoczneWierzchołki

4. dla i ∈ {0..len(W)-1}
5. jeżeli czyWidoczny(W[i]): dodaj W[i] do O
6. dodaj do T krawędzie wychodzące z W[i] i leżące po lewej stronie półprostej B
7. usuń z T krawędzie wychodzące z W[i] i leżące po prawej stronie półprostej B



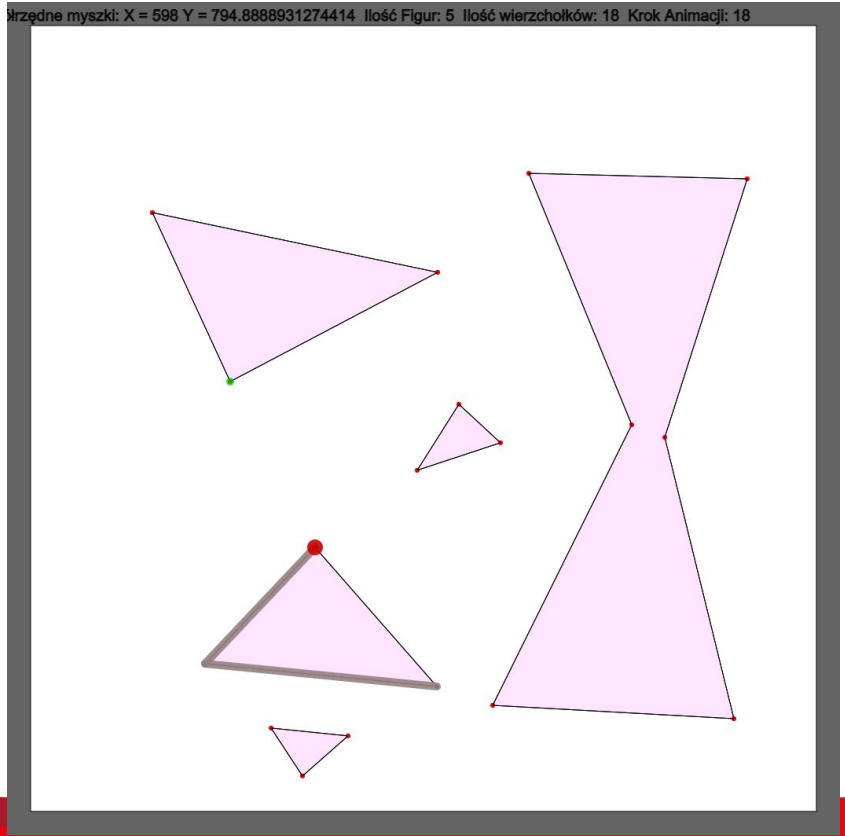
Algorytm widoczneWierzchołki

4. dla i ∈ {0..len(W)-1}
5. jeżeli czyWidoczny(W[i]): dodaj W[i] do O
6. dodaj do T krawędzie wychodzące z W[i] i leżące po lewej stronie półprostej B
7. usuń z T krawędzie wychodzące z W[i] i leżące po prawej stronie półprostej B



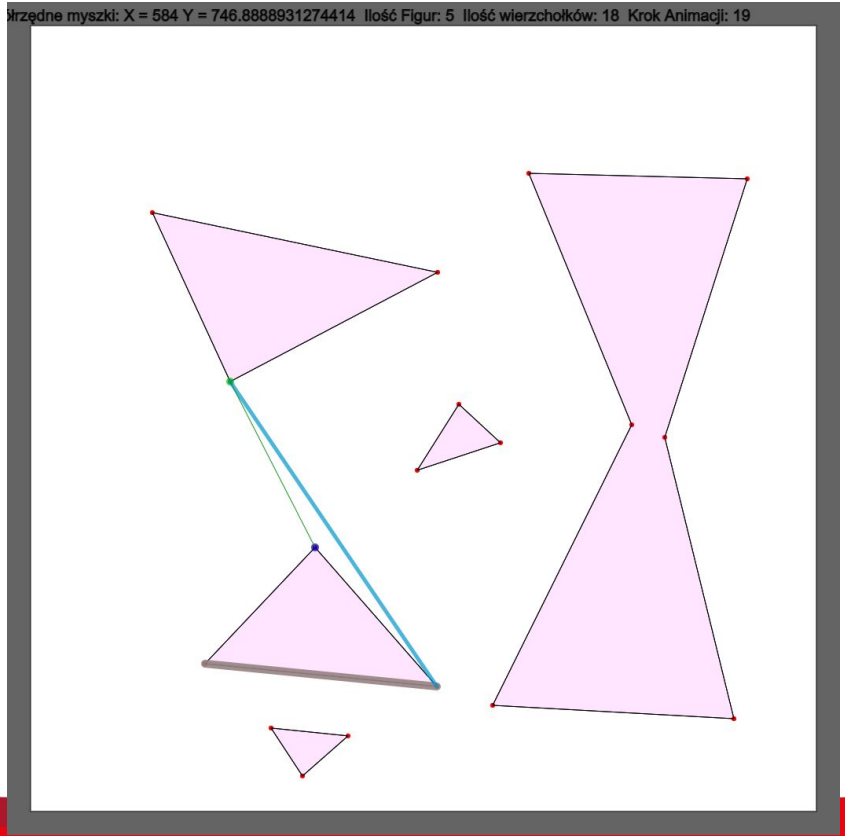
Algorytm widoczneWierzchołki

4. dla i ∈ {0..len(W)-1}
5. **jeżeli czyWidoczny(W[i]): dodaj W[i] do O**
6. dodaj do T krawędzie wychodzące z W[i] i leżące po lewej stronie półprostej B
7. usuń z T krawędzie wychodzące z W[i] i leżące po prawej stronie półprostej B

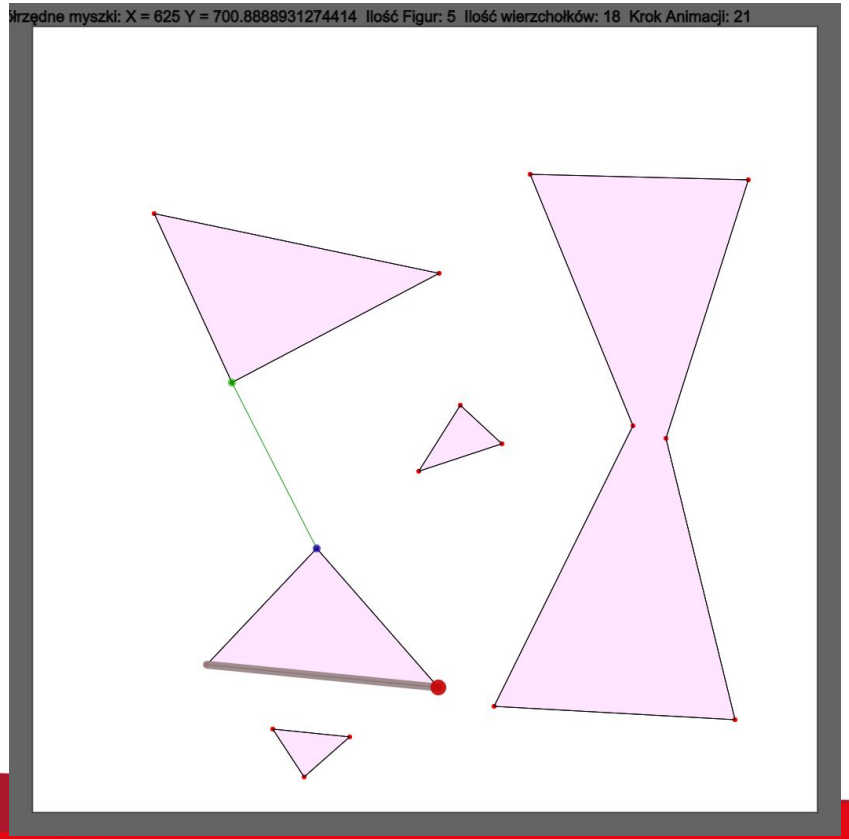


Algorytm widoczneWierzchołki

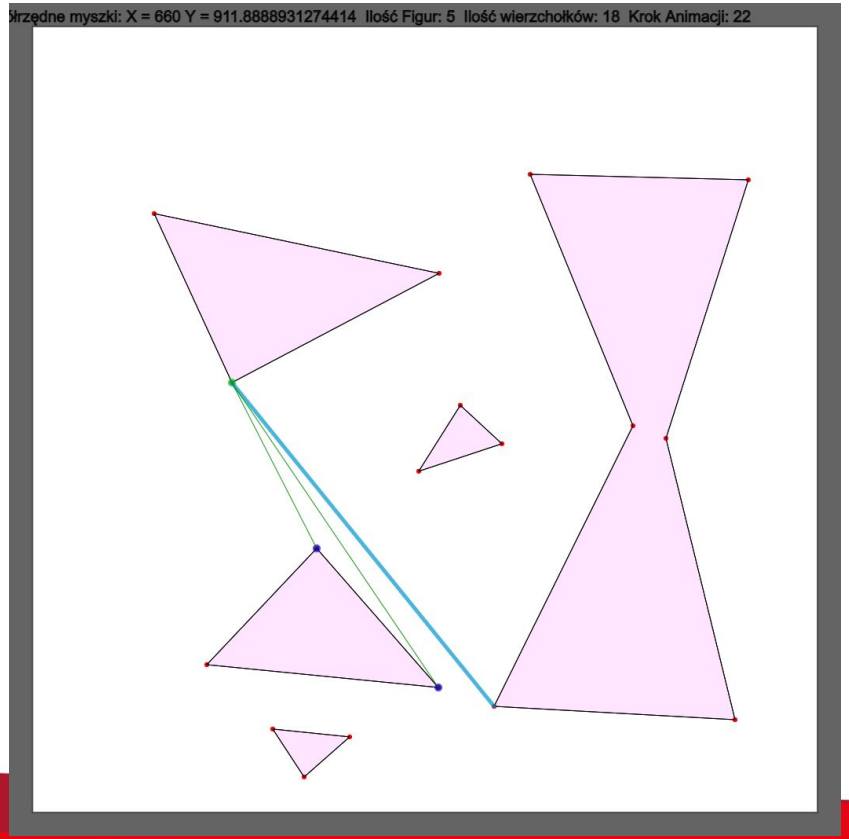
4. dla i ∈ {0..len(W)-1}
5. **jeżeli czyWidoczny(W[i]): dodaj W[i] do O**
6. dodaj do T krawędzie wychodzące z W[i] i leżące po lewej stronie półprostej B
7. usuń z T krawędzie wychodzące z W[i] i leżące po prawej stronie półprostej B



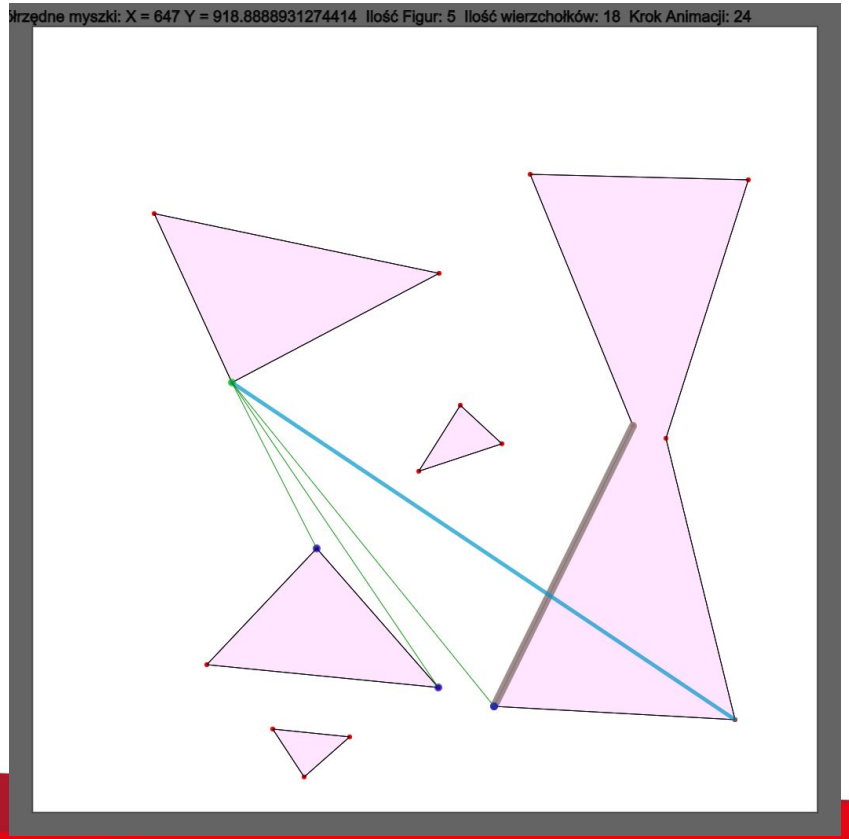
Algorytm widoczneWierzchołki



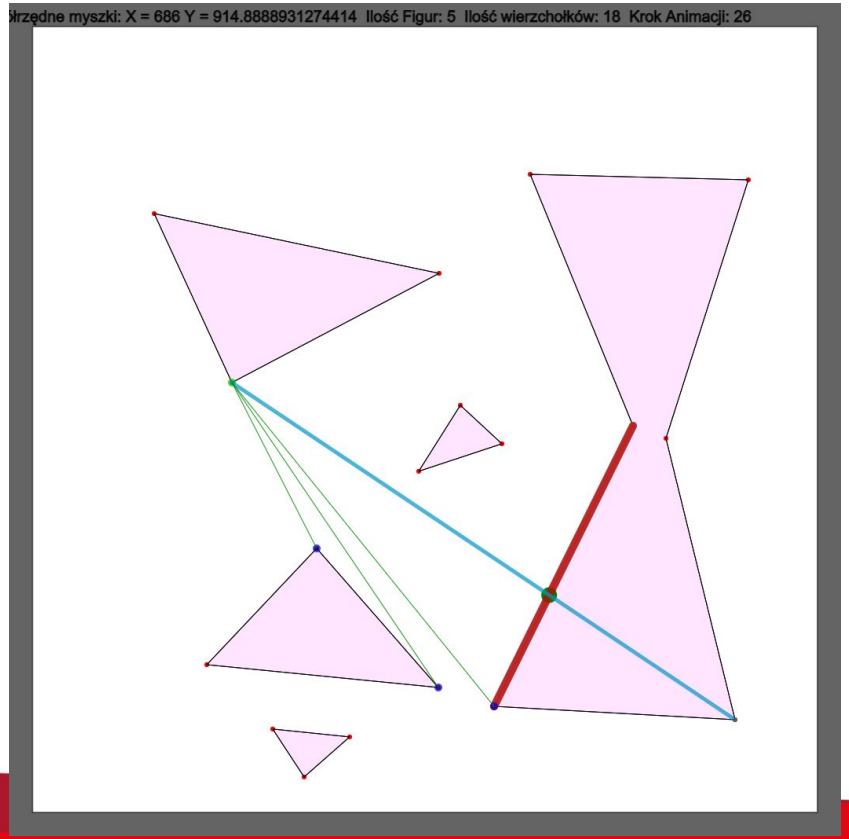
Algorytm widoczneWierzchołki



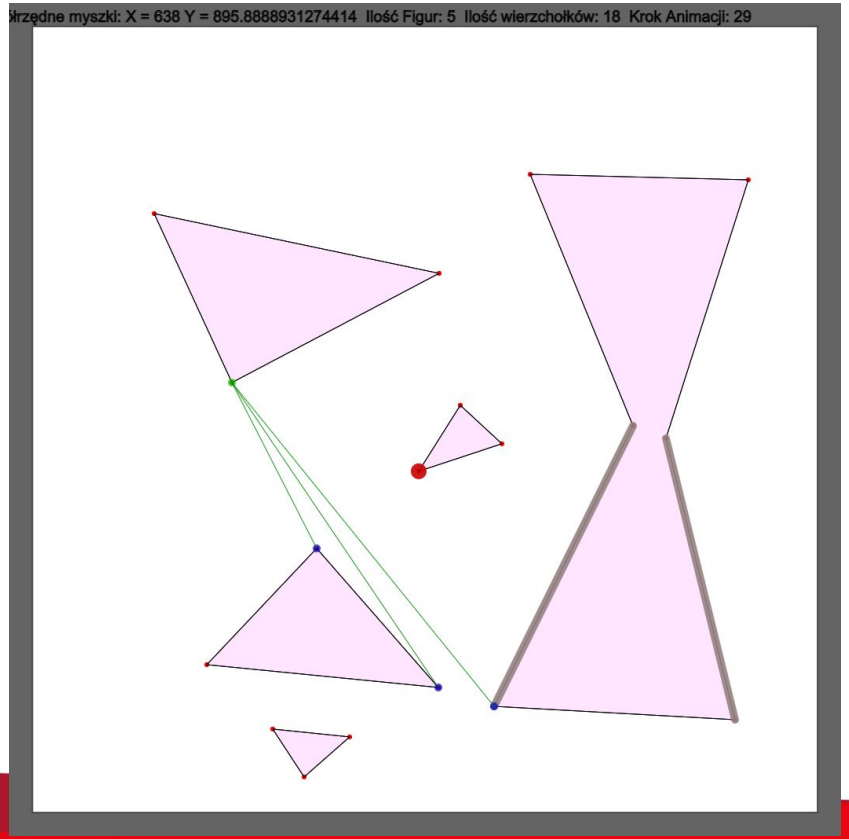
Algorytm widoczneWierzchołki



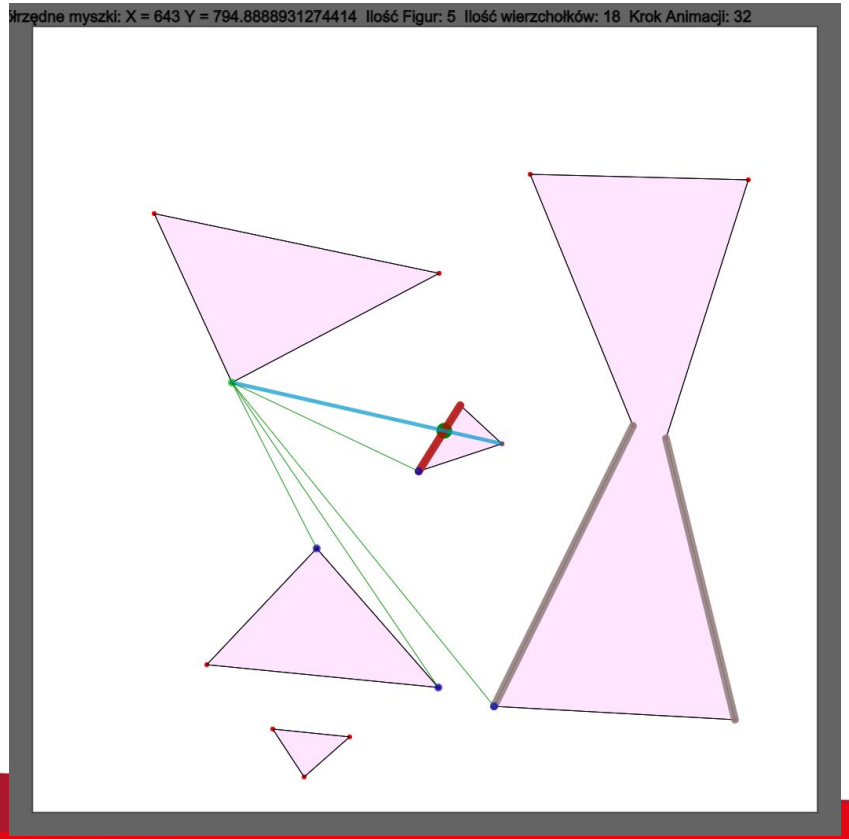
Algorytm widoczneWierzchołki



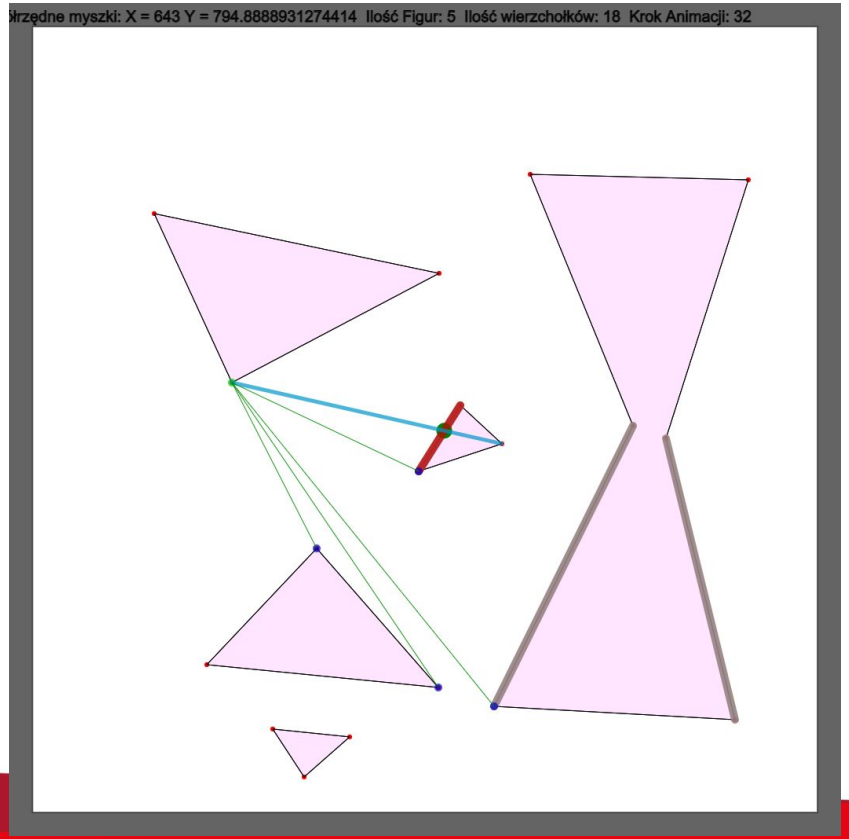
Algorytm widoczneWierzchołki



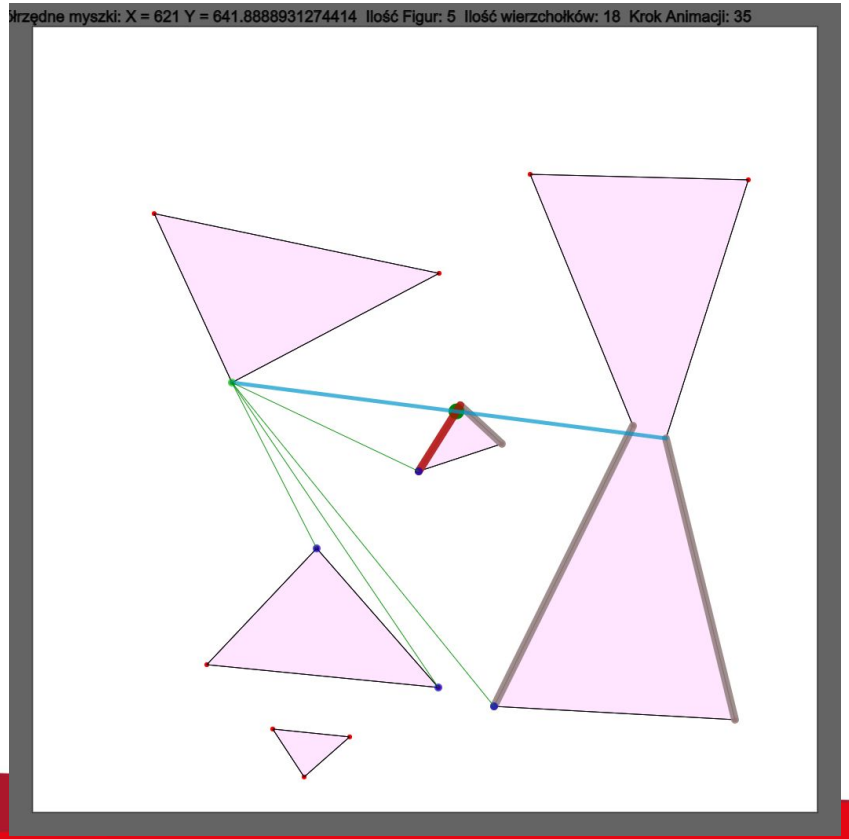
Algorytm widoczneWierzchołki



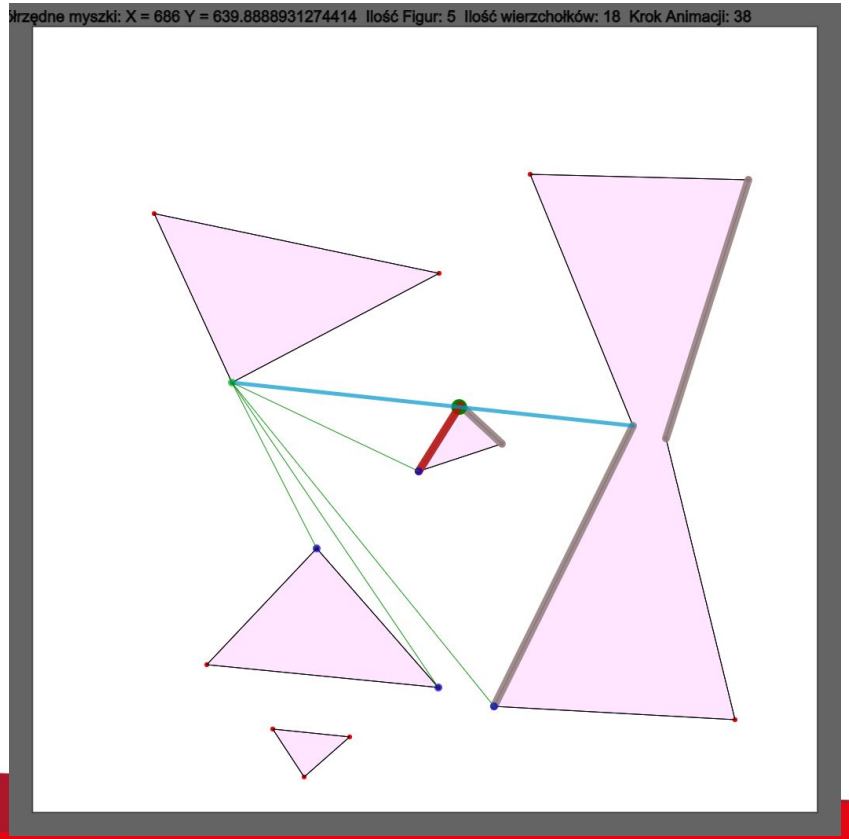
Algorytm widoczneWierzchołki



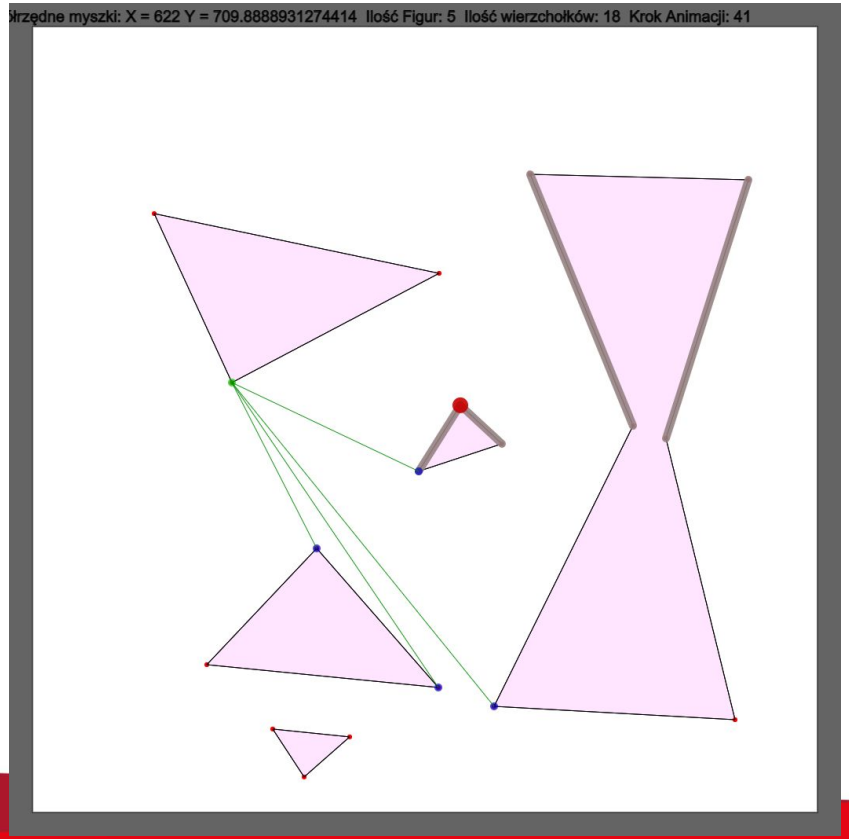
Algorytm widoczneWierzchołki



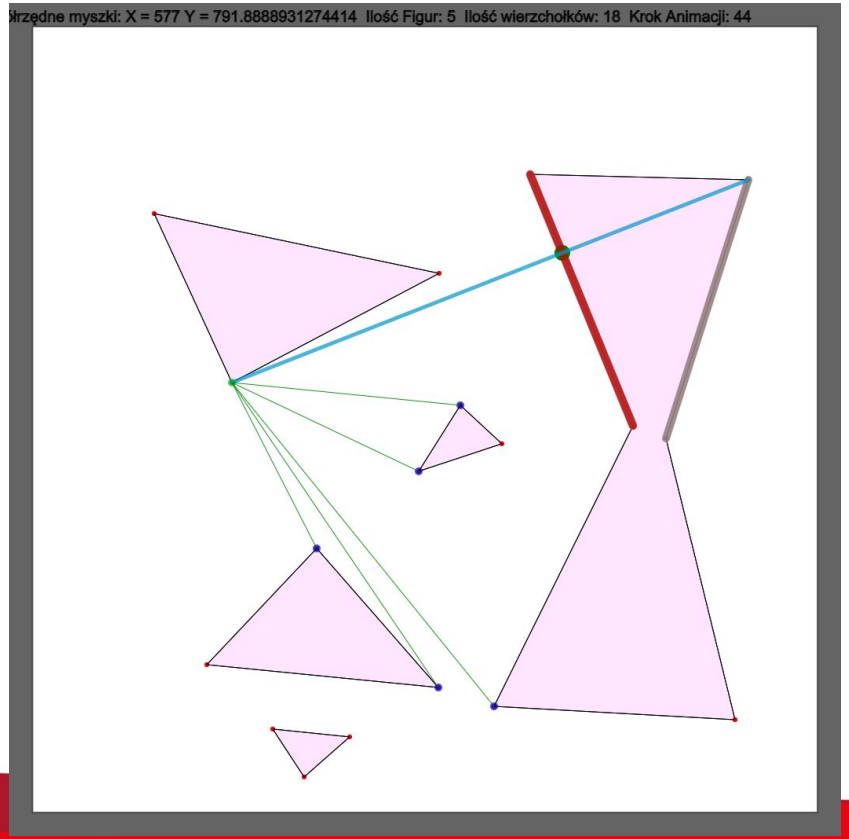
Algorytm widoczneWierzchołki



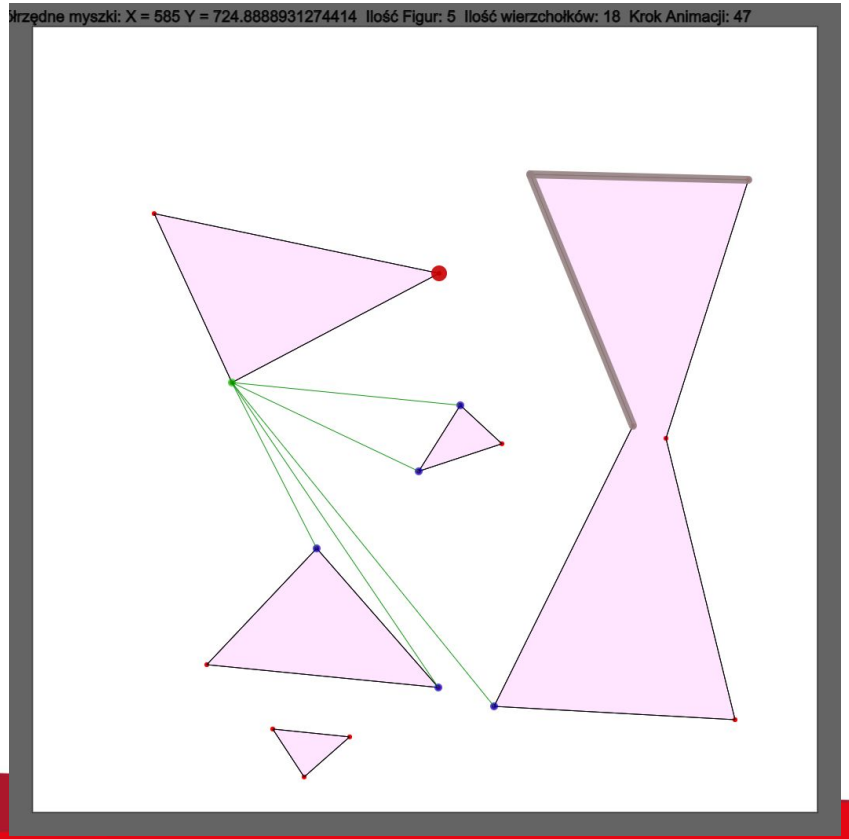
Algorytm widoczneWierzchołki



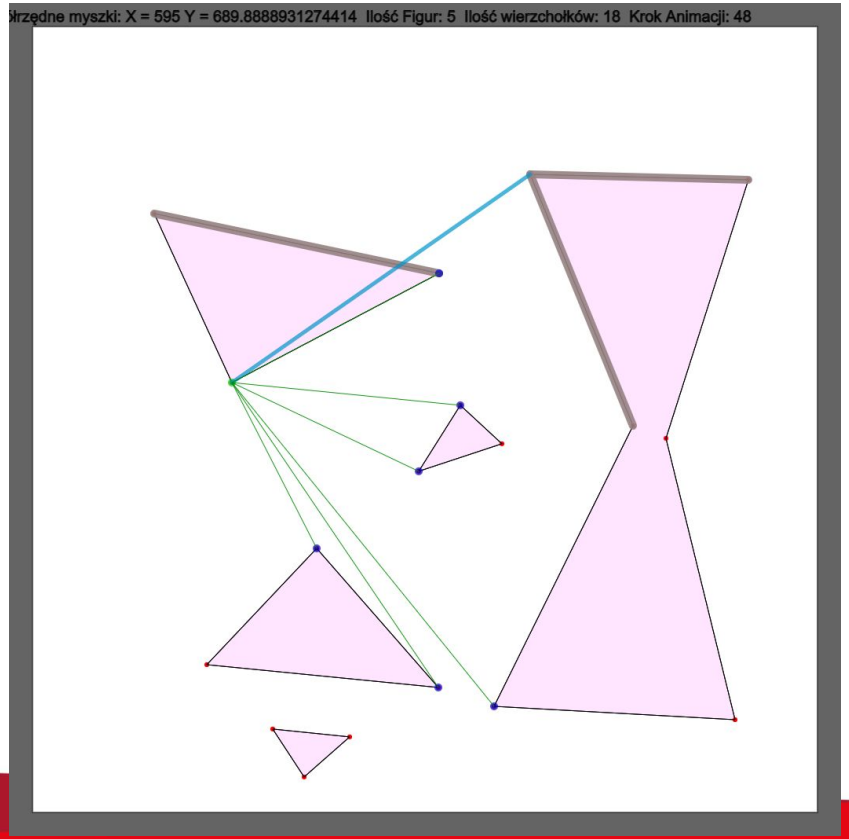
Algorytm widoczneWierzchołki



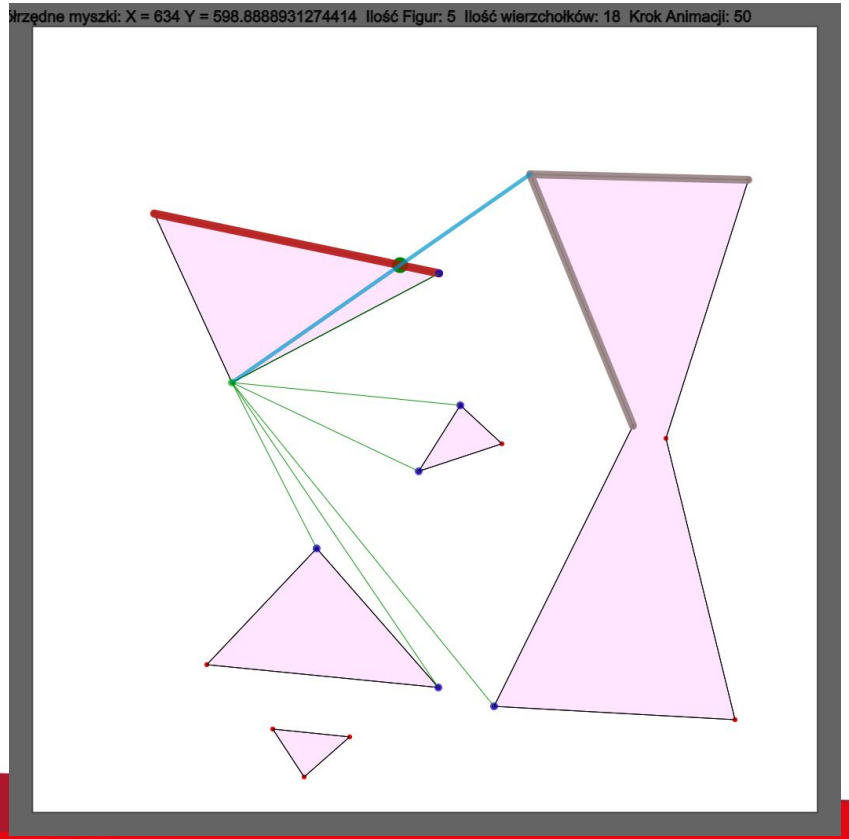
Algorytm widoczneWierzchołki



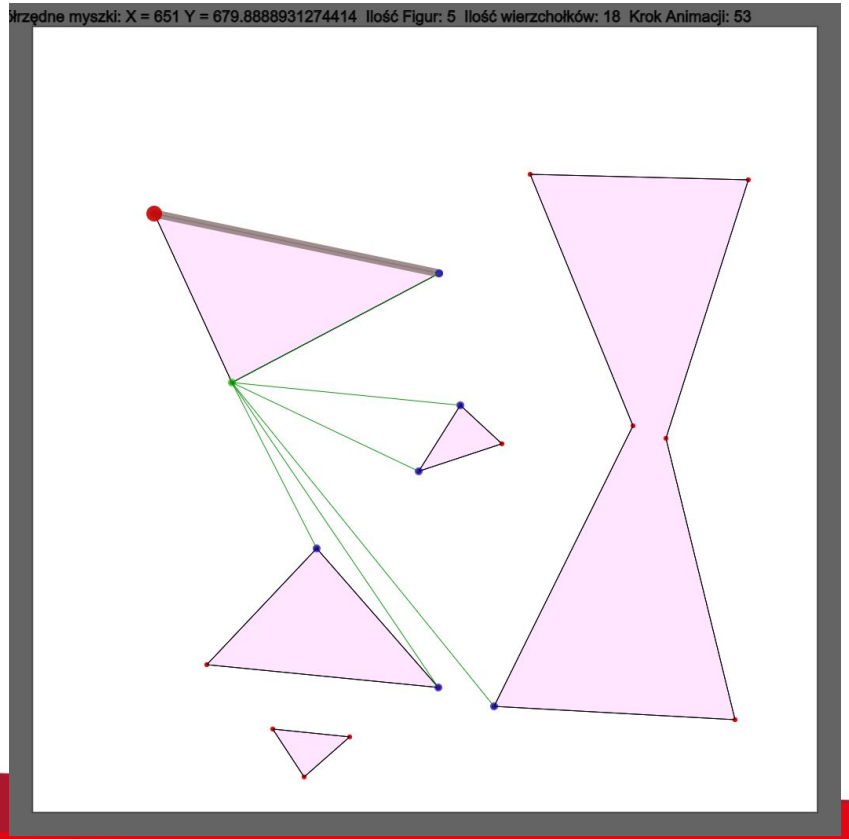
Algorytm widoczneWierzchołki



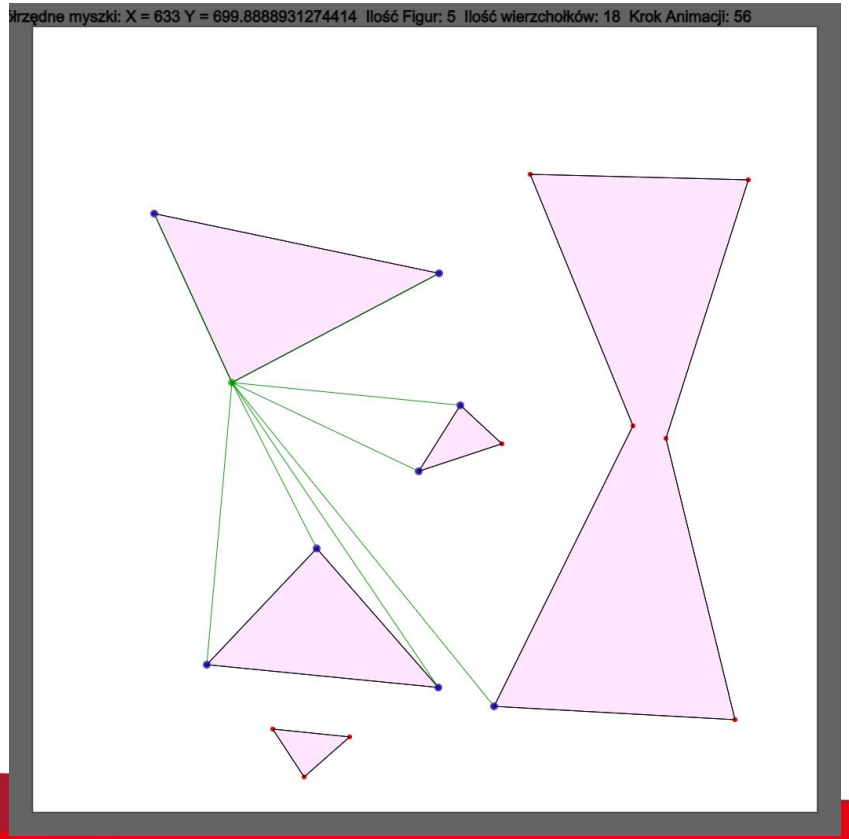
Algorytm widoczneWierzchołki



Algorytm widoczneWierzchołki

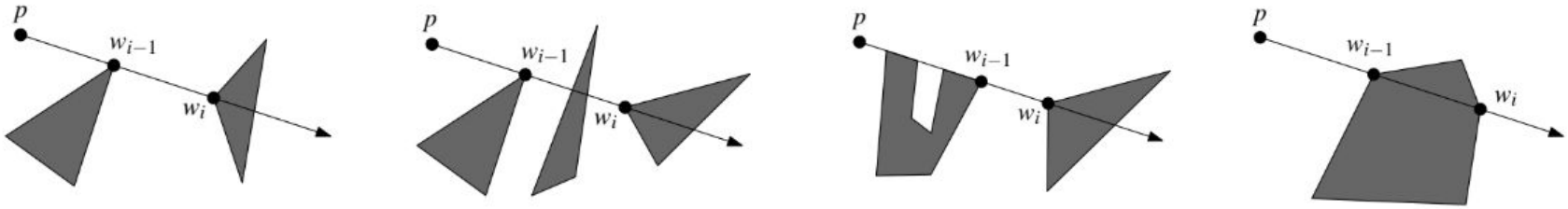


Algorytm widoczneWierzchołki



Zamysł działania algorytmu - widoczność pojedynczego wierzchołka

Sprawdzanie widoczności punktu W (**czyWidoczny**) w większości przypadków sprowadza się do sprawdzenia czy istnieje przecięcie pomiędzy odcinkiem PW, a najbliższą do punktu P krawędzią na miotle. Do pełnego i poprawnego działania algorytmu konieczne będzie również rozpatrywanie sytuacji, gdy napotkane zostaną współliniowe punkty, a także sprawdzanie, czy dwa punkty należące do tego samego wielokąta tworzą odcinek przechodzący przez jego wnętrze.

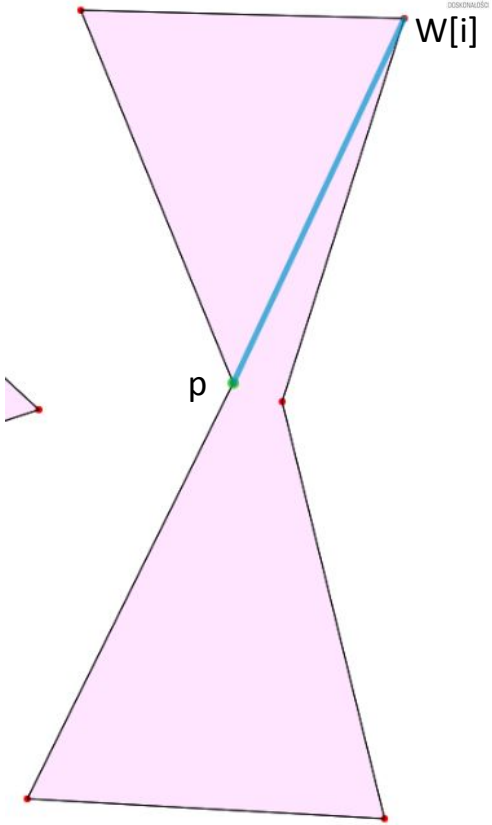


Ilustracje zaczerpnięte z książki:
"Geometria obliczeniowa. Algorytmy i zastosowania" - de Berg M.

Algorytm czyWidoczny

czyWidoczny(W[i]):

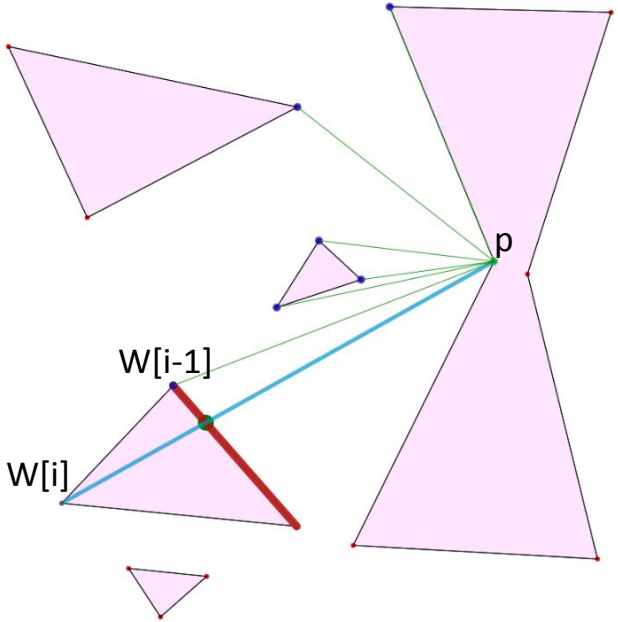
1. **jeżeli prosta pW[i] przechodzi przez wnętrze wielokąta, którego W[i] jest wierzchołkiem: zwróć fałsz**
2. w przeciwnym wypadku, jeżeli i = 0 lub W[i-1] nie leży na prostej pW[i]:
3. e = liść w T, najbardziej po lewej stronie
4. jeżeli e istnieje i pW[i] przecina e: zwróć fałsz
5. w przeciwnym wypadku: zwróć prawdę
6. w przeciwnym wypadku, jeżeli W[i-1] nie jest widoczne: zwróć fałsz
7. w przeciwnym wypadku:
8. znajdź w T krawędź e, która przecina prostą W[i-1]W[i]
9. jeżeli e istnieje: zwróć fałsz
10. w przeciwnym wypadku: zwróć prawdę



Algorytm czyWidoczny

czyWidoczny(W[i]):

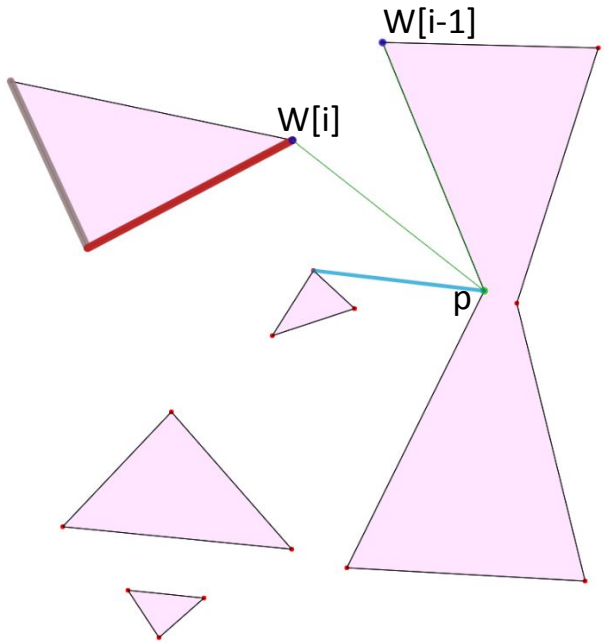
1. jeżeli prosta pW[i] przechodzi przez wnętrze wielokąta, którego W[i] jest wierzchołkiem: zwróć fałsz
2. **w przeciwnym wypadku, jeżeli i = 0 lub W[i-1] nie leży na prostej pW[i]:**
3. e = liść w T, najbardziej po lewej stronie
4. **jeżeli e istnieje i pW[i] przecina e: zwróć fałsz**
5. w przeciwnym wypadku: zwróć prawdę
6. w przeciwnym wypadku, jeżeli W[i-1] nie jest widoczne: zwróć fałsz
7. w przeciwnym wypadku:
8. znajdź w T krawędź e, która przecina prostą W[i-1]W[i]
9. jeżeli e istnieje: zwróć fałsz
10. w przeciwnym wypadku: zwróć prawdę



Algorytm czyWidoczny

czyWidoczny(W[i]):

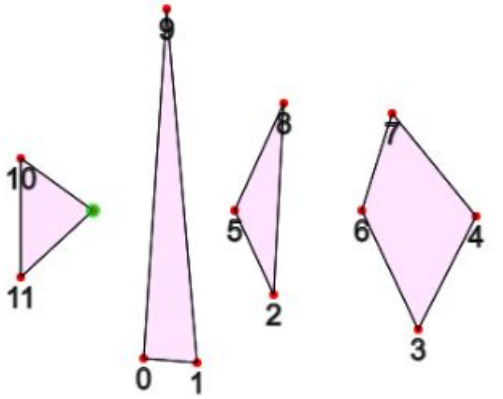
1. jeżeli prosta pW[i] przechodzi przez wnętrze wielokąta, którego W[i] jest wierzchołkiem: zwróć fałsz
2. **w przeciwnym wypadku, jeżeli i = 0 lub W[i-1] nie leży na prostej pW[i]:**
3. e = liść w T, najbardziej po lewej stronie
4. jeżeli e istnieje i pW[i] przecina e: zwróć fałsz
5. **w przeciwnym wypadku: zwróć prawdę**
6. w przeciwnym wypadku, jeżeli W[i-1] nie jest widoczne: zwróć fałsz
7. w przeciwnym wypadku:
8. znajdź w T krawędź e, która przecina prostą W[i-1]W[i]
9. jeżeli e istnieje: zwróć fałsz
10. w przeciwnym wypadku: zwróć prawdę

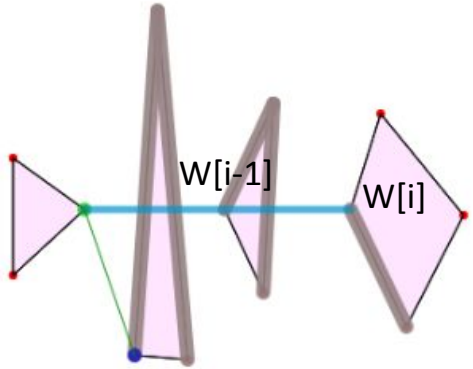


Algorytm czyWidoczny

czyWidoczny(W[i]):

1. jeżeli prosta pW[i] przechodzi przez wnętrze wielokąta, którego W[i] jest wierzchołkiem: zwróć fałsz
2. w przeciwnym wypadku, jeżeli i = 0 lub W[i-1] nie leży na prostej pW[i]:
3. e = liść w T, najbardziej po lewej stronie
4. jeżeli e istnieje i pW[i] przecina e: zwróć fałsz
5. w przeciwnym wypadku: zwróć prawdę
6. **w przeciwnym wypadku, jeżeli W[i-1] nie jest widoczne: zwróć fałsz**
7. w przeciwnym wypadku:
8. znajdź w T krawędź e, która przecina prostą W[i-1]W[i]
9. jeżeli e istnieje: zwróć fałsz
10. w przeciwnym wypadku: zwróć prawdę

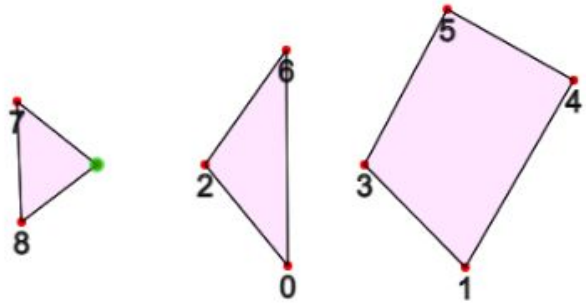


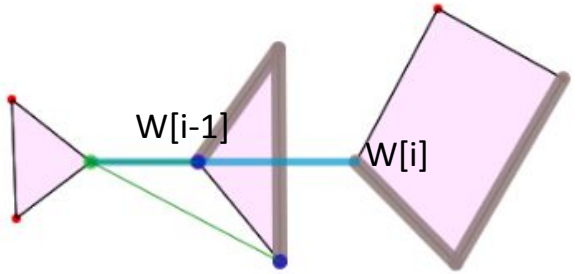


Algorytm czyWidoczny

czyWidoczny(W[i]):

1. jeżeli prosta pW[i] przechodzi przez wnętrze wielokąta, którego W[i] jest wierzchołkiem: zwróć fałsz
2. w przeciwnym wypadku, jeżeli i = 0 lub W[i-1] nie leży na prostej pW[i]:
3. e = liść w T, najbardziej po lewej stronie
4. jeżeli e istnieje i pW[i] przecina e: zwróć fałsz
5. w przeciwnym wypadku: zwróć prawdę
6. w przeciwnym wypadku, jeżeli W[i-1] nie jest widoczne: zwróć fałsz
7. **w przeciwnym wypadku:**
8. **znajdź w T krawędź e, która przecina prostą W[i-1]W[i]**
9. **jeżeli e istnieje: zwróć fałsz**
10. w przeciwnym wypadku: zwróć prawdę

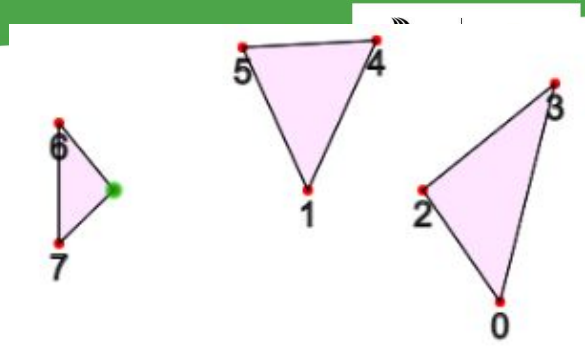


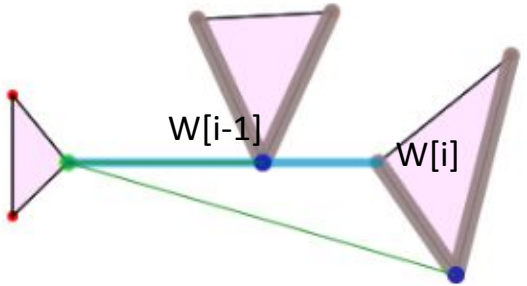


Algorytm czyWidoczny

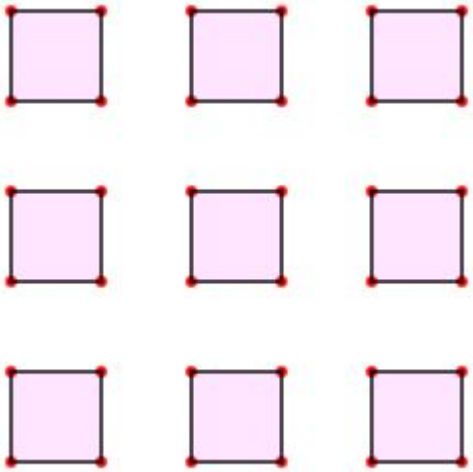
czyWidoczny(W[i]):

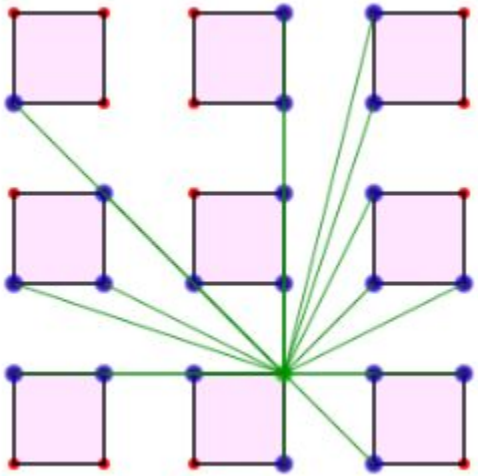
1. jeżeli prosta pW[i] przechodzi przez wnętrze wielokąta, którego W[i] jest wierzchołkiem: zwróć fałsz
2. w przeciwnym wypadku, jeżeli i = 0 lub W[i-1] nie leży na prostej pW[i]:
3. e = liść w T, najbardziej po lewej stronie
4. jeżeli e istnieje i pW[i] przecina e: zwróć fałsz
5. w przeciwnym wypadku: zwróć prawdę
6. w przeciwnym wypadku, jeżeli W[i-1] nie jest widoczne: zwróć fałsz
7. **w przeciwnym wypadku:**
8. **znajdź w T krawędź e, która przecina prostą W[i-1]W[i]**
9. jeżeli e istnieje: zwróć fałsz
10. **w przeciwnym wypadku: zwróć prawdę**

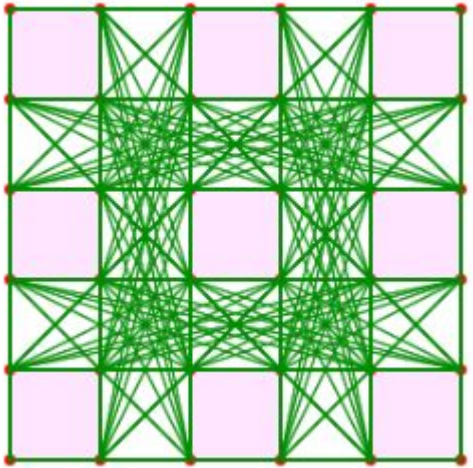




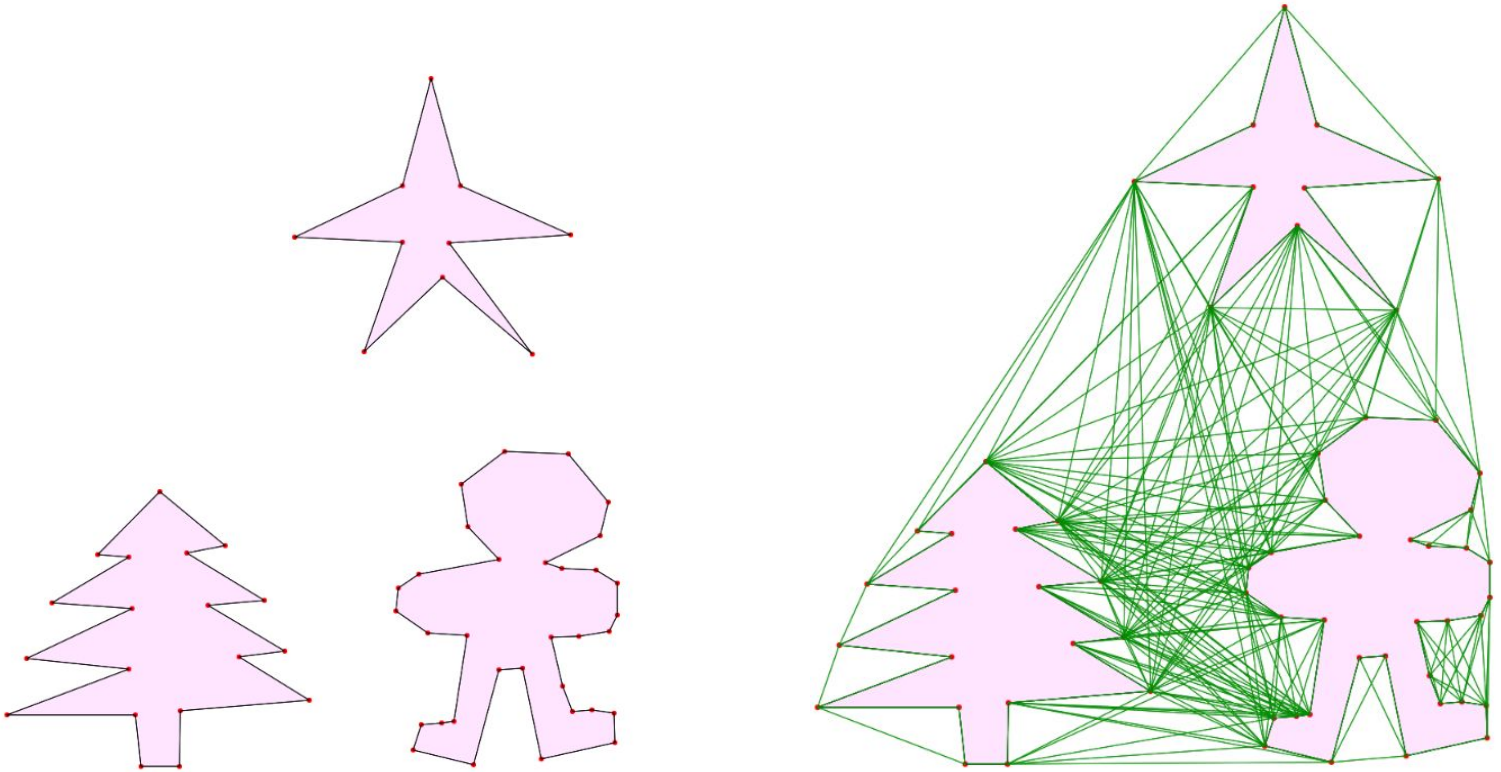
Przykładowe działanie programu - kwadraty



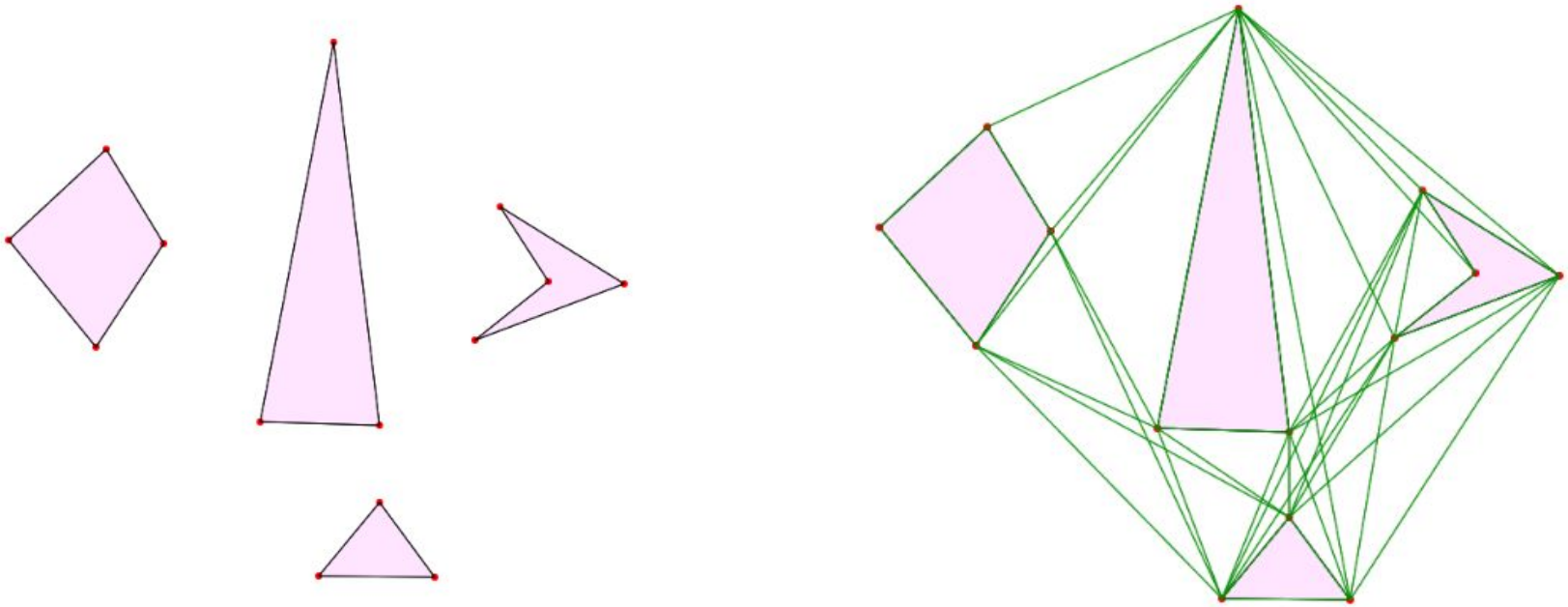




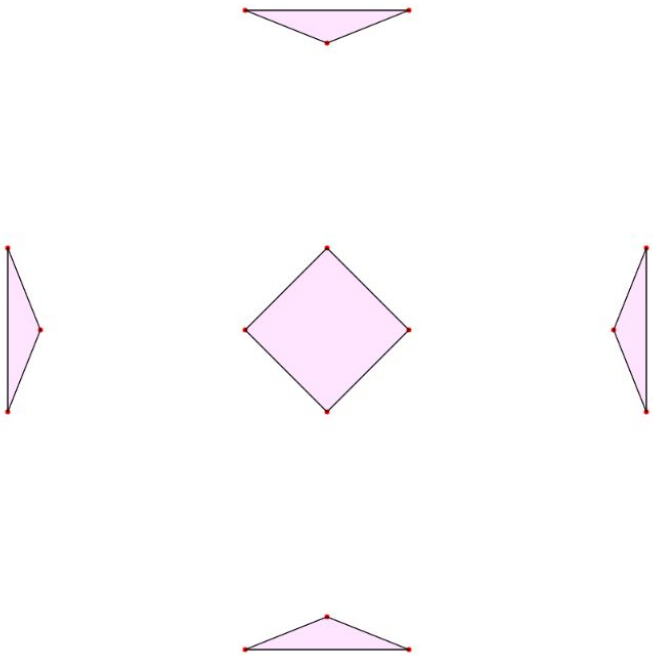
Przykładowe działanie programu - krajobraz świąteczny

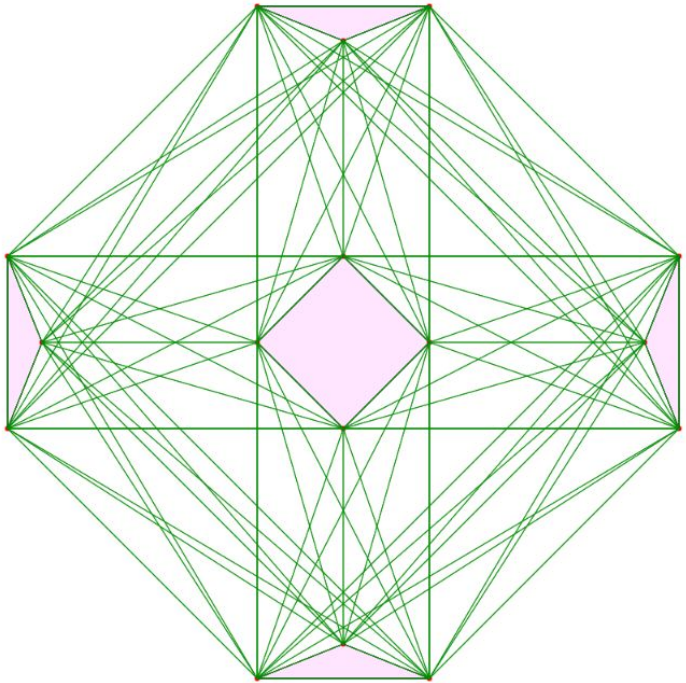


Przykładowe działanie programu



Przykładowe działanie programu





Koniec

Materiały:

- de Berg M., Van Kreveld M., Overmars M. “Geometria obliczeniowa. Algorytmy i zastosowania”

Dziękujemy za uwagę.