Wydział Elektryczny Politechniki Warszawskiej

Języki i Metody Programowania II – projekt 1

Siatkonator

Autor:
Barnaba Turek
barnabaturek@gmail.com

13 czerwca 2012

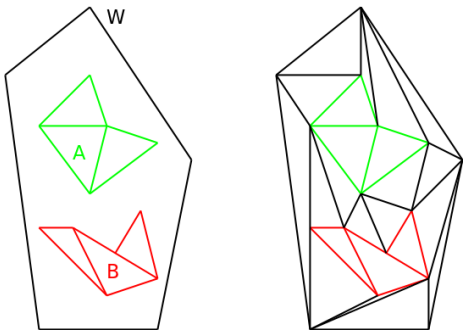
Spis treści

1 Specyfikacja funkcjonalna

1.1 Program

Siatkonator to program sklejający siatki trójkątne. Program przyjmuje jedną lub więcej siatek trójkątnych oraz jeden wielokąt.

Wynikiem działania programu jest siatka trójkątna opisująca zadany wielokąt, która zawiera podane określone siatki.



Rysunek 1: przykład sklejania zadanego wielokątu W i siatek A i B

1.2 Wywołanie

Program nie prowadzi dialogu z użytkownikiem.

Listing 1: Przykładowe wywołanie

```
$ siatkonator -e siatka1.ele -e siatka2.ele polygon.poly
```

Ostatni argument programu określa nazwę pliku (wraz ze ścieżką) w którym opisany jest wielokąt. Nie podanie tego argumentu spowoduje błąd wykonania programu.

Wcześniejsze argumenty określają opcje wykonania programu i nie są wymagane. Kolejność argumentów nie ma znaczenia.

Dostępne są następujące opcje:

`-e <name>` dodaje siatkę, która zostanie "sklejona" z wielokątem. Program zakłada, że istnieje także plik o takiej samej nazwie bazowej z rozszerzeniem `node` opisujący wierzchołki.

`-o <name>` podaje nazwę pliku wyjściowego. W przypadku braku tego parametru wynik zostanie wypisany na standardowe wejście (najpierw plik ele, potem node).

`-q <angle>` określa minimalny kąt w dowolnym trójkącie wygenerowanej siatki (nie zmienia siatek wklejanych).

`-a <area>` określa maksymalną powierzchnie dowolnego trójkąta w generowanej siatce (j.w.).

`-d <eps>` określa epsilon używany do porównywania koordynatów punktów w celu stwierdzenia, czy należy je połączyć w jeden punkt. Domyślnie program przyjmuje 0.005.

W przypadku powodzenia program zwraca zero. W przeciwnym wypadku program zwraca jeden z kodów błędu:

`kod 1` Błąd otwarcia pliku (spowodowany np. brakiem uprawnień lub pliku).

`kod 2` Błąd wczytania pliku (spowodowany błędnym formatem któregoś z plików źródłowych).

`kod 3` Błędny format argumentów linii poleceń.

Ponadto program w czasie działania wypisuje informacje o swoim aktualnym stanie na wyjście `STDERR`.

1.3 Pliki

Siatkonator korzysta z plików w formatach `poly`, `ele` i `node`.

poly format pliku opisującego wielokąt

ele format pliku opisującego z których wierzchołków składają się trójkąty siatki

poly format pliku opisującego wierzchołki siatki

Wszystkie wykorzystywane formaty są zgodne z formatami wykorzystywanymi przez program triangle.

1.4 Obsługa atrybutów

Atrybuty są zachowywane. Wymagane jest, aby wszystkie pliki wejściowe miały taką samą liczbę atrybutów.

W przypadku połączenia dwóch wierzchołków, zostaną zachowane atrybuty jednego z nich.

1.5 Rozwiązywanie konfliktów znaczników wierzchołków

Wczytując pliki program zapamiętuje najniższy i najwyższy znacznik wierzchołka w danym pliku. Następnie program oblicza stałą, o którą należy przesunąć znacznik wierzchołka w tym pliku, aby nuniknąć konfliktu.

Listing 2: Przykład

```
1  Jezeli w~pliku A.poly znaczniki wierzcholkow sa z~zakresu 1..10,
2  I~w~pliku B.node znaczniki sa z~zakresu 3..10,
3  I~po wczytaniu B.node najwiekszy znaleziony znacznik bedzie 10,
4  To po wczytaniu A.poly najmniejszy znaleziony znacznik jest 3,
5  Wiec nalezy wszystkie znaczniki wierzcholkow znalezionych w~A.
       poly przesunac o~(10-3)
```

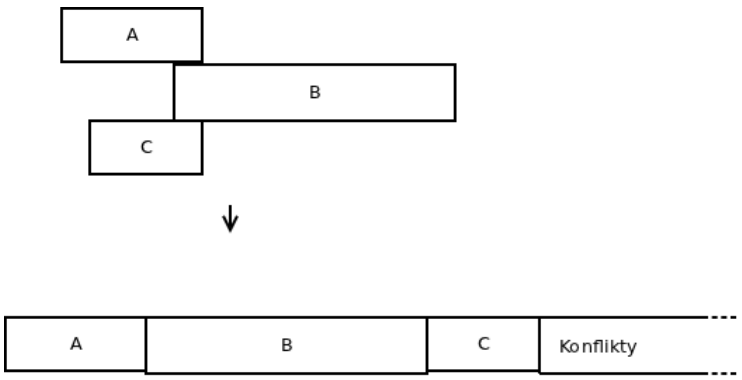
W związku z tym wynik działania programu **zależy** od kolejności podanych plików wejściowych (Z tym ze znaczniki z otoczki zawsze maja najwyższe wartości).

Po takim podziale pozostała przestrzeń zmiennej używana jest do rozwiązywania konfliktów wynikających z tego, że dwa wierzchołki o różnych znacznikach znalazły się w tym samym miejscu.

Jeśli oba wierzchołki zajmujące te same koordynaty mają różną wartość znaczników, zostaje im przydzielona nowa wartość z zakresu wartości przeznaczonego na konflikty za pomocą prostej funkcji mieszającej

```
1 (znacznik\ A + znacznik\ B) % (wolne miejsce w~zakresie INTa) +
      offset zakresu
```

Takie rozwiązanie gwarantuje, że elementy będące połączeniem danych znaczników zawsze będą miały ten sam znacznikw pliku wynikowym.



Rysunek 2: przykład podziału przestrzeni znaczników

1.6 Instalacja

Program nie wymaga instalacji.

1.7 Kompilacja

Program można skompilować korzystając z programu `GNU make`. Proces kompilacji jak i sam program był testowany na systemie Linux.

Program ma jedną zależność, bibliotekę *argtable2*.

Aby uniknąć instalowania pakietu argtable-dev z repozytorium dystrybucji, dołączyłem go do repozytorium SVN. Wywołanie polecenia `make` powinno zbudować tę zależność, jednak kompilacja wymaga programu `cmake`.

2 Specyfikacja implementacyjna

2.1 Struktura programu

Program podzielony jest na następujące moduły:

siatkonator.c Plik zawierający główną funkcję odpowiedzialną za wczytanie argumentów i zależnie od nich wykonanie algorytmu sklejania siatek.

siatkonator_IO Katalog (moduł) zawierający funkcje związane z czytaniem i zapisywaniem plików *node*, *ele* i *poly* do struktur `triangulateio`.

mesh_ops.c Plik zawierający funkcje obsługujące logikę sklejania siatek (znajdowanie wielokąta otaczającego siatkę, wycinanie dziur w siatkach i łączenie siatek).

common.c Plik zawierający funkcje pomocnicze (takie jak wypisywanie wiadomości). `common.h` zawiera także struktury używane do przekazywania danych wewnątrz programu.

markers.c Plik zawierający funkcje służące przesuwaniu i rozwiązywaniu konfliktów znaczników.

siatkonator_hash.c Plik zawierający funkcje i strukturę danych obsługującą tablicę mieszającą używaną do znajdowania wielokąta otaczającego siatkę..

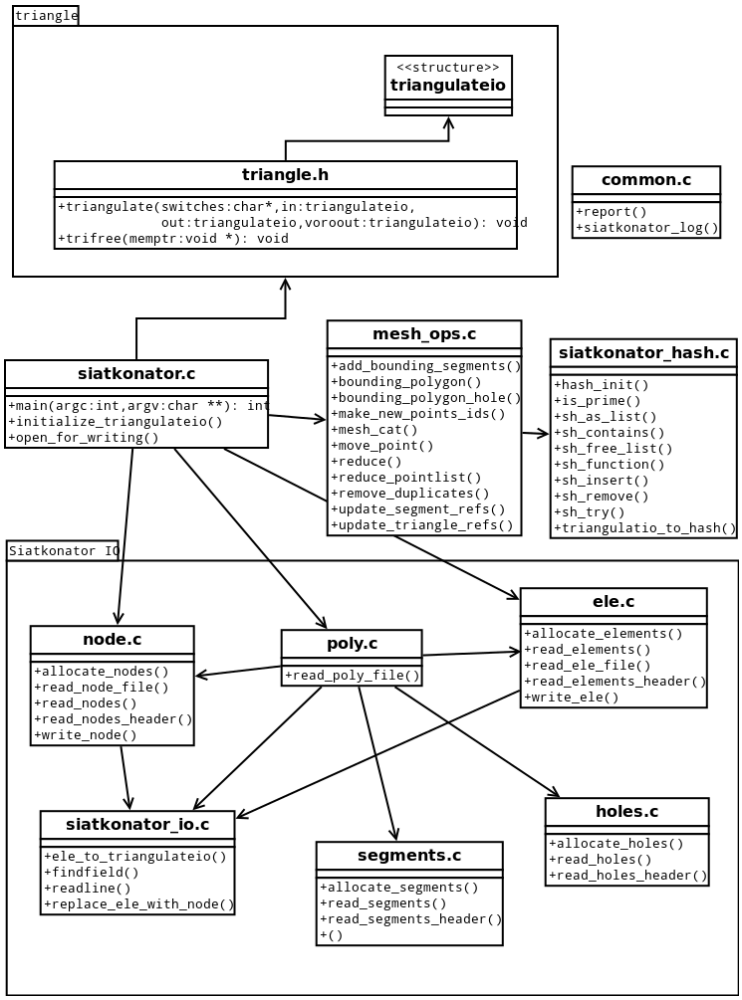
2.1.1 Funkcje modułu `mesh_ops`

`bounding_polygon` funkcja przyjmująca wskaźnik na wczytaną siatkę i znajdująca otaczający ją wielokąt (który należy wyciąć z otoczki przed przeprowadzeniem triangulacji).

`bounding_polygon_hole` funkcja dodająca do otoczki dziurę o współrzędnych równych środkowi ciężkości pierwszego trójkąta z dodawanej siatki.

`add_bounds_as_segments` funkcja dodająca znaleziony wcześniej otaczający wielokąt do otoczki.

`mesh_cat` Funkcja przyjmująca dwa wskaźniki na siatki i dodająca elementy drugiej siatki do elementów pierwszej siatki.



Rysunek 3: Diagram klas

`remove_duplicates` Funkcja usuwająca wierzchołki o identycznych lub podobnych współrzędnych.

2.2 Algorytmy

Listing 3: Pseudokod algorytmu głównej funkcji programu sklejającej siatki

```
1  wczytaj dane z pliku poly do struktury p
2  dla kazdego pliku .ele:
3    wczytaj dane z plikow ele i nodes do struktury E
4    znajdz w strukturze E krawedzie, ktore wystepuja tylko raz
5    dodaj te krawedzie do struktury P jako sekcje
6    dodaj dziure do struktury P wewnatrz pierwszego trojkata ze
         struktury E
7  wykonaj triangulacje na strukturze p do siatki S
8  dla kazdego pliku .ele:
9    wstaw siatke z pliku do dziur w siatce S
```

```
  usun powtarzajace sie wierzcholki
  zapisz wynik triangulacji do plikow ele i node
```

Listing 4: Pseudokod funkcji szukającej wielokąta otaczającego daną siatkę: `bounding_polygon`

```
  H - tablica mieszajaca

  Dla kazdego wierzcholka w:
    jezeli H[w] istnieje:
      Dodaj w do H
    else:
      Usun w z~H

  Dla kazdego wierzcholka w:
    jezeli H[w] istnieje:
      dodaj w~do wielokata otaczajacego
      usun w~z~wielokata otaczajacego

  Dodaj dziure w~wielokacie otaczajacym w~miejscu srodka ciezkosci
      pierwszego elemenetu.
```

2.3 struktury danych

Struktura danych używana do komunikacji między funkcjami to struktura `triangulateio` dostarczona wraz z programem triangle. Struktura ta pozwala opisać siatkę na każdym etapie jej budowania (także siatki wyjściowe).

Jej wadą jest fakt, że nie jest specjalizowana do konkretnych zadań i wiele pól nie będzie miało zastosowania. Mimo to jest już napisana, udokumentowana i mogę być pewien, że zawiera wszystkie potrzebne pola.

Zastosowanie innej struktury, jako struktury pośredniej wymagałoby tłumaczenia jej do `triangulateio` przed wykonaniem funkcji `triangulate()`.

Zastosowana jest także tablica mieszająca, w której konflikty rozwiązywane są za pomocą listy. Tablica potrafi też zwrócić wszystkie swoje elementy w formie listy.

3 Testy aplikacji

3.1 Testy automatyczne

Aplikacja zawiera kilkanaście testów automatycznych napisanych na początku w języku Ruby. Testy te sprawdzają rzeczy łatwe do sprawdzenia z zewnątrz - obsługę argumentów, poprawność wczytywania plików. Przy okazji testy były bardzo pomocne jako wczesny wskaźnik tego, że coś się zepsuło - bardzo podstawowe testy regresyjne, polegające na uruchomieniu programu i sprawdzeniu jego kodu wyjścia.

Zostały zaimplementowane następujące testy:

Listing 5: Testy automatyczne

```
siatkonator IO
  when it reads square.poly
    should read poly file without error
    should read nodes coordinates and attributes properly
```

```
    should read nodes segments and holes properly
  when it reads square-firstnode-1.poly
    should read poly file without error
    should read nodes coordinates and attributes properly
    should read nodes segments and holes properly
  when it reads square-firstnode-1.poly
    should read poly and ele files without error
    should read 1.ele and 1.node properly
    should read 2.ele and 2.node properly

siatkonator argument reader
  should require at least one option
  should be ok when option given
  should fail when the only option is preceded with switch
  should notify that max surface was set
  should notify that min angle was set
  should notify that mesh files were provided
  should notify that output file was set

Finished in 0.09719 seconds
16 examples, 0 failures
```

3.2 Tryb debug

Aplikacja wyposażona jest w tryb wspomagający testowanie, w którym wypisuje swój stan na standardowe wyjście. To, czy aplikacja działa w tym trybie zależy od flag kompilacji.

Ponieważ nie chciałem uzależniać moich funkcji od zmiennych globalnych i wypisywania danych (ze względu na *Single Responsibility Principle*) ograniczyłem całe wypisywanie do funkcji siatkonator log, która w razie czego może zostać zastąpiona pustą funkcją. Zbieranie informacji diagnostycznych w funkcji main byłoby bardzo kłopotliwe i spowodowałoby, że funkcja rozrosłaby się jeszcze bardziej.

Aplikacja wypisuje zawartość struktur `triangulateio` na następujących etapach:

- wczytanie plików (poly, node+ele);
- znalezienie wielokąta ograniczającego;
- dodanie wielokąta ograniczającego i dziury;
- po/przed wywołaniu funkcji triangulate;
- po ponownym dodaniu siatek
- po usunięciu duplikatów

Ponadto program wypisuje różne informacje podsumowujące jego działanie, takie jak np. informacja z jakich plików będzie korzystał i ile wierzchołków wczytał.

3.3 Testy eksploracyjne

Testy, których większość była wykonywana w czasie pisania programu polegające na sprawdzaniu, czy robi to co powinien robić. Do tych testów stosowałem

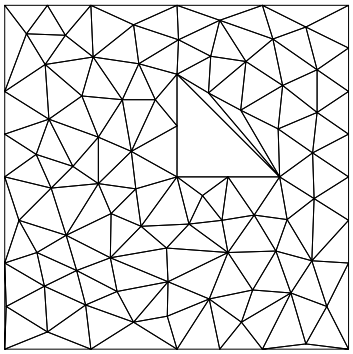
własne pliki, przykładowe pliki dostępne na stronie `wikidyd`, pliki testowe kolegi oraz plik przedstawiający literę 'A' będący częścią dystrybucji Triangle.

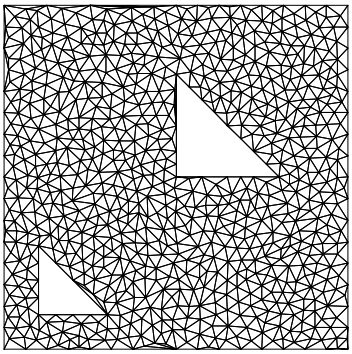
3.3.1 Podstawowe testy

Pierwsze dwa proste testy, które przeprowadziłem dotyczyły kwadratowej otoczki. Testy korzystały z najprostszych możliwych siatek - trójkątów. W jednym teście wklejono do otoczki jedną siatkę, w drugim dwie.

Komendy, za pomoca ktorych uzyskalem te wyniki sa nastepujace[1]:

```
./bin/siatkonator -a 0.01 -e spec/data/2.ele -o test spec/data/
    square-no-holes.poly
./bin/siatkonator -a 0.001 -e spec/data/1.ele -e spec/data/2.ele -o
     test spec/data/square-no-holes.poly
```





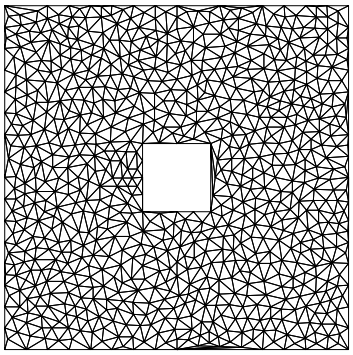
3.3.2 Test wyznaczania otoczki

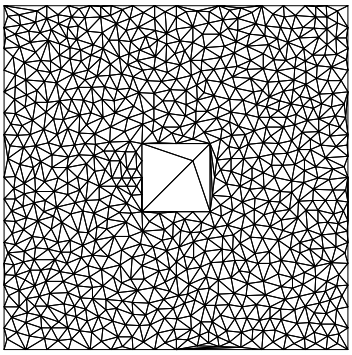
W następnym teście skorzystałem z bardziej złożonej siatki - kwadratu złożonego z trzech trójkątów. Miało to na celu upewnienie się, że funkcje określające wielokąt otaczający działają zgodnie z założeniami.

W ramach tego testu wydałem dwa razy te samą komendę, ale za pierwszym razem wykomentowałem wywołanie funkcji wklejającej siatkę do przeznaczonej dla niej dziury:

```
# wykomentowano 'mesh_cat(&out, meshes + i);'
./bin/siatkonator -a 0.001 -e spec/data/kwadrat.ele -o test spec/
    data/square-no-holes.poly
# odkomentowano
./bin/siatkonator -a 0.001 -e spec/data/kwadrat.ele -o test spec/
    data/square-no-holes.poly
```

[1] wszystkie pliki z których korzystałem do testów są w repozytorium



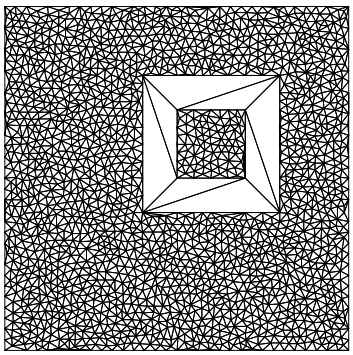


Dzięki temu zabiegowi można się przekonać, że wielokąt otaczający, w tym wypadku kwadrat, został wyznaczony poprawnie oraz że zostały z niego usunięte wszystkie wierzchołki i segmenty, które nie są częścią wielokąta otaczającego.

3.3.3 Element mechaniczny z dziurką

Test polegał na sklejeniu otoczki z siatką, która zawierała dziurę. jego celem jest sprawdzenie, czy dziura zostanie poprawnie wypełniona trójkątami.

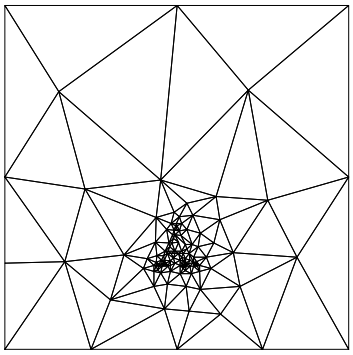
```
./bin/siatkonator -a 0.001 -e spec/data/kwadrat.ele -o test spec/
    data/square-no-holes.poly
```



3.3.4 Test z większym plikiem

W ramach tego testu skorzystałem z pliku poly opisującego literę 'A' złożoną z kilkudziesięciu trójkątów, który jest dystrybuowany razem z trianglem. Test ten zawiera stosunkowo dużo wierzchołków i pozwolił wykryć problem z implementacją hasha.

```
./bin/siatkonator -e spec/data/cieciurm/A.1.ele -o test spec/data/
    square-large.poly
```



3.3.5 Markery i atrybuty

Po włączeniu opcji debug uruchomiłem następujące polecenie, aby sprawdzić że przepisywanie markerów działa zgodnie z założeniami, a atrybuty są zachowywane:

```
./bin/siatkonator -e spec/data/attrs+markers/2attr-1mark.ele spec/
    data/attrs+markers/square-2attr-1mark.poly
```

Pliki wejściowe tego testu mają następującą charakterystykę:

- plik node zawiera atrybuty od 8080 do 8085 i markery od 1 do 3
- plik poly zawiera atrybuty od 1024 do 1031 i markery równe 2

W wyniku uruchomienia programu w trybie `debug`, uzyskałem następującą informację:

```
marker range change for mesh 0: 1-3 --> 0-2
marker range change for mesh 1: 2-2 --> 3-3
```

Należy się więc spodziewać, że w pliku wyjściowym pliki z atrybutami 808* będą miały markery zawierające się w przedziale 0 - 2, natomiast pliki z atrybutami 10** - równe 3.

Wynik testu to potwierdza:

```
0 -2.000000 -2.000000 1024.000000 1025.000000 3
1 -2.000000 3.000000 1026.000000 1027.000000 3
2 3.000000 3.000000 1028.000000 1029.000000 3
3 3.000000 -2.000000 1030.000000 1031.000000 3
4 1.100000 1.100000 8080.000000 8081.000000 0
5 1.300000 1.100000 8082.000000 8083.000000 1
6 1.100000 1.300000 8084.000000 8085.000000 2
```

3.3.6 Pozostałe

Udało mi się wygenerować poprawne siatki korzystając z plików trap i tri zamieszczonych na wikidydzie. Nie udało mi się znaleźć żadnych plików wejściowych o poprawnej strukturze, z których program generowałby nieprawidłowe siatki (z wyjątkiem kombinacji plików tkaich, że siatka nie zawierała się w otoczce).

Pliki o niepoprawnej strukturze powodują nieprawidłowe działąnie programu.