Wydział Elektryczny Politechniki Warszawskiej

Języki i Metody Programowania II – projekt 1

Siatkonator

Autor:
Barnaba Turek
barnabaturek@gmail.com

12 czerwca 2012

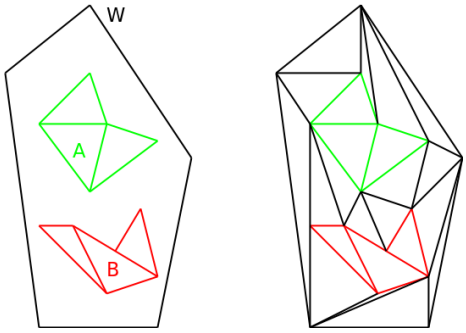
Spis treści

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).

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

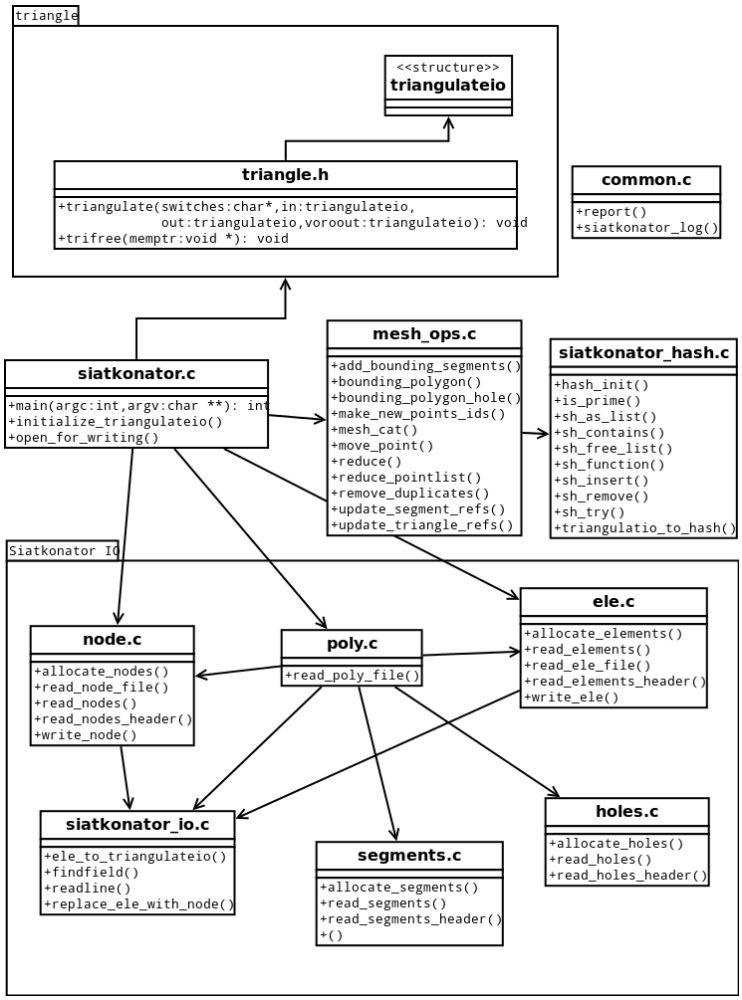
ele format pliku opisującego z których wierzchołków składają się trójkąty siatki

poly format pliku opisującego wierzchołki siatki

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

2 Specyfikacja implementacyjna

2.1 Struktura programu



Rysunek 2: Diagram klas

Program podzielony jest na trzy pliki:

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).

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 gotową 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.

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

2.2 Algorytmy

Listing 2: 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
10 usun powtarzajace sie wierzcholki
11 zapisz wynik triangulacji do plikow ele i node
```

Listing 3: Pseudokod funkcji szukającej wielokąta otaczającego daną siatkę: `bounding_polygon`

```
1  H - tablica mieszajaca
2
3  Dla kazdego wierzcholka w:
4    jezeli H[w] istnieje:
5      Dodaj w do H
6    else:
7      Usun w z~H
8
9  Dla kazdego wierzcholka w:
10   jezeli H[w] istnieje:
11     dodaj w~do wielokata otaczajacego
12     usun w~z~wielokata otaczajacego
13
14 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.

2.4 Obsługa atrybutów i markerów

Atrybuty i markery nie są obsługiwane.

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 4: 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
```

```
25 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

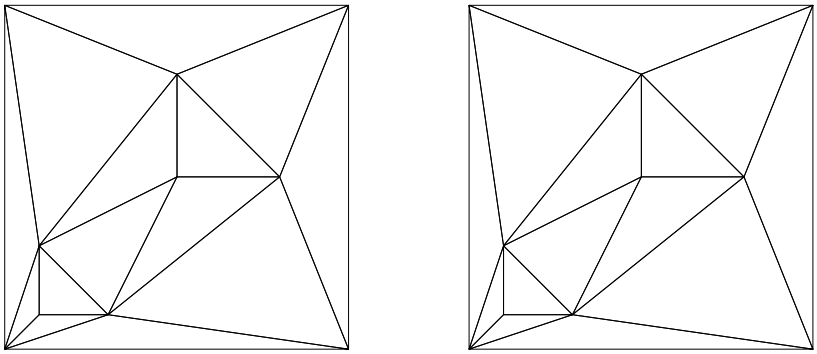
Testy, których większość była wykonywana w czasie pisania programu polegające na sprawdzaniu, czy robi to co powinien robić. Do tych testów stosowałem własne pliki, przykładowe pliki dostępne na stronie `wikidyd`, pliki testowe kolegi oraz plik przedstawiający literę 'A' będący częścią dystrybucji Triangle.

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



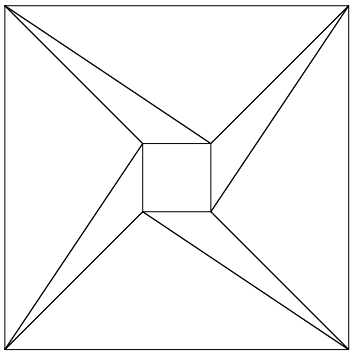
3.3.2 Test wyznaczania otoczki

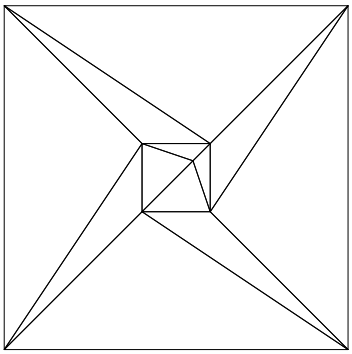
W następnym teście skorzystałem z bardziej złożonej siatki - kwadratu złożonego z trzech trójkątów. Miało to na celu upewnienie się, że funkcje określające wielokąt otaczający działają zgodnie z założeniami.

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

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

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



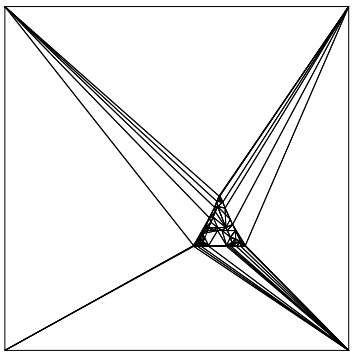


Dzięki temu zabiegowi można się przekonać, że wielokąt otaczający, w tym wypadku kwadrat, został wyznaczony poprawnie oraz że zostały z niego usunięte wszystkie wierzchołki i segmenty, które nie są częścią wielokąta otaczającego.

3.3.3 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.

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



3.3.4 Pozostałe

Nie udało mi się przejść testu `trap.poly`.