SPRAWOZDANIE Z PROJEKTU I

Języki i metodyka programowania 2 - rok akad. 2011/2012

ver 2.5

Mateusz Cieciura
cieciurm@ee.pw.edu.pl

21 maja 2012

1 Specyfikacja funkcjonalna

1.1 Przeznaczenie

Program służy do generowania i scalania siatek trójkątnych w jedną w obrębie podanej otoczki. Powstała siatka jest zgodną topologicznie siatką trójkątną rozmiaru otoczki, zawierającą w sobie siatki wejściowe. W procesie generowania siatek trójkątnych wykorzystywany jest program *Triangle - A Two-Dimensional Quality Mesh Generator and Delaunay Triangulator* (http://www.cs.cmu.edu/ quake/triangle.html).

1.2 Wywołanie

Program wywołuje się w linii komend jako:

```
./mergetr [-p Otoczka][-o Wynik][-q Wartość][-a Wartość] Siatka1 ... SiatkaN
```

Opis opcji:

`-p` Nazwa pliku z otoczką do wczytania - podana jako argument opcji (domyślnie otoczka.poly)

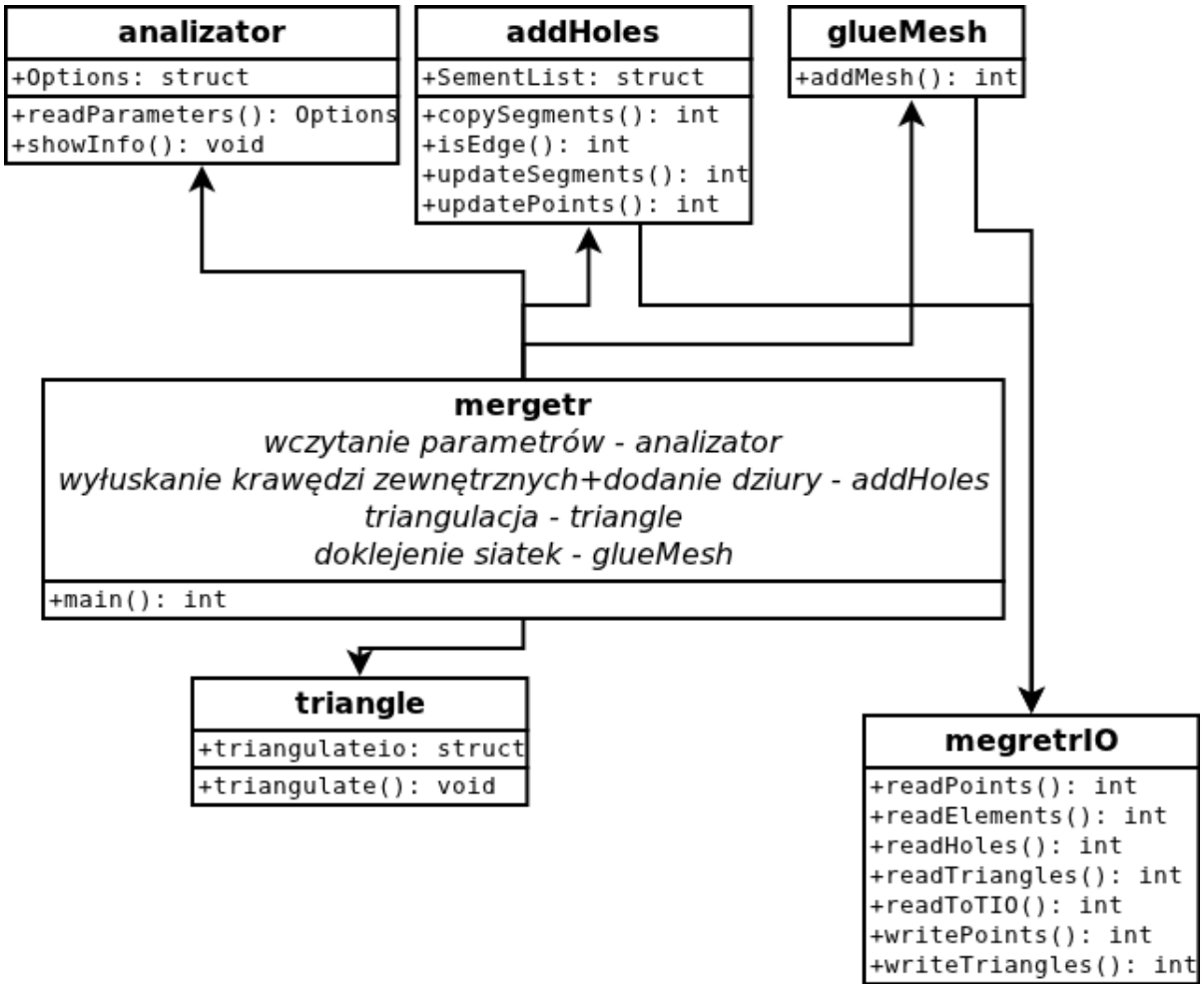
`-o` Nazwa plików wyjściowego - podana jako argument opcji (domyślnie result, powstaną pliki result.node i result.ele)

`-a` Ograniczenie na maksymalne pole trójkąta w siatce - podane jako argument opcji

`-q` Ograniczenie na minimalny kąt trójkąta - podane jako argument opcji

Opis argumentów:

`Siatka1 Siatka2 ... SiatkaN` Nazwy plików zawierających wejściowe siatki (tylko rdzeń nazwy, bez rozszerzeń)



Rysunek 1: Diagram klas programu mergetr

1.3 Dane wejściowe

Na wejściu program dostaje plik *.poly z danymi otoczki oraz tyle par plików *.node oraz *.ele ile zostało podanych siatek składowych. Formaty *.poly, *.node i *.ele są opisane w dokumentacji programu *Triangle*.

1.4 Format wyników

Na wyjściu program tworzy parę plików *.node oraz *.ele, które opisują wynikową siatkę trójkątną o rozmiarze otoczki, która zawiera siatki wejściowe.

2 Specyfikacja implementacyjna

2.1 Struktura programu

Program będzie się składał z następujących modułów:

1. Moduł przetwarzający wywołanie programu z linii komend (parser)
2. Moduł obsługujący wejście/wyjście z plików do struktur programu (i odwrotnie) *Triangle* (mergetrIO)

3. Moduł zamieniający siatki na dziury w otoczce (addHoles)
4. Moduł triangulujący nową figurę (*Triangle*)
5. Moduł doklejający siatki początkowe do wygenerowanej (glueMesh)

2.2 Funkcjonalność poszczególnych modułów

1. Moduł przetwarzający wywołanie programu z linii komend (parser)
   - Za pomocą funkcji *getopt* przetwarza tablicę char** argv
   - Wypełnia strukturę danych niezbędnymi danymi (nazwy plików, parametry *Triangle*)
2. Moduł wejścia/wyjścia plikow i struktur danych (mergetrIO)
   - Czyta dane o wierzchołkach z pliku (czyta plik *.poly i *.node)
   - Czyta dane o trójkątach z pliku (czyta plik *.ele)
   - Czyta dane o dziurach z pliku (czyta plik *.poly)
   - Czyta dane o odcinkach z pliku (czyta plik *.poly)
   - Pisze dane o punktach do pliku (pisze plik *.node)
   - Pisze dane o trójkątach do pliku (pisze plik *.ele)
3. Moduł zamieniający siatki na dziury w otoczce (addHoles)
   - Tworzy listę wszystkich krawędzi (zapamiętuje oba wierzchołki i informację, czy jest krawędzią zewnętrzną)
   - Tworzy listę punktów siatki (zapamiętuje numer w siatce, nowy numer w otoczce i informację, czy jest punktem zewnętrznym)
   - Zaznacza, które krawędzie i punkty są zewnętrze, a które nie
   - Kopiuje zewnętrzne punkty do otoczki
   - Kopiuje zewnętrzne odcinki do otoczki
   - Dodaje dziure w otoczce w miejscu siatki (środek ciężkości pierwszego trójkąta siatki)
4. Moduł triangulujący nową figurę (triangulation)
   - Inicjuje struktury *triangulateio*
   - Trianguluje zmodyfikowany wielokąt (po dodaniu dziur) za pomocą funkcji *triangulate*
5. Moduł doklejający siatki początkowe do wygenerowanej (glueMesh)
   - Kopiuje do striangulowanej otoczki wewnętrzne punkty siatki
   - Uaktualnia numery punktów w otoczce (ponieważ po triangulacji zmienia się porządek punktów)
   - Kopiuje trójkąty z siatki do otoczki według nowej numeracji
   - Usuwa dziurę stworzoną w miejscu siatki

2.3 Szczegółowy opis modułów - struktury danych i prototypy funkcji

Każdy moduł składa się z pliku nagłówkowego, w którym znajdują się deklaracje struktur danych oraz funkcji wykorzystywanych w tym module oraz pliku z rozszerzeniem *.c zawierającym definicje funkcji. Całość jest połączona przez moduł sterujący - mergetr.c. Program *Triangle* jest w programie użyty jako moduł do triangulacji wielokąta.

Oprócz modułów na program składa się plik *makefile*, który umożliwia zbudowanie programu za pomocą narzędzia *GNU Make*.

1. Przetwarzanie wywołania z linii komend (parser)

   Do przechowywania parametrów wywołania służy struktura:

```
typedef struct
{
        char *input;              /* input file nameo */
        char *output;             /* output file name */
        char tr_opt[16];          /* Triangle options */
        int  args_start;          /* index of argv where
           arguments start */
} Options;
```

   - Funkcja odczytująca opcje i argumenty z *char **argv* za pomocą funkcji z biblioteki unistd *getopt*, zwraca poprawnie wypełnioną strukturę. W przypadku, gdy nie zostały podane zostaną ustawione wartości domyślne.

```
Options readParameters (int argc, char **argv);
```

   - W przypadku nie podania argumentów dla programu, podania niewystarczającej ich liczby lub podania błędnego argumentu dla opcji wyświetla informację o argumentach i dostępnych opcjach.

```
void showInfo ();
```

2. Czytanie plików z danymi (megretrIO)

   Moduł korzysta ze struktury *triangulateio* (opisane w dokumentacji *Triangle'a*) - zawierających wszystkie informacje o siatce.

   - Funkcja wypełnia tablicę *pointlist* z podanego pliku.

```
int readPoints (FILE *in, struct triangulateio *x);
```

   - Funkcja wypełnia tablicę *pointargumentlist* z podanego pliku.

```
int readSegments (FILE *in, struct triangulateio *x);
```

   - Funkcja wypełnia tablicę *holelist* z podanego pliku.

```
int readHoles (FILE *in, struct triangulateio *x);
```

   - Funkcja wypełnia tablicę *trianglelist* z podanego pliku.

```
int readTriangles (FILE *in, struct triangulateio *x);
```

- Zapisuje punkty z tablicy *pointlist* do pliku

```
int writePoints (FILE *out, struct triangulateio x);
```

- Zapisuje punkty z tablicy *segmentlist* do pliku

```
int writeSegments (FILE *out, struct triangulateio x);
```

- Zapisuje trójkąty z tablicy *trianglelist* do pliku

```
int writeTriangles (FILE *out, struct triangulateio x);
```

- Zapisuje dziury z tablicy *holelist* do pliku

```
int writeHoles (FILE *out, struct triangulateio x);
```

3. Zamiana siatek wejściowych na dziury (addHoles)

   Moduł do wklejania siatek do otoczki wykorzystuje dwie własne struktury danych:

```
typedef struct {
        int v1, v2;
        int is_border;
} EdgeList;
```

   Lista krawędzi, numery węzłów krawędzi i parametr, czy jest zewnętrzna.

```
typedef struct {
        int no_in_mesh;
        int no_in_otoczka;
        int is_border;
} PointList;
```

   Lista punktów, numer punktu w siatce, numer w otoczce po wklejeniu i parametr, czy jest zewnętrzny.

   - Funkcja tworzy listę krawędzi, zwraca tablicę typu EdgeList

```
EdgeList *createEdgeList (struct triangulateio siatka);
```

   - Funkcja sprawdza, które krawędzie należą tylko do jednego trójkąta i wpisuje wartość "1" do pola is border

```
int markBndEdges (struct triangulateio siatka, EdgeList *
   t);
```

   - Funkcja tworzy listę punktów, zwraca tablicę typu PointList

```
PointList *makePointList (struct triangulateio otoczka,
   struct triangulateio siatka, EdgeList *t);
```

- Dwie funkcje, które modyfikują struktury otoczki i dopisują punkty oraz krawędzie oznaczone jako zewnętrzne. Ostatnia funkcja dodaje dziurę w miejscu środka ciężkości pierwszego trójkąta siatki

```
1 int updatePoints (struct triangulateio *otoczka, struct
     triangulateio siatka, EdgeList *t, PointList *p);
2
3 int updateSegments (struct triangulateio *otoczka, struct
      triangulateio siatka, EdgeList *t, PointList *p);
4
5 int updateHoles (struct triangulateio *otoczka, struct
     triangulateio siatka);
```

4. Triangulacja nowej siatki

- Funkcja (pochodząca z programu *Triangle*) trianguluje obszar opisany w strukturze. Dostaje otoczkę z dodanymi dziurami i zwraca strukturę triangulateio z siatką trójkątną

```
1 void triangulate(char *triswitches, struct triangulateio
     *in, struct triangulateio *out, struct triangulateio *
     vorout)
2 void triangulate(triswitches, in, out, vorout)
3 char *triswitches;
4 struct triangulateio *in;
5 struct triangulateio *out;
6 struct triangulateio *vorout;
```

5. Sklejenie siatek wejściowych z wynikową (glueMesh)

- Ponieważ podczas tworzenia listy krawędzi, krawędzie niezewnętrzne występują dwukrotnie funkcja oznacza odpowiednio tylko po jednej z każdej pary

```
1 int markNotBndEdges (struct triangulateio siatka,
     EdgeList *t);
```

- Ponieważ po triangulacji nie ma pewności, że punkty pozostały w takiej samej kolejności jak przed tym procesem funkcja sprawdza nowe numery punktów (odległość maksymalnie o epsilon - zdefiniowany w programie)

```
1 int fixPointListNumbers (struct triangulateio *out,
     struct triangulateio *otoczka, PointList *p);
```

- Dwie funkcje dodają do struktry punkty i trójkąty z siatki wejściowej. Na końcu zostaje usunięta dziura zrobiona w miejsce siatki

```
1 int glueNotBndPoints (struct triangulateio *out, struct
     triangulateio siatka, PointList *p);
2
3 int glueInteriorTriangles (struct triangulateio *out,
     struct triangulateio siatka, PointList *p);
4
5 int removeHole (struct triangulateio *out);
```

3 Testy

3.1 Przygotowanie testów

Kolejne moduły i dodane funkcjonalności były testowane na bierząco. Na potrzeby testowania ich stworzono zestaw plików testowych składający się z:

- 3 otoczek
- 5 siatek wejściowych

  Poniżej opisano 3 testy, która mogą być automatycznie wykonane przez wywołanie poleceń make - *make test1, make test2, make test3*. Do oglądania wyników można wykorzystać polecenie *make showme*.

3.2 Przeprowadzone testy

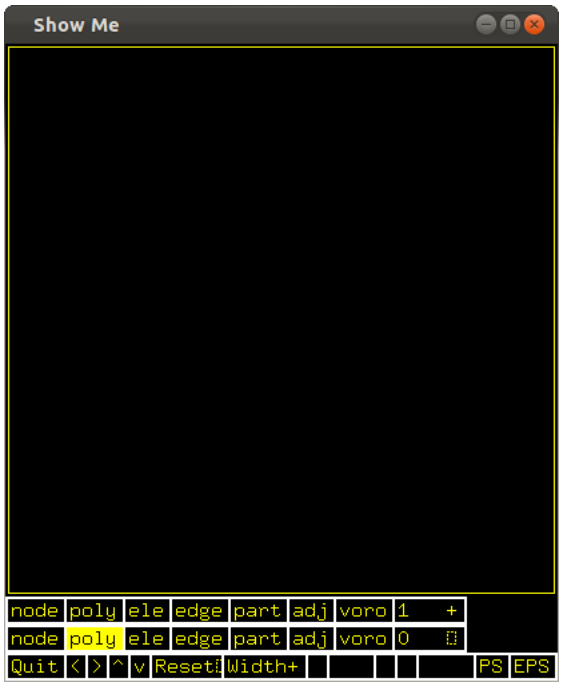
3.2.1 Test 1

Test przeprowadzono poprzez wywołanie programu o składni:

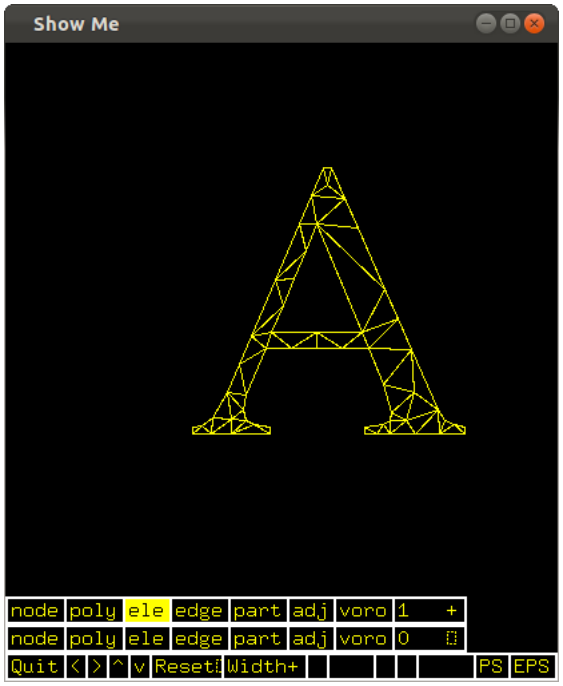
```
./mtr -p test/O1 -a0.01 test/A.1 test/F.1
```

Poniżej przedstawiono wygląd otoczki i siatek składających się na test:

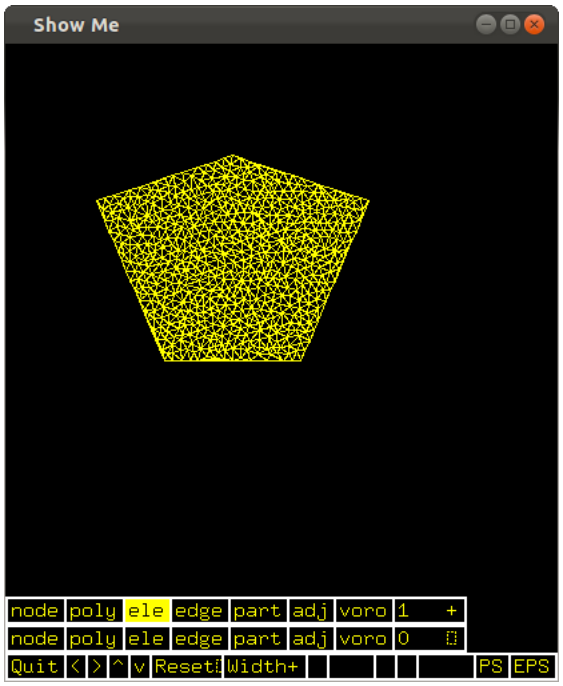
3.2.2 Test2



Rysunek 2: Otoczka - test/O1.poly



Rysunek 3: Siatka nr 1 - test/A.1



Rysunek 4: Siatka nr 2 - test/F.1