Algoritmy v digitální kartografii

Množinové operace s polygony

Zimní semestr 2018/2019

Tereza Kulovaná
Markéta Pecenová

Obsah

1 Zadání

Zadání úlohy bylo staženo ze stránek předmětu 155ADKG.

Vstup: množina n polygonů P=$\{P_1, ..., P_n\}$.

Výstup: množina m polygonů $P' = \{P'_1, ..., P'_m\}$.

S využitím algoritmu pro množinové operace s polygony implementujte pro libovolné dva polygony $P_i, P_j \in P$ následující operace:

- Průnik polygonů $P_i \cap P_j$,
- Sjednocení polygonů $P_i \cup P_j$,
- Rozdíl polygonů: $P_i \cap \overline{P}_j$, resp. $P_j \cap \overline{P}_i$.

Jako vstupní data použijte existující kartografická data (např. konvertované shape fily) či syntetická data, která budou načítána z textového souboru ve Vámi zvoleném formátu.

Grafické rozhraní realizujte s využitím frameworku QT.

Při zpracování se snažte postihnout nejčastější singulární případy: společný vrchol, společná část segmentu, společný celý segment či více společných segmentů. Ošetřete situace, kdy výsledkem není 2D entita, ale 0D či 1D entita.

Pro výše uvedené účely je nutné mít řádně odladěny algoritmy z úlohy 1. Postup ošetření těchto případů diskutujte v technické zprávě, zamyslete se nad dalšími singularitami, které mohou nastat.

Hodnocení:

Krok	**Hodnocení**
Množinové operace: průnik, sjednocení, rozdíl	20b
Konstrukce offsetu (bufferu)	*+10b*
Výpočet průsečíků segmentů algoritmem Bentley & Ottman	*+8b*
Řešení pro polygony obsahující holes (otvory)	*+6b*
Max celkem:	**44b**

Čas zpracování: 2 týdny

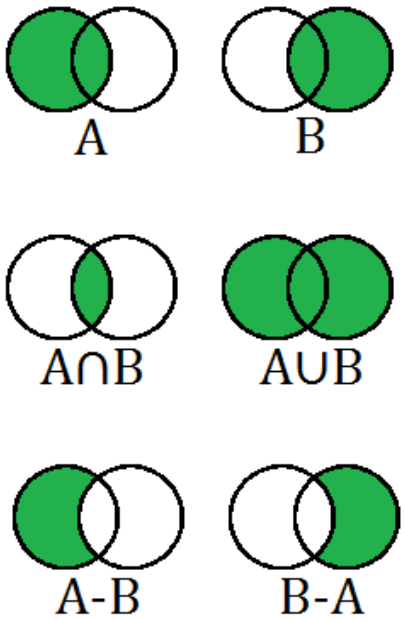
V rámci této úlohy nebyly implementovány žádné bonusové úlohy.

2 Popis a rozbor problému

Úloha **Množinové operace s polygony** se zabývá vytvořením aplikace, která nad libovolnými dvěma vstupními polygony provede zvolenou množinovou operaci.

Mějme vstupní polygony A a B. Množinové operace, které nad nimi lze vykonat, jsou následující:

1. Průnik (Intersection): $\longrightarrow A \cap B$
2. Sjednocení (Union) $\longrightarrow A \cup B$
3. Rozdíl (Difference) $\longrightarrow A \backslash B$ nebo $B \backslash A$



Obrázek 1: Ukázka množinových operací [Zdroj]

3 Algoritmy

Tato kapitola se zabývá popisem algoritmů, které byly v aplikaci implementovány. Vzhledem k jejich složitosti je popis výsledného algoritmu rozdělen do jednotlivých fází. Výsledný algoritmus, který spojuje všechny kroky dohromady, se nazývá *BooleanOper*. Vstupní polygony A a B jsou reprezentovány kruhovými seznamy s orientací proti směru hodinových ručiček (CCW). Body, které oba polygony tvoří, mají nově vytvořený datový typ `QPointFB`.

3.1 Výpočet průsečíků

3.1.1 computePolygonIntersections

dopsat kecy okolo
Průsečíky jsou ukládány do proměnné M datového typu `map`, která funguje na principu hashování. Spolu s výsledkem je ukládán i tzv. klíč, který na něj odkazuje. Za klíč byl zvolen parametr α.

Po nalezení každého nového průsečíku je nutné aktualizovat stávající seznamy bodů obou polygonů. K tomu slouží lokální procedura *ProcessIntersection*. Parametr t reprezentuje směrnice α nebo β, parametr i index daného bodu. DO DOKUMENTACE

Zjednodušený zápis algoritmů lze zapsat způsobem uvedeným níže:

1. Postupně pro všechny $p_i \in A$:
   Vytvoř mapu M
   Postupně pro všechny $p_j \in B$:
   Podmínka ($\exists$ průsečík b_{ij}):
   Přidej: M$[\alpha_i] \leftarrow b_{ij}$
   Zpracuj průsečík pro B: *ProcessIntersection*(b_{ij}, β, B, j)
   Podmínka ($M \neq \emptyset$):
   Postupně pro všechny $b_{ij} \in M$:
   Nalezni průsečík pro A: *ProcessIntersection*(b_{ij}, α, A, i)

Lokální procedura *ProcessIntersection*(b, t, P, i):

1. Podmínka ($|t| < \epsilon$):
   Průsečíkem je počáteční bod: $P[i] \leftarrow inters$
2. Podmínka ($(|t-1| < \epsilon)$:
   Průsečíkem je koncový bod: $P[(i+1)\%m] \leftarrow inters$
3. Jinak:
   $i = i + 1$
   $P \leftarrow (b, i)$

3.1.2 setPositions

Všechny vrcholy polygonu A, resp. B (včetně nalezených průsečíků) jsou následně ohodnoceny, zda leží uvnitř, vně nebo na hraně polygonu B, resp. A. Pro všechny hrany e_i jsou vypočteny jejich středy $\bar{p}$, pro které se následně určuje jejich pozice (ohodnocení g) vůči druhému polygonu. Tato informace je uložena do počátečního bodu hrany e_i do parametru

pozice. K určení pozice bodů vůči polygonu byl použit algoritmus *GetPositionWinding* z úlohy č. 1. Nejprve jsou zpracovány všechny hrany prvního polygonu a stejný postup je analogicky aplikován i pro druhý polygon.

Zjednodušený zápis algoritmů lze zapsat způsobem uvedeným níže:

Postupně pro všechny $p_i \in A$:
$\bar{p} = \frac{p_i(x,y)+p_{i+1}(x,y)}{2}$
pozice = *GetPositionWinding*($\bar{p}, B$)
$p_i[pozice]$ = pozice

3.2 Fragmenty

Sousedící vrcholy se stejným ohodnocením jsou uloženy do samostatných fragmentů f. Seznam bodů každého fragmentu začíná průsečíkem a končí bodem s jiným ohodnocením. Pro odlišení je orientace vrcholů ve fragmentech po směru hodinových ručiček (CW).

3.2.1 createFragments

Do metody vstupuje polygon P o velikosti n, ohodnocení vrcholů g a seznam fragmentů F. Algoritmus obsahuje lokální proceduru *createFragmentFromVertices*, která ze sousedících bodů o stejném ohodnocení vytváří fragment f. i_s je index počátečního bodu fragmentu, i je index přidávaného bodu a g ohodnocení.

Zjednodušený zápis algoritmů lze zapsat způsobem uvedeným níže:

1. Inicializace: $i = n - 1;\ i_s = -1$
2. Dokud $(i > 0)$
   Podmínka ($P[i]$ = průsečík $\wedge\ g(P[i]) = g$)
   $i_s = i;\ i--$
3. Podmínka $(i_s < 0) \rightarrow$ žádný bod neexistuje, ukonči proces
4. Inicializace: $i = i_s$
5. Proveď:
   Podmínka ($P[i]$ = průsečík $\wedge\ g(P[i]) = g$)
   Vytvoř fragment $f = \emptyset$
   Podmínka (*createFragmentFromVertices* vytvořen)
   Je-li potřeba, prohoď orientaci
   Přidej f do seznamu fragmentů F: $F[f[0]] \leftarrow f$
   Jinak inkrementace: $i = (i + 1)\%n$

   Dokud $(i \neq i_s)$

Lokální procedura *createFragmentFromVertices*(i_s, P, g, i, f):

Opakuj:

Přidej bod do fragmentu: $f \leftarrow P[i]$

Inkrementace: $i = (i + 1)\%n$

Podmínka $(i = i_s) \rightarrow$ return FALSE

Podmínka $(g(P[i]) \neq g)$

Přidej bod do fragmentu: $f \leftarrow P[i]$

Return TRUE

3.2.2 mergeFragments

Algoritmus spojuje jednotlivé fragmenty f do výstupních polygonů. Metoda má na vstupu seznam fragmentů F a seznam polygonů C, do kterého jsou ukládány výsledné polygony. Metoda nejprve spojí jednotlivé fragmenty do oblastí a ty jsou následně lokální procedurou *createPolygonFromFragments* převedeny na polygony. Počáteční bod fragmentu je značen jako s, n značí následující bod.

Postupně pro všechna $f \in F$:

Vytvoř: $P = \emptyset$

Najdi startovní bod fragmentu: $s = f.first$

Podmínka (f nebyl ještě zpracován)

Podmínka (*createPolygonFromFragments*)

Přidej: $C \leftarrow P$

Lokální procedura *createPolygonFromFragments*(s, F, P):

Inicializace: $n = s$

Opakuj:

Nalezni další fragment: $f = F.find(n)$

Podmínka (fragment s daným počátečním bodem $\nexists$) $\rightarrow$ return FALSE

Označ fragment za zpracovaný: $f.second.first \leftarrow$ TRUE

Nalezni další bod: $n \leftarrow f.second.second.back()$

Přidej ho bez počátečního bodu: $P \leftarrow f.second.second - f.second.second[0]$

Podmínka $(n = s) \rightarrow$ return TRUE

3.3 BooleanOper

Tento algoritmus spojuje všechny výše uvedené kroky. Na vstupu jsou polygony A a B a typ zvolené množinové operace.

1. Podmínka (orientace $A \vee B \neq$ CCW) $\rightarrow$ prohoď orientaci
2. *computePolygonIntersections*(A, B)
3. *setPositions*(A, B)
4. Vytvoř mapu fragmentů F
5. Zvolení ohodnocení: $pos1 = (oper \equiv intersection \vee oper \equiv DiffAB?Inner : Outer)$
   $pos2 = (oper \equiv intersection \vee oper \equiv DiffBA?Inner : Outer)$
6. Prohození: $swap1 = (oper \equiv DiffAB? : true : false)$
   $swap2 = (oper \equiv DiffBA? : true : false)$
7. *CreateFragments (A, pos1, swap1, F)*

   CreateFragments (B, pos2, swap2, F)
8. *MergeFragments (A,B,C)*

4 Vstupní data

Pro účely této úlohy byla použita data, která byla naměřena v rámci geodetické výuky v terénu v Mariánské u Jáchymova. Souřadnice X a Y byly pro tuto úlohu zredukovány na rozumnou velikost, souřadnice Z byla zachována. Body byly zaměřeny metodou GNSS a totální stanicí a znázorňují tamní louku a část silnice. Seznam vstupních bodů je uložen v textovém souboru testing_data.txt. Soubor je nutné do aplikace nahrát pomocí tlačítka *Load points*. Struktura textového souboru je následující:

Sloupec 1: souřadnice X [m]
Sloupec 2: souřadnice Y [m]
Sloupec 3: souřadnice Z [m]

Po úspěšném/neúspěšném nahrání souboru je uživatel upozorněn hláškou. Uživatel nemůže kliknout na žádné jiné tlačítko pro výpočty, nejsou-li nahrána data (tlačítka jsou zašedivělá). Aplikace dále nedovoluje spustit výpočty, jejichž fungování je závislé na vygenerované trojúhelníkové síti, nebyla-li předtím vytvořena. Uživatel má dále možnost zvolit krok, v jakém se budou vykreslovat vrstevnice. Hodnoty lze měnit šipkami nahoru/dolů po 5 m nebo ručně vepsat hodnotu celého čísla v rozmezí 1 m až 100 m. Delaunayova triangulace, vrstevnice, sklon a orientace se generují stisknutím příslušných tlačítek.

5 Výstupní data

Vstupní množina bodů a nad ní vygenerovaná trojúhelníková síť je zobrazena ve grafickém okně aplikace. Vykreslování vrstevnic, sklonu a orientace je odděleno. U vrstevnic je každá pátá (hlavní) zvýrazněna. Sklon je v odstínech šedi (čím vyšší sklon, tím tmavší barva). Pro zobrazení orientace trojúhelníků ke světovým stranám byla využita prostřední kružnice barvené škály ze stránek společnosti *ESRI*, viz níže. Aplikace je uvedena do výchozího stavu stisknutím tlačítka *Clear*.

6 Aplikace

V následují kapitole je představen vizuální vzhled vytvořené aplikace tak, jak ji vidí prostý uživatel.

7 Dokumentace

Tato kapitola obsahuje dokumentaci k jednotlivým třídám.

7.1 !Algorithms

Třída *Algorithms* obsahuje metody pro výpočet Delaunayovy triangulace a analýzu DTM.

getPositionWinding

Metoda **getPositionWinding** určuje polohu bodu q vzhledem k polygonu P za použití algoritmu *Winding Number*. Na vstupu je bod q a vektor bodů polygonu třídy `QPointFB`. Návratová hodnota typu `TPointPolygon` vrací polohu bodu q vůči polygonu P.

Input:

- `QPointFB` q
- *vector* <`QPointFB`> P

Output:

- `INSIDE` $\rightarrow$ q se nachází uvnitř polygonu P
- `OUTSIDE` $\rightarrow$ q se nachází vně polygonu P
- `ON` $\rightarrow$ q se nachází na hraně polygonu P

getPointLinePosition

Metoda **getPointLinePosition** určuje polohu bodu q vzhledem k přímce tvořené dvěma body. Na vstupu jsou 3 body typu `QPointFB`, návratová hodnota je nově definovaný typ `TPosition`.

Input:

- `QPointFB` q
- `QPointFB` a
- `QPointFB` b

Output:

- `LEFT` $\rightarrow$ bod se nachází vlevo od přímky
- `RIGHT` $\rightarrow$ bod se nachází vpravo od přímky
- `ON` $\rightarrow$ bod se nachází na přímce

get2LinesAngle

Metoda **get2LinesAngle** počítá úhel mezi dvěma přímkami. Na vstupu jsou 4 body typu `QPointFB`, návratová hodnota typu `double` vrací velikost úhlu v radiánech. Body p_1 a p_2 definují první přímku, zbylé dva body druhou přímku.

Input:

- `QPointFB` p_1
- `QPointFB` p_2
- `QPointFB` p_3
- `QPointFB` p_4

Output:

- `double`

get2LinesPosition

Metoda **get2LinesPosition** určuje vzájemnou polohu dvou přímek. Pokud se přímky protínají, metoda vypočte jejich průsečík p_{inters}. Na vstupu jsou 4 body typu `QPointFB`, návratová hodnota je nově definovaný typ `T2LinesPosition`. Body p_1 a p_2 definují první přímku, zbylé dva body druhou přímku.

Input:

- `QPointFB` p_1
- `QPointFB` p_2
- `QPointFB` p_3
- `QPointFB` p_4
- `QPointFB` p_{inters}

Output:

- `PARALLEL` → přímky jsou rovnoběžné
- `COLINEAR` → přímky jsou kolineární
- `INTERSECTING` → přímky se protínají v průsečíku p_{inters}
- `NONINTERSECTING` → přímky se neprotínají

computePolygonIntersections

Metoda **computePolygonIntersections** počítá průsečíky dvou polygonů A a B. Na vstupu jsou dva vektory bodů polygonů, návratová hodnota je typu `void`.

Input:

- *vector* <`QPointFB`> $polA$
- *vector* <`QPointFB`> $polB$

processIntersection

Metoda **processIntersection** slouží k aktualizování seznamu bodů (tzv. map) obou polygonů po přidání nově nalezeného průsečíku. Na vstupu je .... Návratová hodnota je typu `void`.

Input:

- `QPointFB` b
- `double` t
- *vector* <`QPointFB`> pol
- `int` i

setPositions

Metoda **setPositions** počítá orientaci trojúhelníku, který je tvořen třemi body, ke světovým stranám. Návratová hodnota typu `double` vrací orientaci trojúhelníku ve stupních. Orientace je pravotočivá a nabývá hodnot <-180°;180°>.

Input:

- `QPoint3D` p_1
- `QPoint3D` p_2
- `QPoint3D` p_3

createFragments

Metoda **createFragments** vytváří z vektoru hran trojúhelníky a počítá pro ně sklon a orientaci. Vypočtené hodnoty ukládá do datového typu `Triangle`. Návratová hodnota metody je vektor trojúhelníků typu `Triangle`.

Input:

- *vector* <`Edge`> dt

Output:

- *vector* <`Triangle`>

createFragmentFromVertices

Metoda **createFragmentFromVertices** počítá poloměr kružnice, která je tvořena 3 body. Na vstupu jsou 4 body typu `QPoint3D`, návratová hodnota typu `double` vrací velikost poloměru kružnice.

Input:

- `QPoint3D` p_1
- `QPoint3D` p_2
- `QPoint3D` p_3
- `QPoint3D` $c \rightarrow$ střed kružnice

mergeFragments

Metoda **mergeFragments** počítá vzdálenost mezi dvěma body. Na vstupu jsou 2 body typu `QPoint3D`, návratová hodnota typu `double` vrací vzdálenost mezi dvěma body.

Input:

- `QPoint3D` p_1
- `QPoint3D` p_2

createPolygonFromFragments

Metoda **createPolygonFromFragments** slouží k nalezení nejbližšího bodu z množiny bodů vzhledem k danému bodu p. Na vstupu je daný bod p a vektor bodů typu `QPoint3D`. Návratová hodnota typu `int` vrací index nejbližšího bodu.

Input:

- `QPoint3D` p
- *vector* <`QPoint3D`> *points*

getPolygonOrientation

Metoda **getPolygonOrientation** slouží k nalezení třetího bodu trojúhelníku, který splňuje Delaunayovo kritérium nejmenší opsané kružnice. Na vstupu jsou dva body typu `QPoint3D`, které představují orientovanou hranu, a vektor bodů typu `QPoint3D`. Návratová hodnota typu `int` vrací index hledaného bodu.

Input:

- `QPoint3D` $s \rightarrow$ počáteční bod hrany
- `QPoint3D` $e \rightarrow$ koncový bod hrany
- *vector* <`QPoint3D`> *points*

BooleanOper

Metoda **BooleanOper** počítá průsečík hrany trojúhelníku tvořené dvěma body typu `QPoint3D` s rovinou o dané výšce Z. Návratová hodnota je typu `QPoint3D`.

Input:

- `QPoint3D` p_1
- `QPoint3D` p_2
- `double` z

resetIntersections

Metoda **resetIntersections** vytváří z vektoru hran trojúhelníky a počítá pro ně sklon a orientaci. Vypočtené hodnoty ukládá do datového typu `Triangle`. Návratová hodnota metody je vektor trojúhelníků typu `Triangle`.

Input:

- *vector* <`Edge`> *dt*

Output:

- *vector* <`Triangle`>

lineOffset

Metoda **lineOffset** vytváří z vektoru hran trojúhelníky a počítá pro ně sklon a orientaci. Vypočtené hodnoty ukládá do datového typu `Triangle`. Návratová hodnota metody je vektor trojúhelníků typu `Triangle`.

Input:

- *vector* <`Edge`> *dt*

Output:

- *vector* <`Triangle`>

lineOffset

Metoda **lineOffset** vytváří z vektoru hran trojúhelníky a počítá pro ně sklon a orientaci. Vypočtené hodnoty ukládá do datového typu `Triangle`. Návratová hodnota metody je vektor trojúhelníků typu `Triangle`.

Input:

- *vector* <`Edge`> *dt*

Output:

- *vector* <`Triangle`>

sampleArc

Metoda **sampleArc** vytváří z vektoru hran trojúhelníky a počítá pro ně sklon a orientaci. Vypočtené hodnoty ukládá do datového typu `Triangle`. Návratová hodnota metody je vektor trojúhelníků typu `Triangle`.

Input:

- *vector* <`Edge`> *dt*

Output:

- *vector* <`Triangle`>

polygonOffset

Metoda **polygonOffset** vytváří z vektoru hran trojúhelníky a počítá pro ně sklon a orientaci. Vypočtené hodnoty ukládá do datového typu `Triangle`. Návratová hodnota metody je vektor trojúhelníků typu `Triangle`.

Input:

- *vector* <`Edge`> *dt*

Output:

- *vector* <`Triangle`>

7.2 Draw

Třída *Draw* obsahuje metody, které nahrávají a vykreslují vstupní množinu bodů. Dále zajišťuje vykreslení a smazání všech operací, kterou jsou nad množinou prováděny.

paintEvent

Metoda **paintEvent** vykresluje vstupní množinu bodů, Delaunayovu triangulaci, vrstevnice a sklon a orientaci trojúhelníků.

clearDT

Metoda **clearDT** slouží k vymazání všech vykreslených dat.

setAB

Metoda **setAB** slouží k získání vektoru bodů z kreslící plochy. Metoda vrací vektor bodů typu `QPoint3D`.

setRes

Metoda **setRes** slouží k získání vektoru hran z kreslící plochy. Metoda vrací vektor hran typu `Edge`.

setA

Metoda **setA** slouží k převedení Delaunayovy triangulace do kreslícího okna.

getA

Metoda **getA** slouží k převedení digitálního modelu terénu do kreslícího okna.

setB

Metoda **setB** slouží k načtení vstupních dat do aplikace. Součástí metody je i kontrola, zda se soubor úspěšně nahrál. Návratová hodnota je typu *QString* vrací hlášku, zda byly polygony úspěšně nahrány či nikoli.

getB

Metoda **getB** slouží k vykreslení sklonu trojúhelníků.

setBuffer

Metoda **setBuffer** slouží k vykreslení orientace trojúhelníků.

7.3 !QPointFB

Třída *QPointFB* slouží k definování nového datového typu `QPointFB`, který je odvozen od typu `QPointF` a který navíc obsahuje směrnice přímek *alfa* a *beta*, informaci, zda bod je průsečíkem, a poloha bodu vůči druhému polygonu. Defaultně je nastaveno, že bod není průsečíkem a hodnoty obou směrnic jsou rovny nule. A co poloha????

getAlfa

Metoda **getAlfa** slouží k získání směrnice *alfa*.

setAlfa

Metoda **setAlfa** slouží k nastavení směrnice *alfa*.

getBeta

Metoda **getBeta** slouží k získání směrnice *beta*.

setBeta

Metoda **setBeta** slouží k nastavení směrnice *beta*.

getInters

Metoda **getInters** slouží k získání informace, zda bod je průsečík či nikoli.

setInters

Metoda **setInters** slouží k nastavení informace, zda bod je průsečík či nikoli.

getPosition

Metoda **getPosition** slouží k získání polohy bodu.

setPosition

Metoda **setPosition** slouží k nastavení polohy bodu.

7.4 Types

Třída *Types* slouží k definování nových datových typů výčtového typu.

TPointPolygon

Datový typ **TPointPolygon** definuje polohu bodu q vůči polygonu P.

- `INSIDE` $\rightarrow q \in P$
- `OUTSIDE` $\rightarrow q \notin P$
- `ON` $\rightarrow q$ leží na P

TBooleanOperation

Datový typ **TBooleanOperation** definuje množinovou operaci, která je nad polygony A a B prováděna.

- `INTERSECTION` $\rightarrow A \cap B$
- `UNION` $\rightarrow A \cup B$
- `DIFFAB` $\rightarrow A \backslash B$
- `DIFFBA` $\rightarrow B \backslash A$

T2LinesPosition

Datový typ **T2LinesPosition** definuje polohu dvou přímek a a b.

- `PARALLEL` $\rightarrow a \parallel b$
- `COLINEAR` $\rightarrow a = b$
- `INTERSECTING` $\rightarrow a \cap b \neq \emptyset$
- `NONINTERSECTING` $\rightarrow a \cap b = \emptyset$

TPointLinePosition

Datový typ **TPointLinePosition** definuje polohu bodu q a přímky a.

- `LEFT` → bod q leží vlevo od přímky a
- `RIGHT` → bod q leží vpravo od přímky a
- `COL` → bod q leží na přímce a

7.5 Widget

Metody třídy *Widget* slouží pro práci uživatele s aplikací. Metody na vstupu nemají žádné parametry a návratové hodnoty jsou typu `void`.

on_delaunay_button_clicked

Metoda **on_delaunay_button_clicked** nad vstupní množinou bodů zobrazí Delaunayovu triangulaci.

on_clear_button_clicked

Metoda **on_clear_button_clicked** vrací aplikaci do výchozí polohy smazáním všeho, co bylo vykresleno.

on_contours_button_clicked

Metoda **on_contours_button_clicked** nad vygenerovanou trojúhelníkovou sítí z Delaunayovy triangulace vykreslí vrstevnice.

on_slope_button_clicked

Metoda **on_slope_button_clicked** obarví trojúhelníky vygenerované Delaunayovou triangulací do odstínů šedi podle hodnoty sklonu daného trojúhelníku.

on_aspect_button_clicked

Metoda **on_aspect_button_clicked** obarví trojúhelníky vygenerované Delaunayovou triangulací na základě jejich orientace ke světové straně.

on_load_button_clicked

Metoda **on_load_button_clicked** načítá data z textového souboru. Uživatel sám vyhledává cestu k požadovanému souboru.

8 Závěr

V rámci úlohy *Digitální model terénu a jeho analýzy* byla vytvořena aplikace, která ze vstupní množiny bodů vytváří digitální model terénu. Implementace některých algoritmů byla náročná, avšak výsledek je obstojný. Z kartografického hlediska by aplikace mohla být vylepšena. Jedná se zejména o přidání možnosti navolení povinných a lomových hran, které by zpřesnily výsledný DMT. Algoritmus generuje přijatelné výsledky pro terén, který neobsahuje příliš výrazné terénní hrany. Je také nevhodný pro vstupní data, která jsou rozmístěna pravidelně na mřížce, jelikož hledaná kružnice s nejmenším poloměrem je pro tyto body nejednoznačná. Dále by bylo vhodné zajistit aspoň zevrubní vyhlazení vrstevnic, jelikož působí kostrbatým dojmem.

Do budoucna by bylo vhodné přidat aspoň popis hlavních vrstevnic, případně barevnou hypsometrii. Autorky jsou s výslednou podobou aplikace spokojené.

9 Zdroje

1. *BAYER, Tomáš*. Množinové operace s polygony [online][cit. 4. 1. 2019]. Dostupné z: https://web.natur.cuni.cz

2. *Elementary Set Theory* [online] [cit. 4. 1. 2019]. Dostupné z: http://www.efgh.com/