ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA STAVEBNÍ, OBOR GEODÉZIE A KARTOGRAFIE
KATEDRA GEOMATIKY

název předmětu

Algoritmy digitální kartografie a GIS

název úlohy

Digitální model terénu

školní rok	studijní skup.	číslo zadání	zpracoval, email	datum	klasifikace
2021/22	60	-	Jakub Šimek, Jan Kučera	7.12.21	

Obsah

1. Zadání

Úloha č. 3: Digitální model terénu

Vstup: množina $P = \{p_1, ..., p_n\}$, $p_i = \{x_i, y_i, z_i\}$.

Výstup: polyedrický DMT nad množinou P představovaný vrstevnicemi doplněný vizualizací sklonu trojúhelníků a jejich expozicí.

Metodou inkrementální konstrukce vytvořte nad množinou P vstupních bodů 2D Delaunay triangulaci. Jako vstupní data použijte existující geodetická data (alespoň 300 bodů) popř. navrhněte algoritmus pro generování syntetických vstupních dat představujících významné terénní tvary (kupa, údolí, spočinek, hřbet, ...).

Vstupní množiny bodů včetně níže uvedených výstupů vhodně vizualizujte. Grafické rozhraní realizujte s využitím frameworku QT. Dynamické datové struktury implementujte s využitím STL.

Nad takto vzniklou triangulací vygenerujte polyedrický digitální model terénu. Dále proveďte tyto analýzy:

- S využitím lineární interpolace vygenerujte vrstevnice se *zadaným krokem* a v *zadaném intervalu*, proveďte jejich vizualizaci s rozlišením zvýrazněných vrstevnic.
- Analyzujte sklon digitálního modelu terénu, jednotlivé trojúhelníky vizualizujte v závislosti na jejich sklonu.
- Analyzujte expozici digitálního modelu terénu, jednotlivé trojúhelníky vizualizujte v závislosti na jejich expozici ke světové straně.

Zhodnoťte výsledný digitální model terénu z kartografického hlediska, zamyslete se nad slabinami algoritmu založeného na 2D Delaunay triangulaci. Ve kterých situacích (různé terénní tvary) nebude dávat vhodné výsledky? Tyto situace graficky znázorněte.

Zhodnocení činnosti algoritmu včetně ukázek proveďte alespoň na **3 strany** formátu A4.

Hodnocení:

Krok	Hodnocení
Delaunay triangulace, polyedrický model terénu.	10b
Konstrukce vrstevnic, analýza sklonu a expozice.	10b
Triangulace nekonvexní oblasti zadané polygonem.	*+5b*
Výběr barevných stupnic při vizualizaci sklonu a expozice.	*+3b*
Automatický popis vrstevnic.	*+3b*
Automatický popis vrstevnic respektující kartografické zásady (orientace, vhodné rozložení).	*+10b*
Algoritmus pro automatické generování terénních tvarů (kupa, údolí, spočinek, hřbet, ...).	*+10b*
3D vizualizace terénu s využitím promítání.	*+10b*
Barevná hypsometrie.	*+5b*
Max celkem:	**65b**

2. Bonusové úlohy

Ke standartnímu zadání byly zpracovány tyto bonusové úlohy:

1. Výběr barevných stupnic při vizualizaci sklonu a expozice.
2. Automatický popis vrstevnic.
3. Automatický popis vrstevnic respektující kartografické zásady.
4. Algoritmus pro automatické generování terénních tvarů.

3. Popis a rozbor problému

Na vstupu máme množinu bodů $P\{p_i\}_{i=0}^{n}$, kde každý bod p_i je trojice souřadnic $\{x_i, y_i, z_i\}$. Úkolem je najít Delaunayho triangulaci (DT) tvořenou trojúhelníky t_j. Tyto trojúhelníky poté slouží k dalším analýzám terénu.

3.1. Delaunayho triangulace

Výsledkem tohoto algoritmu jsou trojúhelníky, které se nejvíce přibližují rovnostranným trojúhelníkům.
Výsledná triangulace má čtyři vlastnosti, a to:

1) v kružnici opsané trojúhelníku t_j neleží žádný bod p_i,
2) DT maximalizuje minimální úhel $\forall t$, avšak neminimalizuje maximální úhel v t,
3) DT je lokálně i globálně optimální vůči kritériu minimálního úhlu,
4) DT je jednoznačná, pokud nenastane případ 4 bodů na jedné kružnici.

3.2. Lineární interpolace vrstevnic

Trojúhelník t_j je tvořen hranami e_1, e_2 a e_3. Pro vztah hrany trojúhelníku a vodorovné roviny s výškou z vznikají tři případy:

1) $(z - z_i)(z - z_{i+1}) < 0 \Rightarrow e_i$ protíná vodorovnou rovinu,
2) $(z - z_i)(z - z_{i+1}) > 0 \Rightarrow e_i$ neprotíná vodorovnou rovinu,
3) $(z - z_i)(z - z_{i+1}) = 0 \Rightarrow e_i$ leží ve vodorovné rovině.

Jestliže pro hranu platí bod 1), pak spočteme souřadnice průsečíků vodorovné roviny s hranou e_i, podle rovnic:

$x = \frac{(x_2 - x_1)(z - z_1)}{(z_2 - z_1)} + x_1,$

$y = \frac{(y_2 - y_1)(z - z_1)}{(z_2 - z_1)} + y_1.$

3.3. Expozice

Orientace trojúhelníku je dána azimutem průmětu gradientu $\nabla\rho$ do roviny x, y. Azimut je určen podle vzorce:

$A = \arctan\left(\frac{n}{m}\right),$

kde n, m jsou vektorové součiny vektorů $\vec{u}, \vec{v}$, tvořící rovinu trojúhelníku.

Hodnota azimutu byla převedena na interval $< 0, 2\pi >$. Hodnota azimutu byla následně transformována na interval barvy $< 0, 200 >$ vztahem:

$barva = 55 + \frac{200}{\pi}|A - \pi|.$

Tento vztah byl použit za účelem plynulého navázání azimutů blízkých nule a azimutů blízkých 360°. Použitím tohoto vztahu získáme osvícení terénu směrem od severu.

Hodnota $barva$ je pak využita k vizualizaci expozice. Vizualizaci je možné provézt v červené/modré/zelené/šedé.

3.4. Sklon

Sklon je úhel, který svírá svislice s normálou trojúhelníku. Sklon je určen vztahem:

$\varphi = \arccos\left(\frac{\vec{n} * \vec{n_t}}{|\vec{n}||\vec{n_t}|}\right),$

kde $\vec{n} = (0,0,1)$, $\vec{n_t} = \vec{u} \times \vec{v}$.

Ze všech trojúhelníku byly vybrány maximální a minimální hodnoty sklonu. Hodnota $barva$ pak byla určena vztahem:

$barva = (slope - minSlope)\frac{255}{maxSlope - minSlope}.$

Hodnota $barva$ je pak využita k vizualizaci sklonu. Vizualizaci je možné provézt v červené/modré/zelené/šedé. Barva trojúhelníku byla určena jako:

1. Červená – $RGB(barva, 0, 0)$
2. Modrá – $RGB(0, barva, 0)$
3. Zelená – $RGB(0,0, barva)$
4. Šedá – $RGB(barva, barva, barva)$

4. Popis algoritmů

4.1. Delaunayova triangulace

Pro tvorbu DT byla zvolena inkrementální konstrukce. To znamená, že vybraný Delaunayho bod musí ležet v levé polorovině orientované úsečky. Poloměr opsané kružnice trojúhelníku musí být minimální a střed této kružnice musí ležet v pravé polorovině. Jestliže takový bod neexistuje, pak je otočena orientace úsečky.

4.1.1 Implementace DT

1) $p_1 = rand(P), \|p_2 - p_1\|_2 = min$, náhodný a nejbližší bod
2) Vytvoř hranu $e = (e_1, e_2)$
3) $\underline{p} = \arg min_{\forall p_i \sigma_L(e)} r'(k_i), k_i = (a, b, p_i), e = (a, b)$
4) Pokud $\nexists \underline{p}$, prohoď orientaci $e \leftarrow (p_2, p_1)$, jdi na 3)
5) $e_2 = \left(p_2, \underline{p}\right), e_3 = (\underline{p}, p_1)$, zbývající hrany trojúhelníku
6) $AEL \leftarrow e, AEL \leftarrow e_2, AEL \leftarrow e_3$, přidání 3 hran do AEL
7) $DT \leftarrow e, DT \leftarrow e_2, DT \leftarrow e_3$, přidání 3 hran do DT
8) while AEL not empty:
9) $\quad AEL \rightarrow e, e = (p_1, p_2)$, vezmi první hranu z AEL
10) $\quad e = (p_2, p_1)$, prohoď její orientaci
11) $\quad \underline{p} = \arg min_{\forall p_i \sigma_L(e)} r'(k_i), k_i = (a, b, p_i), e = (a, b)$
12) $\quad if\ \exists \underline{p}$:
13) $\quad\quad e_2 = \left(p_2, \underline{p}\right), e_3 = (\underline{p}, p_1)$, zbývající hrany trojúhelníku
14) $\quad\quad DT \leftarrow e$, přidej hranu do DT ale ne do AEL
15) $\quad\quad add(e_2, AEL, DT), add(e_3, AEL, DT)$, přidej do DT i do AEL

4.1.2 Implementace add

1) Vytvoř hranu $e' = (b, a)$
2) $if\ (e' \in AEL)$
3) $\quad AEL \rightarrow e'$, odstraň z AEL
4) $else$
5) $\quad AEL \leftarrow e$, přidej do AEL
6) $DT \leftarrow (a, b)$, přidej do DT

4.2 Generování terénních tvarů

V aplikaci jsou implementovány tři druhy terénních tvarů, a to: kupa (Pile), hřbet (Ridge) a sedlo (Saddle).

4.2.1 Kupa

Pro výpočet kupy byl využit tvar rotačního paraboloidu, který je daný hodnotou:
$z = -(x_i - x_t)^2 - (y_i - y_t)^2$,
kde hodnoty s indexem t značí souřadnice těžiště všech bodů. Hodnotám souřadnic z byl přiřazen náhodný šum.

4.2.2 Hřbet

Tvar hřbetu byl vygenerován následující rovnicí:
$z = \sqrt{y_t^2 - (y_i - y_t)^2} + x_i$.
Hodnotě z byl následně přiřazen náhodný šum.

4.2.3 Sedlo

Tvar sedla byl vygenerován rovnicí:
$z = (x_i - x_t)^2 (y_i - y_t)^2$.
Hodnotě z byl následně přiřazen náhodný šum.

4.2.4 Implementace

1. V prvním kroku je vygenerováno n bodů s náhodnými souřadnicemi x, y v rozsahu $Canvasu$. Dále jsou spočteny souřadnice těžiště.
2. Cyklus for pro všech n bodů:
3. Souřadnice z: podle rovnic z předchozích kapitol Kupa/Hřbet/Sedlo.

4.3 Generování vrstevnic a jejich popisů

Poté, co byla vytvořena Delaunayova triangulace nad zadanými daty, je možné dál pokračovat v procesech jako je třeba generování vrstevnic. Toto téma se dá rozdělit na dvě části. Na generování vrstevnic, ať už hlavních či vedlejších, a na generování jejich popisů. Takto bude rozdělena i tato kapitola.

4.3.1 Generování vrstevnic

Předpokladem je již vytvořená trojúhelníková síť. Všechny vrstevnice jsou generovány funkcí *getContourLines*. Hlavní vrstevnice, které je potřeba oddělit, protože se symbolizují výraznější linií než vedlejší vrstevnice, jsou pak generovány funkcí *getMainContourLines*. Interpolace bodů pro vrstevnice je realizovaná funkcí *getContourPoint*.

4.3.1.1 Implementace *getContourLines*

Vstup: DT = {$e_1, e_2, \ldots$}, kde e_i = {s_i, e_i}, s_i = {x_{is}, y_{is}, z_{is}}, e_i = {x_{ie}, y_{ie}, z_{ie}}; z – výška vrstevnice

1) Pro $t_j \in$ DT, kde t_j = ($e_{j\cdot i}, e_{(j\cdot i)+1}, e_{(j\cdot i)+2}$)
2) $dz_{j1} = z_{j1} - z$, $dz_{j2} = z_{j2} - z$, $dz_{j3} = z_{j3} - z$, výškové rozdíly
3) $dz_{j12} = dz_{j1} \cdot dz_{j2}$, $dz_{j23} = dz_{j2} \cdot dz_{j3}$, $dz_{j31} = dz_{j3} \cdot dz_{j1}$, součiny výškových rozdílů
4) If ($dz_{i1} = 0$) && ($dz_{i2} = 0$) && ($dz_{i3} = 0$), komplanarita rovin
5) Continue
6) Else if ($dz_{j1} == 0$) && ($dz_{j2} == 0$)
7) *Contours* ← $e_{j\cdot i}$, přidej do vrstevnic
8) Else if ($dz_{j2} == 0$) && ($dz_{j3} == 0$)
9) *Contours* ← $e_{(j\cdot i)+1}$, přidej do vrstevnic
10) Else if ($dz_{j3} == 0$) && ($dz_{j1} == 0$)
11) *Contours* ← $e_{(j\cdot i)+2}$, přidej do vrstevnic
12) Else if (($dz_{j12} <= 0$) && ($dz_{j23} < 0$)) || (($dz_{j12} < 0$) && ($dz_{j23} <= 0$))
13) A = *getContourPoint*(p_{j1}, p_{j2}, *z*)
14) B = *getContourPoint*(p_{j2}, p_{j3}, *z*)
15) *Contours* ← e(A, B), přidej do vrstevnic
16) Else if (($dz_{j23} <= 0$) && ($dz_{j31} < 0$)) || (($dz_{j23} < 0$) && ($dz_{j31} <= 0$))
17) A = *getContourPoint*(p_{j2}, p_{j3}, *z*)
18) B = *getContourPoint*(p_{j3}, p_{j1}, *z*)
19) *Contours* ← e(A, B), přidej do vrstevnic
20) Else if (($dz_{j31} <= 0$) && ($dz_{j12} < 0$)) || (($dz_{j31} < 0$) && ($dz_{j12} <= 0$))
21) A = *getContourPoint*(p_{j3}, p_{j1}, *z*)
22) B = *getContourPoint*(p_{j1}, p_{j2}, *z*)
23) *Contours* ← e(A, B), přidej do vrstevnic

4.3.1.2 Implementace *getContourPoints*

Vstup: $p_1 = \{x_1, y_1\}$, $p_2 = \{x_2, y_2\}$, z

1) $x = \frac{(x_2 - x_1)(z - z_1)}{(z_2 - z_1)} + x_1$
2) $y = \frac{(y_2 - y_1)(z - z_1)}{(z_2 - z_1)} + y_1$

4.3.1.3 Implementace *getMainContourLines*

Vstup: *Contours* = $\{e_1, e_2, \ldots\}$, *k* – každá *k*-tá vrstevnice je hlavni, *dz* výškový interval vstevnic

1) *dh = dz · k*
2) *For e ϵ Contours*
3) *z = getZ(e)*, získej výšku hrany
4) *If (z%dh == 0), zbytek po dělení*
5) *ContoursMain* ← *pair<z, e>*, přidej hranu s výškou jako pár do struktury *map*

4.3.2 Generování popisů

Generování popisů, aby splňovali kartografické zásady je potřeba rozdělit na několik částí. První částí je, že struktura, ve které máme uložené hlavní vrstevnice, vlastně obsahuje multipolygony (neuzavřené samozřejmě). Tedy jinak řečeno se zde může nacházet více různých souvislých vrstevnic se stejnou výškou. Ty je potřeba oddělit, aby následující funkce *getDistancedEdges* mohla správně vzdálenostně odfiltrovat hrany vrstevnice, na kterých se popisek vypíše. Jinak by docházelo ke kolizím s ostatním souvislými vrstevnicemi stejné výšky a popisek by se vypsal jen zřídkakdy. U vlnitých terénů by to byl veliký problém. Toto rozdělení na souvislé vrstevnice se děje pomocí funkce *chainEdges*, která na principu postupného zřetězování hran, vrátí souvislé vrstevnice. Druhou částí prosté odfiltrování hran pro zadaný vzdálenostní práh, aby popisky nebyly všude (na každé hraně). Třetí částí uvážení vlivu sklonu a následné zorientování popisku směrem vzrůstající výšky. To už není provedeno funkcí, ale při vykreslování popisků. Byl tu totiž problém, že C++ ignoroval změnu orientace hrany a ukládal hranu v nějakém výchozím režimu, který tuto změnu při vykreslování ignoroval.

4.3.2.1 Implementace *getLabelContours*

1) *ContoursMain = getContoursMain(Contours),* vytvoř hlavní vrstevnice ze všech vrstevnic
2) *For c: ContoursMain*
3) *ContinuousContours = chainEdges(ContoursMain),* vrať souvislé vrstevnice
4) *DistancedEdges = getDistancedEdges(ContinuousContours),* vrať vzdálenostně odfiltrované hrany
5) *Labels* ← *DistancedEdges,* přidej hrany do seznamu hran s popisky

4.3.2.2 Implementace *chainEdges*

Vstup: *Edges* = {e1, e2, …}

1) *For e: Edges*
2) *If* EdgesChained is empty
3) EdgesChained ← {e} přidej nový vektor s hranou e
4) *Else*
5) *For vektor:* EdgesChained
6) *For e_ch: vektor*
7) *If (e ~ e_ch),* pokud se hrany dotýkají
8) *Vektor_ids* ← *id,* přidej id vektoru, kde existuje styk
9) *If (Vektor_ids is empty)*
10) EdgesChained ← {e} přidej nový vektor s hranou e
11) *Else if (Vektor_ids.size == 1)*
12) EdgesChained[*id*] ← {e} přidej hranu e

13) *Else*
14) *Id0 = Vektor_ids[0]*
15) EdgesChained[*id0*] ← {e} přidej hranu e
16) *Vektor_ids – id*, odeber první index
17) While *Vektor_ids* not empty
18) *Id = Vektor_ids[0]*
19) EdgesChained[*Id0*] ← EdgesChained[*Id*], přidej vektor hran, který kde je styk hran
20) EdgesChained[*Id*], odeber připojený vector hran
21) *Vektor_ids – id*, odeber index *id*

4.3.2.3 Implementace *getDistancedEdges*

1) *DistancedEdges* ← *random e*, přidej libovolnou hranu
2) *For e:Edges*
3) *If (dist(e,* ∀e ∈ *DistancedEdges) > t)*, pokud je hrana dostatečně daleko od všech ve vektoru distancovaných hran
4) *DistancedEdges* ← *e*, přidej hranu

4.3.2.4 Implementace *orientování popisku do kopce*

1) *For edge:Edges*, kde *edge* = {*s*, *e*}, *z* – výška této hrany (vrstevnice)
2) *i_left = getDelaunayPoint(s, e, points)*, získej nejbližší bod nalevo od hrany
3) *i_right = getDelaunayPoint(e, s, points)*, získej nejbližší bod nalevo od hrany
4) *If* (*i_left != -1 && i_right != -1*), pokud aspoň jeden nejbližší bod existuje
5) *z_left = points[i_left].z*, získej výšku bodu nalevo
6) *z_right = points[i_right].z*, získej výšku bodu napravo
7) *s* = směrník *e (*funkcí *atan2)*, směrník hrany tedy natočení popisku
8) *If (((z_left - z) > 0) || ((z_right - z) < 0)*, pokud hrana směřuje z kopce
9) *s+= 200gon*

5. Vstupní data

Vstupními daty pro tuto úlohu je množina bodů. Existují celkem 3 způsoby, jak tyto body vykreslit na obrazovku (Canvas):

1) Klikáním myši
2) Generováním terénních tvarů
3) Načtením ze souboru CSV

5.1 Klikání myši

Prvním způsobem, jakým dostat do aplikace data je klikáním do $Widgetu$. Po kliknutí do $Widgetu$ se uloží souřadnice *x, y*. Souřadnice *z* je vygenerována generátorem náhodných čísel v rozmezí $< 0{,}1000 >$.

5.2 Generování terénních tvarů

Druhou možností vstupních dat je generátor terénních tvarů (princip vysvětlen v předchozí kapitole). Ve widgetu $ComboBox$ lze zvolit jeden ze tří terénních tvarů a po kliknutí na tlačítko $Generate\ points$ se body objeví ve $Widgetu$. Vygenerované body už mají spočtenou souřadnici z odpovídajícího tvaru terénu.

5.3 Načtení ze souboru CSV

Poslední možností vstupních dat jsou data externí. Pro tento vstup se nachází v aplikaci tlačítko *Load data*. Po kliknutí na toto tlačítko se otevře *Windows* průzkumník souborů, ve kterém uživatel zvolí cestu k vstupnímu soboru.
Vstupní soubor musí být v textovém souboru ve formátu *CSV*. Požadavkem je, aby na prvním řádku byla hlavička souboru:
"*ID, X, Y, Z*".
V tomto pořadí jsou následně řazeny hodnoty bodů terénu. Oddělovačem jednotlivých hodnot na řádku musí být čárka.
Ukázková vstupní data jsou v repositáři ve složce *Data/DMT*. Složka obsahuje čtyři soubory: potok_klukovice, potok_radic, test a vyskopis_10. První dva soubory jsou z geodetického měření, třetí soubor na testování čtení souboru a posledním souborem jsou výškopisná data poskytována službou *INSPIRE*.

Proces načítání můžeme rozdělit na dvě hlavní části, a to načtení souřadnic z CSV souboru a následná transformace na Canvas.

5.3.1 Načtení ze souboru

Princip této části je vysvětlen v úvodu této podkapitoly, kde je stěžejním prvkem, aby data byla v pravoúhlém systému XYZ, a pořadí dat ve vstupním CSV, jinak následná transformace neproběhne správně a data se zobrazí zdeformovaně.

5.3.2 Transformace dat

Jelikož data mohou být v různých souřadnicových systémech, je nutné je transformovat, pokud jsou v pravoúhlém systému. Transformační klíč obsahuje pouze dva parametry. Je to posun a měřítko.

5.3.2.1 Měřítko

Měřítko je spočteno poměrem šířky (resp. dély) zobrazovacího okna (Canvasu) a šířky (resp. dély) datasetu (minmax boxu).

```
//Get size ratio for transformation to canvas
double x_ratio = canvas_width/dataset_width;
double y_ratio = canvas_height/dataset_height;
```

Aby nedošlo k deformaci poměru stran, je z uvedených poměrů *x* a *y* rozměrů vzat do transformace menší z nich.

```
//Get proper scale for whole dataset visibility without deformation
double scale;
if (x_ratio < y_ratio)
    scale = x_ratio;
else
    scale = y_ratio;
```

5.3.2.2 Posun

Posun je spočten posunem počátku datasetu vůči zobrazovacímu oknu. Počátky jsou v obou případech definovány levým minimálními souřadnicemi, tedy vzhledem k orientaci souřadných systémů se jedná o horní levé rohy.

```
//Get translation parameter for transformation
double x_trans = x_min - x_left_top;
double y_trans = y_min - y_left_top;
```

5.3.2.3 Výsledná transformace a vliv posunutého počátku Canvasu

Transformace je realizována funkcí *transformPoints* z třídy *Algorithms*. Zde je potřeba zmínit zvláštní pozici počátku Canvasu, který je lokalizován na souřadnicích [x,y] = [11,11]. Tento posun je počátku je v transformaci potřeba uvážit.

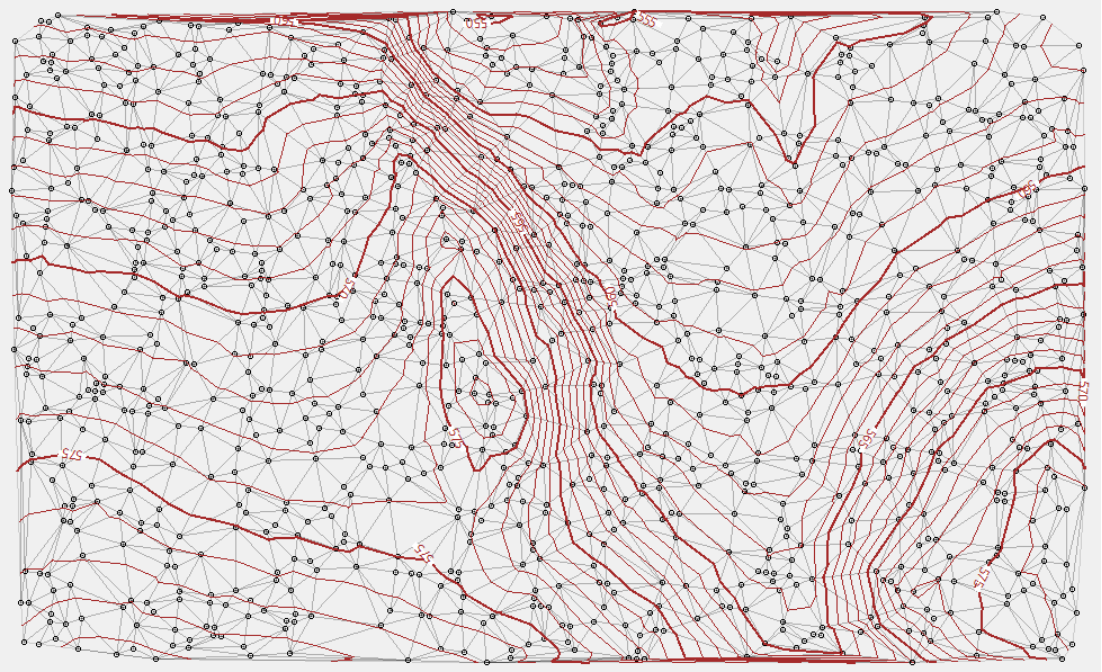
```
//Transform polygon coorinates by basic transformation based on minmax box of dataset
//x_min, x_max, y_min, y_max represent boundaries of dataset minmax box
std::vector<QPoint3D> points_transformed;

for (QPoint3D p : points_3d)
{
    //Translation with slight offset due to canvas origin set on [11,11] coors
    double dx = p.x()-trans_x-offset_x;
    double dy = p.y()-trans_y-offset_y;

    //Data scaling
    double ddx = dx*(scale/1.1);
    double ddy = dy*(scale/1.1);

    //Translate data back to visible part of Canvas
    double x0 = ddx + offset_x;
    double y0 = ddy + offset_y;

    points_transformed.push_back(QPoint3D(x0, y0, p.getZ()));
}

//Compute transformation key
return points_transformed;
```

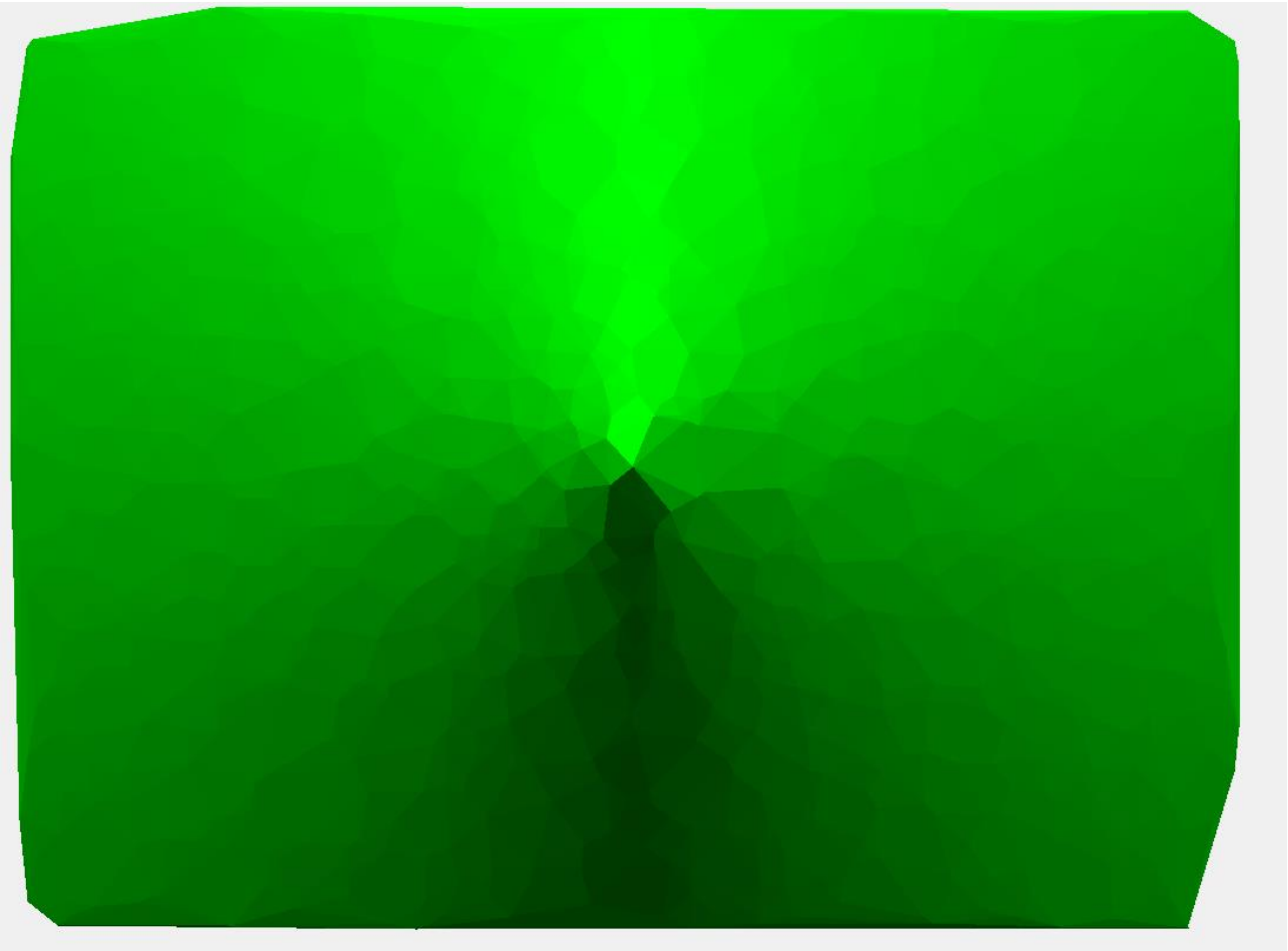
Jak je na obrázku vidět, data jsou nejdříve posunuta (parametr *trans*) s uvážením posunu počátku (parametr *offset*). Poté jsou délkově redukována měřítkem (parametr *scale*) a nakonec pro viditelnost dat znovu posunuta o ofset počátku.

6. Výstupní data

Aplikace slouží především k zobrazení výsledků analýzy terénu v $Canvasu$.
Mohla by nastat situace, že by si uživatel mohl chtít výsledek uložit. Pro tento případ je zde tlačítko $Save$. Po kliknutí se otevře průzkumník souborů, v němž uživatel ukáže, kam by chtěl uložit výstup. Poté je soubor uložen ve formátu PNG.

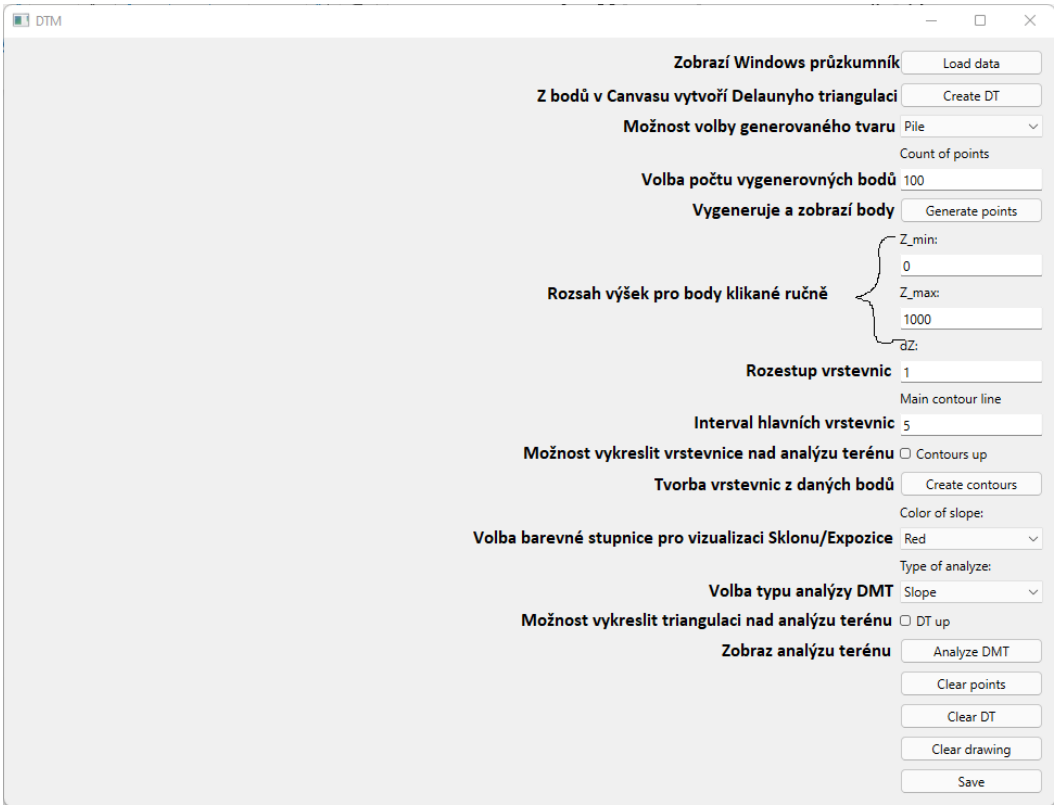
6.1 Ukázky výstupů





7. Vzhled aplikace

Po spuštění aplikace se objeví následující okno:

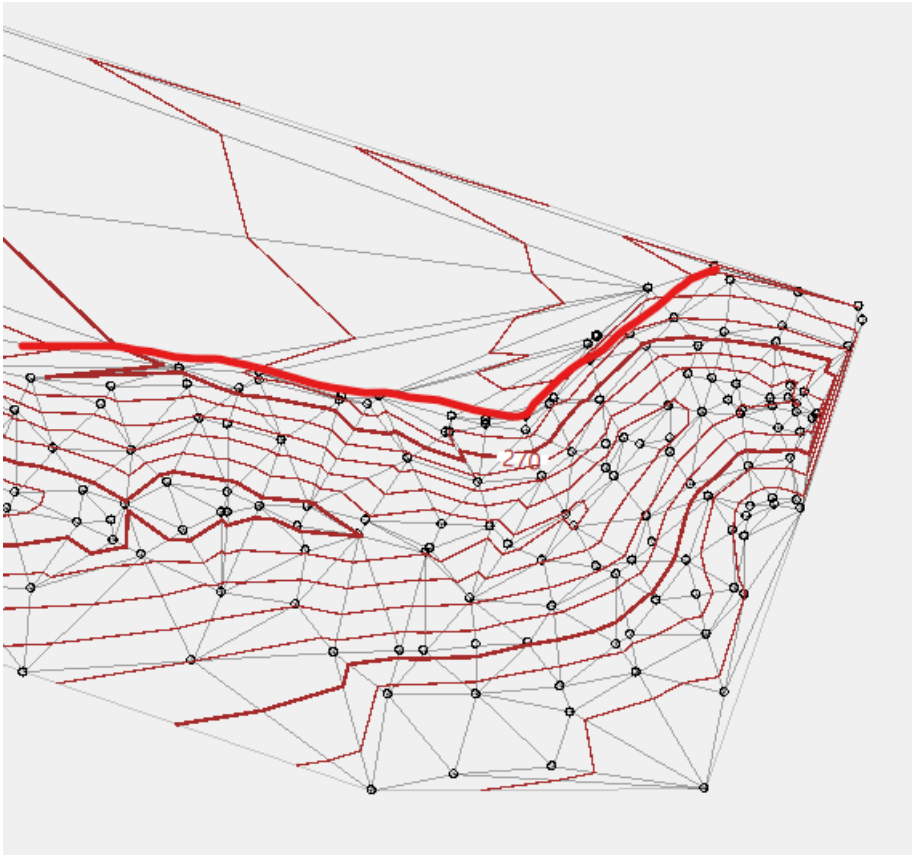


8. Zhodnocení

V této kapitole bude rozebráno, v jakých případech nemusí dávat aplikace dobré výsledky triangulace. Obecně Delaunyho triangulace dává dobré výsledky pro terény, které jsou hezky hladké a spojité. Problematické situace tedy mohou být například skalní převisy, koryta řek nebo například různá lidská díla, typově třeba liniové stavby a úpravy terénu kolem nich.

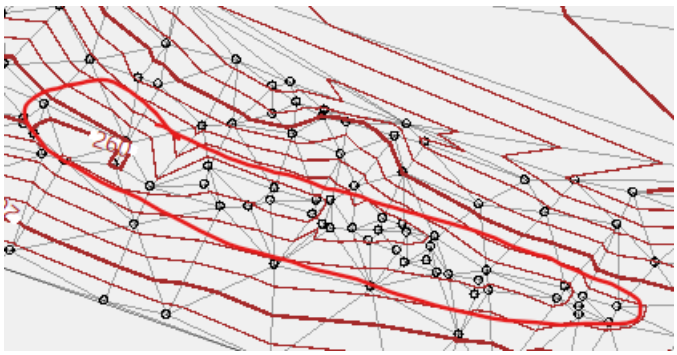
Nekonvexní oblasti

Prvním takovým případem jsou nekonvexní oblasti. V aplikaci není implementována možnost triangulovat nekonvexní oblasti. Na následujícím obrázku, získaného z geodetického měření, lze vidět následek tohoto problému. Zatímco v dolní části oblasti algoritmus dává celkem dobré výsledky, tak v horní reprezentuje skutečnost velmi špatně. Řešením by bylo implementovat funkci, která by odebrala trojúhelníky, které nepatří do zájmové oblasti.



Ostré hrany

Dalším problém nastává u ostrých změn v průběhu terénu. Na obrázku níže je uveden příklad této problematiky. Uprostřed červeně označené oblasti se nachází koryto potoka. Koryta potok většinou mají ostré hrany, avšak z obrázku jednoznačně nelze vidět nějaké koryto. Tento problém by mohl řešen vkládáním takzvaných povinných hran. Pomocí této hrany bychom mohli vynutit, aby se vrstevnice na nich lámaly a poté by plasticky lépe vyniklo toto koryto.



4.1 Dokumentace

1.1 Třída *Algorithms*

a) double pointDist(QPoint3D &p1, QPoint3D &p2);
   - vrací vzdálenost dvou bodů

b) int getPointLinePosition(QPoint &a,QPoint &p1,QPoint &p2)
   - analyzuje vzájemný vztah bodu a přímky

c) tuple<QPoint3D,double> getCircleCenterAndRadius(QPoint3D &p1,QPoint3D &p2,QPoint3D &p3);
   - vrací střed kruhu a jeho poloměr

d) int getDelaunayPoint(QPoint3D &s,QPoint3D &e,std::vector<QPoint3D> &points);
   - vrací index Delaunayho bodu v rámci vektoru bodů

e) int getNearestPoint(QPoint3D &p, std::vector<QPoint3D> &points);
   - vrací index nejbližšího bodu k bodu p

f) vector<Edge> dT(std::vector<QPoint3D> &points);
   - vytvoří DT ze vstupních bodů

g) void updateAEL(Edge &e, std::list<Edge> &ael);
   - aktualizuje hranu v listu hran

h) double getSlope(QPoint3D &p1, QPoint3D &p2, QPoint3D &p3);
   - vrací hodnotu sklonu trojúhelníku

i) double getExposition(QPoint3D &p1, QPoint3D &p2, QPoint3D &p3);
   - vrátí hodnotu expozice trojúhelníku

j) std::vector<QPoint3D> generatePile(std::vector<QPoint3D> &points);
   - vygeneruje z souřadnice pro kupu

k) int findMaxZ(std::vector<QPoint3D> &points);
   - vrátí největší z souřadnici

l) int findMinZ(std::vector<QPoint3D> &points);
   - vrátí nejmenší z souřadnici

m) std::vector<QPoint3D> generateRandomPoints(QSize &size_canvas, int n);
   - vrátí n náhodných bodů v canvasu

n) std::vector<QPoint3D> generateSaddle(std::vector<QPoint3D> &points);
   - vygeneruje z souřadnice pro sedlo

o) std::vector<QPoint3D> generateRidge(std::vector<QPoint3D> &points);
   - vygeneruje z souřadnice pro hřbet

p) QPoint3D getContourPoint(QPoint3D &p1, QPoint3D &p2, double z);
   - vrací interpolovaný bod

q) std::vector<Edge> getContourLines(std::vector<Edge> &dt, double zmin, double zmax, int dz);
   - vrací hrany vrstevnic

r) std::map<double, std::vector<Edge>> getMainContourLines(std::vector<Edge> &contours, int contour_interval, double dz);
   - vrací hlavní vrstevnice rozdělené podle výšky

s) std::vector<QPoint3D> transformPoints(std::vector<QPoint3D> &points_3d, double &trans_x, double &trans_y, double &scale, int &offset_x, int &offset_y);
   - transformuje vektor bodů podle zadaných parametrů transformace

t) std::vector<Edge> getLabeledContours(std::vector<Edge> &contours, std::vector<Edge> &contours_main, int contour_interval, double dz, double &distance_threshold, double &length_threshold, double &offset);
   - vrátí vektor hran, na kterých se vypíšou popisy vrstevnic

u) std::vector<Edge> getDistancedEdges(std::vector<Edge> &edges, double &threshold);
   - vrátí vektor hran splňující vzdálenostní rozestup

v) std::vector<Edge> chainEdges(std::vector<Edge> &edges);
   - vrátí souvislou vrstevnici (zřetězené hrany)

w) double getMinSlope(std::vector<Triangle> &triangles);
   - vrací nejmenší hodnotu sklonu

x) double getMaxSlope(std::vector<Triangle> &triangles);

- Vrací maximální hodnotu sklonu

1.2 Třída *CSV*

a) vector<QPolygon> read_csv(string &filename)
- načte řádky vstupního *csv* souboru

b) std::vector<QPoint3D> getPoints3D(std::vector<std::vector<std::string>> &csv_content, double &x_min, double &x_max, double &y_min, double &y_max);
- vrací vektor QPoint3D z načtených řádek CSV

1.3 Třída *Draw*

a) void paintEvent(QPaintEvent *event)
- vykreslí polygony na Canvas

b) void mousePressEvent(QMouseEvent *event)
- vrátí souřadnice kurzoru po kliknutí na Canvas

c) void clearPoints()
- vyčistí vektor bodů

d) void clearDT()
- vyčistí vektor DT

e) void clearContours()
- vyčistí vektor souřadnic

f) void clearTriangles()
- vyčistí vektor trojúhelníků

g) void drawCSVPoints(std::vector<QPoint3D> &points_3d);
- vykreslí načtené body

1.4 Třída *Widget*

a) void on_pushButton_2_clicked();

b) void on_pushButton_clicked();

c) void on_pushButton_cleardt_clicked();

d) void on_lineEdit_editingFinished();

e) void on_lineEdit_2_editingFinished();

f) void on_lineEdit_3_editingFinished();

g) void on_pushButton_3_clicked();

h) void on_pushButton_4_clicked();

i) void on_lineEdit_4_editingFinished();

j) void on_pushButton_5_clicked();

k) void on_pushButton_7_clicked();

l) void on_lineEdit_5_editingFinished();

m) void on_pushButton_Load_clicked();

n) void on_pushButton_6_clicked();

o) void on_checkBox_clicked();

p) void on_checkBox_2_clicked();

q) void on_checkBox_2_stateChanged(int arg1);

r) void on_save_clicked();

s) void on_save_canvas_clicked();

t) void on_dmtUP_clicked();

4.2 Závěr

Výsledkem je aplikace DMT s grafickým uživatelským rozhraním, která je uvolněna pod licencí GNU GPL. Aplikace je psána v jazyce C++. Pro grafické uživatelské rozhraní byl použit framework QT.
Aplikace umožňuje nahrát body z textového souboru, naklikat si vlastní nebo použít generátor terénních tvarů. Pro tvorbu triangulace byl zvolena metoda inkrementální konstrukce DT. Výsledek analýzy terénu lze ukládat ve formátu PNG.

4.3 Zdroje

[1] Rovinné triangulace a jejich využití [online]. [cit. 2021-12-08]. Dostupné z: https://web.natur.cuni.cz/~bayertom/images/courses/Adk/adk5.pdf

Jakub Šimek, Jan Kučera