

TECHNICKÁ ZPRÁVA

ÚLOHA Č.3: DIGITÁLNÍ MODEL TERÉNU

1. ZADÁNÍ:

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

Výstup: polyedrický DMT nad množinou P představovaný vrstevnicemi doplněný vizualizací sklonu trojúhelníku 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. navrhnete 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.

SKUPINA: Kateřina Chromá, Štěpán Šedivý

2. ZPRACOVANÉ DOBROVOLNÉ ČÁSTI ÚKOLU:

- *Automatický popis vrstevnic*

3b

3. VSTUPNÍ DATA:

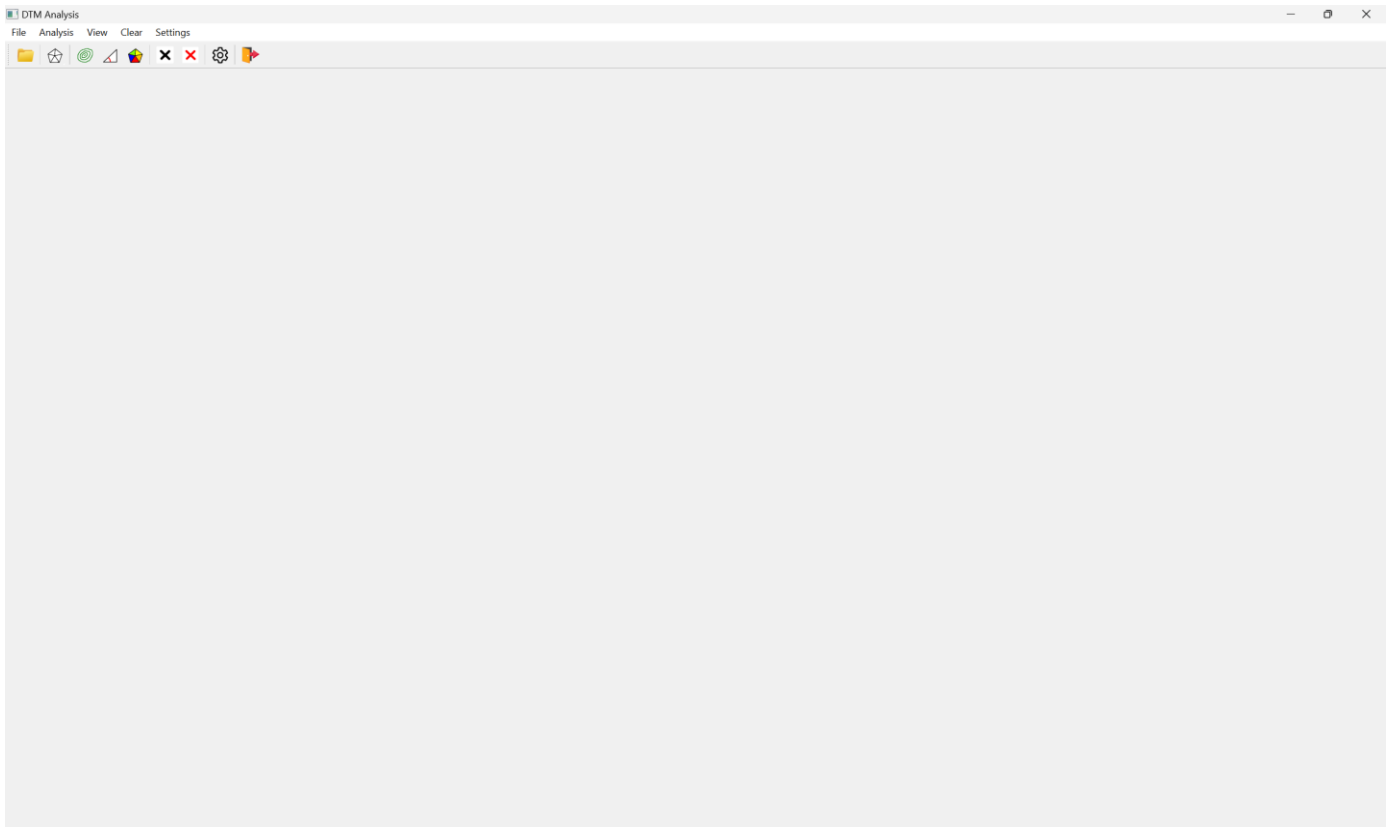
Za vstupní data jsme zvolili data DMR 5. Data jsme zgeneralizovali, tak abychom měly maximálně 700 bodů. Tyto body do aplikace vstupují ve formátu TXT a jsou vyjádřeny pomocí souřadnic X, Y a Z.

4. VÝSTUPNÍ DATA:

Výstupem tohoto úkolu je aplikace generující vrstevnice, slope a aspect.










5. VYTVOŘENÁ APLIKACE:

Grafické rozhraní bylo vytvořeno s využitím frameworku QT, kde bylo vytvořeno grafické rozložení (umístění tlačítek, ...). Toto rozhraní bylo zkonvertováno do kódu programovacího jazyka Python.



Obrázek 1: Grafické rozhraní

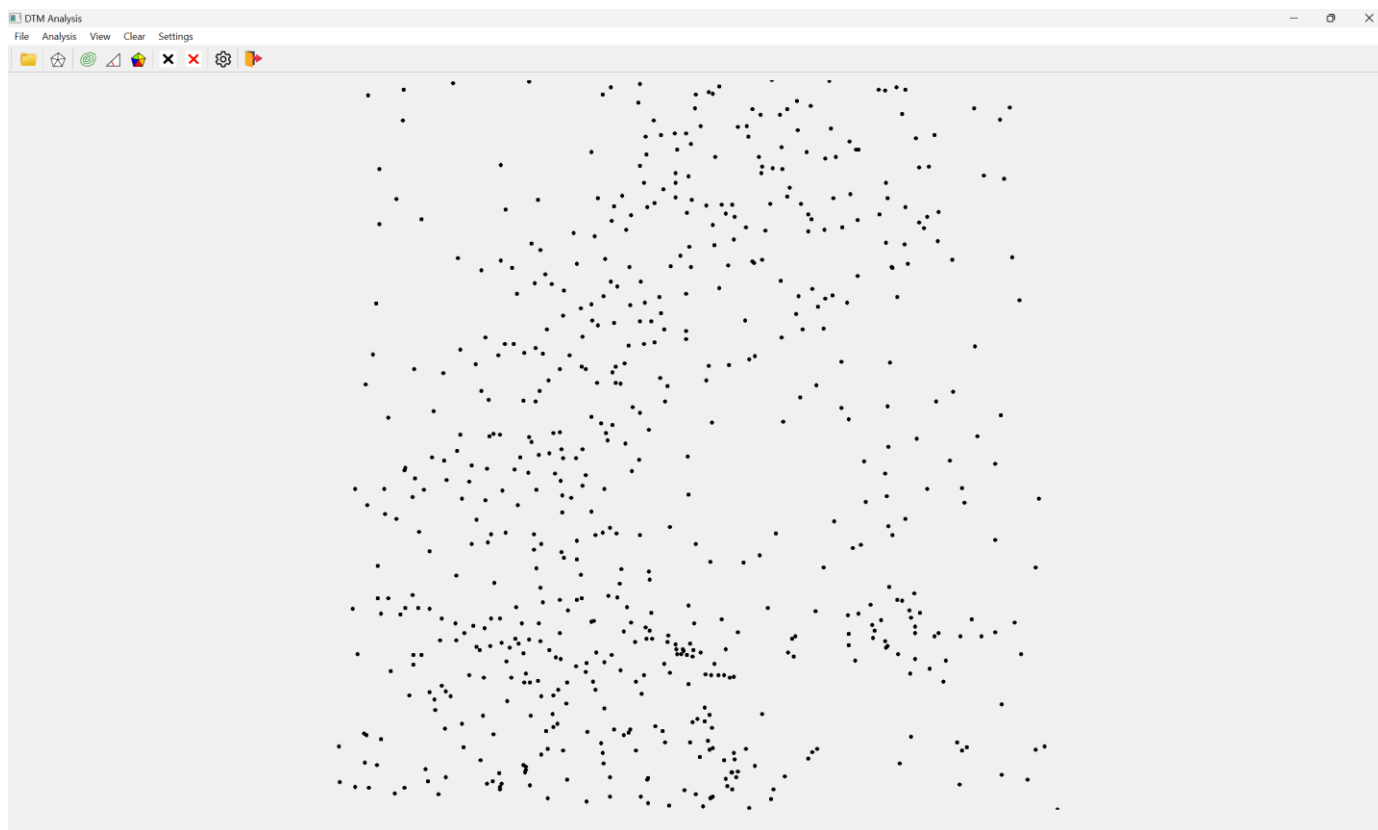
V grafickém rozhraní bylo vytvořeno pět tlačítek:

-  : otevření souboru (umožňuje pouze otevřít soubory formátu *shapefile*),
-  : vykreslení triangulační sítě pomocí *Delaunay triangulace*
-  : vykreslení vrstevnic,
-  : vykreslení metody *slope* (zobrazení sklonu svahu),
-  : vykreslení metody *aspect* (zobrazení směru svahu),
-  : odstranění výsledků,
-  : odstranění všech prvků,
-  : nastavení parametrů vrstevnic,
-  : vypnutí aplikace.

Všechny tyto možnosti jsou také spustitelné z MenuBar.

V MenuBaru je také možné zvolit si jaké metody chceme zobrazit.

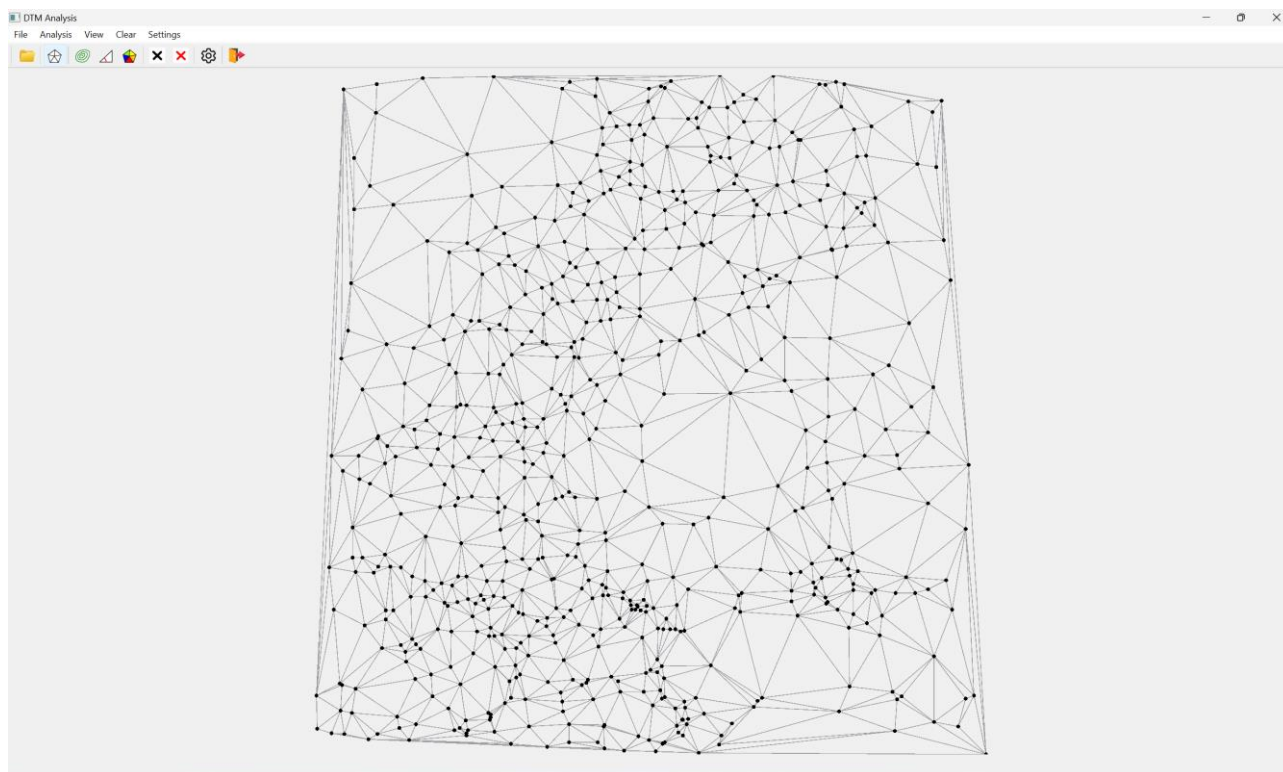
Uživatel si do aplikace může nahrát textový soubor obsahující seznam souřadnic bodů.



Obrázek 2: Grafické rozhraní – import bodů

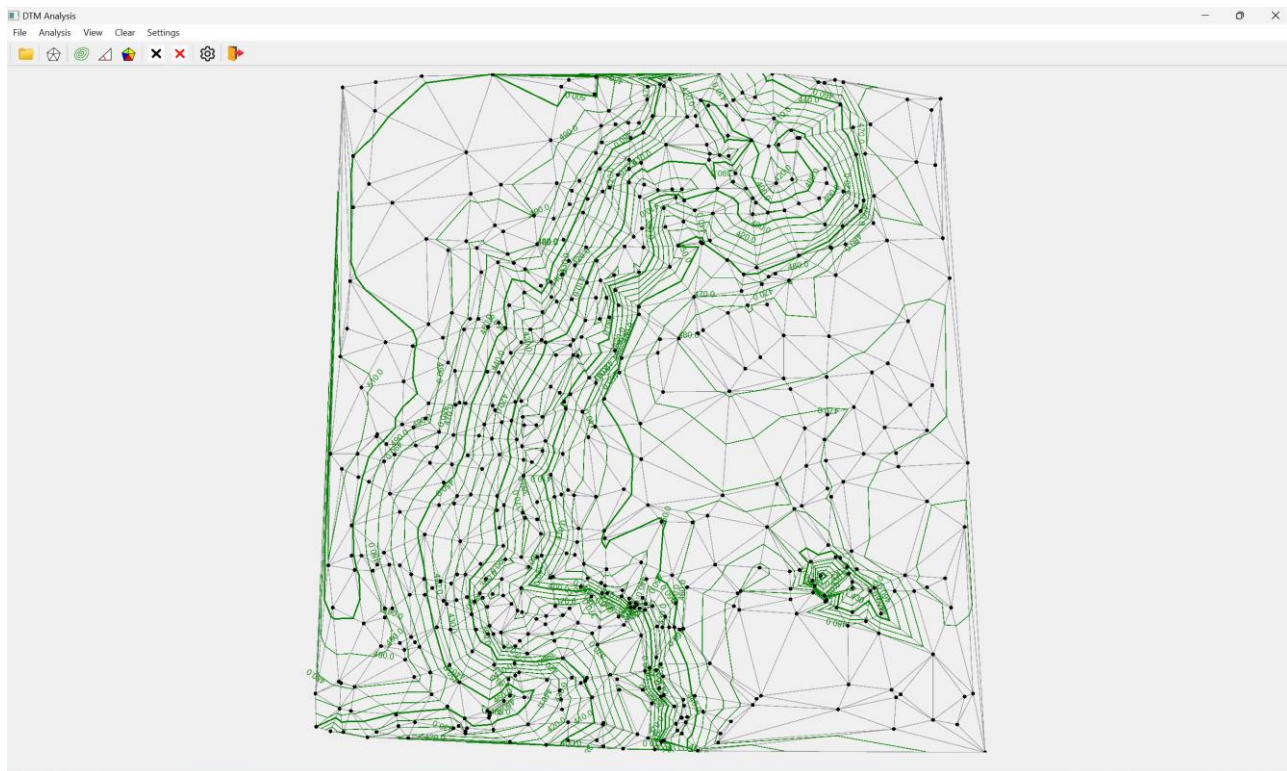
Po zvolení určité funkce se uživateli zobrazí její vyhodnocení.

1. Delaunay triangulace



Obrázek 3: Grafické rozhraní – Delaunay triangulace

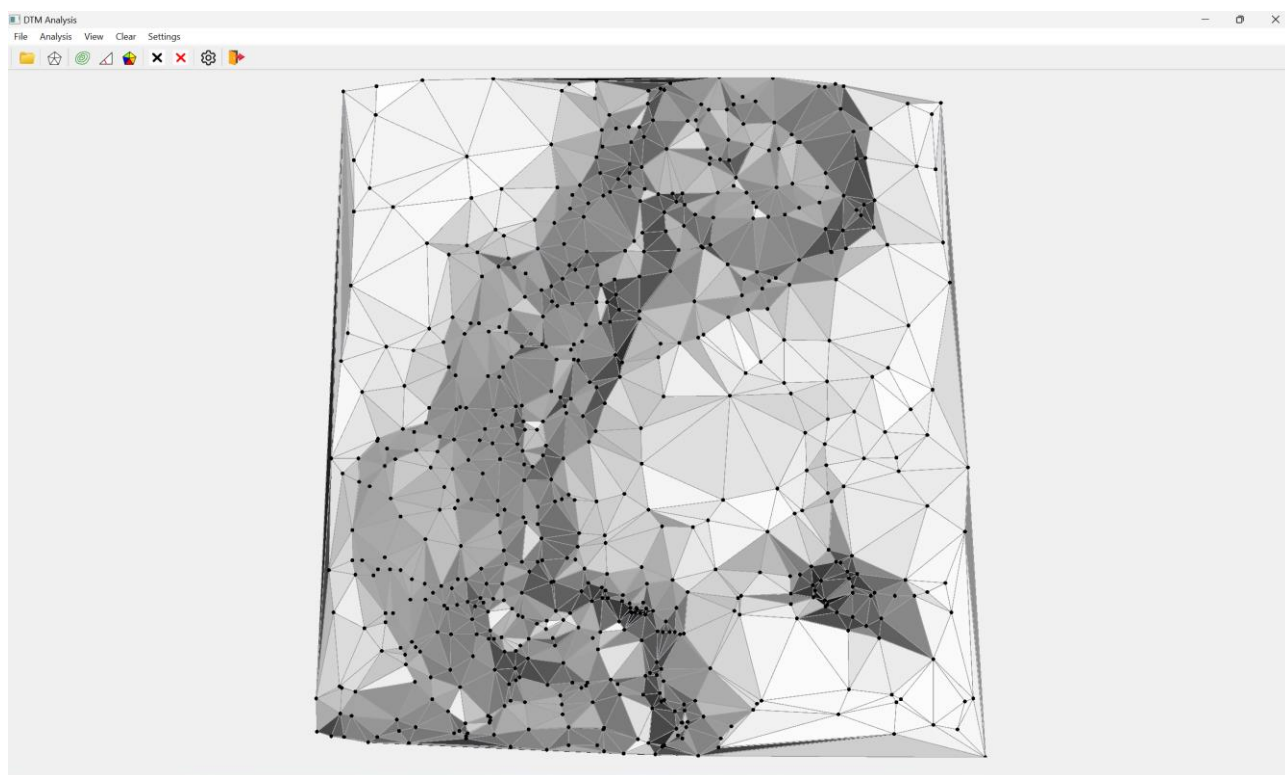
2. Vrstevnice



Obrázek 4: Grafické rozhraní – vrstevnice

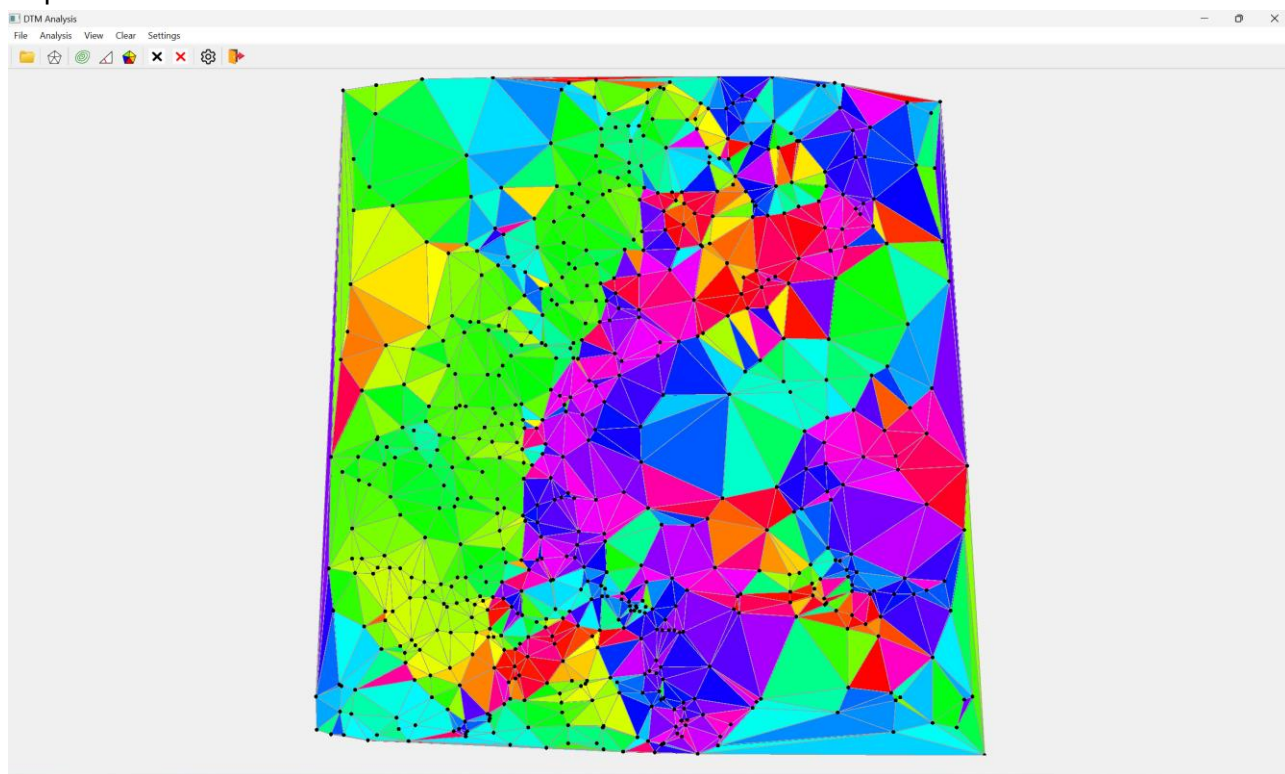
3. Slope – sklon svahu

čím tmavší barva, tím větší sklon

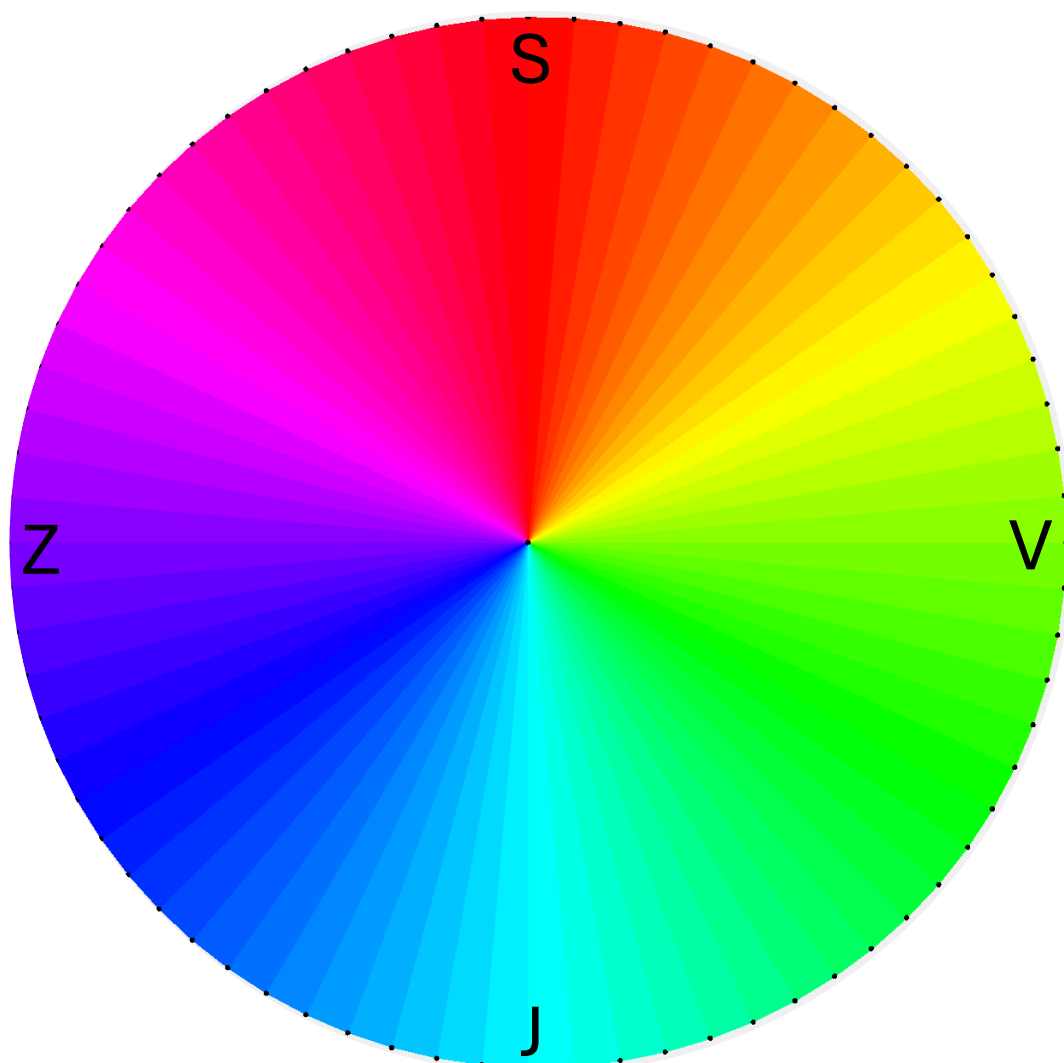


Obrázek 5: Grafické rozhraní – slope

4. Aspect – směr svahu



Obrázek 6: Grafické rozhraní – aspect



Obrázek 7: Grafické rozhraní – rozložení barev na světové strany

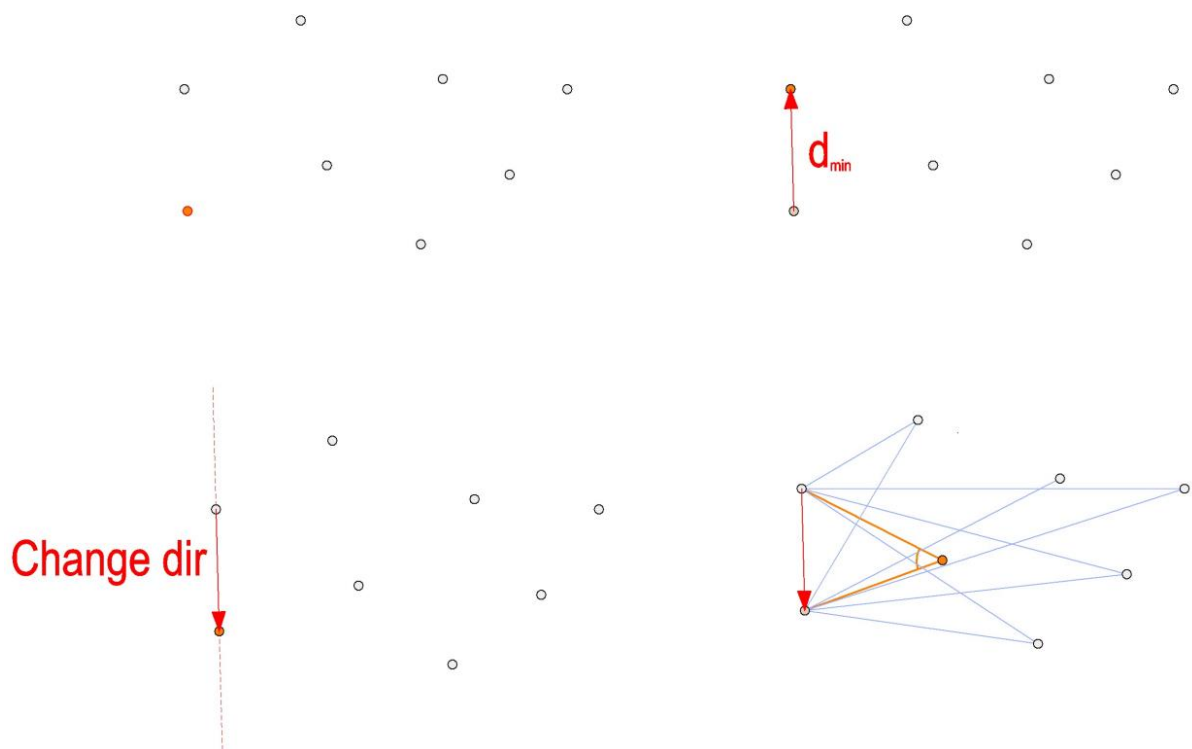
6. POSTUP ZPRACOVÁNÍ:

Hlavním cílem tohoto úkolu bylo vytvořit aplikaci, která provede triangulaci, tedy vytvoří trojúhelníkovou síť nad vstupním mračnem bodů. Nad takto vzniklou trojúhelníkovou sítí následně aplikace provede výpočet sklonu svahu (slope), směru svahu (aspect) a vrstevnic.

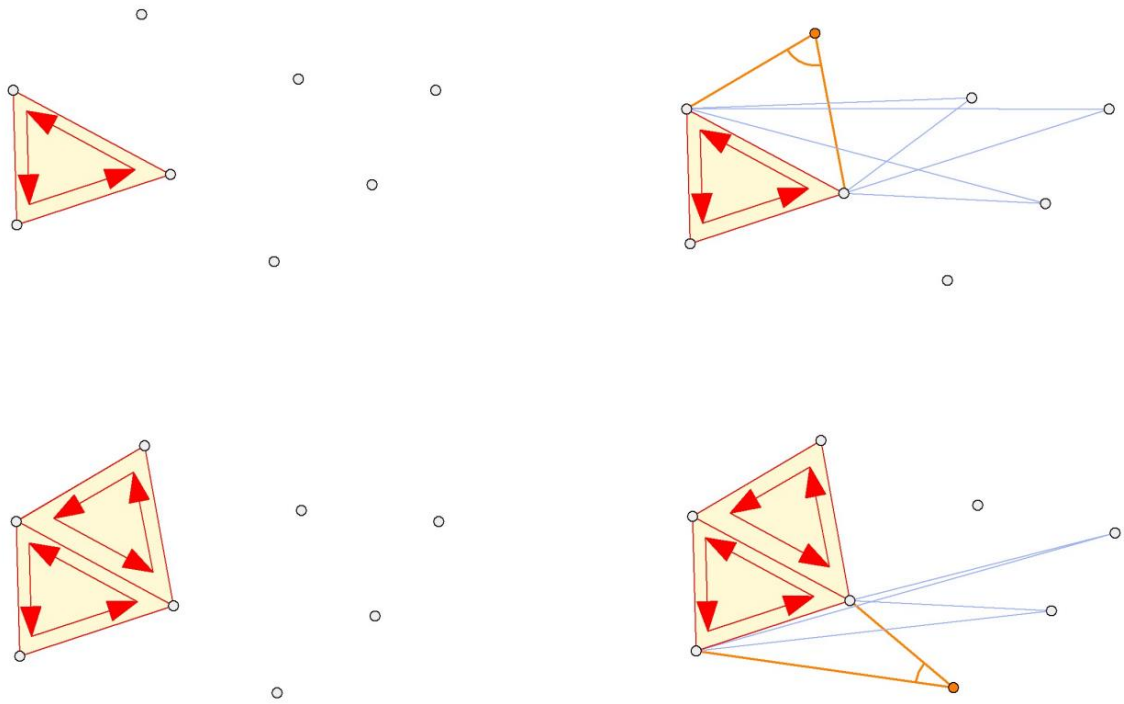
a. *Delaunayova triangulace*

Pomocí Delaunayovi triangulace byla ze vstupních bodů bodového mračna vytvořena síť. Na této síti poté pracují všechny ostatní algoritmy. V této úloze byl využit algoritmus pro metodu inkrementální konstrukce.

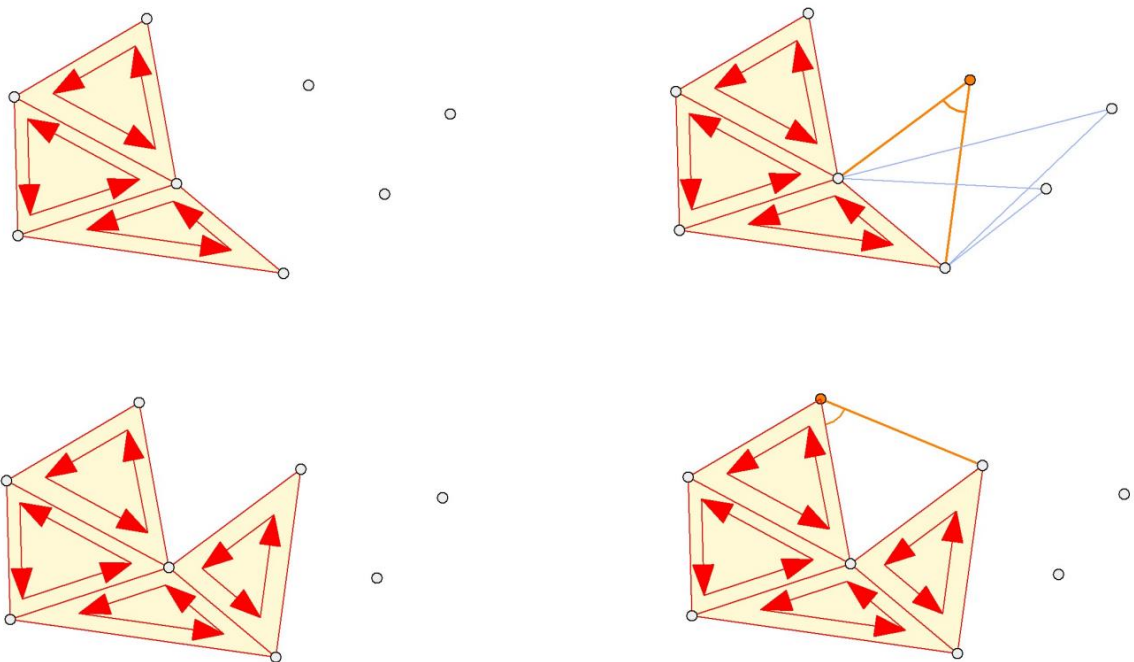
Metoda je založena na postupném přidávání bodů do již vytvořené triangulace. K určité hraně je vždy dohledáván vhodný bod, který splňuje podmínku, že úhel u tohoto bodu mezi body prohledávané hrany musí být maximální. Takto vyhledaný bod tvoří nový trojúhelník. Při konstrukci algoritmu je využíván Active Edge List (AEL), ve kterém se ukládají hrany, pro které hledáme vhodný bod.



Obrázek 8: Metoda inkrementální konstrukce [1]



Obrázek 9: Metoda inkrementální konstrukce [1]



Obrázek 10: Metoda inkrementální konstrukce [1]

```
def createDT(self, points: list[QPoint3DF]):
    #Create Delaunay triangulation with incremental method
    if points is empty:
        pass

    dt = []
    ael = []
```

```

#Sort points by x
p1 = min(points, key = lambda k: k.x() )

#Find nearest point
p2 = self.getNearestPoint(p1, points)

#Create edges
e = Edge(p1, p2)
e_op = Edge(p2, p1)

#Add both edges to ael
ael.append(e)
ael.append(e_op)

# Repeat until ael is empty
while ael:
    # Take first edge
    e1 = ael.pop()

    #Change orientation
    e1_op = e1.changeOrientation()

    #Find optimal Delaunay point
    p_dt = self.getDelaunayPoint(e1_op.getStart(), e1_op.getEnd(), points)

    #Did we find a suitable point?
    if p_dt != None:
        #create remaining edges
        e2 = Edge(e1_op.getEnd(), p_dt)
        e3 = Edge(p_dt, e1_op.getStart())

        #Create Delaunay triangle
        dt.append(e1_op)
        dt.append(e2)
        dt.append(e3)

        #Update AEL
        self.updateAEL(e2, ael)
        self.updateAEL(e3, ael)

return dt

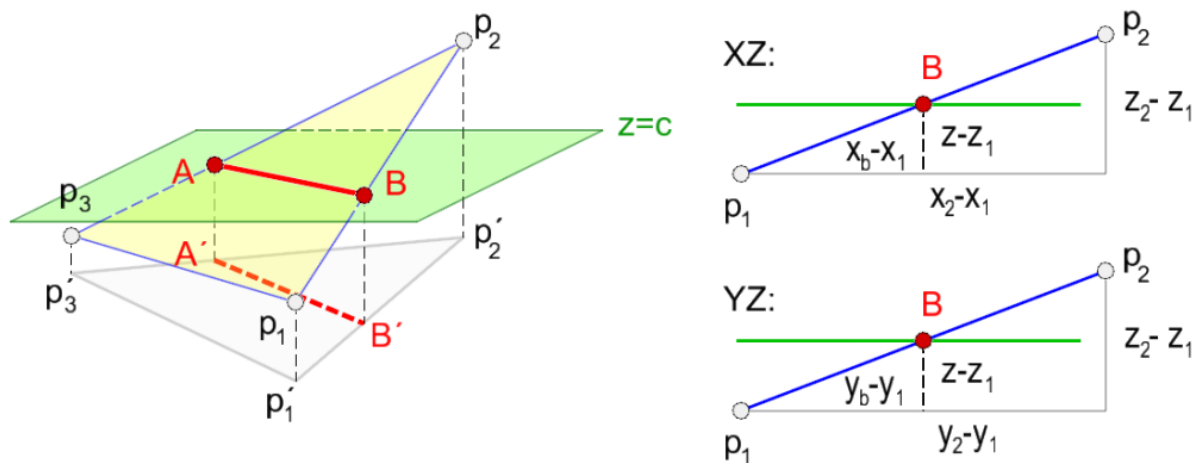
```

Algoritmus nejprve seřadí body podle souřadnice x, přičemž první bod p1 je určen jako ten s nejnižší hodnotou x. Následně se najde nejbližší bod p2 k bodu p1. Mezi těmito dvěma body se vytvoří hrany a přidají se do aktivního seznamu hran (AEL). Algoritmus poté iterativně zpracovává hrany z AEL, kde pro každou hranu najde optimální Delaunayův bod a vytvoří nové hrany, které se přidají zpět do AEL. Tento proces pokračuje, dokud není AEL prázdný. Výsledkem je seznam Delaunayových trojúhelníků, které jsou vráceny jako konečný výstup algoritmu.

b. Tvorba vrstevnic

Trojúhelníky, které byly vytvořeny Delaunayovou triangulací, byly použity pro vytvoření vrstevnic prostřednictvím metody lineární interpolace. Tato metoda vycházela z principů analytické geometrie. V každém z těchto trojúhelníků byla hledána průsečnice s horizontální rovinou v konkrétní výšce. Pro výpočet průsečnic byly využity následující vzorce

$$x_a = \frac{x_3 - x_1}{z_3 - z_1}(z - z_1) + x_1, \quad x_b = \frac{x_2 - x_1}{z_2 - z_1}(z - z_1) + x_1,$$
$$y_a = \frac{y_3 - y_1}{z_3 - z_1}(z - z_1) + y_1, \quad y_b = \frac{y_2 - y_1}{z_2 - z_1}(z - z_1) + y_1.$$



Obrázek 11: Konstrukce vrstevnic lineární interpolací [1]

```
def createContourLines(self, dt, zmin, zmax, dz):
    #Create contour lines defined by interval and step
    contours = []

    #Process all triangles
    for i in range(0, len(dt), 3):
        #Get vertices of triangle
        p1 = dt[i].getStart()
        p2 = dt[i].getEnd()
        p3 = dt[i + 1].getEnd()

        #Get z coordiantes
        z1 = p1.getZ()
        z2 = p2.getZ()
        z3 = p3.getZ()

        #Create all contour lines
        for z in arange(zmin, zmax, dz):
```

```

#Compute edge height differences
dz1 = z - z1
dz2 = z - z2
dz3 = z - z3

#skip coplanar triangle
if dz1 == 0 and dz2 == 0 and dz3 == 0:
    continue

#Edge p1 and p2 is colinear
elif dz1 ==0 and dz2 ==0:
    contours.append(dt[i])

#Edge p2 and p3 is colinear
elif dz2 ==0 and dz3==0:
    contours.append(dt[i+1])

#Edge p3 and p1 is colinear
elif dz3 ==0 and dz1==0:
    contours.append(dt[i+2])

#Edges p1, p2 and p2, p3 intersected by plane
elif dz1 * dz2 <= 0 and dz2 * dz3 <= 0:

    #Compute intersections
    a = self.getContourPoint(p1, p2, z)
    b = self.getContourPoint(p2, p3, z)

    #Create edge
    e1 = Edge(a, b)

    #Add edge to contour lines
    contours.append(e1)

#Edges p2, p3 and p3, p1 intersected by plane
elif dz2 * dz3 <= 0 and dz3 * dz1 <= 0:

    #Compute intersections
    a = self.getContourPoint(p2, p3, z)
    b = self.getContourPoint(p3, p1, z)

    #Create edge
    e1 = Edge(a, b)

    #Add edge to contour lines
    contours.append(e1)

#Edges p3, p1 and p1, p2 intersected by plane
elif dz3 * dz1 <= 0 and dz1 * dz2 <= 0:

    #Compute intersections
    a = self.getContourPoint(p3, p1, z)

```

```

        b = self.getContourPoint(p1, p2, z)

        #Create edge
        e1 = Edge(a, b)

        #Add edge to contour lines
        contours.append(e1)

    return contours

```

Algoritmus prochází všechny trojúhelníky v triangulaci, přičemž pro každý trojúhelník získá jeho vrcholy a jejich z-souřadnice. Pro každou zadanou výšku v intervalu $z_{min} - z_{max}$ algoritmus vypočítá rozdíly výšek mezi zadanou výškou a z-souřadnicemi vrcholů trojúhelníka. Pokud je trojúhelník koplanární s rovinou z , ignoruje ho. V případě, že jsou některé hrany trojúhelníka kolinéární s rovinou, přidá je do seznamu vrstevnic. Pokud rovina z protíná hrany trojúhelníka, algoritmus vypočítá průsečíky a vytvoří nové hrany, které přidá do seznamu vrstevnic. Tento proces se opakuje, dokud nejsou zpracovány všechny trojúhelníky a všechny zadané výšky. Závěrem algoritmus vrátí seznam vytvořených vrstevnicových čar.

c. Výpočet sklonu

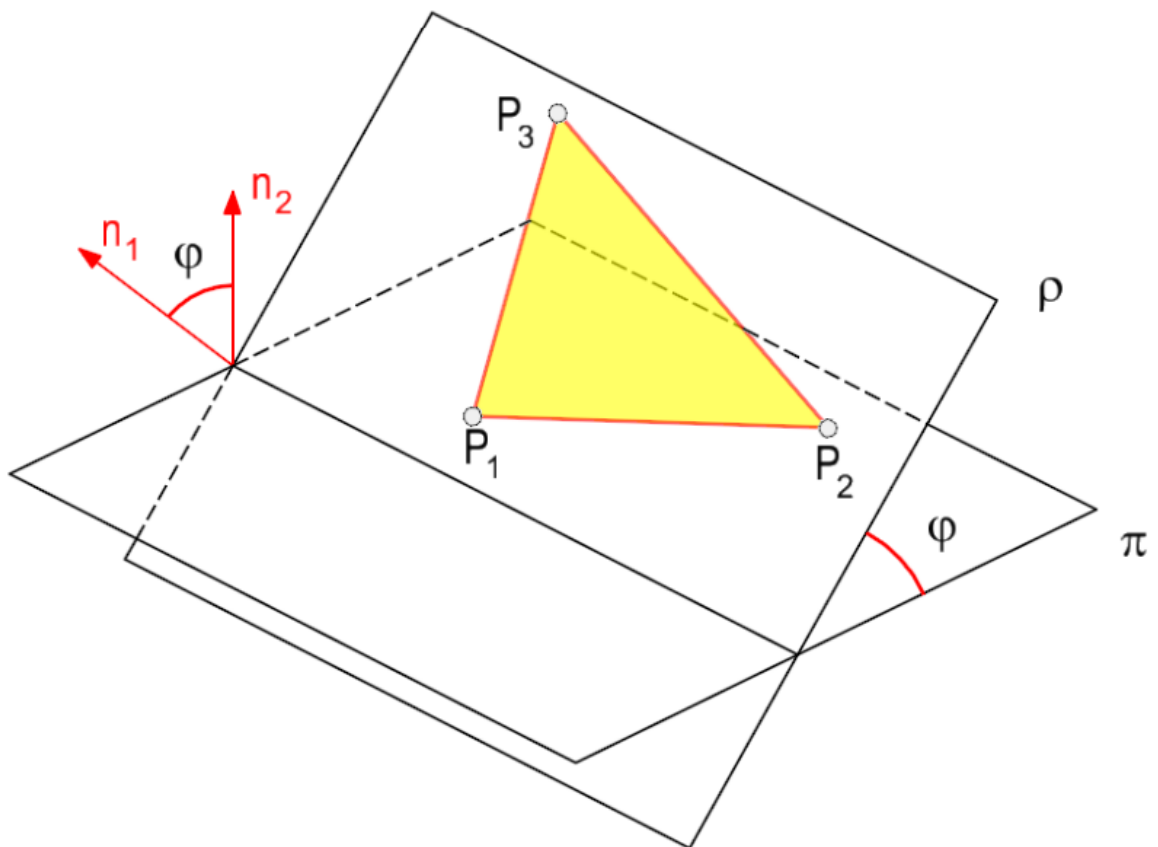
Analýza sklonu probíhá pro každý trojúhelník samostatně. Vždy byl počítán úhel mezi normálovým vektorem vodorovné roviny a normálovým vektorem vypočteným pro daný trojúhelník, podle následujícího vzorce

$$\varphi = \arccos \frac{n_1 \cdot n_2}{\|n_1\| \cdot \|n_2\|} = \arccos \frac{c}{\|n_1\|},$$

$$n_1 = (a, b, c),$$

$$n_2 = (0, 0, 1).$$

Vypočtený sklon φ je následně vizualizován ve stupních šedi pro každý trojúhelník.



Obrázek 12: Výpočet sklonu terénu [1]

```
def computeSlope(self, p1:QPoint3DF, p2:QPoint3DF, p3:QPoint3DF):
    #Compute triangle slope

    #Directions
    ux = p1.x() - p2.x()
    uy = p1.y() - p2.y()
    uz = p1.getZ() - p2.getZ()

    vx = p3.x() - p2.x()
    vy = p3.y() - p2.y()
    vz = p3.getZ() - p2.getZ()

    #Normal vector
    nx = uy * vz - vy * uz
    ny = - (ux*vz - vx * uz)
    nz = ux * vy - vx * uy

    #Norm
    norm = (nx**2 +ny**2 + nz**2)**(1/2)

    return acos(abs(nz)/norm)
```

d. Výpočet expozice

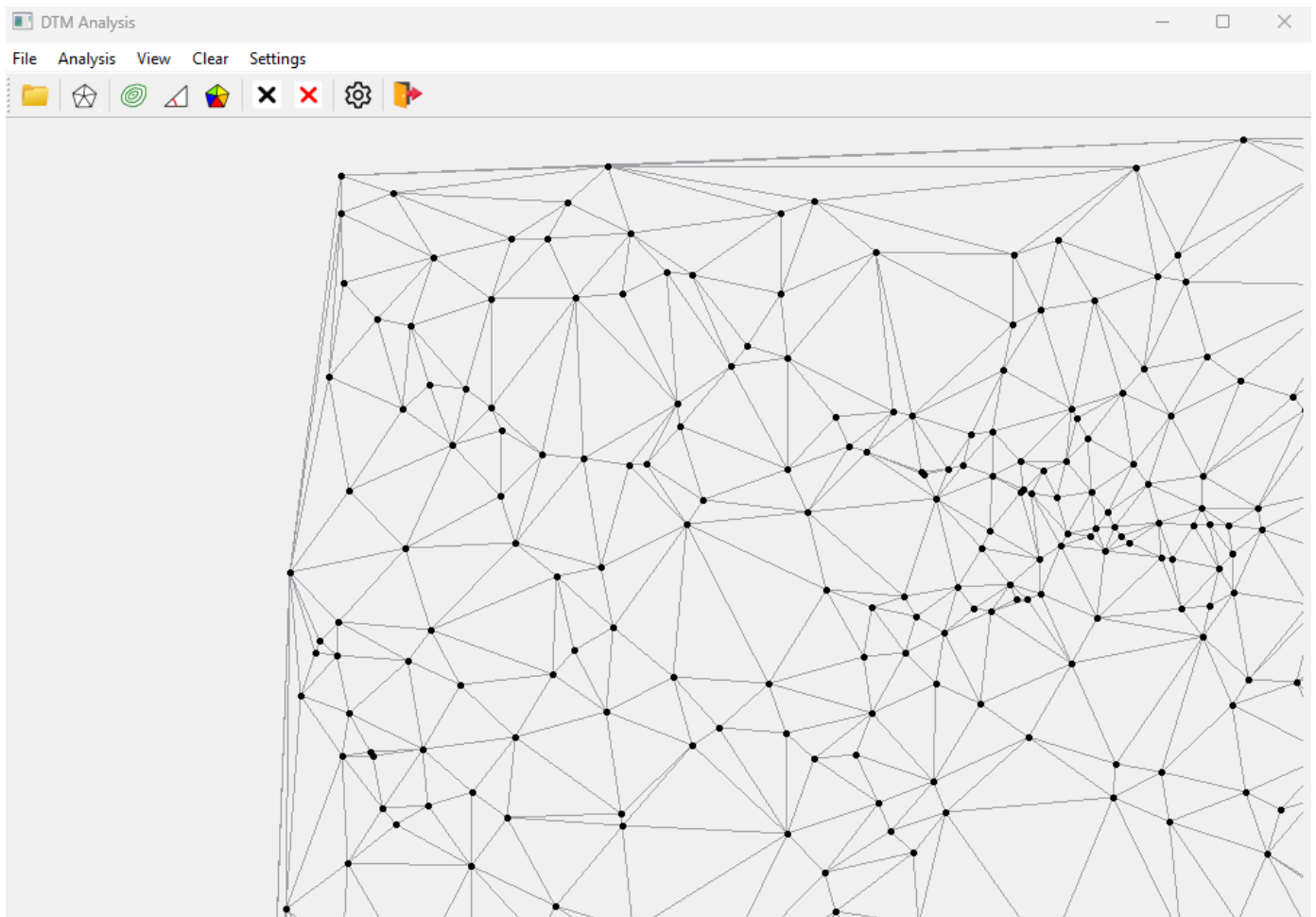
Jako u analýzy sklonu se i analýza expozice digitálního modelu terénu provádí postupně pro každý trojúhelník zvlášť. Výpočet probíhá po směru hodinových ručiček, počítán je ve vodorovné rovině od severu k průmětu normálového vektoru daného trojúhelníka, dle následujícího vzorce

$$A = \text{atan} \frac{n_{1x}}{n_{1y}}.$$

Vypočtená expozice je vizualizovaná barevným přechodem, který je vyobrazen na obrázku 7.

```
def computeAspect(self, p1:QPoint3DF, p2:QPoint3DF, p3:QPoint3DF):  
    #Compute triangle aspect  
  
    #Directions  
    ux = p1.x() - p2.x()  
    uy = p1.y() - p2.y()  
    uz = p1.getZ() - p2.getZ()  
  
    vx = p3.x() - p2.x()  
    vy = p3.y() - p2.y()  
    vz = p3.getZ() - p2.getZ()  
  
    #Normal vector  
    nx = uy * vz - vy * uz  
    ny = - (ux*vz - vx * uz)  
  
    return atan2(nx, ny)
```

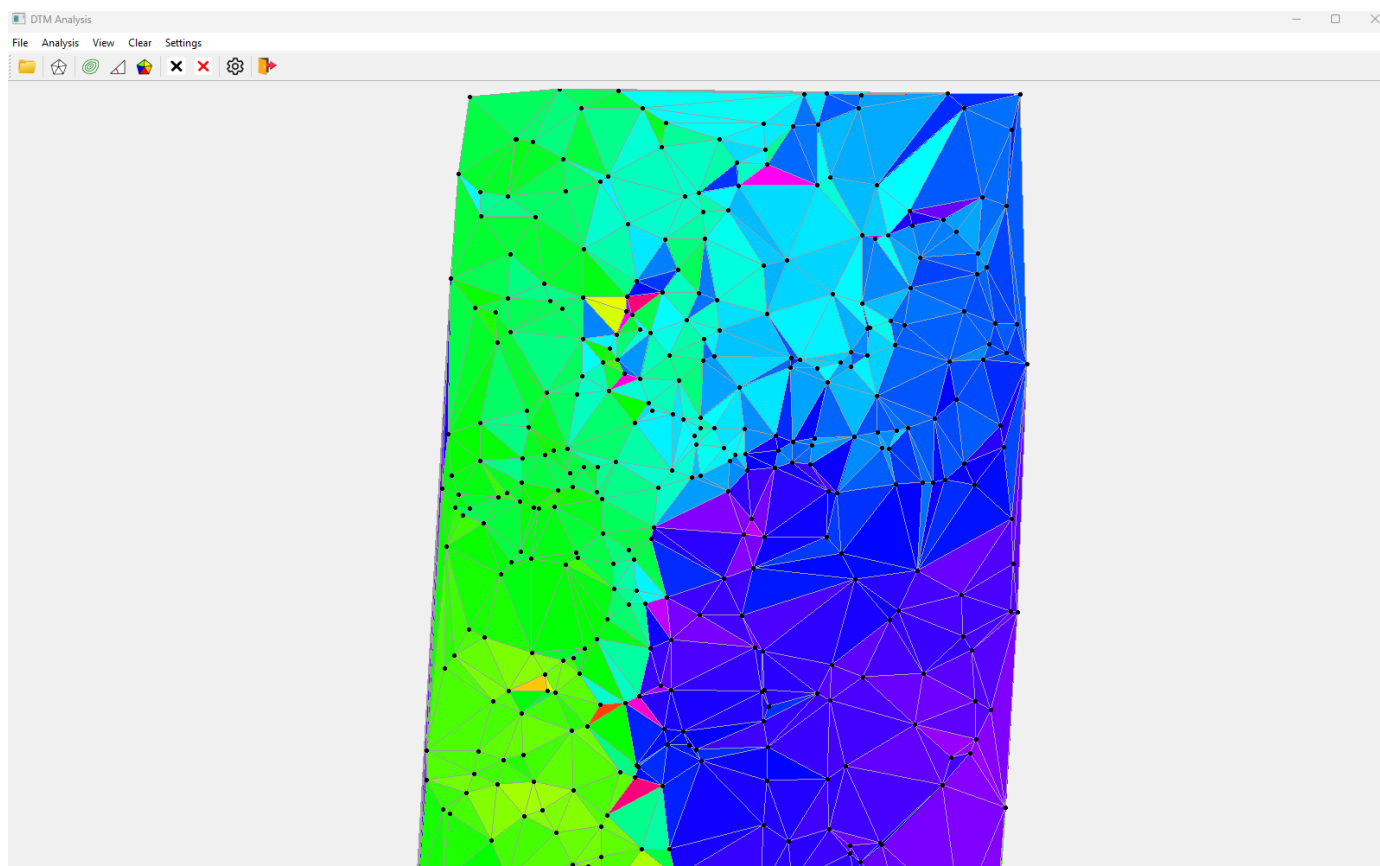
8. VÝSLEDKY:



Obrázek 13: Delaunayova triangulace, kupa

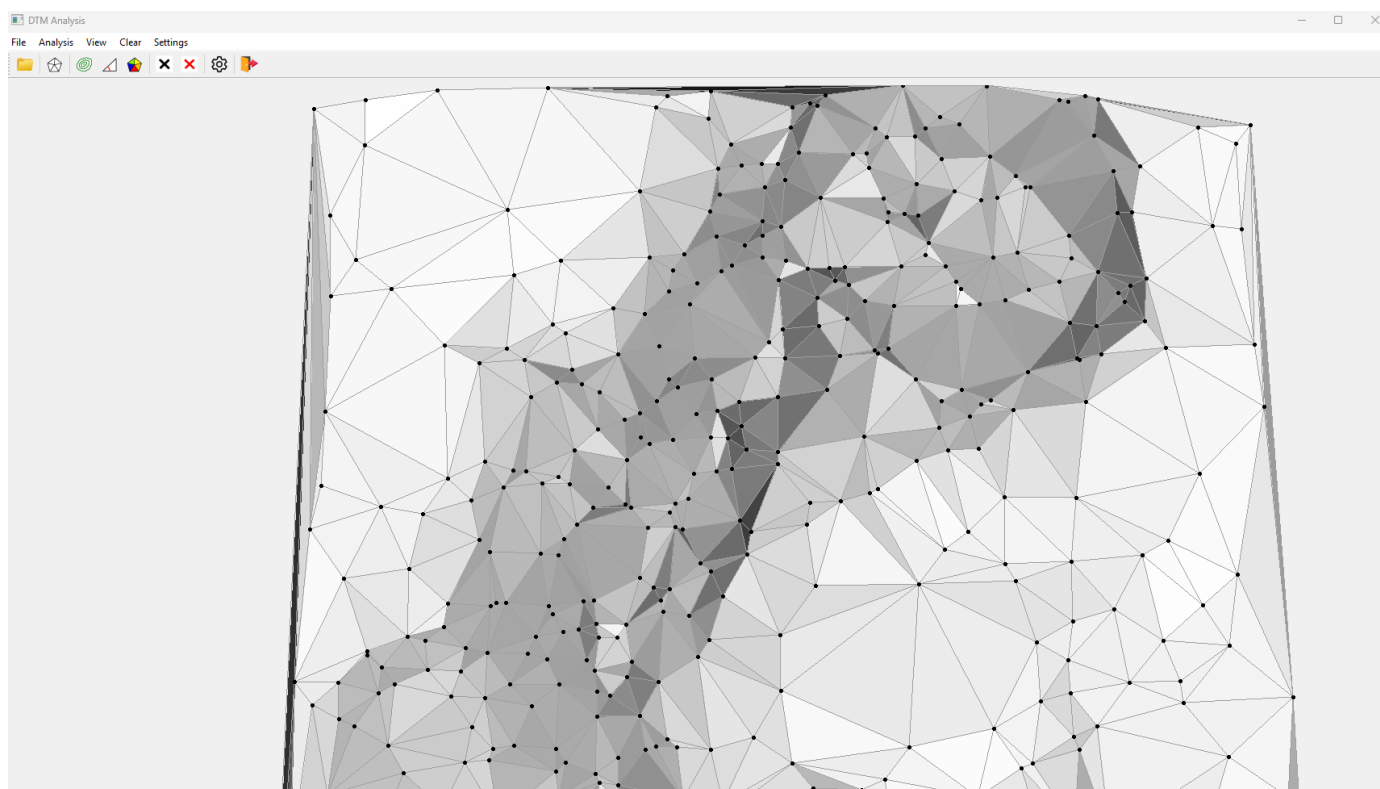
Na obrázku 13 můžeme vidět největší nedostatek Delaunayovi triangulace, a to konkrétně trojúhelníky s dlouhými stranami na okrajích zájmového území. V těchto místech má trojúhelník k reprezentaci terénu v tomto místě skutečnosti daleko a v praxi je nutné tato místa řešit. Jako způsoby řešení se nabízejí například manuální editace trojúhelníků na okrajích, případně odstranění těch nadbytečných. Dalším řešením by mohlo být nastavení prahových hodnot pro délky stran trojúhelníka. Vyšší kvalité výsledné trojúhelníkové sítě lze jít také naproti vhodným rozložením výchozích bodů po mapované oblasti. Vstupní body triangulace by měli být do maximální možné míry pravidelně rozloženy a tato síť by měla být následně zahuštěna v místech bohatých na terénní tvary. Pokud k zahuštění sítě nedojde algoritmus je nucen interpolovat více než je vhodné. V takovém případě nejen, že se z modelu vytratí detaily, ale v oblastech málo posetých body můžeme vlivem interpolace dojít k úplně špatným výsledům.

Pro zvýšení vypovídající hodnoty a rychlé získání přehledu o zobrazovaném území jsou v rámci aplikace počítány také analýzy trojúhelníkové sítě vzniklé pomocí Delaunayovi triangulace, konkrétně potom analýza expozice a sklonu svahů. Analýzy jsou počítány pro každý trojúhelník zvlášť a jednotlivým hodnotám je při vizualizaci přiřazena barva dle definovaných schémat. Jako drobnou nevýhodu těchto analýz spatřujeme právě to, že jsou prováděny pro každý trojúhelník zvlášť a v kontextu celého zobrazovaného území mohou místy působit zavádějícím způsobem.



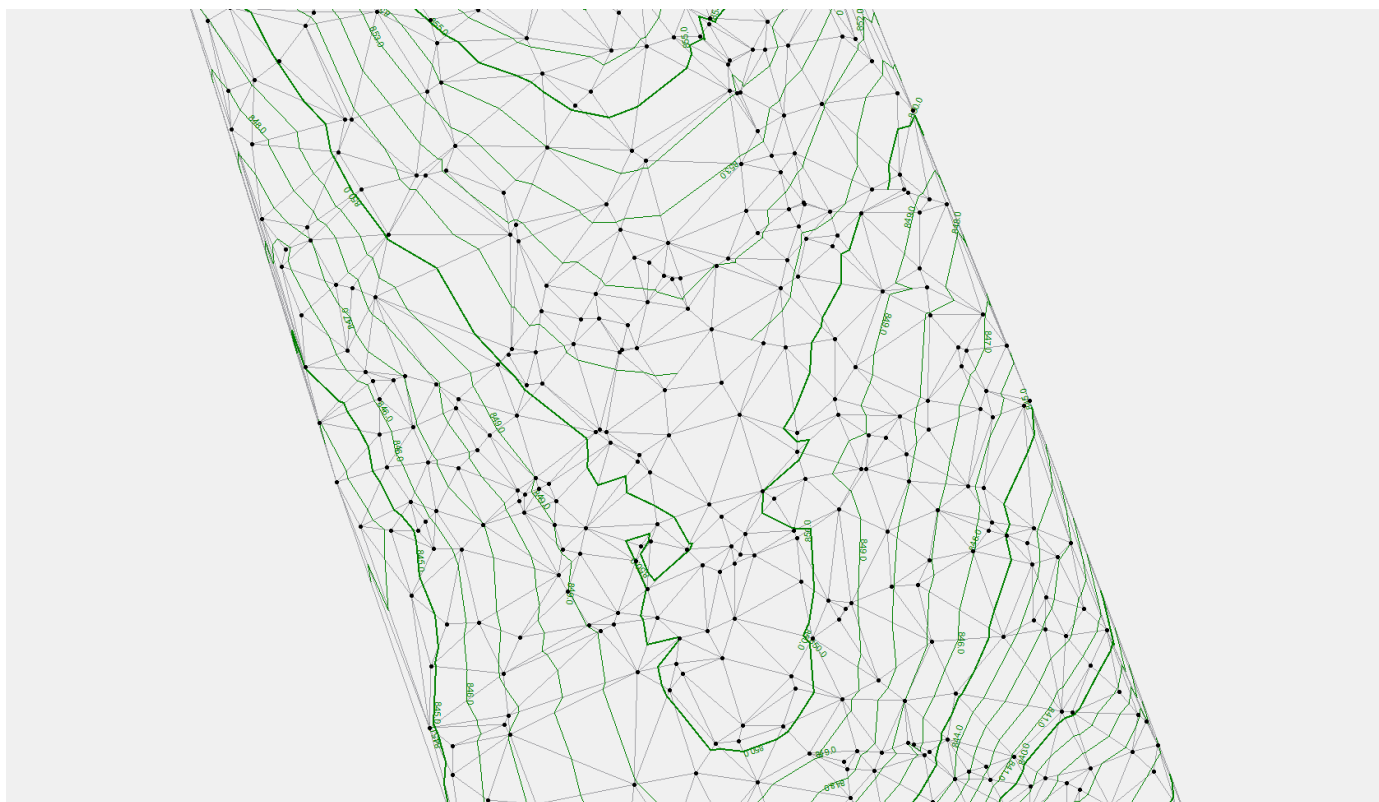
Obrázek 14: Delaunayova triangulace a výpočet expozice, údolí

Obrázek 14 obsahuje trojúhelníkovou síť údolí, na které lze vhodně představit analýzu expozice. Na základě výše definovaného barevného schématu lze snadno poznat, že na pravé straně je svah orientovaný k západu a na levé straně obrázku je svah orientovaný k jihu. Údolí tedy ve směru nahoru směřuje k jihu a výškové rozdíly se v jižním směru stávají pozvolnějšími.



Obrázek 15: Delaunayova triangulace a výpočet sklonu, okolí propasti Macocha

Na obrázku 15 můžeme vidět trojúhelníkovou síť okolí propasti Macocha. Pro znázornění převýšení byla v tomto případě využita analýza sklonu svahu. Po okrajích má zobrazovaná oblast pozvolnou změnu nadmořské výšky, trojúhelníky jsou proto bílé, nebo velmi světlé šedé barvy. Směrem doprostřed sledované oblasti trojúhelníky sítě začínají postupně tmavnout, podle toho lze velmi intuitivně poznat, ve které části se propast nachází.



Obrázek 16: Delaunayova triangulace a interpolace vrstevnic, spočinek

Další funkcionalitou aplikace je interpolace vrstevnic nad vytvořenou trojúhelníkovou sítí. V tomto případě jsme pro vizualizaci zvolili terénní tvar spočinek. Na obrázku 16 lze pomocí vrstevnic zelené barvy spočinek dobře znatelný. Generování vrstevnic se dá v rámci aplikace nastavovat. Mezi nastavitelné parametry patří nastavení rozsahu intervalu generování vrstevnic (minimální a maximální výška) a interval po kterém se budou vrstevnice generovat.

Mezi hlavní problémy generování vrstevnic tímto způsobem patří problémy s hladkostí jejich průběhu. Zejména v okolí spočinku má vrstevnice velmi kostrbatý tvar a pro další využití by bylo vhodné ji manuálně upravit, či případně do aplikace vložit některý vyhlazovací algoritmus. Dalším problémem jsou již výše zmíněné dlouhé trojúhelníky na krajích bodového mračna. Tyto trojúhelníky vrstevnici prudce zalomí do míst, ve kterých se výška reprezentovaná konkrétní vrstevnicí vůbec vyskytovat nemusí. Mezi problémy této metody tvorby vrstevnic také patří, že algoritmus ignoruje přirozené hrany terénního reliéfu, jednoduše proto, že z výpočtu nevyplývají. Tyto hrany by bylo nutné do modelu přidat ručně, jelikož se jejich tvorba bude automatizovat jen velmi těžko. Při vizualizaci vrstevnic v okně aplikace by mohlo dojít k dalším kartografickým problémům, například k překrývání popisků vrstevnic sousedními vrstevnicemi.

9. ZÁVĚR:

Bylo vytvořené grafické rozhraní, do kterého uživatel může prostřednictvím textového souboru nahrát mračno bodů. Nad takto nahraným bodovým mračnem potom může uživatel vytvořit trojúhelníkovou síť metodou Delaunayovi triangulace. Pro vzniklou trojúhelníkovou síť může následně provádět implementované analýzy. Konkrétně vypočítat sklon svahů, jejich orientaci vzhledem ke světovým stranám a generovat vrstevnice s nastavitelným rozestupem.

10. ZDROJE:

Obrázky [8], [9], [10], [11], [12] a informace o průběhu výpočtu:

Rovinné triangulace a jejich využití. Online. BAYER, Tomáš. Rovinné triangulace a jejich využití. 2024. Dostupné z: https://web.natur.cuni.cz/~bayertom/images/courses/Adk/adk5_new.pdf. [cit. 2024-05-25].

V Plzni dne 25.5.2024

Kateřina Chromá, Štěpán Šedivý