

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

název předmětu

ALGORITMY DIGITÁLNÍ KARTOGRAFIE A GIS

číslo
úlohy

3

název úlohy

Digitální model terénu a jeho analýzy

školní rok

2024

studijní skup.

C-101

číslo zadání

-

Zpracoval:

Josef Jehlička

datum

3.6.
2024

klasifikace

TECHNICKÁ ZPRÁVA

1) *Zadání:* Úkolem bylo vytvořit polyedrický digitální model terénu (DMT) nad množinou 3D bodů doplněný vizualizací sklonu trojúhelníků a jejich expozicí. Metodou inkrementální konstrukce je potřeba vytvořit nad množinou P vstupních bodů 2D Delaunay triangulaci. Jako vstupní data bylo nutno použít existující geodetická data (alespoň 300 bodů).

Vstupní hodnoty bylo potřeba vhodně vizualizovat v grafickém rozhraní s využitím QT frameworku.

Dále bylo nutné s využitím lineární interpolace vygenerovat vrstevnice s daným krokem a intervalem. Je potřeba nastavit jejich vykreslení s výrazněním hlavních vrstevnic. Poté analyzovat sklon a expozici jednotlivých trojúhelníků a tyto jevy vhodně vizualizovat.

2) *Zpracované bonusové úlohy:*

1. Výběr barevných stupnic při vizualizaci sklonu a expozice
2. Automatický popis vrstevnic
3. Barevná hysometrie

3) *Popis problému:*

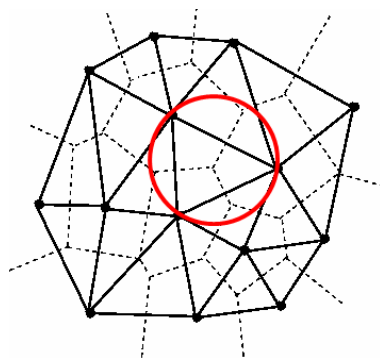
Delaunayho triangulace je efektivní metoda pro tvorbu kvalitní trojúhelníkové sítě (TIN) z množiny bodů v rovině. Hlavní výhodou Delaunayho triangulace je, že vytváří dobře tvarované trojúhelníky, které nejsou příliš protáhlé nebo úzké.

Algoritmus Delaunayho triangulace funguje následovně [1][2][3]:

- Vezme se množina bodů P v rovině, kde každý bod má souřadnici Z (výšku).
- Provede se triangulace této množiny bodů P . Výsledkem je síť trojúhelníků, která aproximuje terén definovaný body P .
- Důležitou vlastností Delaunayho triangulace je, že uvnitř kružnice opsané každému trojúhelníku neleží žádný jiný bod množiny P .
- Algoritmus lze realizovat inkrementální konstrukcí nebo metodou rozdělení a panuj (divide and conquer).
- Existuje také metoda lokálního prohazování hran, která převede libovolnou triangulaci na Delaunayho triangulaci.

Delaunayho triangulace má řadu užitečných vlastností [1][2]:

- Vytváří dobře tvarované trojúhelníky, což je výhodné pro interpolaci a aproximaci terénu.
- Je duální ke Thiessenovu diagramu (Voroného diagramu).
- Interpolací hodnot na hranách takové trojúhelníkové sítě lze získat průběh vrstevnic.

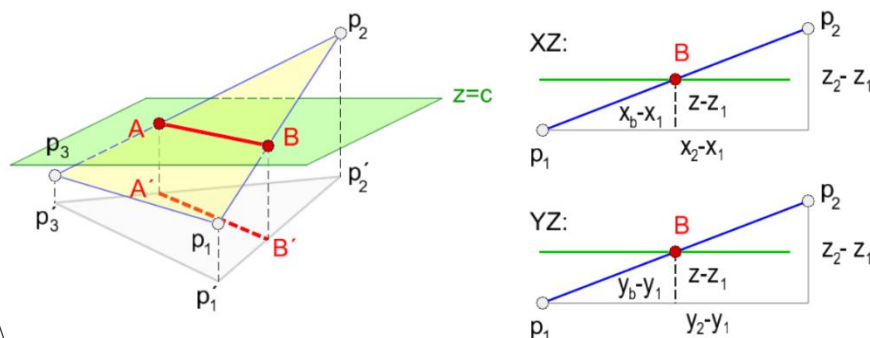


Obr.1: Delaunayho triangulace [2]

Po získání trojúhelníkové sítě lze v modelu vykreslit vrstevnice metodou lineární interpolace. V daném trojúhelníku je potřeba najít průsečnici s rovinou o určité výšce. Průsečíky s rovinou určíme jako:

$$x_a = \frac{x_3 - x_1}{z_3 - z_1}(z - z_1) + x_1, \quad x_b = \frac{x_2 - x_1}{z_2 - z_1}(z - z_1) + x_1,$$

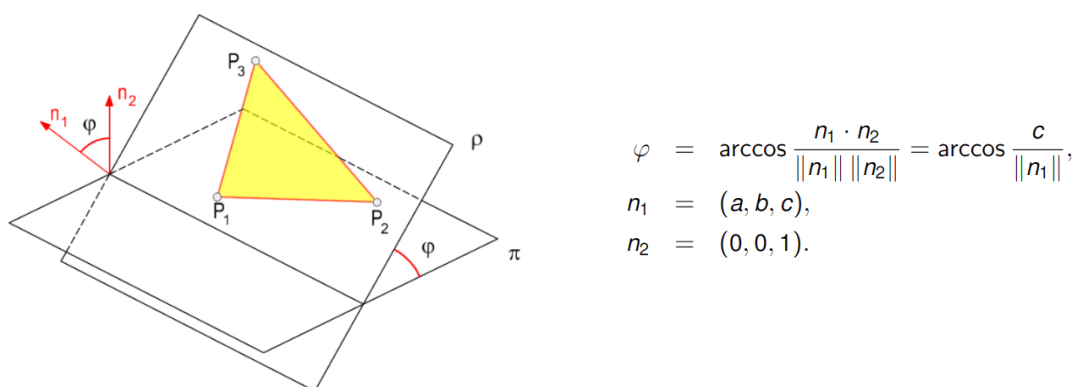
$$y_a = \frac{y_3 - y_1}{z_3 - z_1}(z - z_1) + y_1, \quad y_b = \frac{y_2 - y_1}{z_2 - z_1}(z - z_1) + y_1.$$



Obr. 2: Vztahy pro nalezení vrstevnice [3]

Sklon (Slope) je ukazatelem toho, jak se mění výška reliéfu ve směru jeho největšího spádu. Sklon je prostředkem, kterým gravitace řídí pohyb vody a jiných materiálů. Je tak jednou z nejvýznamnějších vlastností reliéfu ovlivňujících hydrologii a geomorfologii. [4]

Jeho hodnota je získávána v jednotlivých trojúhelnících jako úhel mezi normálovým vektorem vodorovné roviny a normálovým vektorem trojúhelníku.



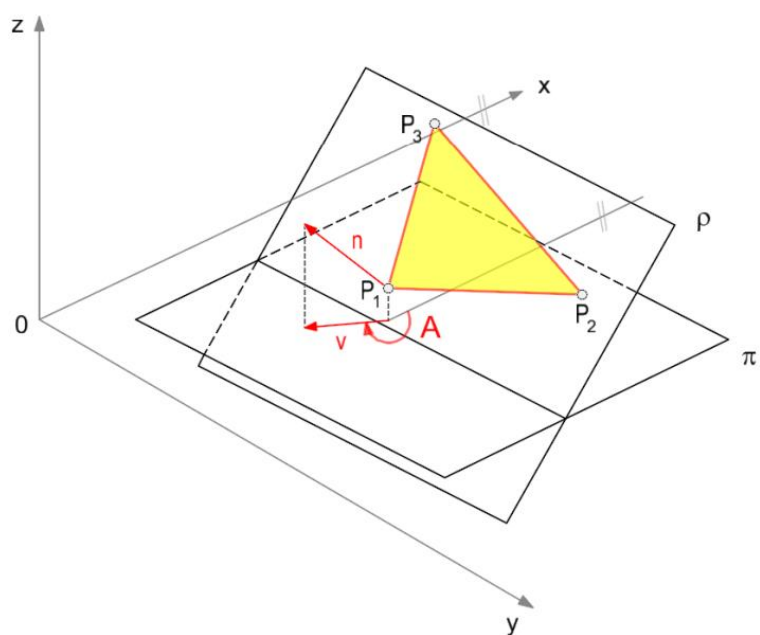
Obr. 3: Vztah pro nalezení hodnoty sklonu [3]

Expozice (Aspect) charakterizuje orientaci svahu ke světovým stranám. Jedná se o orientaci svahu podle jeho největšího spádu a je obvykle měřen od severu ve směru hodinových ručiček. Orientace svahů má především význam pro určování množství dopadajícího slunečního záření. Je také využíván pro vizualizaci reliéfu. Význam orientace svahu je větší v oblastech s větší svažitostí, v rovinných oblastech jeho význam upadá. [4]

Počítá se jako úhel od severu ve směru hodinových ručiček k průmětu normálového vektoru daného trojúhelníka.

$$v = \left(\frac{\partial \rho}{\partial x}(x_0), \frac{\partial \rho}{\partial y}(y_0), 0 \right) = (a, b, 0)$$

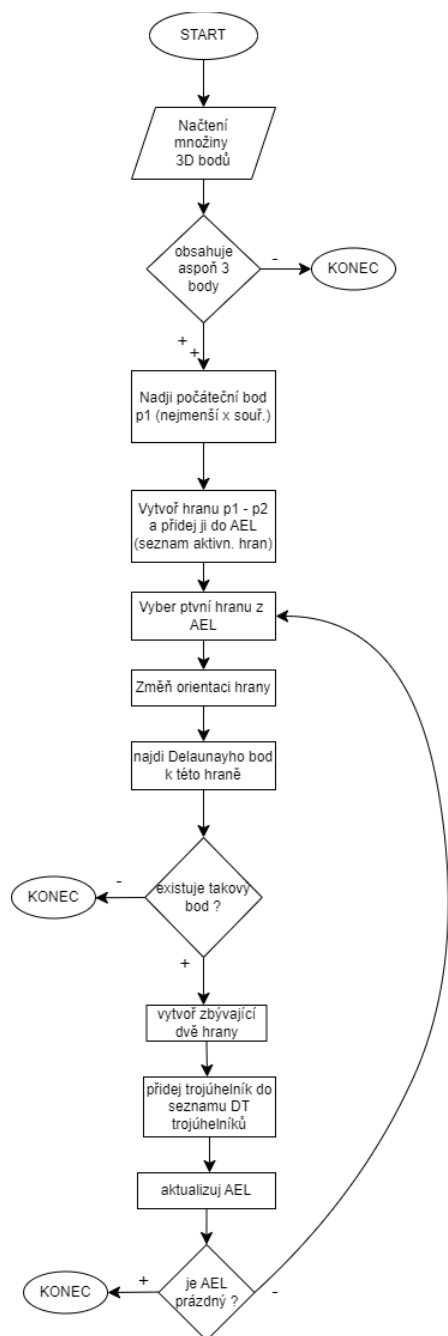
$$A = \arctan \left(\frac{a}{b} \right)$$



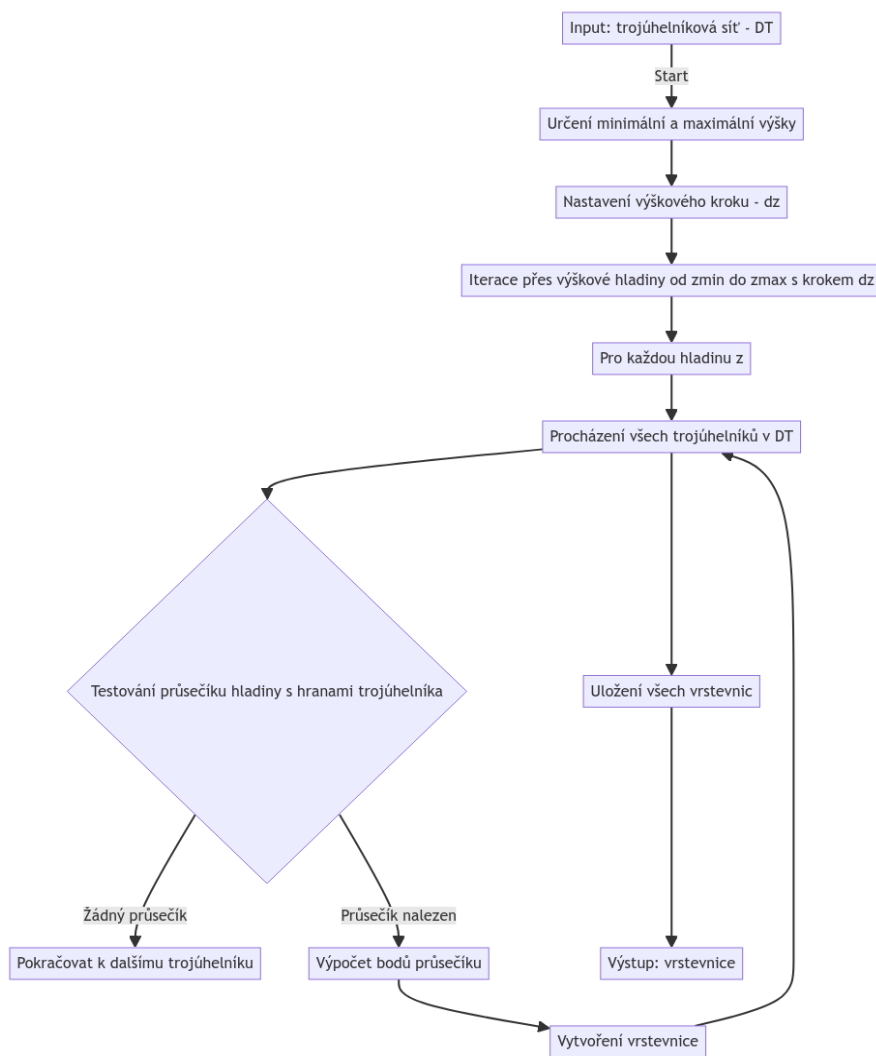
Obr. 4: Vztah pro nalezení hodnoty expozice [3]

4) Popis algoritmů formálním jazykem:

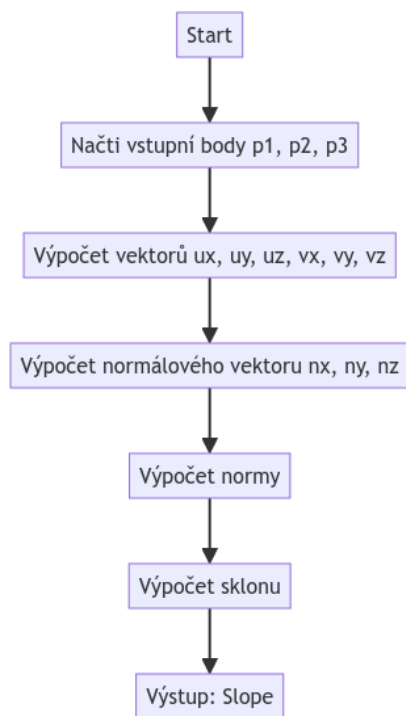
Delaunayho triangulace



Tvorba vrstevnice



Výpočet sklonu:



Výpočet expozice:



5) Problematické situace při řešení:

Výběr barevných stupnic při vizualizaci sklonu a expozice

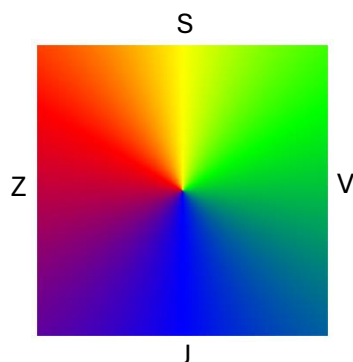
Sklon je vizualizován ve stupních šedi, kde bílá barva reprezentuje rovinu a černá maximální sklon. Toho je docíleno pomocí normalizace hodnot sklonu do rozsahu 0 až 255 a přidání této hodnoty stejně do všech složek RGB.



Obr.4: Barevná stupnice reprezentující sklon

Expozice je reprezentována RGB barvami, kdy sklon na západ je reprezentován barvou červenou, na sever žlutou, na východ zelenou a na jih modrou. Hodnota sklonu je reprezentována hodnotami 0-1 od jihu směrem na východ.

- Pokud je sklon mezi hodnotou 0 a 1/4 – barevný přechod od modré po zelenou
- Pokud je sklon mezi hodnotou 1/4 a 1/2 – barevný přechod od zelené po žlutou
- Pokud je sklon mezi hodnotou 1/2 a 3/4 - barevný přechod od žluté po červenou
- Pokud je sklon mezi hodnotou 3/4 a 1 - barevný přechod od červené po modrou



Obr.5: Barevná stupnice reprezentující expozici

Automatický popis vrstevnic

Každý pátý interpolovaný bod na trojúhelníkové síti je popsán jeho hodnotou Z. V malých sítích tak mohou vzniknout vrstevnice bez popisu. Tento popis je odsazen o 5 px, aby byl čitelný.

Barevná hypsometrie

Nejprve se zjišťují minimální a maximální hodnoty nadmořských výšek všech trojúhelníků. Poté se pro každý trojúhelník získají hodnoty z nadmořské výšky pro jeho vrcholy a vypočte se průměrná hodnota nadmořské výšky pro trojúhelník. Tato průměrná hodnota nadmořské výšky se normalizuje do rozsahu 0 až 1 podle minimální a maximální hodnoty.

$$\bar{z} = \frac{z_0 + z_{min}}{z_{max} - z_{min}}$$

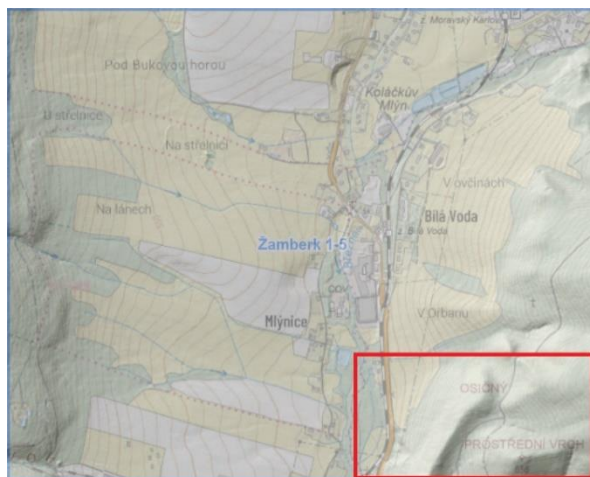
- Pokud je hodnota normalizovaného sklonu mezi 0–1/3 – barevný přechod od černé po zelenou.
- Pokud je hodnota normalizovaného sklonu mezi 1/3–2/3 – barevný přechod od zelené po žlutou.
- Pokud je hodnota normalizovaného sklonu mezi 2/3–1 – barevný přechod od žluté po červenou.



Obr.6: Barevná stupnice reprezentující barevnou hypsometrii

6) Vstupní data:

Vstupní data jsou pořízena z dlaždice DMR5G (Žamberk 1-5) vydávaného ČÚZK. Ta byla převedena z formátu LAZ do vektorových bodů. Z nich byla vybrána jen zájmová oblast (jihovýchodní část). V této oblasti byly body vyfiltrovány, tak aby jejich počet byl blízko tisíci (1078).

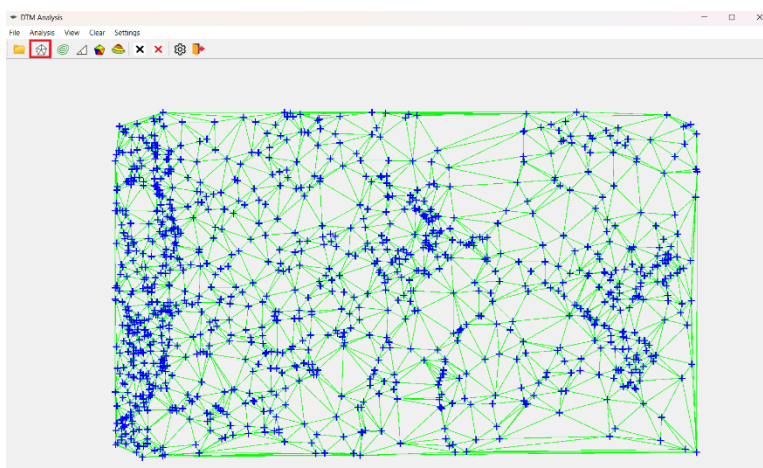


Obr.3 : Znázornění zájmové oblasti na dané dlaždici DMR5G

7) Ukázka aplikace:

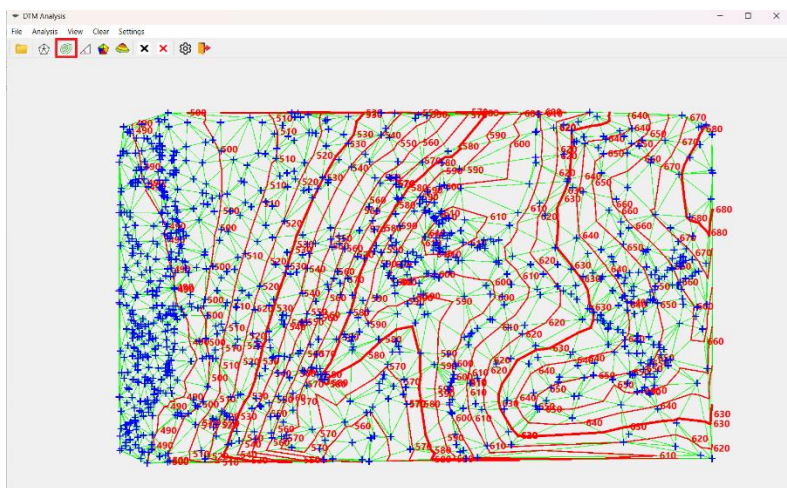
Aplikace obsahuje v hlavní části vykreslovací okno, do kterého lze zadávat polohy bodů po stisknutí tlačítka myši s náhodnou souřadnicí Z. Rozsah generování této hodnoty s rozestupem vrstevnic lze upravit ve vyskakovacím okně s nastavením. Do vykreslovací plochy lze nahrát soubor s 3D body ve formátu shapefile (.shp). Tato akce vymaže předchozí vstupy. V záložce „view“ lze nastavit, která vizualizace analýzy se bude zobrazovat.

První tlačítko zleva otevře dialogové okno pro výběr souboru, a to druhé vytvoří trojúhelníkovou síť pomocí Delaunayho triangulace.



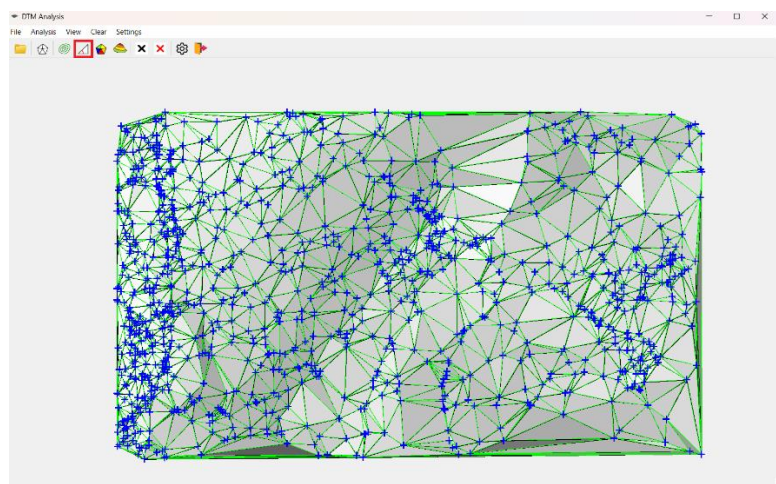
Obr. 7: Výsledek Delaunayho triangulace

Třetí tlačítko vyhotoví vrstevnice s popisy a každou pátou vrstevnicí dvojnásobě tučnou.



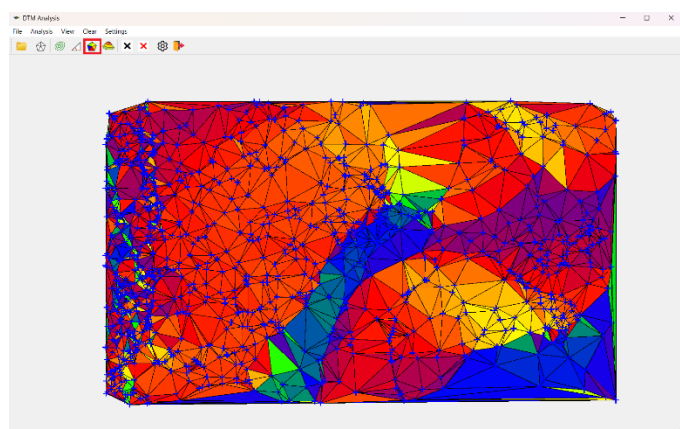
Obr. 8: Výsledek tvorby vrstevnic

Čtvrté tlačítko spustí analýzu sklonu.



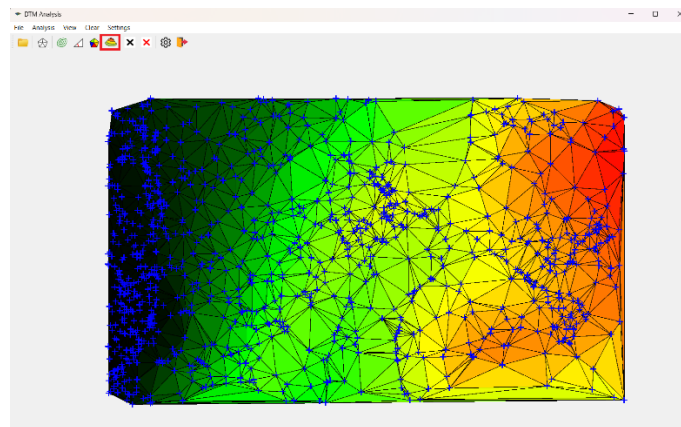
Obr. 9: Výsledek analýzy sklonu

Páté tlačítko spustí analýzu expozice.



Obr. 10: Výsledek analýzy expozice

Šesté tlačítko spustí tvorbu barevné hypsometrie.



Obr. 11: Výsledek barevné hypsometrie

Sedmé tlačítko vymaže výsledky, osmé tlačítko smaže výsledky i vložené body. Deváté tlačítko spustí otevře již zmíněné vyskakovací okno s nastavením a poslední tlačítko ukončí aplikaci.

8) Závěr:

Aplikace úspěšně tvoří polyedrický digitální model terénu (DMT) na základě množiny 3D bodů. Proces zahrnuje tvorbu 2D Delaunay triangulace pomocí inkrementální konstrukce, vizualizaci sklonu a expozice trojúhelníků a generování vrstevnic pomocí lineární interpolace. Výstupy byly efektivně zobrazeny v grafickém rozhraní využívajícím QT framework.

Bonusové úlohy, včetně výběru barevných stupnic pro vizualizaci sklonu a expozice, automatického popisu vrstevnic a barevné hypsometrie, byly úspěšně realizovány.

Za nedostatek považuji, že při zvoleném způsobu popisu vrstevnic mohou některé krátké vrstevnic zůstat nepopsány.

9) Přílohy:

- GitHub repozitář - https://github.com/jehlijos/ADKI_2024_Jehlicka_Predota
- Dokumentace zdrojového kódu

..

10) Seznam literatury:

- [1] JONES, Chris B. Geographical Information Systems and Computer Cartography. PEARSON Education, 1996. ISBN 978-0582044395.
- [2] JEDLIČKA, Karel. Nepravidelná trojúhelníková síť: způsob reprezentace povrchu. Vysokoškolská prezentace. Západočeská univerzita v Plzni.
- [3] BAYER, Tomáš. Rovinné triangulace a jejich využití: Greedy Triangulation. Delaunay Triangulation. Constrained Delaunay Triangulation. Data Dependent Triangulation. DMT. Vysokoškolská prezentace. Katedra aplikované geoinformatiky a kartografie. Přírodovědecká fakulta UK
- [4] PENÍŽEK, Vít; ZÁDOROVÁ, Tereza; KODEŠOVÁ, Radka a KLEMENT, Aleš. Optimalizace vzorkovací sítě pomocí využití analýzy reliéfu pro popis prostorové variability půdních vlastností v rámci půdních bloků, CERTIFIKOVANÁ METODIKA. Česká zemědělská univerzita v Praze. 2014, s. 41. ISBN 978-80-213-2533-3.

DOKUMENTACE ZDROJOVÉHO KÓDU

Třída **Ui MainForm** (MainForm.py):

- Metoda **setUpUi**

Metoda **setUpUi** je zodpovědná za nastavení uživatelského rozhraní hlavního okna aplikace PyQt6. Tato metoda je vygenerována pomocí nástroje Qt Designer a nastavuje různé prvky uživatelského rozhraní, včetně menu, toolbaru a hlavního widgetu.. Je vygenerována v sw. QtDesigner.

```
def setUpUi(self, MainWindow):
    MainWindow.setObjectName("MainWindow")
    MainWindow.resize(1111, 1015)
    MainWindow.setWindowIcon(QtGui.QIcon("images/icons/applogo.ico"))
    ...
    QtCore.QMetaObject.connectSlotsByName(MainWindow)
```

- Nastavuje název objektu, velikost a ikonu hlavního okna aplikace.
- Přidává centrální widget (Canvas) a horizontální rozvržení do hlavního okna.
- Vytváří menu bar, status bar a tool bar s různými akcemi, jako je otevření souboru, ukončení aplikace, vytváření Delaunay triangulace, analýza svahu, apod.
- Načítá ikony a propojuje vytvořená tlačítka s odpovídajícími metodami.

- Metoda **openClick**

Metoda **openClick** je zodpovědná za zpracování události, kdy je spuštěna akce "Otevřít". Načítá data ze souboru pomocí třídy IO, nastavuje plátno pro zobrazení načtených bodů a aktualizuje plátno.

```
def openClick(self):
    try:
        # Instantiate the IO class
        io = IO()

        # Load data from file
        width = self.Canvas.width()
        height = self.Canvas.height()
        points = io.loadData(width, height)

        # Set points on the canvas
        self.Canvas.SetPoints(points)

        # Repaint the canvas
        self.Canvas.repaint()
    except:
        pass
```

- Vytváří instanci třídy **IO**.
- Načítá data ze souboru pomocí metody **loadData**.
- Nastavuje načtené body na plátno a aktualizuje plátno.

..

- Metoda createDTClick

Metoda createDTClick je zodpovědná za vytvoření a zobrazení Delaunay triangulace na plátně.

```
def createDTClick(self):
    # Set flag and view button to true
    self.actionDT.setChecked(True)
    self.Canvas.draw_dt = True

    # Get input data
    points = self.Canvas.getPoints()

    # Run DT
    a = Algorithms()
    dt = a.createDT(points)

    # Set results
    self.Canvas.setDT(dt)

    # Repaint
    self.Canvas.repaint()
    o   Nastavuje příznak pro zobrazení Delaunay triangulace na True.
    o   Získává vstupní data (body) z plátna.
    o   Vytváří Delaunay triangulaci pomocí třídy Algorithms a nastavuje výsledky na plátno.
    o   Aktualizuje plátno.
```

- Metoda createContourLinesClick

Metoda createContourLinesClick je zodpovědná za vytvoření a zobrazení vrstevnic na plátně.

```
def createContourLinesClick(self):
    # Set flag and view button to true
    self.actionContour_lines_2.setChecked(True)
    self.Canvas.draw_contours = True
    self.actionDT.setChecked(True)
    self.Canvas.draw_dt = True

    a = Algorithms()

    # No DT
    if len(self.Canvas.getDT()) == 0:
        points = self.Canvas.getPoints()
        dt = a.createDT(points)
        self.Canvas.setDT(dt)

    # Get DT
    else:
        dt = self.Canvas.getDT()

    # Create contour lines
    with open("settings.conf", "r") as file:
        lines = file.readlines()
        zmin = int(round(float((lines[0]))))
        zmax = int(round(float((lines[1]))))
        dz = int(round(float((lines[2]))))

    contours = a.CreateCountourLines(dt, zmin, zmax, dz)

    # Set result
    self.Canvas.setContours(contours)

    # Repaint
    self.Canvas.repaint()
```

- Nastavuje příznaky pro zobrazení vrstevnic a Delaunay triangulace na **True**.
 - Načítá hodnoty z konfiguračního souboru **settings.conf** a vytváří vrstevnice pomocí třídy **Algorithms**.
 - Nastavuje vytvořené vrstevnice na plátno a aktualizuje plátno.
- Metoda `analyzeSlopeClick`
Metoda `analyzeSlopeClick` je zodpovědná za analýzu sklonu digitálního modelu terénu (DTM) a zobrazení výsledků na plátně.

```
def analyzeSlopeClick(self):
    # Set flag and view button to true
    self.actionSlope.setChecked(True)
    self.Canvas.draw_slope = True
    self.actionDT.setChecked(True)
    self.Canvas.draw_dt = True
    self.Canvas.draw_aspect = False
    self.actionExposition.setChecked(False)
    self.actionHypso.setChecked(False)
    self.Canvas.draw_hypso = False

    a = Algorithms()

    # No DT
    if len(self.Canvas.getDT()) == 0:
        points = self.Canvas.getPoints()
        dt = a.createDT(points)
        self.Canvas.setDT(dt)

    # Get DT
    else:
        dt = self.Canvas.getDT()

    # Analyze slope
    triangles = a.analyzeDTMSlopeAndAspect(dt)

    # Set result
    self.Canvas.setTriangles(triangles)

    # Repaint
    self.Canvas.repaint()
```

- Nastavuje příznaky pro zobrazení sklonu a Delaunay triangulace na **True**, ostatní příznaky na **False**.
 - Získává nebo vytváří Delaunay triangulaci a analyzuje sklon pomocí třídy **Algorithms**.
 - Nastavuje analyzované trojúhelníky na plátno a aktualizuje plátno.
- Metoda `analyzeExpositionClick`
Metoda `analyzeExpositionClick` je zodpovědná za analýzu expozice (orientace) digitálního modelu terénu (DTM) a zobrazení výsledků na plátně.

```
def analyzeExpositionClick(self):
    # Set flag and view button to true
    self.actionExposition.setChecked(True)
    self.Canvas.draw_aspect = True
    self.actionSlope.setChecked(True)
    self.Canvas.draw_slope = True
    self.actionHypso.setChecked(False)
    self.Canvas.draw_hypso = False
```

```

a = Algorithms()

# No DT
if len(self.Canvas.getDT()) == 0:
    points = self.Canvas.getPoints()
    dt = a.createDT(points)
    self.Canvas.setDT(dt)

# Get DT
else:
    dt = self.Canvas.getDT()

# Analyze aspect
triangles = a.analyzeDTMSlopeAndAspect(dt)

# Set result
self.Canvas.setTriangles(triangles)

# Repaint
self.Canvas.repaint()

```

- Nastavuje příznaky pro zobrazení expozice a sklonu na **True**, ostatní příznaky na **False**.
- Získává nebo vytváří Delaunay triangulaci a analyzuje expozici pomocí třídy **Algorithms**.
- Nastavuje analyzované trojúhelníky na plátno a aktualizuje plátno.

- Metoda settingsClick

Metoda settingsClick je zodpovědná za otevření dialogového okna s nastavením aplikace.

```

def settingsClick(self):
    # Create a new instance of QDialog
    self.dialog = QtWidgets.QDialog()

    # Create a new instance of Ui_Dialog
    self.ui = Ui_Dialog()

    # Setup the dialog
    self.ui.setupUi(self.dialog)

    # Show the dialog
    self.dialog.show()

```

- Vytváří novou instanci dialogového okna QDialog.
- Nastavuje uživatelské rozhraní dialogového okna pomocí třídy **Ui_Dialog**.
- Zobrazuje dialogové okno.

- Metoda retranslateUi

Metoda retranslateUi je zodpovědná za nastavení textových popisků a titulků v uživatelském rozhraní. Je vygenerována v sw. QtDesigner.

Třída **Algorithms** (algorithms.py):

- Metoda `getPointPolPosition`

Metoda `getPointPolPosition` určuje pozici bodu vůči polygonu pomocí algoritmu ray crossing.

```
# Point and polygon position, ray crossing algorithm
k = 0
n = len(pol)

# Process all vertices of the polygon
for i in range(n):
    # Reduce coordinates
    x_ir = pol[i].x() - q.x()
    y_ir = pol[i].y() - q.y()

    x_ilr = pol[(i + 1) % n].x() - q.x()
    y_ilr = pol[(i + 1) % n].y() - q.y()

    # Appropriate segment intersection the ray
    if ((y_ilr > 0) and (y_ir <= 0)) or ((y_ir > 0) and (y_ilr <= 0)):

        # Compute intersection coordinate
        xm = (x_ilr * y_ir - x_ir * y_ilr) / (y_ilr - y_ir)

        # Appropriate intersection, increment k
        if xm > 0:
            k = k + 1

# Inside
if k % 2 == 1:
    return 1

# Outside
return 0
```

- Prochází všechny vrcholy polygonu.
- Určuje počet průsečíků paprsku s hranami polygonu.
- Vrací 1, pokud je bod uvnitř polygonu, jinak 0.

- Metoda `getSlope`

Metoda `getSlope` vypočítává sklon trojúhelníku.

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

    # Vectors
    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 - uz * vy
    ny = -ux * vz + uz * vx
    nz = ux * vy - uy * vx

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

    # Slope
    return acos(fabs(nz) / norm)
```


- Vytváří vektory z bodů trojúhelníku.
 - Vypočítává normálový vektor.
 - Vrací sklon trojúhelníku v radiánech.
- Metoda `get2VectorsAngle`
Metoda `get2VectorsAngle` vypočítává úhel mezi dvěma vektory.

```
def get2VectorsAngle(self, p1: QPoint3DF, p2: QPoint3DF, p3: QPoint3DF,
p4: QPoint3DF):
    # Angle between two vectors
    ux = p2.x() - p1.x()
    uy = p2.y() - p1.y()

    vx = p4.x() - p3.x()
    vy = p4.y() - p3.y()

    # dot product
    dot = ux * vx + uy * vy

    # Norms
    nu = (ux ** 2 + uy ** 2) ** 0.5
    nv = (vx ** 2 + vy ** 2) ** 0.5

    # Correct interval
    arg = dot / (nu * nv)
    arg = max(-1, min(1, arg))

    return acos(arg)
```

- Vypočítává skalární součin a normy dvou vektorů.
 - Vrací úhel mezi dvěma vektory v radiánech.
- Metoda `createCH`
Metoda `createCH` vytváří konvexní obal pomocí Jarvis Scan algoritmu.

```
def createCH(self, pol: QPolygonF):
    # create Convex Hull using Jarvis Scan
    ch = QPolygonF()

    # Find pivot q (minimize y)
    q = min(pol, key=lambda k: k.y())

    # Find left-most point (minimize x)
    s = min(pol, key=lambda k: k.x())

    # Initial segment
    pj = q
    pj1 = QPoint3DF(s.x(), q.y())

    # Add to CH
    ch.append(pj)

    # Find all points of CH
    while True:
        # Maximum and its index
        omega_max = 0
        index_max = -1

        # Browse all points
```

```

for i in range(len(pol)):

    if pj != pol[i]:

        # Compute omega
        omega = self.get2VectorsAngle(pj, pj1, pj, pol[i])

        # Actualize maximum
        if (omega > omega_max):
            omega_max = omega
            index_max = i

    # Add point to the convex hull
    ch.append(pol[index_max])

    # Reassign points
    pj1 = pj
    pj = pol[index_max]

    # Stopping condition
    if pj == q:
        break

return ch

```

- Inicializuje prázdný QPolygonF pro konvexní obálku (ch).
- Najde pivotní bod q s minimální y-souřadnicí.
- Najde nejlevější bod s s minimální x-souřadnicí.
- Iterativně vybírá body, které tvoří vnější hranici, výpočtem úhlu omega a aktualizací maximálního úhlu.
- Přidává vybrané body do konvexní obálky, dokud nebude dosažen výchozí bod q.

- **Metoda createMMB**

Metoda createMMB vytváří minimální obdélníkový obal a počítá jeho plochu.

```

def createMMB(self, pol: QPolygonF):
    # Create min max box and compute its area

    # Points with extreme coordinates
    p_xmin = min(pol, key=lambda k: k.x())
    p_xmax = max(pol, key=lambda k: k.x())
    p_ymin = min(pol, key=lambda k: k.y())
    p_ymax = max(pol, key=lambda k: k.y())

    # Create vertices
    v1 = QPoint3DF(p_xmin.x(), p_ymin.y())
    v2 = QPoint3DF(p_xmax.x(), p_ymin.y())
    v3 = QPoint3DF(p_xmax.x(), p_ymax.y())
    v4 = QPoint3DF(p_xmin.x(), p_ymax.y())

    # Create new polygon
    mmb = QPolygonF([v1, v2, v3, v4])

    # Area of MMB
    area = (v2.x() - v1.x()) * (v3.y() - v2.y())

    return mmb, area

```

- Najde body s extrémními souřadnicemi.
- Vytváří vrcholy minimálního obdélníkového obalu.
- Vrací obdélníkový obal a jeho plochu.

- **Metoda LH**

Metoda LH počítá plochu polygonu pomocí L'Huillierových vzorců

```
# Compute polygon area using LH formula
area = 0
n = len(pol)

# Compute area
for i in range(n):

    area = area + pol[i].x() * (pol[(i + 1) % n].y() -
                                pol[(i - 1 + n) % n].y())

return abs(area) / 2
```

- Výpočet spočívá v rozdělení plochy na soustavu lichoběžníků a během vlastního výpočtu pak dochází ke sčítání a odčítání ploch těchto lichoběžníků.

- **Metoda rotatePolygon**

Metoda rotatePolygon otáčí polygon podle zadaného úhlu.

```
def rotatePolygon(self, pol: QPolygonF, sig: float):
    # Rotate polygon according to a given angle
    pol_rot = QPolygonF()

    # Process all polygon vertices
    for i in range(len(pol)):
        # Rotate point
        x_rot = pol[i].x() * cos(sig) - pol[i].y() * sin(sig)
        y_rot = pol[i].x() * sin(sig) + pol[i].y() * cos(sig)

        # Create QPoint
        vertex = QPoint3DF(x_rot, y_rot)

        # Add vertex to rotated polygon
        pol_rot.append(vertex)

    return pol_rot
```

- Otočí všechny vrcholy polygonu podle zadaného úhlu.
- Vrací otočený polygon.

- **Metoda createMBR**

Metoda createMBR vytváří minimální obalový obdélník s minimální plochou.

```
def createMBR(self, pol: QPolygonF):
    # Create minimum area enclosing rectangle

    # Create convex hull
    ch = self.createCH(pol)

    # Get min-max box, area and sigma
    mmb_min, area_min = self.createMMB(ch)
    sigma_min = 0
```

```

# Process all segments of ch
for i in range(len(ch) - 1):

    # Compute sigma
    dx = ch[i + 1].x() - ch[i].x()
    dy = ch[i + 1].y() - ch[i].y()
    sigma = atan2(dy, dx)

    # Rotate convex hull by sigma
    ch_rot = self.rotatePolygon(ch, -sigma)

    # Find min-max box over rotated convex hull
    mmb, area = self.createMMB(ch_rot)

    # Actualize minimum area
    if area < area_min:
        area_min = area
        mmb_min = mmb
        sigma_min = sigma

# Rotate min-max box
er = self.rotatePolygon(mmb_min, sigma_min)

# Resize rectangle
er_r = self.resizeRectangle(er, pol)

return er_r

```

- Vytvoří konvexní obal polygonu.
- Iterativně otáčí konvexní obal a hledá minimální obalový obdélník.
- Vrací minimální obalový obdélník.

- **Metoda resizeRectangle**

Metoda resizeRectangle mění velikost obdélníku podle plochy polygonu.

```

def resizeRectangle(self, er: QPolygonF, pol: QPolygonF):
    # Building area
    Ab = abs(self.LH(pol))

    # Enclosing rectangle area
    A = abs(self.LH(er))

    # Fraction of Ab and A
    k = Ab / A

    # Center of mass
    x_t = (er[0].x() + er[1].x() + er[2].x() + er[3].x()) / 4
    y_t = (er[0].y() + er[1].y() + er[2].y() + er[3].y()) / 4

    # Vectors
    u1_x = er[0].x() - x_t
    u2_x = er[1].x() - x_t
    u3_x = er[2].x() - x_t
    u4_x = er[3].x() - x_t
    u1_y = er[0].y() - y_t
    u2_y = er[1].y() - y_t
    u3_y = er[2].y() - y_t
    u4_y = er[3].y() - y_t

    # Coordinates of new vertices
    v1_x = x_t + sqrt(k) * u1_x
    v1_y = y_t + sqrt(k) * u1_y

```

```

v2_x = x_t + sqrt(k) * u2_x
v2_y = y_t + sqrt(k) * u2_y

v3_x = x_t + sqrt(k) * u3_x
v3_y = y_t + sqrt(k) * u3_y

v4_x = x_t + sqrt(k) * u4_x
v4_y = y_t + sqrt(k) * u4_y

# Create new vertices
v1 = QPoint3DF(v1_x, v1_y)
v2 = QPoint3DF(v2_x, v2_y)
v3 = QPoint3DF(v3_x, v3_y)
v4 = QPoint3DF(v4_x, v4_y)

# Create rectangle
er_r = QPolygonF([v1, v2, v3, v4])

return er_r

```

- Mění velikost obdélníku podle poměru ploch polygonu a obdélníku.
- Vrací upravený obdélník.

- **Metoda createERPCA**

Metoda createERPCA vytváří obalující obdélník pomocí PCA (Principal Component Analysis).

```

def createERPCA(self, pol: QPolygonF):
    # Create enclosing rectangle using PCA
    x = []
    y = []

    # Add x,y coordinates to the list
    for p in pol:
        x.append(p.x())
        y.append(p.y())

    # Invert to matrix
    A = array([x, y])

    # Covariance matrix
    C = cov(A)

    # Angular value decomposition
    [U, S, V] = svd(C)

    # Compute sigma
    sigma = atan2(V[0][1], V[0][0])

    # Rotate polygon
    pol_rot = self.rotatePolygon(pol, -sigma)

    # Find min-max box over rotated building
    mmb, area = self.createMMB(pol_rot)

    # Rotate min-max box
    er = self.rotatePolygon(mmb, sigma)

    # Resize rectangle
    er_r = self.resizeRectangle(er, pol)

    return er_

```

- Nejprve se z polygonu pol získají všechny x a y souřadnice a uloží se do seznamů x a y.
- seznamy souřadnic se převádějí na dvourozměrnou matici A.
- Vypočítá se kovarianční matice C z matice A.
- Proveďte se SVD (Singular Value Decomposition) na kovarianční matici C, což nám poskytne matice U, S a V.
- Úhel sigma se vypočítá jako arcustangens z prvků matice V.
- Polygon pol se otočí o úhel -sigma.
- Najde se minimálně-maximum obalující obdélník (min-max box) pro otočený polygon.
- Min-max box se otočí zpět o úhel sigma.
- Obdélník se přizpůsobí původnímu polygonu pol.
- Upravený obdélník se vrátí jako výsledek.

- Metoda `getNearestPoint`

Metoda `getNearestPoint` vrací bod nejbližší k danému bodu q.

```
def getNearestPoint(self, q: QPoint3DF, points: list[QPoint3DF]):
    # Return point nearest to q

    d_min = inf
    i_min = -1

    # Process all points of the cloud
    for i in range(len(points)):

        # q different from points[i]
        if q != points[i]:
            # Compute distance
            dx = q.x() - points[i].x()
            dy = q.y() - points[i].y()

            d = sqrt(dx ** 2 + dy ** 2)

            # Update minimum
            if d < d_min:
                d_min = d
                i_min = i

    return d_min, i_min
```

- Proměnné `d_min` a `i_min` jsou inicializovány na nekonečno a -1.
- Prochází se všechny body v seznamu `points`.
- Kontroluje se, jestli bod q není stejný jako aktuální bod v iteraci.
- Vzdálenost mezi body q a aktuálním bodem se vypočítá pomocí Pythagorovy věty.
- Pokud je vypočtená vzdálenost menší než `d_min`, aktualizují se hodnoty `d_min` a `i_min`.
- Po iteraci se vrací minimální vzdálenost a index nejbližšího bodu.

- Metoda `getPointAndLinePosition`

Metoda `getPointAndLinePosition` analyzuje pozici bodu vůči linii.

```
def getPointAndLinePosition(self, p: QPoint3DF, p1: QPoint3DF, p2: QPoint3DF):
    # Analyze point and line position

    # Compute vectors u, v
    ux = p2.x() - p1.x()
    uy = p2.y() - p1.y()

    vx = p.x() - p1.x()
    vy = p.y() - p1.y()
```

```

# Compute test
t = ux * vy - uy * vx

# Point if the left half plane
if t > 0:
    return 1

# Point if the right half plane
if t < 0:
    return 0

# Point on the line
return -1

```

- Vypočítají se dva vektory u a v , kde u je vektor z p_1 do p_2 a v je vektor z p_1 do p .
- Hodnota t se vypočítá jako determinant matice složené z vektorů u a v .
- Na základě hodnoty t se určí pozice bodu p vůči linii. Pokud je t kladné, bod je vlevo; pokud je záporné, bod je vpravo; pokud je nula, bod je na linii.

- **Metoda getDelaunayPoint**

Metoda getDelaunayPoint vrací Delaunayův bod.

```

def getDelaunayPoint(self, start: QPoint3DF, end: QPoint3DF, points:
                                list[QPoint3DF]):
    # Return Delaunay point

    omega_max = 0
    i_max = -1

    # Process all points of the cloud
    for i in range(len(points)):

        # Start and end different from points[i]
        if start != points[i] and end != points[i]:

            # Point in left half-plaine
            if self.getPointAndLinePosition(points[i], start, end) == 1:

                # Compute angle
                omega = self.get2VectorsAngle(points[i], start,
                                                points[i], end)

                # Update maximum
                if omega > omega_max:
                    omega_max = omega
                    i_max = i

    return omega_max, i_max

```

- Proměnné ω_{\max} a i_{\max} jsou inicializovány na 0 a -1.
- Prochází se všechny body v seznamu `points`.
- Kontroluje se, jestli body `start` a `end` nejsou stejné jako aktuální bod v iteraci.
- Pokud je aktuální bod vlevo od linie `start-end`, vypočítá se úhel mezi vektory.
- Pokud je vypočtený úhel větší než ω_{\max} , aktualizují se hodnoty ω_{\max} a i_{\max} .
- Po iteraci se vrací maximální úhel a index bodu tvořícího tento úhel.

- Metoda createDT
Metoda createDT generuje Delaunayho triangulaci z daného seznamu bodů. Začíná tím, že identifikuje počáteční bod a jeho nejbližšího souseda, aby vytvořila první hranu. Poté metoda iterativně přidává hrany do seznamu aktivních hran (AEL) a konstruuje trojúhelníky tím, že hledá optimální body Delaunayovy triangulace, dokud v AEL nezůstanou žádné další hrany.

```
def createDT(self, points: list[QPoint3DF]):

    if len(points) < 3:
        return []

    # Create Delaunay triangulation
    dt = []
    ael = []

    # Find initial point
    p1 = min(points, key=lambda k: k.x())

    # Find nearest point
    d_min, idx = self.getNearestPoint(p1, points)
    p2 = points[idx]

    # Create edge and opposite edge
    e = Edge(p1, p2)
    e_op = Edge(p2, p1)

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

    # Repeat until ael is empty
    while ael:
        # Take the 1st edge
        e1 = ael.pop()

        # Switch orientation
        e1_op = e1.changeOrientation()

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

        # Is there any Delaunay point
        if idx >= 0:
            # Create remaining edges
            e2 = Edge(e1_op.getEnd(), points[idx])
            e3 = Edge(points[idx], e1_op.getStart())

            # Add triangle to dt
            dt.append(e1_op)
            dt.append(e2)
            dt.append(e3)

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

    return dt
```


- Zkontroluje, zda je počet bodů menší než 3; pokud ano, vrátí prázdný seznam.
- Identifikuje počáteční bod (p1) s nejmenší x-ovou souřadnicí a najde jeho nejbližšího souseda (p2).
- Vytvoří počáteční hranu (e) a její opačnou hranu (e_op), poté je přidá do seznamu aktivních hran (AEL).
- Dokud není AEL prázdný, vyndá hranu, změní její orientaci a najde optimální bod Delaunayovy triangulace.
- Vytvoří nové hrany pro vytvoření trojúhelníku, přidá je do seznamu triangulace (dt) a aktualizuje AEL.

- **Metoda updateAEL**

Metoda updateAEL aktualizuje seznam aktivních hran (AEL) buď přidáním nebo odebráním hrany na základě její orientace. Pokud hrana s opačnou orientací existuje v AEL, odejme ji; jinak ji přidá do seznamu.

```
def updateAEL(self, e: Edge, ael: list[Edge]):
    # Update list of valid Delaunay edges

    # Change orientation
    e_op = e.changeOrientation()

    # Is edge in AEL?
    # Yes, remove edge
    if e_op in ael:
        ael.remove(e_op)

    # No, add to the list
    else:
        ael.append(e)
```

- Metoda přijímá hranu e a seznam aktivních hran AEL.
- Změní orientaci hrany e a získá e_op.
- Zkontroluje, zda je e_op v AEL.
- Pokud je e_op nalezeno, odstraní e_op z AEL.
- Pokud e_op není nalezeno, přidá původní hranu e do AEL.

- **Metoda getContourPoint**

Metoda getContourPoint vypočítá průsečík úsečky definované dvěma 3D body (p1 a p2) s horizontální rovinou na dané z-ové souřadnici. Vrátí tento průsečík jako nový objekt **QPoint3DF**.

```
def getContourPoint(self, p1: QPoint3DF, p2: QPoint3DF, z: float):
    # Compute intersection point

    xb = (p2.x() - p1.x()) / (p2.getZ() - p1.getZ()) * (z - p1.getZ()) +
        p1.x()
    yb = (p2.y() - p1.y()) / (p2.getZ() - p1.getZ()) * (z - p1.getZ()) +
        p1.y()

    return QPoint3DF(xb, yb, z)
```

- Vypočítá x-ovou a y-ovou souřadnici (xb, yb) průsečíku pomocí lineární interpolace mezi body p1 a p2 na základě dané hodnoty z.
- Vrátí objekt **QPoint3DF** s xb, yb a vloženého z

- Metoda Create CountourLines

Metoda CreateCountourLines generuje vrstevnice pro daný soubor trojúhelníků (Delaunayho triangulace) v určeném rozsahu hodnot z a kroku. Identifikuje průsečíky mezi horizontálními rovinami na různých rovinách z a hranami trojúhelníků, vytvářející izolinie tam, kde se tyto průsečíky objevují.

```
def CreateCountourLines(self, dt: list[Edge], zmin: float, zmax: float,
dz: float):
    # Create contour lines inside interval zmin, zmax and with step dz
    contours = []

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

        # Z of points
        z1 = p1.getZ()
        z2 = p2.getZ()
        z3 = p3.getZ()

        # Test horizontal plane and triangle intersections
        for z in range(zmin, zmax, dz):
            # mpute height differences
            dz1 = z - z1
            dz2 = z - z2
            dz3 = z - z3

            # Triangle is coplanar
            if dz1 == 0 and dz2 == 0 and dz3 == 0:
                continue

            # Edge (1, 2) in plane
            elif dz1 == 0 and dz2 == 0:
                contours.append(dt[i])

            # Edge (2, 3) in plane
            elif dz2 == 0 and dz3 == 0:
                contours.append(dt[i + 1])

            # Edge (3, 1) in plane
            elif dz3 == 0 and dz1 == 0:
                contours.append(dt[i + 2])

            # Edges (1,2) & (2,3) intersected by plane
            elif dz1 * dz2 <= 0 and dz2 * dz3 < 0 or dz1 * dz2 < 0 and
                dz2 * dz3 <= 0:

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

                # Create new edge/line
                e = Edge(a, b)

                # Add edge to the list
                contours.append(e)

            # Edges (2,3) & (3,1) intersected by plane
            elif dz2 * dz3 <= 0 and dz3 * dz1 < 0 or dz2 * dz3 < 0 and
                dz3 * dz1 <= 0:

                # Contour intersections
                a = self.getContourPoint(p2, p3, z)
```

```

        b = self.getContourPoint(p3, p1, z)

        # Create new edge/line
        e = Edge(a, b)

        # Add edge to the list
        contours.append(e)

    # Edges (3,1) & (1,2) intersected by plane
    elif dz3 * dz1 <= 0 and dz1 * dz2 < 0 or dz3 * dz1 < 0 and
        dz1 * dz2 <= 0:

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

        # Create new edge/line
        e = Edge(a, b)

        # Add edge to the list
        contours.append(e)

    return contours

```

- Inicializuje prázdný seznam contours pro uložení výsledných kót.
- Prochází seznam hran po krocích 3, aby zpracoval každý trojúhelník.
- Pro každý trojúhelník získá z-hodnoty jeho vrcholů.
- Pro každou z-hodnotu v určeném rozsahu zkontroluje průsečíky mezi horizontální rovinou a hranami trojúhelníka.
- Pokud jsou nalezeny průsečíky, vytvoří nové hrany představující vrstevnice a přidá je do seznamu contours.

- Metoda getAspect

Metoda getAspect vypočítá expozici trojúhelníku tvořeného třemi body v 3D prostoru. Expozice je určena výpočtem normálového vektoru k rovině trojúhelníku a následným nalezením úhlu mezi normálovým vektorem a y-ovou osou pomocí funkce atan2.

```

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

    # Vectors
    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 - uz * vy
    ny = -ux * vz + uz * vx

    # Aspect
    return atan2(nx, ny)

```

- Vypočtete vektory u a v od p1 k p2 a od p3 k p2.
- Vypočtete normálový vektor n k rovině trojúhelníku pomocí vektorového součinu u a v.
- Vypočtete hodnotu expozice nalezením úhlu mezi normálovým vektorem a y-ovou osou pomocí funkce atan2.

- **Metoda analyzeDTMSlopeAndAspect**
Metoda analyzeDTMSlopeAndAspect zpracovává seznam hran Delaunayovy triangulace k výpočtu sklonu a expozice pro každý vytvořený trojúhelník. Vrací seznam objektů **Triangle** obsahujících tyto vypočtené hodnoty.

```
def analyzeDTMSlopeAndAspect(self, dt):
    # Analyze DTM slope and aspect
    triangles = []

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

        # Compute slope
        slope = self.getSlope(p1, p2, p3)

        # Compute aspect
        aspect = self.getAspect(p1, p2, p3)

        # Create triangle
        triangle = Triangle(p1, p2, p3, slope, aspect)

        # Add triangle to list
        triangles.append(triangle)

    return triangles
```

- Inicializuje prázdný seznam triangles.
- Prochází seznam dt po krocích 3, aby zpracoval každý trojúhelník.
- Extrahuje vrcholy každého trojúhelníku.
- Vypočte sklon a expozici pro každý trojúhelník.
- Vytvoří objekt **Triangle** s vrcholy, sklonem a expozicí.
- Připojí objekt **Triangle** do seznamu triangles.

Třída IO (inpout.py)

Třída, která obsahuje metody pro načítání a zpracování Shapefile souborů.

- **Metoda loadGeometries**
Metoda pro načítání 3D bodových geometrií ze Shapefile. Iteruje přes každý záznam ve Shapefile a vytváří objekty **QPoint3DF** pro každý bod.

```
def loadGeometries(self, fileName):
    # Method to load 3D point geometries from a Shapefile
    points = []
    # Opening the Shapefile
    with fiona_open(fileName) as shapefile:
        # Iterating through each record in the Shapefile
        for record in shapefile:
            geom = shape(record['geometry'])
            if isinstance(geom, Point):
                x, y, z = geom.x, geom.y, geom.z if geom.has_z
                else 0
                points.append(QPoint3DF(x, y, z))
    # Returning list of 3D points
    return points
```

- Inicializuje prázdný seznam points.
 - Otevře Shapefile soubor pomocí fiona_open.
 - Iteruje každý záznam v Shapefile souboru.
 - Převeďte geometrii každého záznamu na Shapely bod.
 - Extrahuje souřadnice x, y a z z bodu a připojí je jako objekty **QPoint3DF** do seznamu points.
 - Vráť seznam objektů **QPoint3DF**.

- Metoda createPoints

Metoda pro vytváření PyQt 3D bodů z bodů Shapely. Iteruje přes každý bod a vytváří objekty **QPoint3DF**.

```
def createPoints(self, points):
    # Method to create PyQt 3D points from Shapely points
    qpoints = []
    # Iterating through each point
    for pt in points:
        qpoint = QPoint3DF(pt.x(), pt.y() * (-1), pt.getZ())
        qpoints.append(qpoint)
    return qpoints
```

- Inicializuje prázdný seznam qpoints.
 - Iteruje přes každý bod ve vstupním seznamu points.
 - Pro každý bod vytvoří objekt **QPoint3DF** se souřadnicí x, převrácenou souřadnicí y a souřadnicí z.
 - Přidá vytvořený objekt **QPoint3DF** do seznamu qpoints.
 - Vráť seznam qpoints.
 - Metoda scaleAndTranslatePoints
- Metoda scaleAndTranslatePoints zmenší a posune seznam 3D bodů (objektů **QPoint3DF**) na základě zadaného měřítka a hodnot posunu souřadnic x a y.

```
def scaleAndTranslatePoints(self, points, s, shift_x, shift_y):
    # Method to scale and translate points
    scaled_translated_points = [QPoint3DF(point.x() * s - shift_x,
                                           point.y() * s - shift_y, point.getZ())
                                for point in points]
    return scaled_translated_points
```

- Iteruje přes každý bod v seznamu bodů.
 - Pro každý bod změní měřítko souřadnic x a y koeficientem s.
 - Od zmenšených souřadnic x a y odečte hodnoty posunu shift_x a shift_y.
 - Vytvoří se nový objekt **QPoint3DF** se zmenšenými a posunutými souřadnicemi a původní souřadnicí z.
 - Nový objekt **QPoint3DF** přidá do seznamu scaled_translated_points.
 - Metoda processPointCoordinates
- Metoda processPointCoordinates získá souřadnice x, y a z ze seznamu objektů **QPoint3DF** a vrátí je jako tři samostatné seznamy.

```
def processPointCoordinates(self, points):
    # Method to process coordinates of points
    x_crds = [point.x() for point in points]
    y_crds = [point.y() for point in points]
    z_crds = [point.getZ() for point in points]
    return x_crds, y_crds, z_crds
```

- Inicializuje tři seznamy pro uložení souřadnic x, y a z.
 - Iteruje přes každý objekt **QPoint3DF** v seznamu points.
 - Získá souřadnice x, y a z z každého bodu.
 - Vrábí tři seznamy obsahující souřadnice x, y a z.
- Metoda loadData
Metoda loadData načte 3D body ze Shapefile souboru, zpracuje jejich souřadnice, upraví měřítko a posun tak, aby se vešly do zadané velikosti okna, a vrátí transformované body (**QPoint3DF**).

```
def loadData(self, w, h):
    # Method to load Shapefile after selecting it
    if self.dia.exec():
        selected_files = self.dia.selectedFiles()
        if not selected_files:
            # For user closing the dialog without selecting any file
            exit()

        # Initializing extreme values
        x_min = float('inf')
        y_min = float('inf')
        x_max = float('-inf')
        y_max = float('-inf')

        # Loading point geometries from selected Shapefile
        points = self.loadGeometries(selected_files[0])
        # Creating QPoint3DF from loaded geometries
        qpoints = self.createPoints(points)

        all_x, all_y, all_z = self.processPointCoordinates(qpoints)

        # Updating extreme coordinates of points if necessary
        x_min = min(all_x)
        y_min = min(all_y)
        x_max = max(all_x)
        y_max = max(all_y)

        # Calculating height and width of bounding box
        H = y_max - y_min
        W = x_max - x_min

        # Calculating height ratio and width ratio
        ratio_h = h / H
        ratio_w = w / W

        mean_x = mean(all_x)
        mean_y = mean(all_y)

        # Calculating scaling factor
        scale = min(ratio_w, ratio_h) * 0.9

        center_X = mean_x * scale
        center_Y = mean_y * scale

        # Calculating center of the window
        center_x = w / 2
        center_y = h / 2

        # Calculating shift in X-direction and Y-direction
        shift_x = center_X - center_x
        shift_y = center_Y - center_y

        # Scaling and translating points
```

```

        Data = self.scaleAndTranslatePoints(qpoints, scale, shift_x,
shift_y)

        return Data
    else:
        pass

```

- Otevře dialogové okno pro výběr Shapefile souboru.
- Načte 3D body z vybraného Shapefile souboru.
- Zpracuje souřadnice bodů a vyhledá extrémní hodnoty.
- Vypočítá měřítko a posuny, aby se body vešly do zadané velikosti okna.
- Změní měřítko a posun bodů a vrátí transformované body.

Třída **Draw** (draw.py):

Třída vykresluje body, vrstevnice a analýzy terénu.

- Metoda `mousePressEvent`

Metoda `mousePressEvent` zpracovává kliknutí myši ve vykreslovacím okně. Zachytí polohu kurzoru, načte konfigurační hodnoty z-ových souřadnic, vygeneruje náhodnou z-ovou souřadnici v zadaném rozsahu, vytvoří nový 3D bod, přidá jej do mračna bodů a spustí překreslení obrazovky.

```

def mousePressEvent(self, e: QMouseEvent):
    # Block using in Pane class in TD.py

    # Get cursor position
    x = e.position().x()
    y = e.position().y()

    # Get zmin and zmax values from the configuration file
    with open("settings.conf", "r") as file:
        # Reading the configuration file
        lines = file.readlines()
        # Extracting the values of zmin, zmax
        zmin = int(round(float(lines[0].strip())))
        zmax = int(round(float(lines[1].strip())))

    z = random() * (zmax - zmin) + zmin

    # Create new point
    p = QPoint3DF(x, y, z)

    # Add point to point cloud
    self.points.append(p)

    # Repaint screen
    self.repaint()

```

- Zachytí polohu kurzoru z myši.
 - Přečte hodnoty `zmin` a `zmax` z konfiguračního souboru.
 - Vygeneruje náhodnou z-ovou souřadnici v rozsahu `zmin` a `zmax`.
 - Vytvoří nový objekt **QPoint3DF** se získanými hodnotami `x`, `y` a vygenerovanou hodnotou `z`.
 - Přidá nový bod do seznamu bodů a překreslí widget kreslicího plátna.
- Metoda `paintEvent`
- Metoda `paintEvent` je zodpovědná za vykreslení různých grafických prvků na widgetu. Vykresluje barevnou hypsometrii, sklon, expozici, Delaunayovu triangulaci, vrstevnice a body na základě aktuálního stavu instance třídy **Draw**.

```

def paintEvent(self, e: QPaintEvent):
    # Draw situation
    qp = QPainter(self)
    qp.begin(self)

    # Draw hypsometric tints
    if self.draw_hypso and self.triangles:

        # Get zmin and zmax values from vertices for all triangles
        zmin = min(t.getMinZ() for t in self.triangles)
        zmax = max(t.getMaxZ() for t in self.triangles)

        # Paint lowest triangle green and highest brown
        for t in self.triangles:
            # Get the Z values of the triangle vertices
            z1, z2, z3 = t.getZValues()
            meanZ = (z1 + z2 + z3) / 3

            # Normalize the Z value from zmin to zmax to a range of 0 to
1            normalizedZ = (meanZ - zmin) / (zmax - zmin)

            # Calculate RGB values based on the normalized Z value
            # Using colors scheme from dark green to red
            #
            if normalizedZ < 1 / 3:
                r = 0
                g = int(255 * (normalizedZ * 3))
                b = 0

            #
            elif normalizedZ < 2 / 3:
                r = int(255 * (normalizedZ - 1 / 3) * 3)
                g = 255
                b = 0
            else:
                r = 255
                g = int(255 * (1 - (normalizedZ - 2 / 3) * 3))
                b = 0

            # Create a QColor from the calculated RGB values
            col = QColor(r, g, b)

            # Create a QPolygonF from the triangle vertices
            polygon = t.getVertices()

            # Set the brush to the calculated color
            qp.setBrush(col)

            # Draw the polygon
            qp.drawPolygon(polygon)

    if self.draw_slope:
        # Draw triangles: slope
        for t in self.triangles:
            vertices = t.getVertices()
            slope = t.getSlope()
            RGB = int(255 - slope * 2 * 255 / pi)
            col = QColor(RGB, RGB, RGB)
            qp.setBrush(col)
            qp.drawPolygon(vertices)

    if self.draw_aspect:
        # Draw triangles: aspect

```



```

for t in self.triangles:
    vertices = t.getVertices()
    aspect = t.getAspect()

    # Normalize the aspect value from -pi to +pi to a range of 0
    # to 1
    normalized_aspect = (aspect + pi) / (2 * pi)

    # Calculate RGB values based on the normalized aspect value
    if 0 <= normalized_aspect < 1 / 4: # Blue to Green
        r = 0
        g = int(255 * (normalized_aspect * 4))
        b = int(255 * (1 - normalized_aspect * 4))
    elif 1 / 4 <= normalized_aspect < 1 / 2: # Green to Yellow
        r = int(255 * (normalized_aspect - 1 / 4) * 4)
        g = 255
        b = 0
    elif 1 / 2 <= normalized_aspect < 3 / 4: # Yellow to Red
        r = 255
        g = int(255 * (1 - (normalized_aspect - 1 / 2) * 4))
        b = 0
    else: # Red to Blue
        r = int(255 * (1 - (normalized_aspect - 3 / 4) * 4))
        g = 0
        b = int(255 * ((normalized_aspect - 3 / 4) * 4))

    # Create a QColor from the calculated RGB values
    col = QColor(r, g, b)

    # Set the brush to the calculated color
    qp.setBrush(col)

    # Draw the polygon with the specified vertices
    qp.drawPolygon(vertices)

# Draw Delaunay triangulation
if self.draw_dt:
    qp.setPen(QPen(Qt.GlobalColor.green))
    for e in self.dt:
        qp.drawLine(int(e.getStart().x()), int(e.getStart().y()),
                    int(e.getEnd().x()), int(e.getEnd().y()))

# Draw contour lines
qp.setPen(QPen(Qt.GlobalColor.red, 2))

if self.draw_contours:
    # Draw contour lines
    # Get unique Z values and sort them
    unique_z_values = sorted(set(e.getEnd().getZ() for e in
                                self.contours))

    # Create a dictionary to map Z values to indices
    z_value_to_index = {z: idx for idx, z in
                        enumerate(unique_z_values)}

    # Counter to keep track of the number of contour segments drawn
    text_counter = 0

    for e in self.contours:
        z_value = e.getEnd().getZ()
        if (z_value_to_index[z_value] + 1) % 5 == 0:
            qp.setPen(QPen(Qt.GlobalColor.red, 4))
            # Twice as wide for every 5th unique Z value
        else:
            qp.setPen(QPen(Qt.GlobalColor.red, 2))

```

```

qp.drawLine(int(e.getStart().x()), int(e.getStart().y()),
            int(e.getEnd().x()), int(e.getEnd().y()))

# Set font size and draw Z value only for every 5th contour
segment

text_counter += 1
if text_counter % 5 == 0:
    font = qp.font()
    font.setPointSize(12)
    font.setBold(True)
    qp.setFont(font)
    qp.drawText(int(e.getEnd().x() + 5), int(e.getEnd().y() +
                                           5), str(z_value))

# Draw points as crosses
qp.setPen(QPen(Qt.GlobalColor.blue, 2))
length = 5
# if p is NoneType, pass
if self.points:
    for p in self.points:
        qp.drawLine(int(p.x()), int(p.y() - length), int(p.x()),
                    int(p.y() + length))
        qp.drawLine(int(p.x() - length), int(p.y()), int(p.x() +
                    length), int(p.y()))
else:
    # Make p a empty list and not NoneType
    self.points = []

qp.end()

```

- Inicializuje objekt QPainter, který se stará o kreslení.
- Vykreslí barevnou hypsometrii, pokud je povoleno **draw_hypso** a jsou k dispozici trojúhelníky.
- Vykreslí stínování svahu, pokud je povoleno **draw_slope**.
- Vykreslí stínování expozice, pokud je povoleno **draw_aspect**.
- Vykreslí Delaunayovy trojúhelníky, pokud je povoleno **draw_dt**.
- Vykreslí vrstevnice, pokud je povolena funkce **draw_contours**.
- Vykreslí body jako křížky.

- Metoda **getPoints**

Metoda **getPoints** je jednoduchá funkce, která vrací seznam bodů aktuálně uložených v třídě **Draw**.

```

def getPoints(self):
    # Return points
    return self.points

```

- Metoda **clearAll**

Metoda **clearAll** vymaže veškeré vykreslené objekty na plátně, včetně bodů, Delaunayovy triangulace, vrstevnic a trojúhelníků, a poté překreslí obrazovku.

```

def clearAll(self):
    # Clear points
    self.points.clear()

    # Clear triangles
    self.dt.clear()

    # Clear DT
    self.dt.clear()

    # Clear contour lines

```

```

self.contours.clear()

# Clear triangles
self.triangles.clear()

# Repaint screen
self.repaint()

```

- Vymaže seznam bodů, Delaunayovu triangulaci, seznam vrstevnic a seznam trojúhelníků (s hodnotami sklonu a expozice)
- Překreslí obrazovku.

- **Metoda clearResults**

Metoda clearResults vymaže výsledky vykreslování, včetně Delaunayovy triangulace, DT, vrstevnic a trojúhelníků, a poté překreslí obrazovku. Natozdíl od funkce **clearAll** zachová vykreslené body.

```

def clearResults(self):
    # Clear triangles
    self.dt.clear()

    # Clear DT
    self.dt.clear()

    # Clear contour lines
    self.contours.clear()

    # Clear triangles
    self.triangles.clear()

    # Repaint screen
    self.repaint()

```

- Vymaže Delaunayovu triangulaci, seznam vrstevnic a seznam trojúhelníků (s hodnotami sklonu a expozice)
- Překreslí obrazovku.

- **Metoda setDT**

Metoda setDT nastaví Delaunayovu triangulaci na zadaný seznam hran dt.

```

def setDT(self, dt: list[Edge]):
    # Set DT
    self.dt = dt

```

- **Metoda getDT**

Metoda getDT vrátí aktuální Delaunayovu triangulaci (DT).

```

def getDT(self):
    # Get DT
    return self.dt

```

- **Metoda setTriangles**

Nastaví trojúhelníky na zadaný seznam triangles.

```

def setTriangles(self, triangles: list[Triangle]):
    # Set triangles
    self.triangles = triangles

```

- Metoda `setDrawDT`
Nastaví, zda se má vykreslovat Delaunayova triangulace podle hodnoty `draw_dt`.

```
def setDrawDT(self, draw_dt):
    # Set draw DT
    self.draw_dt = draw_dt
```

- Metoda `setPoints`
Nastaví body na zadaný seznam `points`.

```
def setPoints(self, points):
    # Set points
    self.points = points
```

- Metoda `setDrawContours`
Nastaví, zda se mají vykreslovat vrstevnice podle hodnoty `draw_contours`.

```
def setDrawContours(self, draw_contours):
    # Set draw contour lines
    self.draw_contours = draw_contours
```

- Metoda `setDrawSlope`
Nastaví, zda se má vykreslovat sklon podle hodnoty `draw_slope`.

```
def setDrawSlope(self, draw_slope):
    # Set draw slope
    self.draw_slope = draw_slope
```

- Metoda `setDrawAspect`
Nastaví, zda se má vykreslovat expozice podle hodnoty `draw_aspect`.

```
def setDrawAspect(self, draw_aspect):
    # Set draw aspect
    self.draw_aspect = draw_aspect
```

- Metoda `getContours`
Metoda `getContours` vrátí aktuální vrstevnice.

```
def getContours(self):
    # Get contour lines
    return self.contours
```

Třída `Ui_Dialog` (`Settings.py`)

Třída `Ui_Dialog` je zodpovědná za tvorbu dialogového okna v aplikaci PyQt6. Načte počáteční nastavení z konfiguračního souboru, zobrazí je v dialogovém okně a umožní uživateli tato nastavení upravit a uložit. Metoda je generována v sw. Qt Designer.

- Metoda `setupUi`
Metoda `setupUi` nastaví uživatelské rozhraní dialogového okna, načte konfigurační hodnoty a inicializuje grafické prvky.

```
def setupUi(self, Dialog):
    self.Dialog = Dialog # Store the Dialog instance as an attribute
    with open("settings.conf", "r") as file:
        # Reading the configuration file
        lines = file.readlines()
        # Extracting the values of zmin, zmax, and dz
        self.zmin = int(round(float((lines[0]))))
```

```

        self.zmax = int(round(float((lines[1]))))
        self.dz = int(round(float((lines[2]))))

        Dialog.setObjectName("Dialog")
        Dialog.resize(421, 309)
        Dialog.setWindowIcon(QtGui.QIcon("images/icons/settings.png"))
        . . .
self.retranslateUi(Dialog)
    self.Settings.accepted.connect(self.saveSettings) # type: ignore
    self.Settings.rejected.connect(Dialog.reject) # type: ignore
    QtCore.QMetaObject.connectSlotsByName(Dialog)

```

- Načte konfigurační hodnoty zmin, zmax, a dz ze souboru settings.conf.
- Nastaví vlastnosti dialogového okna a jeho grafických prvků, včetně tlačítek a vstupních polí.
- Připojí signály k odpovídajícím tlačítkům.

- **Metoda retranslateUi**

Metoda retranslateUi je zodpovědná za nastavení textových popisků a titulků v uživatelském rozhraní. Je vygenerována v sw. QtDesigner.

- **Metoda getZmin**

Získá hodnotu z textového pole lineEdit a vrátí ji jako minimální výšku vrstevnice.

```

def getZmin(self):
    # Get the minimum contour line height
    self.zmin = float(self.lineEdit.text())
    return self.zmin

```

- **Metoda getZmax**

Získá hodnotu z textového pole lineEdit_2 a vrátí ji jako maximální výšku vrstevnice.

```

def getZmax(self):
    # Get the maximum contour line height
    self.zmax = float(self.lineEdit_2.text())
    return self.zmax

```

- **Metoda getDz**

Získá hodnotu z textového pole lineEdit_3 a vrátí ji jako interval výšky vrstevnice.

```

def getDz(self):
    # Get the contour line height interval
    self.dz = float(self.lineEdit_3.text())
    return self.dz

```

- **Metoda saveSettings**

Metoda saveSettings uloží nová nastavení do konfiguračního souboru settings.conf a uzavře dialogové okno.

```

def saveSettings(self):
    # Get the new settings
    zmin = self.getZmin()
    zmax = self.getZmax()
    dz = self.getDz()

    # Write the new settings to the settings.conf file
    with open("settings.conf", "w") as file:
        file.write(f"{zmin}\n{zmax}\n{dz}\n")

    # Close the dialog
    self.Dialog.accept()

```

- Získá nové hodnoty zmin, zmax, a dz z textových polí.
- Zapiše nové hodnoty do konfiguračního souboru settings.conf.
- Uzavře dialogové okno přijetím dialogu.

Třída **QPoint3DF** (QPoint3DF.py)

Třída QPoint3DF rozšiřuje třídu QPointF o třetí rozměr z.

- Metoda getZ
Metoda getZ vrátí z-tovou souřadnici 3D bodu

```
def getZ(self):
    # Method to get the z-coordinate of the point
    return self.z
```

Třída **Edge** (Edge.py)

Třída Edge představuje úsečku ve 3D prostoru definovanou dvěma body (začátek a konec) typu **QPoint3DF**.

- Metoda getStart
Metoda getStart vrátí počáteční bod linie (**QPoint3DF**)

```
def getStart(self):
    return self.start
```

- Metoda getEnd
Metoda getEnd vrátí koncový bod linie (**QPoint3DF**)

```
def getEnd(self):
    return self.end
```

- Metoda changeOrientation
Metoda changeOrientation vytvoří nový objekt Edge s prohozeným počátečním a koncovým bodem.

```
def changeOrientation(self):
    return Edge(self.end, self.start)
```

Třída **Triangle** (Triangle.py)

třída Triangle představuje 3D trojúhelník definovaný třemi vrcholy, z nichž každý je reprezentován objektem QPoint3DF. Obsahuje též hodnotu expozice a sklonu pro takový trojúhelník.

- Metoda getVertices
Metoda getVertices vrací 2D projekci vrcholů trojúhelníku jako objekt **QPolygonF**.

```
def getVertices(self):
    # Method to get the vertices of the triangle
    return QPolygonF([QPointF(p.x(), p.y()) for p in self.vertices])
```

- Metoda getSlope
Metoda getSlope vrací hodnotu sklonu objektu Triangle.

```
def getSlope(self):
    # Method to get the slope of the triangle
    return self.slope
```

- Metoda `getAspect`
Metoda `getAspect` vrací hodnotu expozice objektu `Triangle`.

```
def getAspect(self):  
    # Method to get the aspect of the triangle  
    return self.aspect
```

- Metoda `getZValues`
Metoda `getZValues` získá z-ové souřadnice všech vrcholů objektu `Triangle`.

```
def getZValues(self):  
    # Method to get the z-values of the vertices of the triangle  
    return [p.getZ() for p in self.vertices]
```

- Metoda `getMinZ`
Metoda `getMinZ` vrací minimální hodnotu z z vrcholů trojúhelníku.

```
def getMinZ(self):  
    # Method to get the minimum z-value of the triangle  
    return min(self.getZValues())
```

- Metoda `getMaxZ`
Metoda `getMaxZ` vrací maximální hodnotu z z vrcholů trojúhelníku.

```
def getMaxZ(self):  
    # Method to get the maximum z-value of the triangle  
    return max(self.getZValues())
```