

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

Název předmětu:

Algoritmy digitální kartografie a GIS

Úloha:

U3

Název úlohy:

Digitální model terénu a jeho analýzy

Akademický rok:

2024/2025

Semestr:

letní

Studijní skupina:

101C

Vypracoval:

Michal Kovář
Filip Roučka

Datum:

12. 5. 2025

Klasifikace:

1 Zadání

Zadáním této úlohy bylo metodou inkrementální konstrukce vytvořit digitální model terénu (DMT) nad množinou 3D bodů pomocí 2D Delaunay triangulace. Výstupem je polyedrický DMT reprezentovaný vrstevnicemi, doplněný o vizualizaci sklonu trojúhelníků a jejich expozice.

- **Vstup:** Množina bodů $P = \{p_1, p_2, \dots, p_n\}$, kde každý bod p_i je reprezentován třemi souřadnicemi $p_i = (x_i, y_i, z_i)$, kde x_i a y_i jsou prostorové souřadnice bodu v rovině a z_i je výška nad referenčním bodem.
- **Výstup:** Polyedrický digitální model terénu (DMT), reprezentovaný triangulacemi a vrstevnicemi, spolu s vizualizacemi sklonu a expozice terénu.
- Delaunay triangulace, polyedrický model terénu.
- Konstrukce vrstevnic, analýza sklonu a expozice.

2 Bonus

- *Automatický popis vrstevnic.*
- *Automatický popis vrstevnic respektující kartografické zásady.*
- *Algoritmus pro automatické generování terénních tvarů.*
- *3D vizualizace terénu s využitím promítání.*

3 Popis problému

Cílem úlohy je vytvořit digitální model terénu (DMT) nad množinou 3D bodů, kde výstupem bude polyedrický DMT reprezentovaný vrstevnicemi doplněný o vizualizace sklonu trojúhelníků a jejich expozice. Tento model bude vytvořen metodou inkrementální konstrukce pomocí 2D Delaunay triangulace. Součástí úkolu je také analýza terénu s využitím lineární interpolace pro generování vrstevnic, analýza sklonu a expozice jednotlivých trojúhelníků a jejich vizualizace.

3.1 Formulace problému

Dáno:

- Množina $P = \{p_1, p_2, \dots, p_n\}$ bodů, kde každý bod $p_i = (x_i, y_i, z_i)$ je definován prostorovými souřadnicemi x_i , y_i a výškou z_i .

Určováno:

- Polyedrický digitální model terénu (DMT) nad množinou bodů P , reprezentovaný triangulacemi a vrstevnicemi, spolu s vizualizacemi sklonu a expozice terénu.

3.2 Techniky řešení problému

- **2D Delaunay triangulace:** Hlavní metodou pro konstrukci digitálního modelu terénu je vytvoření 2D Delaunay triangulace nad množinou bodů. Tento algoritmus umožňuje získat triangulaci, která minimalizuje úhly mezi trojúhelníky a zabraňuje vytvoření ostrých úhlů, což je klíčové pro správnou reprezentaci terénu[1].
- **Generování vrstevnic:** Na základě triangulace se generují vrstevnice pomocí lineární interpolace. Tyto vrstevnice jsou zobrazeny v daném intervalu s příslušným krokem a zvýrazněním hlavních vrstevnic pro lepší čitelnost a analýzu terénu.
- **Analýza sklonu:** Po vytvoření triangulace se analyzuje sklon jednotlivých trojúhelníků. K tomu se používají geometrické vlastnosti trojúhelníků, jako je úhel mezi roviny a horizontem, což umožňuje určit sklon terénu v každé oblasti.

- **Analýza expozice:** Expozice terénu je určena na základě orientace trojúhelníků vůči světovým stranám. Tato analýza pomáhá identifikovat orientace svahů a zjistit, které oblasti terénu jsou více vystaveny slunečnímu záření, což má významné ekologické a klimatické důsledky.
- **Triangulace nekonvexní oblasti zadané polygonem:** Pokud jsou některé části terénu definovány nekonvexními oblastmi, je nutné aplikovat algoritmus pro triangulaci těchto oblastí, aby byl model terénu správně reprezentován i v složitějších geografických formacích.
- **Vizualizace terénu:** Terén bude vizualizován v 3D s využitím promítání a barevné hypsometrie, která zobrazuje výškové rozdíly pomocí barevného spektra. Dále budou použity barevné stupnice pro vizualizaci sklonu a expozice, což umožní intuitivní pochopení terénních tvarů.

4 Popis algoritmů

4.1 2D Delaunay triangulace

Delaunayho triangulace je algoritmus používaný k vytvoření trojúhelníkové sítě (triangulace) z množiny bodů v rovině tak, aby žádný bod neležel uvnitř kružnice opsané žádnému z trojúhelníků. Výsledná triangulace má vlastnost, že maximalizuje minimální úhel všech trojúhelníků, čímž se minimalizuje výskyt trojúhelníků s ostrými úhly.

Nejprve je nalezen bod p_1 který má minimální x souřadnici:

$$p_1 = \min(x_i) \quad (1)$$

Následně je vybrán bod p_2 který se nachází nejbližší bodu p_2 :

$$p_2 = \min(s_{p_1, p_2}) \quad (2)$$

Následně byla vytvořena hrana e_1 , která je tvořena body p_1, p_2 a hrana e_2 kterou tvoří stejné body, ale s opačnou orientací tudíž p_2, p_1 Hrany jsou přidány do pomocné konstrukce *AEI*

$$AEI.push_back(e_1, e_2) \quad (3)$$

Prochází se proměnou *AEI* dokud není prázdná. Vezme se první hrana a změní se její orientace. Prochází se pouze body, které leží vpravo od hrany. Dále se najde bod, který s počátečním a koncovým bodem hrany svírá největší úhel. Takovému bodu se říká Delaunayovský bod:

$$p_d = \max \angle(p_d, p_{end}, p_d, p_{start}) \quad (4)$$

Pokud se takový bod najde, všechny hrany jsou přidány do *DT*:

$$dt.push_back(e_1, e_{p_1, d}, e_{p_2, d}) \quad (5)$$

Nakonec je zkontrolováno zda se hrana s opačnou orientací nenachází v *AEI*. Pokud ano hrana s opačnou orientací je odstraněna z *AEI*, pokud ne hrana je přidána do *AEI*

Metoda DT – Delaunayova triangulace (inkrementálně)

```
1: Vstup: množina bodů  $P$ 
2: Výstup: množina hran Delaunayovy triangulace DT
3:  $DT \leftarrow \emptyset$  {výsledná triangulace}
4:  $AEL \leftarrow \emptyset$  {aktivní hrany (Active Edge List)}
5:  $p_1 \leftarrow$  nejlevější bod z  $P$  {pivot podle  $x$ -ové souřadnice}
6:  $p_2 \leftarrow$  nejbližší bod k  $p_1$  z  $P$ 
7:  $e_1 \leftarrow (p_1, p_2)$ 
8:  $e_2 \leftarrow (p_2, p_1)$ 
9: přidej  $e_1, e_2$  do AEL
10: while  $AEL \neq \emptyset$  do
11:    $e \leftarrow$  poslední prvek z AEL
12:   odstraň  $e$  z AEL
13:    $e_s \leftarrow$  změň orientaci hrany  $e$ 
14:    $p_d \leftarrow \text{findDelaunayPoint}(e_s.\text{start}, e_s.\text{end}, P)$ 
15:   if  $p_d \neq \text{null}$  then
16:      $e_{2s} \leftarrow (e_s.\text{end}, p_d)$ 
17:      $e_{3s} \leftarrow (p_d, e_s.\text{start})$ 
18:     přidej  $e_s, e_{2s}, e_{3s}$  do DT
19:     updateAEL( $e_{2s}$ , AEL)
20:     updateAEL( $e_{3s}$ , AEL)
21:   end if
22: end while
23: return DT
```

Metoda getPointAndLinePosition

```
1: Vstup: bod  $p$ , počáteční bod  $p_1$ , koncový bod  $p_2$ 
2: Výstup: pozice bodu  $p$  vůči orientované přímce  $(p_1, p_2)$ 
3: Návratové hodnoty:
   • 1 pokud je bod  $p$  vlevo od přímky
   • 0 pokud je bod  $p$  vpravo od přímky
   • -1 pokud leží na přímce

4:  $\varepsilon \leftarrow 10^{-6}$ 
5:  $u_x \leftarrow p_2.x - p_1.x$ 
6:  $u_y \leftarrow p_2.y - p_1.y$ 
7:  $v_x \leftarrow p.x - p_1.x$ 
8:  $v_y \leftarrow p.y - p_1.y$ 
9:  $t \leftarrow u_x \cdot v_y - u_y \cdot v_x$  {determinant pro orientaci}
10: if  $t > \varepsilon$  then
11:   return 1
12: else if  $t < -\varepsilon$  then
13:   return 0
14: else
15:   return -1
16: end if
```

Metoda get2LinesAngle

```
1: Vstup: body  $p_1, p_2$  definující první přímku a  $p_3, p_4$  definující druhou přímku
2: Výstup: úhel mezi přímkami v radiánech
3:  $u_x \leftarrow p_2.x - p_1.x$ 
4:  $u_y \leftarrow p_2.y - p_1.y$ 
5:  $v_x \leftarrow p_4.x - p_3.x$ 
6:  $v_y \leftarrow p_4.y - p_3.y$ 
7: skalární součin  $\leftarrow u_x \cdot v_x + u_y \cdot v_y$ 
8:  $n_u \leftarrow \sqrt{u_x^2 + u_y^2}$  {norma vektoru  $u$ }
9:  $n_v \leftarrow \sqrt{v_x^2 + v_y^2}$  {norma vektoru  $v$ }
10: return  $\arccos\left(\frac{\text{skalární součin}}{n_u \cdot n_v}\right)$ 
```

Metoda findDelaunayPoint

```
1: Vstup: body  $p_1, p_2$ , množina bodů  $P$ 
2: Výstup: index bodu  $p_d \in P$ , který tvoří největší úhel  $\angle p_1 p_d p_2$  a leží vlevo od přímky  $(p_1, p_2)$ 
3:  $i_{\max} \leftarrow -1$  {žádný bod zatím nebyl nalezen}
4:  $\omega_{\max} \leftarrow 0$ 
5: for  $i \leftarrow 0$  to  $\text{size}(P) - 1$  do
6:   if  $\text{getPointAndLinePosition}(P[i], p_1, p_2) = 1$  then
7:      $\omega \leftarrow \text{get2LinesAngle}(P[i], p_1, P[i], p_2)$ 
8:     if  $\omega > \omega_{\max}$  then
9:        $\omega_{\max} \leftarrow \omega$ 
10:       $i_{\max} \leftarrow i$ 
11:     end if
12:   end if
13: end for
14: return  $i_{\max}$ 
```

Metoda get2DDistance

```
1: Vstup: body  $p_1, p_2$ 
2: Výstup: 2D eukleidovská vzdálenost mezi body  $p_1$  a  $p_2$ 
3:  $dx \leftarrow p_1.x - p_2.x$ 
4:  $dy \leftarrow p_1.y - p_2.y$ 
5: return  $\sqrt{dx^2 + dy^2}$ 
```

Metoda findNearestPoint

```
1: Vstup: bod  $p$ , množina bodů  $P$ 
2: Výstup: index bodu  $p_{\min} \in P$ , který je nejbližší bodu  $p$ 
3:  $i_{\min} \leftarrow -1$ 
4:  $\varepsilon \leftarrow 10^{-16}$  {tolerance pro porovnání}
5:  $d_{\min} \leftarrow 10^{16}$  {počáteční nekonečná vzdálenost}
6: for  $i \leftarrow 0$  to  $\text{size}(P) - 1$  do
7:   if  $p \neq P[i]$  then
8:      $d \leftarrow \text{get2DDistance}(p, P[i])$ 
9:     if  $d < d_{\min}$  then
10:       $d_{\min} \leftarrow d$ 
11:       $i_{\min} \leftarrow i$ 
12:     end if
13:   end if
14: end for
15: return  $i_{\min}$ 
```

Metoda updateAEL

```
1: Vstup: hrana  $e$ , seznam aktivních hran AEL
2: Výstup: aktualizovaný seznam AEL
3:  $e_s \leftarrow \text{changeOrientation}(e)$  {změní orientaci hrany}
4:  $i_e \leftarrow \text{index hrany } e_s \text{ v AEL}$  {vyhledání opačné hrany}
5: if  $e_s$  není nalezena v AEL then
6:   přidej  $e$  do AEL {hrana je nová, trojúhelník vzniká poprvé}
7: else
8:   odstraň  $e_s$  z AEL {opačná hrana už existuje, trojúhelník je kompletní}
9: end if
```

4.2 Tvorba vrstevnic

Je algoritmus, který za použití lineární interpolace vytváří vrstevnice z Delaunayova triangulace.

$$x_a = \frac{x_3 - x_1}{z_3 - z_1}(z - z_1) + x_1; \quad y_a = \frac{y_3 - y_1}{z_3 - z_1}(z - z_1) + y_1 \quad (6)$$

Vrstevnice jsou reprezentovány pomocí vektoru hran. Při výpočtu se pracuje na intervalu od z_{min} do z_{max} s krokem δ_z . V prvním kroku se vybere první trojúhelník vytvořený Delaunayovou triangulací:

$$p_1 = dt_i; \quad p_2 = dt_{i+1}; \quad p_3 = dt_{i+3} \quad (7)$$

Od aktuální hodnoty z jsou odečteny hodnoty jednotlivých bodů:

$$\delta_{z_1} = z - p_{1z}; \quad \delta_{z_2} = z - p_{2z}; \quad \delta_{z_3} = z - p_{3z} \quad (8)$$

Před výpočtem souřadnic x a y je nutné ověřit, jak je umístěn trojúhelník v prostoru vůči rovině tvořené souřadnicí z . Pokud δ_{z_1} , δ_{z_2} a δ_{z_3} jsou všechny rovny nule trojúhelník je koplanární proto ho není potřeba řešit. Pokud dvojice rozdílů je rovna 0, hrana tvoří vrstevnici. V ostatních případech násobky rozdílů:

$$\delta_{z_1}\delta_{z_2}; \quad \delta_{z_1}\delta_{z_3}; \quad \delta_{z_2}\delta_{z_3} \quad (9)$$

Pomocí lineární interpolace se vypočítají souřadnice průsečíku pouze v případě, že násobek je < 0 . V ostatních případech výpočet nemá smysl.

Metoda createContourLines

```
1: Vstup: triangulace  $dt$ , minimální výška  $z_{\min}$ , maximální výška  $z_{\max}$ , krok  $dz$ 
2: Výstup: seznam vrstevnicových hran  $contour\_lines$ 
3:  $contour\_lines \leftarrow \emptyset$ 
4: for  $z \leftarrow z_{\min}$  to  $z_{\max}$  s krokem  $dz$  do
5:   for  $i \leftarrow 0$  to  $\text{size}(dt) - 1$  po třech do
6:      $p_1 \leftarrow \text{start}(dt[i])$ 
7:      $p_2 \leftarrow \text{start}(dt[i + 1])$ 
8:      $p_3 \leftarrow \text{start}(dt[i + 2])$ 
9:      $dz_1 \leftarrow z - p_1.z$ 
10:     $dz_2 \leftarrow z - p_2.z$ 
11:     $dz_3 \leftarrow z - p_3.z$ 
12:    if  $dz_1 = 0$  and  $dz_2 = 0$  then
13:      přidej  $dt[i]$  do  $contour\_lines$ 
14:      continue
15:    end if
16:    if  $dz_2 = 0$  and  $dz_3 = 0$  then
17:      přidej  $dt[i + 1]$  do  $contour\_lines$ 
18:      continue
19:    end if
20:    if  $dz_3 = 0$  and  $dz_1 = 0$  then
21:      přidej  $dt[i + 2]$  do  $contour\_lines$ 
22:      continue
23:    end if
24:    if  $dz_1 \cdot dz_2 \leq 0$  then
25:       $a \leftarrow \text{countourLinePoint}(p_1, p_2, z)$ 
26:      if  $dz_2 \cdot dz_3 \leq 0$  then
27:         $b \leftarrow \text{countourLinePoint}(p_2, p_3, z)$ 
28:        přidej hranu  $(a, b)$  do  $contour\_lines$ 
29:      else if  $dz_3 \cdot dz_1 \leq 0$  then
30:         $b \leftarrow \text{countourLinePoint}(p_3, p_1, z)$ 
31:        přidej hranu  $(a, b)$  do  $contour\_lines$ 
32:      end if
33:    else if  $dz_2 \cdot dz_3 \leq 0$  then
34:       $a \leftarrow \text{countourLinePoint}(p_2, p_3, z)$ 
35:      if  $dz_3 \cdot dz_1 \leq 0$  then
36:         $b \leftarrow \text{countourLinePoint}(p_3, p_1, z)$ 
37:        přidej hranu  $(a, b)$  do  $contour\_lines$ 
38:      end if
39:    end if
40:  end for
41: end for
42: return  $contour\_lines$ 
```

Metoda countourLinePoint

```
1: Vstup: body  $p_1, p_2$ , výšková hodnota  $z$ 
2: Výstup: bod  $p$  na vrstevnici  $z$  mezi  $p_1$  a  $p_2$ 
3:  $x_b \leftarrow \left( \frac{p_2.x - p_1.x}{p_2.z - p_1.z} \right) \cdot (z - p_1.z) + p_1.x$ 
4:  $y_b \leftarrow \left( \frac{p_2.y - p_1.y}{p_2.z - p_1.z} \right) \cdot (z - p_1.z) + p_1.y$ 
5: return bod  $(x_b, y_b, z)$ 
```

4.3 Analýza sklonu

Pro jednotlivé trojúhelníky je vypočítán sklon daného trojúhelníku.

Pokud je vektor trojúhelníků prázdný, nebo byly přidány nové body, je přepočítán vektor trojúhelníků. Z třech bodů trojúhelníku $p_1 = (x_1, y_1, z_1)$, $p_2 = (x_2, y_2, z_2)$ a $p_3 = (x_3, y_3, z_3)$.

$$\vec{u} = (x_3 - x_2, y_3 - y_2, z_3 - z_2) \quad (10)$$

$$\vec{v} = (x_1 - x_2, y_1 - y_2, z_1 - z_2) \quad (11)$$

Normálový vektor \vec{n} je určen jako vektorový součin $\vec{u} \times \vec{v}$:

$$n_x = (y_3 - y_2)(z_1 - z_2) - (z_3 - z_2)(y_1 - y_2) \quad (12)$$

$$n_y = -[(x_3 - x_2)(z_1 - z_2) - (z_3 - z_2)(x_1 - x_2)] \quad (13)$$

$$n_z = (x_3 - x_2)(y_1 - y_2) - (y_3 - y_2)(x_1 - x_2) \quad (14)$$

Velikost normálového vektoru:

$$\|\vec{n}\| = \sqrt{n_x^2 + n_y^2 + n_z^2} \quad (15)$$

Sklon trojúhelníka vůči rovině XY (v radiánech):

$$\theta = \arccos\left(\frac{n_z}{\|\vec{n}\|}\right) \quad (16)$$

Pro zobrazení sklonu svahu byla použita barevná stupnice v odstínech šedi. Kdy nejmenší sklon zobrazuje bílá barva a největší sklon černá barva.

Metoda `edgesToTriangle`

```
1: Vstup: seznam hran triangulace dt
2: Výstup: seznam trojúhelníků triangles
3: for  $i \leftarrow 0$  to  $\text{size}(dt) - 1$  po třech do
4:    $p_1 \leftarrow \text{start}(dt[i])$ 
5:    $p_2 \leftarrow \text{start}(dt[i + 1])$ 
6:    $p_3 \leftarrow \text{start}(dt[i + 2])$ 
7:    $t \leftarrow \text{nový trojúhelník}(p_1, p_2, p_3)$ 
8:   přidej  $t$  do triangles
9: end for
```

Metoda `analyzeSlope`

```
1: Vstup: triangulace dt, seznam trojúhelníků triangles, boolean click
2: Výstup: aktualizovaný seznam triangles s hodnotami sklonu
3: if  $\text{size}(\text{triangles}) = 0$  or click then
4:   edgesToTriangle(dt, triangles)
5: end if
6: for  $i \leftarrow 0$  to  $\text{size}(\text{triangles}) - 1$  do
7:    $p_1 \leftarrow \text{triangles}[i].\text{getP1}()$ 
8:    $p_2 \leftarrow \text{triangles}[i].\text{getP2}()$ 
9:    $p_3 \leftarrow \text{triangles}[i].\text{getP3}()$ 
10:   $s \leftarrow \text{computeSlope}(p_1, p_2, p_3)$ 
11:  triangles[i].setSlope(s)
12: end for
```

```

1: Vstup: body  $p_1, p_2, p_3$ 
2: Výstup: sklon trojúhelníku v radiánech
3:  $u \leftarrow p_2\vec{p}_3 = (u_x, u_y, u_z) = (p_3.x - p_2.x, p_3.y - p_2.y, p_3.z - p_2.z)$ 
4:  $v \leftarrow p_2\vec{p}_1 = (v_x, v_y, v_z) = (p_1.x - p_2.x, p_1.y - p_2.y, p_1.z - p_2.z)$ 
5:  $n_x \leftarrow u_y \cdot v_z - u_z \cdot v_y$ 
6:  $n_y \leftarrow -(u_x \cdot v_z - u_z \cdot v_x)$ 
7:  $n_z \leftarrow u_x \cdot v_y - u_y \cdot v_x$ 
8:  $n \leftarrow \sqrt{n_x^2 + n_y^2 + n_z^2}$  {velikost normálového vektoru}
9: return  $\arccos\left(\frac{n_z}{n}\right)$ 

```

4.4 Orientace svahu

Pro jednotlivé trojúhelníky je vypočítána orientace daného trojúhelníku.

Pokud je vektor trojúhelníků prázdný, nebo byly přidány nové body, je přepočítán vektor trojúhelníků. Z třech bodů trojúhelníku $p_1 = (x_1, y_1, z_1)$, $p_2 = (x_2, y_2, z_2)$ a $p_3 = (x_3, y_3, z_3)$.

$$\vec{u} = (x_3 - x_2, y_3 - y_2, z_3 - z_2) \quad (17)$$

$$\vec{v} = (x_1 - x_2, y_1 - y_2, z_1 - z_2) \quad (18)$$

Normálový vektor \vec{n} je určen jako vektorový součin $\vec{u} \times \vec{v}$:

$$n_x = (y_3 - y_2)(z_1 - z_2) - (z_3 - z_2)(y_1 - y_2) \quad (19)$$

$$n_y = -[(x_3 - x_2)(z_1 - z_2) - (z_3 - z_2)(x_1 - x_2)] \quad (20)$$

$$n_z = (x_3 - x_2)(y_1 - y_2) - (y_3 - y_2)(x_1 - x_2) \quad (21)$$

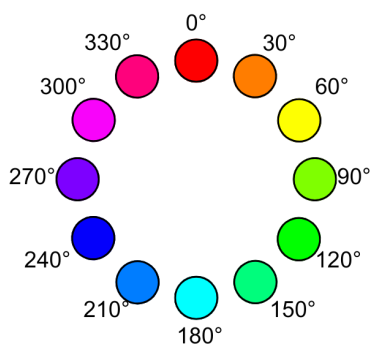
Velikost normálového vektoru:

$$\|\vec{n}\| = \sqrt{n_x^2 + n_y^2 + n_z^2} \quad (22)$$

Orientace trojúhelníka vůči rovině XY (v radiánech):

$$\theta = \arctan 2 \left(\frac{n_z}{\|\vec{n}\|} \right) \quad (23)$$

Pro zobrazení orientace trojúhelníku byla využita tato barevná stupnice:



Obrázek 1: Barevná stupnice

Metoda `analyzeAspect`

```
1: Vstup: triangulace dt, seznam trojúhelníků triangles, boolean click
2: Výstup: aktualizovaný seznam triangles s hodnotami orientace
3: if size(triangles) = 0 or click then
4:   edgesToTriangle(dt, triangles)
5: end if
6: for  $i \leftarrow 0$  to size(triangles) - 1 do
7:    $p_1 \leftarrow \text{triangles}[i].\text{getP1}()$ 
8:    $p_2 \leftarrow \text{triangles}[i].\text{getP2}()$ 
9:    $p_3 \leftarrow \text{triangles}[i].\text{getP3}()$ 
10:   $a \leftarrow \text{computeAspect}(p_1, p_2, p_3)$ 
11:  triangles[i].setAspect(a)
12: end for
```

Metoda `computeAspect`

```
1: Vstup: body  $p_1, p_2, p_3$ 
2: Výstup: Orientace trojúhelníku v radiánech
3:  $u \leftarrow p_2\vec{p}_3 = (u_x, u_y, u_z) = (p_3.x - p_2.x, p_3.y - p_2.y, p_3.z - p_2.z)$ 
4:  $v \leftarrow p_2\vec{p}_1 = (v_x, v_y, v_z) = (p_1.x - p_2.x, p_1.y - p_2.y, p_1.z - p_2.z)$ 
5:  $n_x \leftarrow u_y \cdot v_z - u_z \cdot v_y$ 
6:  $n_y \leftarrow -(u_x \cdot v_z - u_z \cdot v_x)$ 
7:  $n_z \leftarrow u_x \cdot v_y - u_y \cdot v_x$ 
8:  $n \leftarrow \sqrt{n_x^2 + n_y^2 + n_z^2}$  {velikost normálového vektoru}
9: return  $\arctan 2 \left( \frac{n_z}{n} \right)$ 
```

5 Problematické situace

5.1 Transformace souřadnic bodů ze souboru

Při načítání souřadnicových dat ze souborů (např. `.txt` nebo `.xyz`) do aplikace je nutné body transformovat tak, aby odpovídaly souřadnicovému systému vykreslovacího okna. Data obvykle pocházejí z reálného terénu a mají velké absolutní hodnoty souřadnic. Tyto hodnoty je třeba převést (škálovat) na oblast viditelnou v aplikaci.

- **Načtení bodů ze souboru**

Načítají se trojice souřadnic (x, y, z) , přičemž aplikace podporuje formát oddělený čárkou, mezerou nebo jejich kombinací. Před načtením se vymažou předchozí body.

- **Zjištění rozsahu hodnot (bounding box)**

Během načítání jsou pro každý bod určeny minimální a maximální hodnoty souřadnic x, y a z , které později slouží k výpočtu měřítka.

- **Zjištění rozměrů vykreslovacího okna**

Pomocí funkcí `width()` a `height()` jsou získány rozměry okna aplikace, do kterého se mají data zobrazit.

- **Výpočet měřítka**

Souřadnice bodů se transformují pomocí škálování. Měřítka se počítá zvlášť pro osu x a y :

$$\text{scaleX} = \frac{\text{windowWidth}}{\text{maxX} - \text{minX}}, \quad \text{scaleY} = \frac{\text{windowHeight}}{\text{maxY} - \text{minY}} \quad (24)$$

- **Výpočet posunu (offsetu)**

Aby byly body zarovnány k okraji okna, použije se posun:

$$\text{offsetX} = -\text{minX}, \quad \text{offsetY} = -\text{minY} \quad (25)$$

- **Transformace souřadnic bodů**

Každému bodu je následně vypočítána nová souřadnice takto:

$$\text{transformedX} = (x + \text{offsetX}) \cdot \text{scaleX} \quad (26)$$

$$\text{transformedY} = (y + \text{offsetY}) \cdot \text{scaleY} \quad (27)$$

- **Uložení bodů a překreslení**

Po transformaci jsou nové souřadnice uloženy do vektoru a aplikace je překreslena funkcí `repaint()`.

6 Vstupní data

Vstupní data pro tvorbu digitálního modelu terénu mohou být do aplikace zadána několika způsoby, což umožňuje flexibilní testování i realistické modelování různých terénních situací:

- **Ručně uživatelem** – body lze vkládat interaktivně kliknutím do grafického rozhraní. Souřadnice x a y odpovídají poloze kurzoru, výška z je generována náhodně.
- **Generování syntetických dat** – aplikace umožňuje automaticky vygenerovat reprezentativní morfologické tvary terénu:

- *Kupa*
- *Údolí*
- *Hřbet*
- *Spočinek*
- *Sedlo*

- **Načtení ze souboru** – aplikace podporuje import bodových dat ze souborů ve formátu `.txt` nebo `.xyz`. Každý řádek obsahuje trojici hodnot: x , y , z , přičemž hodnoty mohou být odděleny mezerou nebo čárkou. Jako oddělovač desetinných míst se používá tečka. Ukázka vstupního souboru:

```
120.0,280.0,320.0
160.0,670.0,370.0
260.0,310.0,410.0
...
```

Vzorková data použitá v této úloze pocházejí z digitálního modelu reliéfu České republiky – DMR 5G. Zdroj: **Český úřad zeměměřický a katastrální (ČÚZK)** – dostupné online na: Zdroj dat: Český úřad zeměměřický a katastrální (ČÚZK) [2].

7 Výstupní data

Výsledky jsou prezentovány formou interaktivní vizualizace přímo v aplikaci. Uživatel má možnost zobrazit vstupní body, Delaunay triangulaci, vrstevnice, sklony a orientace trojúhelníků. Jednotlivé složky lze podle potřeby zapínat nebo vypínat přímo v grafickém rozhraní. Vrstevnice jsou generovány s možností nastavení intervalu dz , hlavní vrstevnice jsou automaticky popisovány v souladu s kartografickými zásadami. Aplikace neumožňuje export výsledků (např. do souboru), slouží primárně k vizuální a analytické interpretaci vstupních dat a jejich geometrické struktury. Výsledky jsou tedy určeny pouze pro prohlížení v rámci samotné aplikace.

Analytické možnosti vizualizace

Součástí vizualizace je i možnost analyzovat jednotlivé aspekty digitálního modelu terénu:

- **Vrstevnice** – generovány lineární interpolací podle zadaného kroku a rozsahu, s podporou hlavních vrstevnic a jejich popisu.
- **Sklon** – zobrazen pomocí stupnice odstínů šedi, kde tmavší barva značí strmější svah.

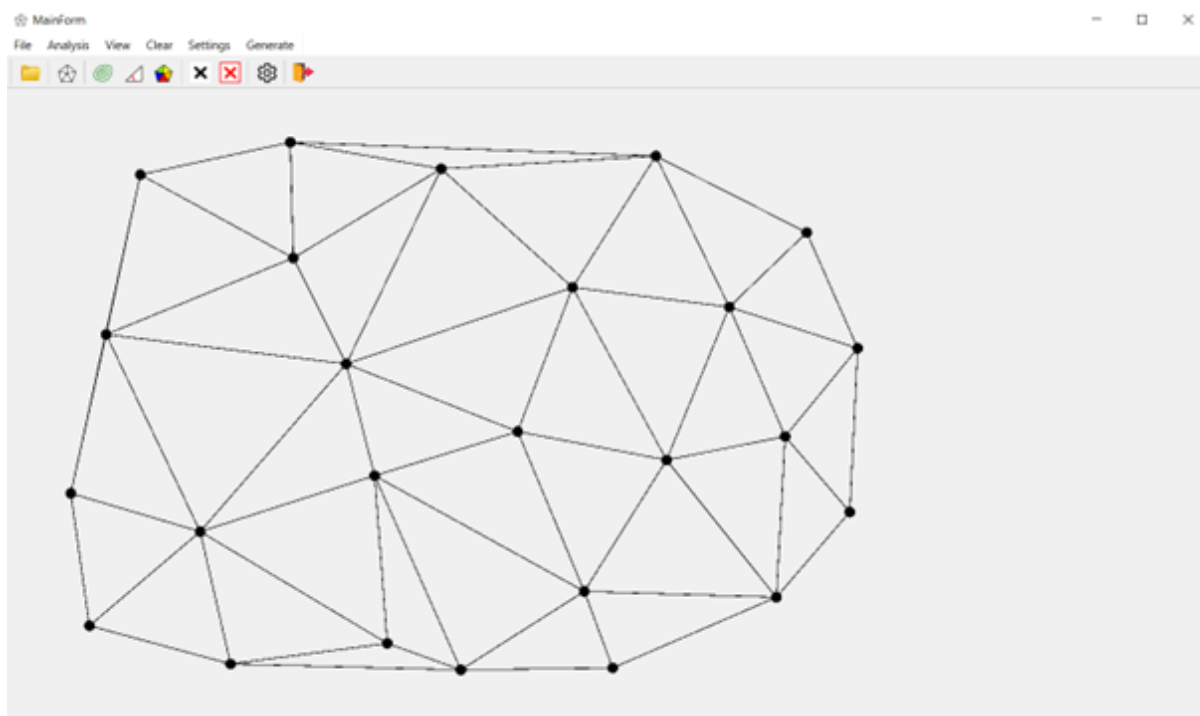
- **Expozice (aspekt)** – vyjádřena barevnou orientací trojúhelníků podle směru jejich normály ke světovým stranám.
- **3D vizualizace** – zjednodušená ortogonální projekce, kterou lze interaktivně rotovat a přibližovat.

8 Výsledná aplikace

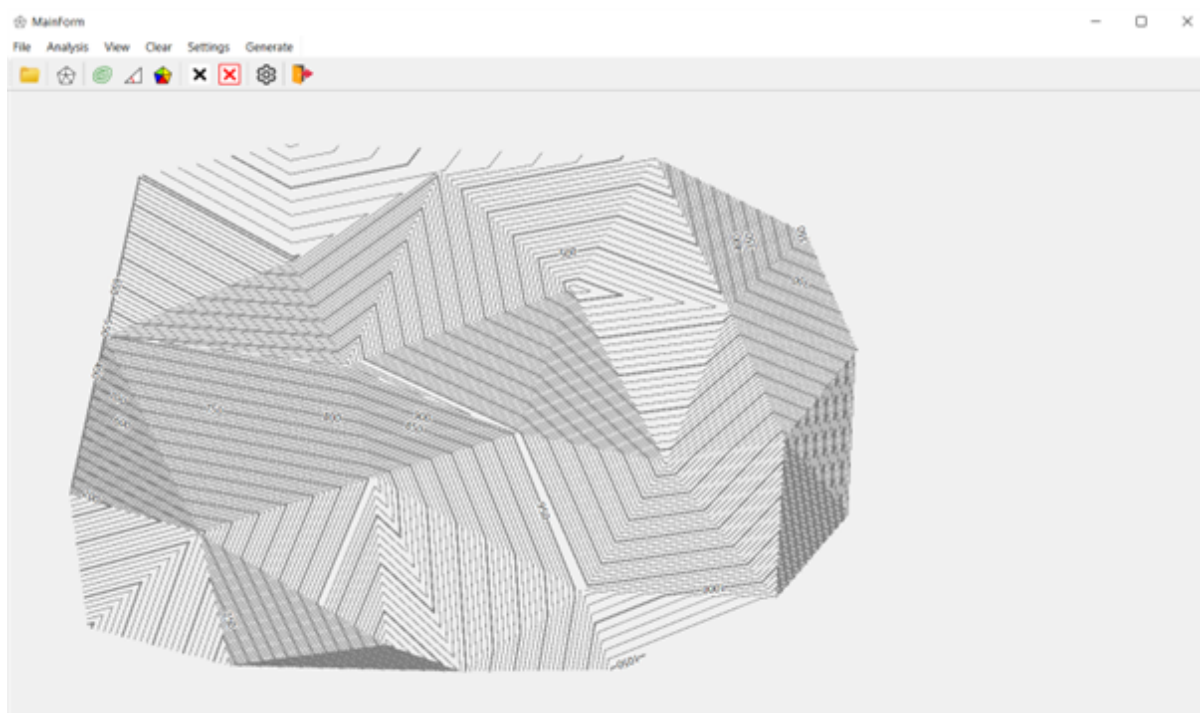
Následující obrázky ukazují jednotlivé části a funkce výsledné aplikace, která slouží k tvorbě digitálního modelu terénu pomocí Delaunay triangulace, generování vrstevnic, výpočtu sklonu a expozice a základní 3D vizualizaci. Aplikace umožňuje načíst vlastní data nebo generovat syntetické tvary terénu.



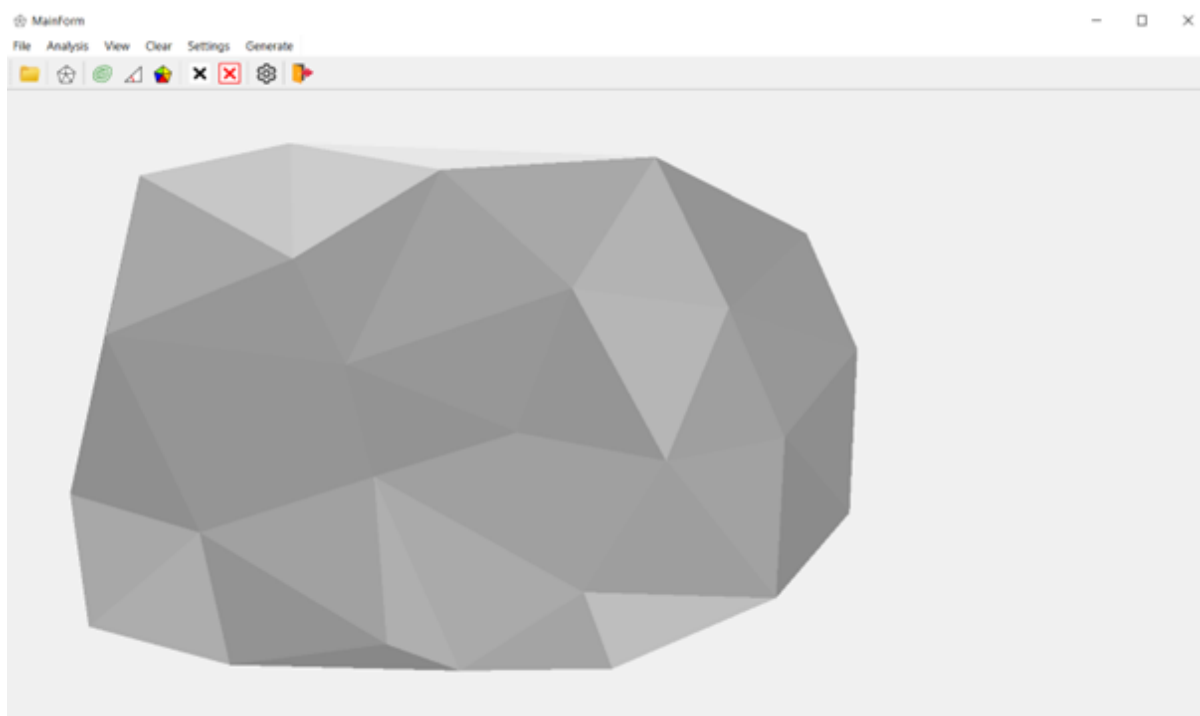
Obrázek 2: Hlavní okno aplikace s ovládacími prvky



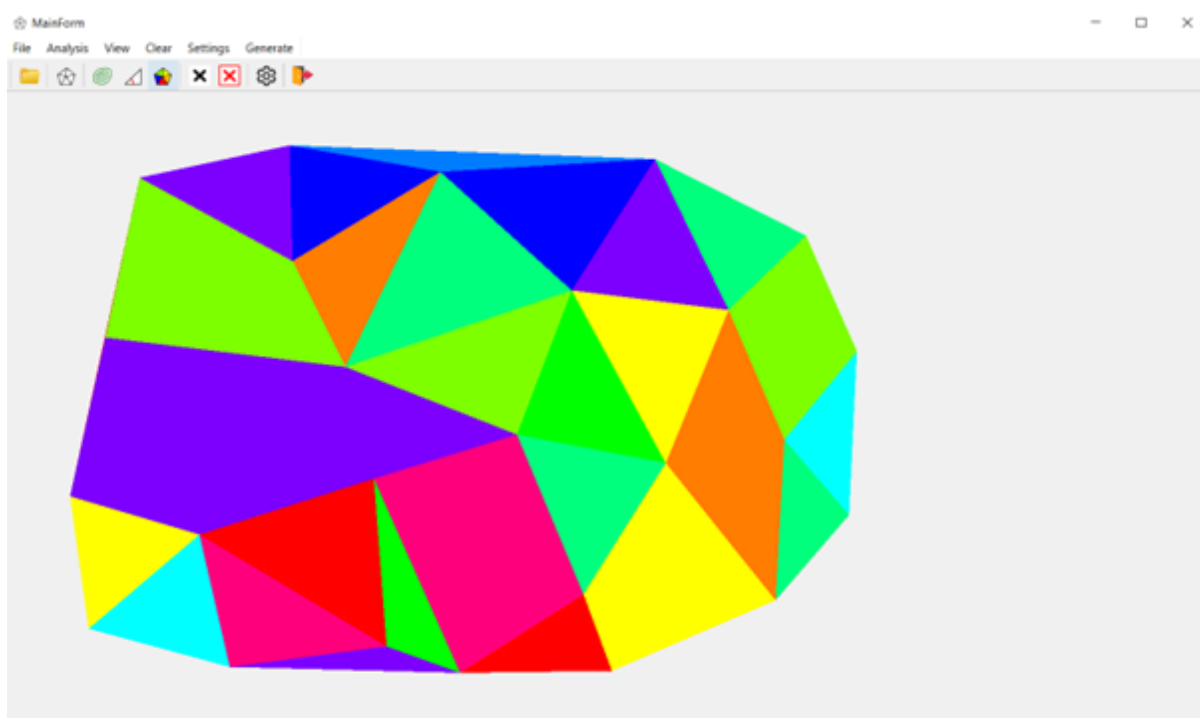
Obrázek 3: Vytvořená Delaunay triangulace (TIN) ze zadaných bodů



Obrázek 4: Vizualizace vrstevnic vytvořených lineární interpolací



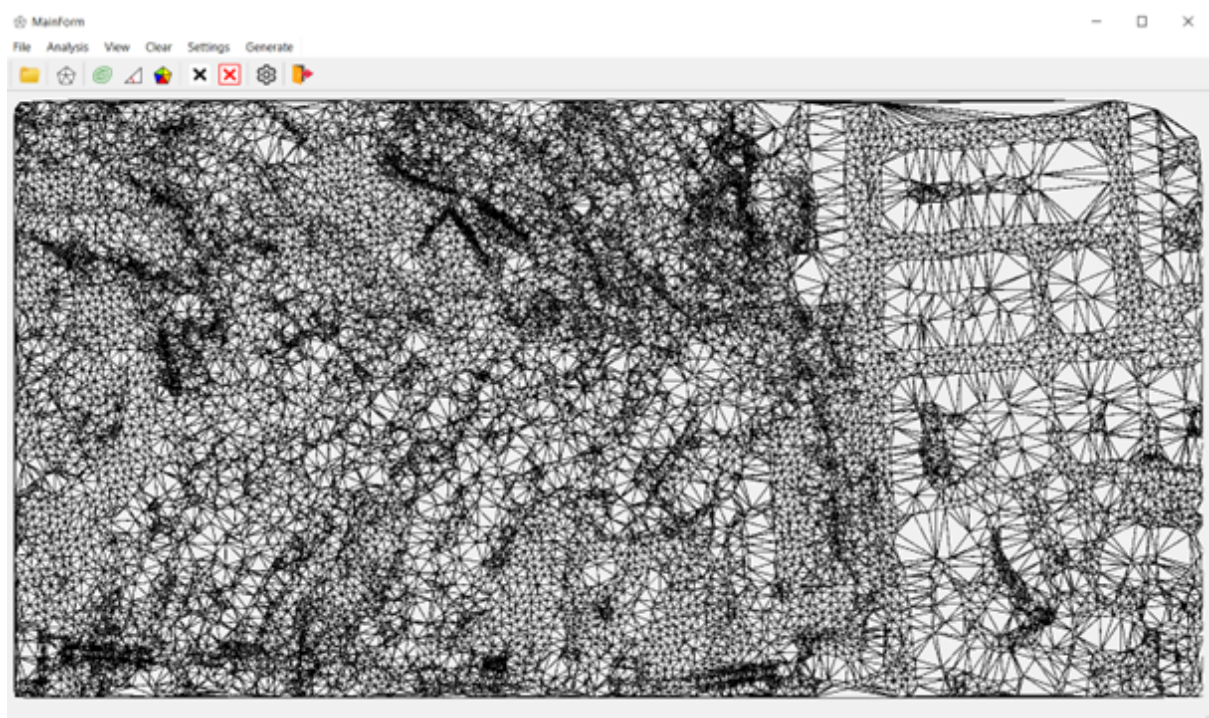
Obrázek 5: Barevná vizualizace sklonu (slope) jednotlivých trojúhelníků



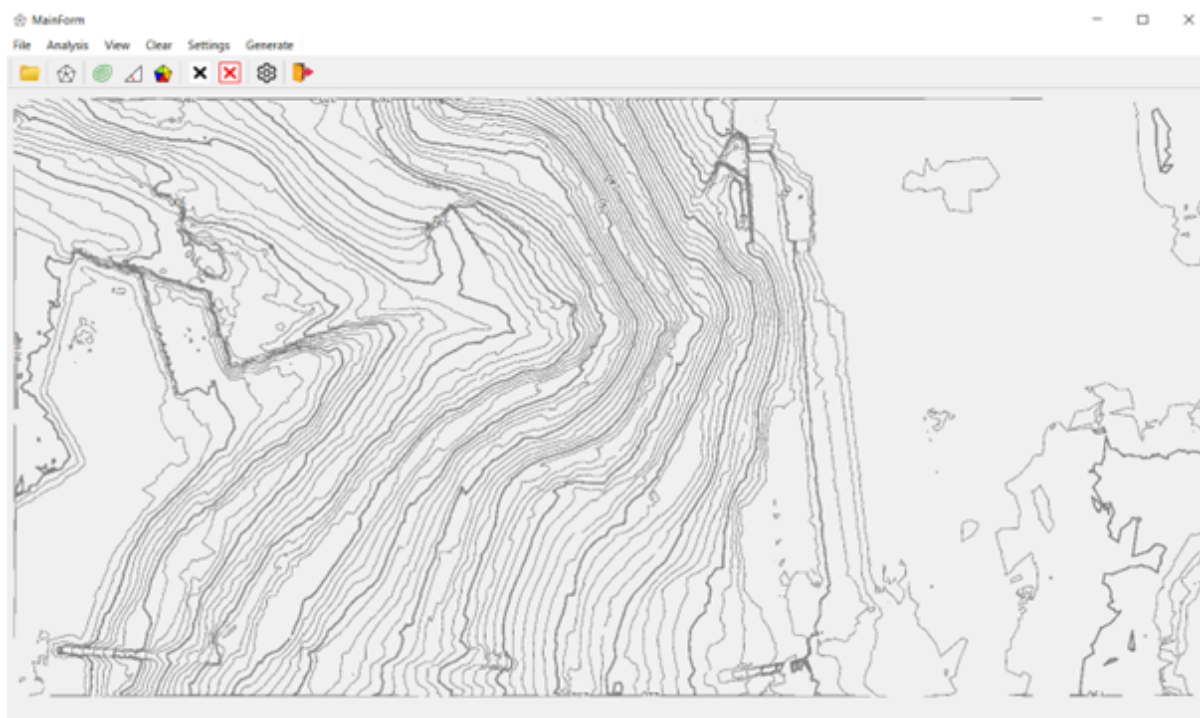
Obrázek 6: Barevná vizualizace expozice (aspekt) jednotlivých trojúhelníků



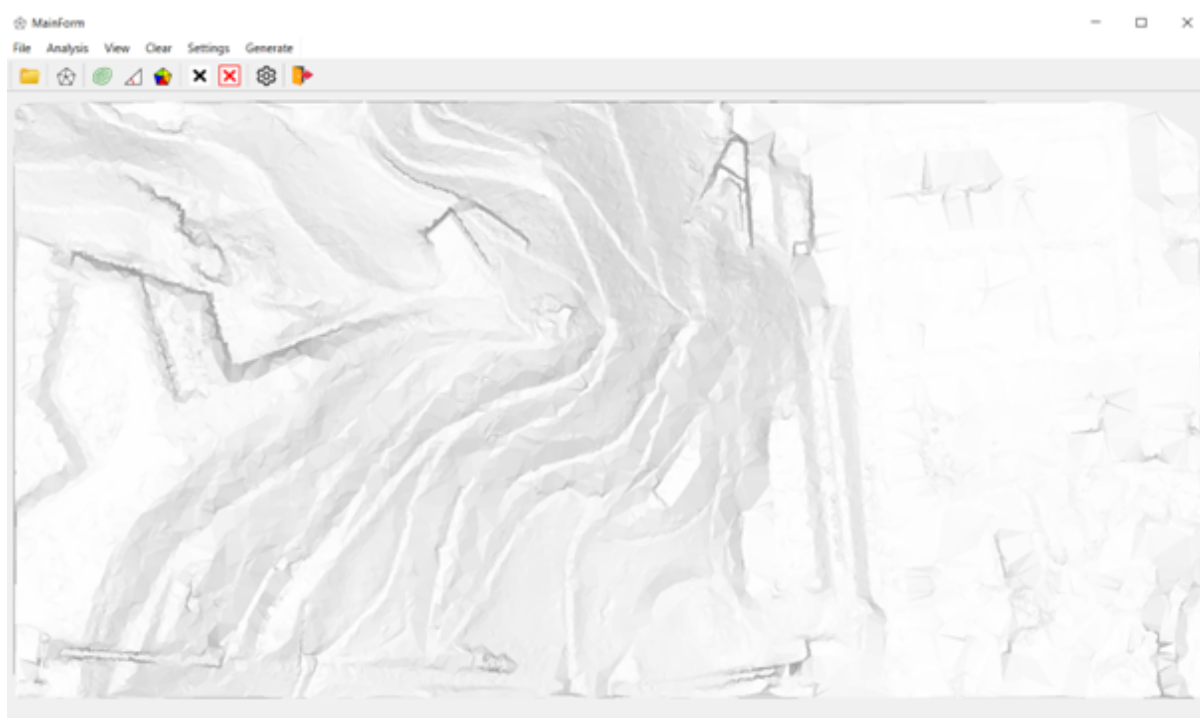
Obrázek 7: Načtení reálných dat – okolí Petřína (data z DMR 5G)



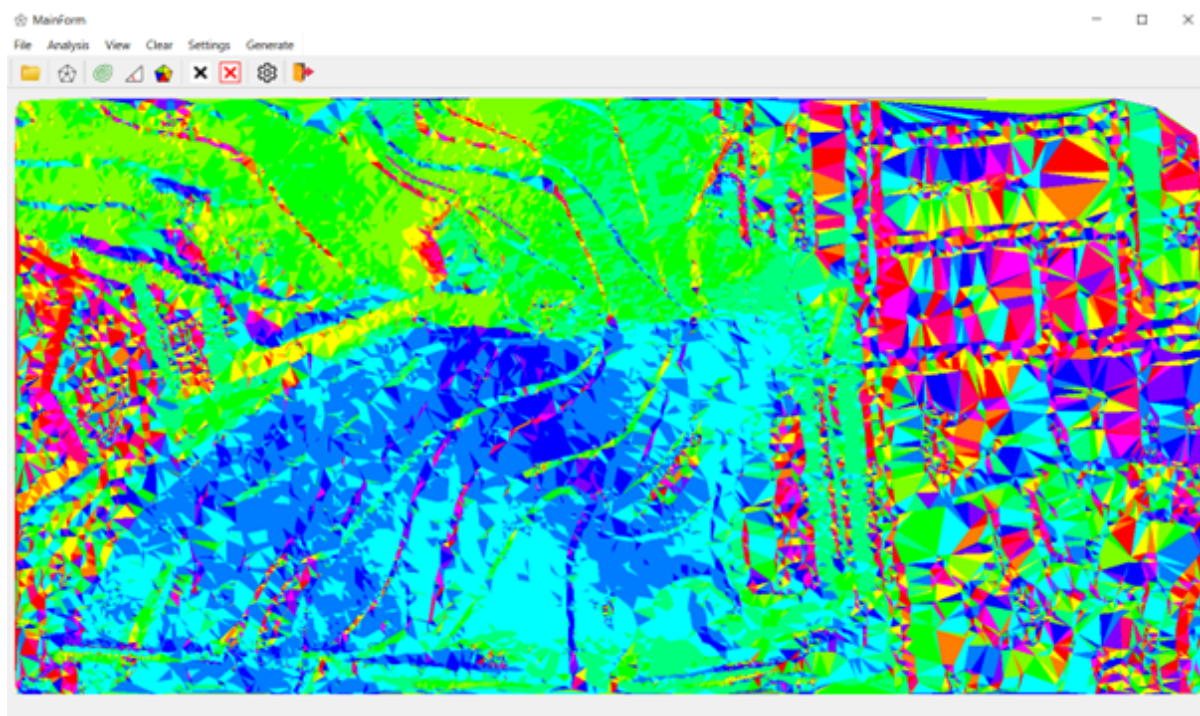
Obrázek 8: Delaunayova triangulace – okolí Petřína (data z DMR 5G)



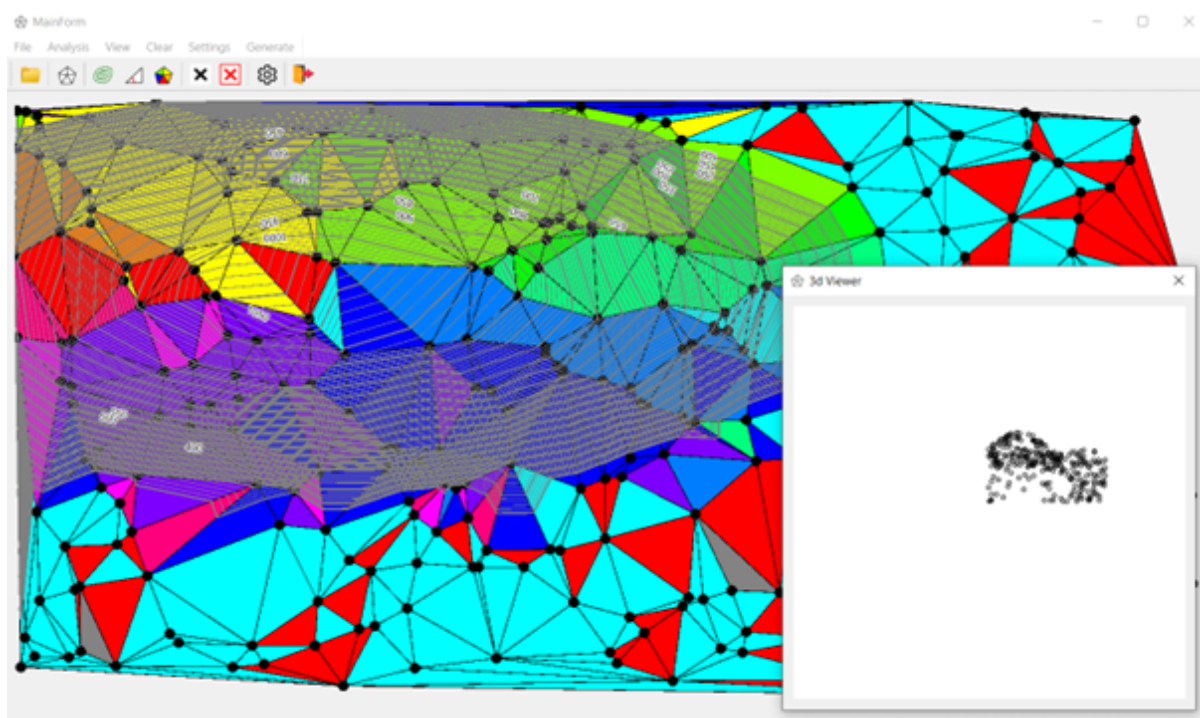
Obrázek 9: Vrstevnice – okolí Petřína (data z DMR 5G)



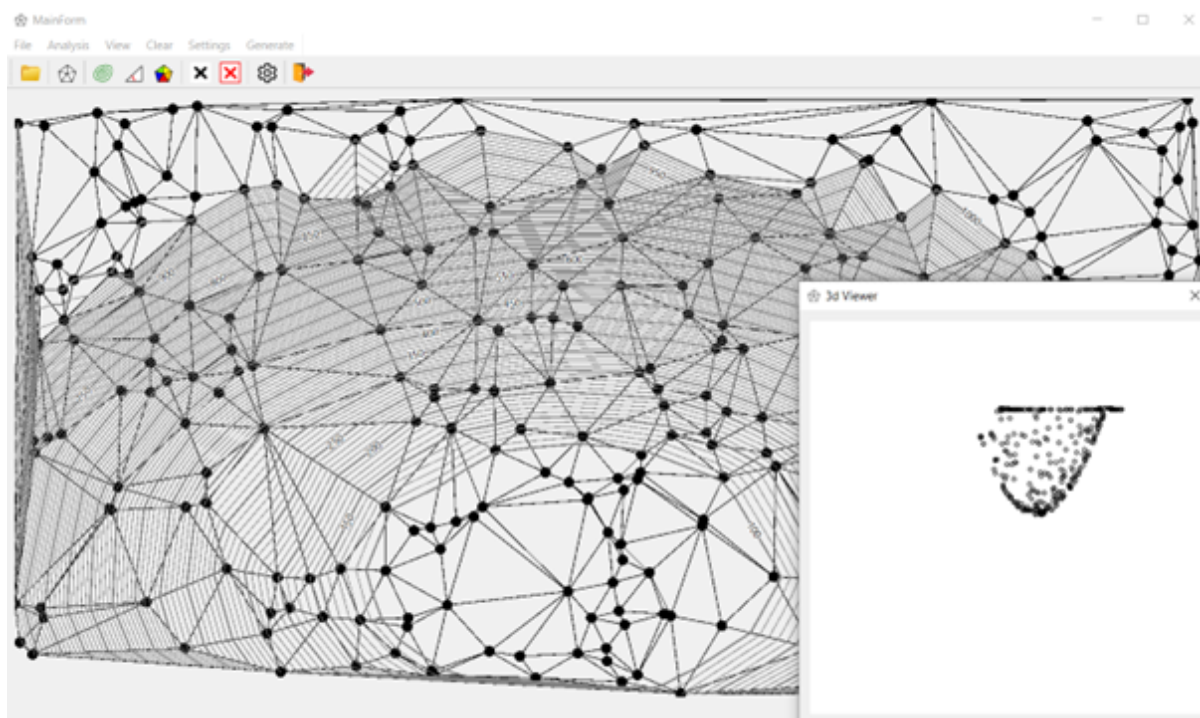
Obrázek 10: Sklon (slope) – okolí Petřína (data z DMR 5G)



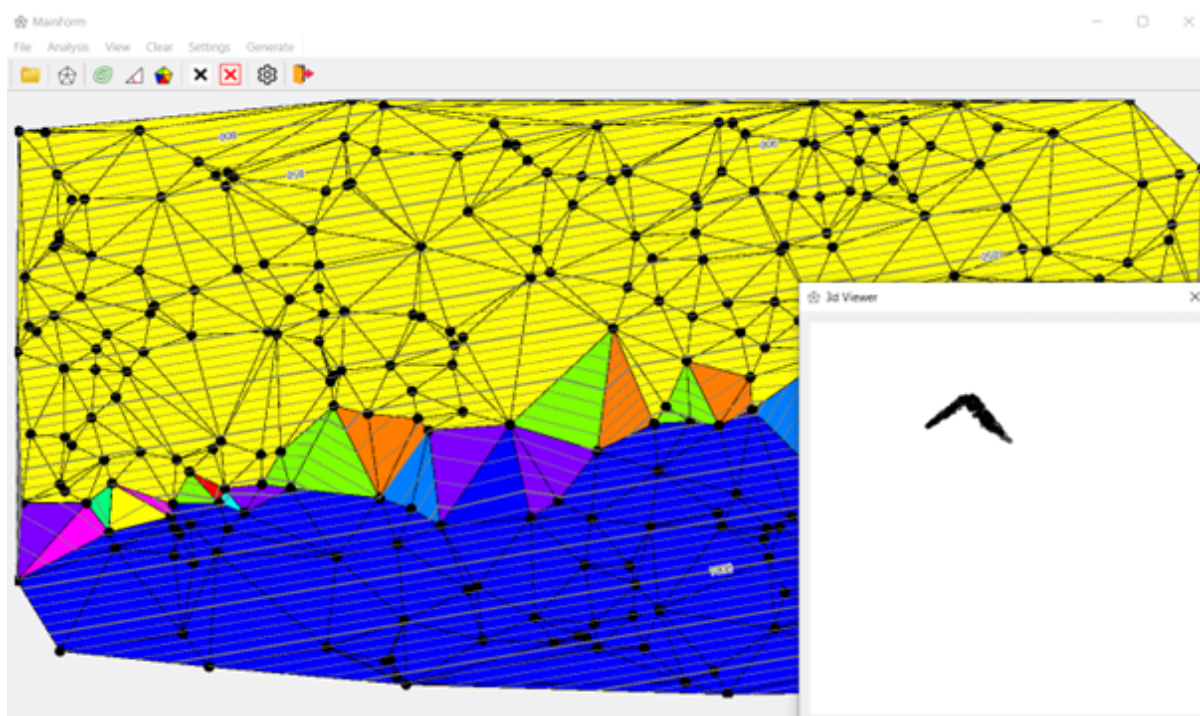
Obrázek 11: Expozice (aspect) – okolí Petřína (data z DMR 5G)



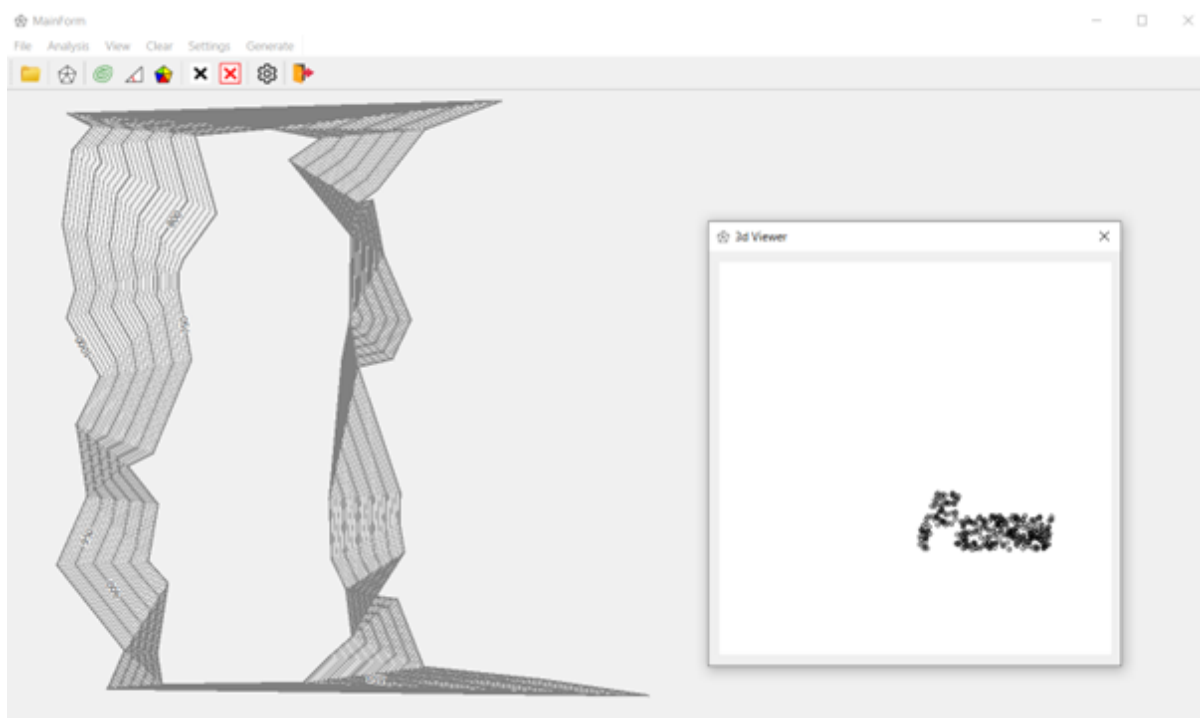
Obrázek 12: Ukázka uměle vygenerované kupy



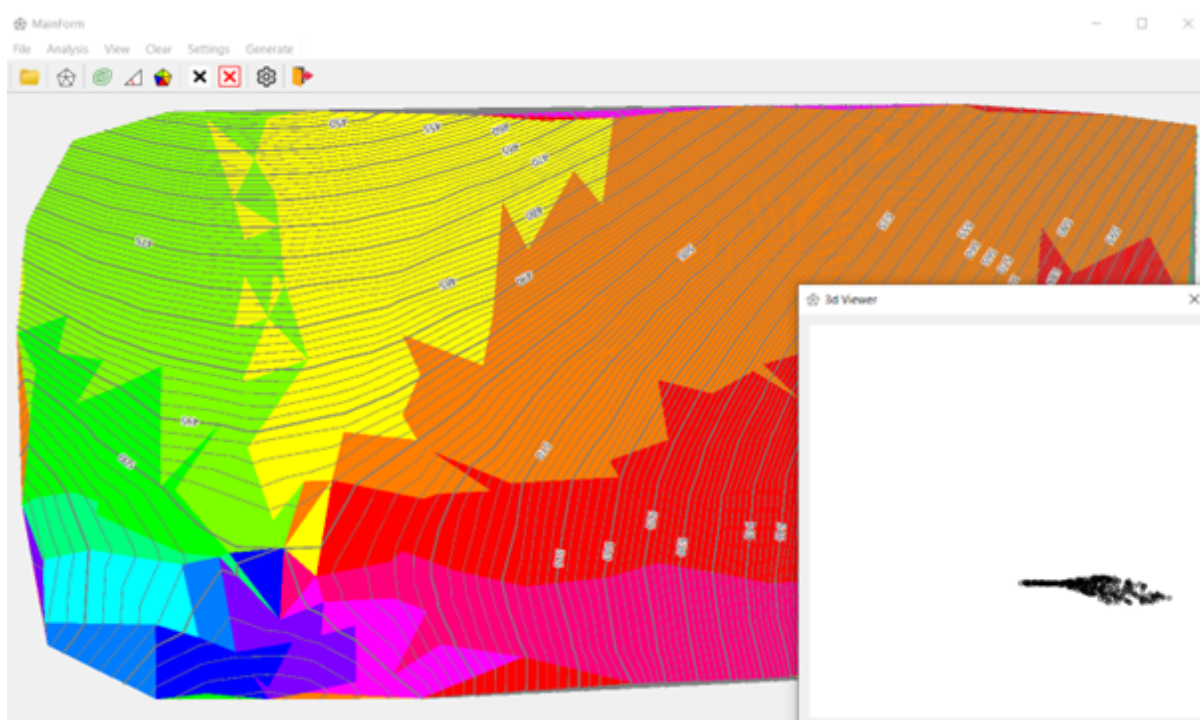
Obrázek 13: Ukázka uměle vygenerovaného údolí



Obrázek 14: Ukázka uměle vygenerovaného hřebetu



Obrázek 15: Ukázka uměle vygenerovaného spočinku



Obrázek 16: Ukázka uměle vygenerovaného sedla

9 Dokumentace

9.1 Použité knihovny

Kód využívá následující knihovny:

- Qt – Knihovna která slouží jak pro tvorbu grafického rozhraní, tak pro vlastní vizualizaci dat. Využívány jsou zejména třídy pro 2D grafiku (QPainter, QPointF, QPolygonF), správu událostí a widgetů (QWidget, QDialog, QMouseEvent) a základní kontejnery a typy (QVector, QString).

9.2 Třída Algorithms

Třída metody pro vytvoření 2D Delaunay triangulace, výpočet vrstevnic, analýzu sklonu a orientace trojúhelníků a generování syntetických terénních tvarů (např. kupa, údolí, hřbet, spočinek, sedlo).

Veřejné metody:

- `short getPointAndLinePosition(const QPoint3DF &p, const QPoint3DF &p1, const QPoint3DF &p2)` – Určuje polohu bodu vůči přímce ve 3D prostoru.
- `double get2LinesAngle(const QPoint3DF &p1, const QPoint3DF &p2, const QPoint3DF &p3, const QPoint3DF &p4)` – Vypočítá úhel mezi dvěma přímkami definovanými dvojicemi bodů.
- `int findDelaunayPoint(const QPoint3DF &p1, const QPoint3DF &p2, const std::vector<QPoint3DF> &points)` – Vrací index bodu, který tvoří s danou hranou Delaunay trojúhelník.
- `int findNearestPoint(const QPoint3DF &p, const std::vector<QPoint3DF> &points)` – Najde nejbližší bod k zadanému bodu ze seznamu bodů.
- `double get2DDistance(const QPoint3DF &p1, const QPoint3DF &p2)` – Vypočítá vzdálenost mezi dvěma body v rovině XY.
- `std::vector<Edge> DT(const std::vector<QPoint3DF> &points)` – Vytvoří Delaunay triangulaci ze vstupních bodů.
- `void updateAEL(const Edge &e, std::list<Edge> &ael)` – Aktualizuje aktivní seznam hran při triangulaci.
- `QPoint3DF countourLinePoint(const QPoint3DF &p1, const QPoint3DF &p2, double z)` – Vypočítá průsečík vrstevnice se zadanou hranou.
- `std::vector<Edge> createContourLines(const std::vector<Edge> &dt, const double zmin, const double zmax, const double dz)` – Generuje vrstevnice z Delaunay triangulace pro daný výškový rozsah a interval.
- `double computeSlope(const QPoint3DF &p1, const QPoint3DF &p2, const QPoint3DF &p3)` – Spočítá sklon trojúhelníka definovaného třemi body.
- `void analyzeSlope(const std::vector<Edge> &dt, std::vector<Triangle> &triangles, const bool &click)` – Analyzuje sklony trojúhelníků na základě triangulace.
- `double computeAspect(const QPoint3DF &p1, const QPoint3DF &p2, const QPoint3DF &p3)` – Vypočítá orientaci trojúhelníku.
- `void analyzeAspect(const std::vector<Edge> &dt, std::vector<Triangle> &triangles, const bool &click)` – Analyzuje orientaci trojúhelníků na základě triangulace.
- `void edgesToTriangle(const std::vector<Edge> &dt, std::vector<Triangle> &triangles)` – Převádí hrany triangulace na trojúhelníky.
- `static std::vector<QPoint3DF> generateHill(int n, int width, int height, int cx, int cy, int rx, int ry, int maxZ)` – Vygeneruje soubor bodů simulujících tvar kopce (kupy), jehož výška kvadraticky klesá od středu eliptické základny směrem k okraji.
- `static std::vector<QPoint3DF> generateValley(int n, int width, int height, int cx, int cy, int rx, int ry, int depth)` – Vygeneruje údolí, jehož dno leží uprostřed elipsy a výška roste směrem k okrajům, čímž vzniká konkávní terénní tvar.
- `static std::vector<QPoint3DF> generateRidge(int n, int width, int height, int x1, int y1, int x2, int y2, int maxZ)` – Vytvoří hřbet (ridge) s výškou klesající od centrální osy – čím blíže k ní, tím vyšší hodnota Z.

- `static std::vector<QPoint3DF> generateBench(int n, int width, int height, int stepStartX, int stepEndX, int depthZ)` – Generuje spočinek jako mírný terasovitý pokles výšky v zadaném vodorovném pásmu.
- `static std::vector<QPoint3DF> generateSaddle(int n, int width, int height, int cx, int cy, int scaleX, int scaleY)` – Vytváří tvar sedla (hyperbolický paraboloid) – v ose X roste výška, v ose Y klesá.

9.3 Třída `sortPointsByX`

Třída pro seřazení bodů podle souřadnice x.

- `bool operator()(const QPointF &p1, const QPointF &p2)` – Vratí `True`, pokud `x.p1` je menší než `x.p2`.

9.4 Třída `sortPointsByY`

Třída pro seřazení bodů podle souřadnice y.

- `bool operator()(const QPointF &p1, const QPointF &p2)` – Vratí `True`, pokud `y.p1` je menší než `y.p2`.

9.5 Třída `Draw`

Třída zajišťuje vizualizaci dat (body, triangulace, vrstevnice, sklon, aspekt) a interakci uživatele s vykreslenou scénou.

Veřejné metody:

- `explicit Draw(QWidget *parent = nullptr)` – Konstruktor inicializující widget.
- `void mousePressEvent(QMouseEvent *e)` – Obsluhuje kliknutí myši pro interakci s plátnem.
- `void paintEvent(QPaintEvent *event)` – Metoda pro vykreslení obsahu widgetu (body, hrany, vrstevnice apod.).
- `std::vector<QPoint3DF> getPoints() const` – Vrací aktuálně zobrazené body.
- `std::vector<Edge> getDT() const` – Vrací aktuální triangulaci.
- `void setDT(const std::vector<Edge> &dt)` – Nastaví hrany triangulace.
- `void setCL(const std::vector<Edge> &contour_lines)` – Nastaví seznam vrstevnic.
- `void setTR(const std::vector<Triangle> &triangles)` – Nastaví trojúhelníky pro analýzu sklonu/aspektu.
- `void setViewPoints(const bool &view_points)` – Zapne/vypne zobrazení bodů.
- `void setViewDT(const bool &view_dt)` – Zapne/vypne zobrazení triangulace.
- `void setViewContourLines(const bool &view_contour_lines)` – Zapne/vypne zobrazení vrstevnic.
- `void setViewSlope(const bool &view_slope)` – Zapne/vypne zobrazení sklonu.
- `void setViewAspect(const bool &view_aspect)` – Zapne/vypne zobrazení aspektu.
- `void setClicked(const bool &clicked)` – Nastaví příznak kliknutí.
- `bool getClicked()` – Vrací příznak, zda bylo kliknuto.
- `void clearResults()` – Vymaže triangulaci, trojúhelníky a vrstevnice.
- `void clearAll()` – Vymaže všechna data včetně bodů.
- `void loadPointsFromTextfile(const QString &fileName)` – Načte body ze souboru ve formátu TXT s oddělovači (mezery, čárky). Automaticky provede převod souřadnic do velikosti okna.
- `void setPoints(const std::vector<QPoint3DF> &newPoints)` – Umožňuje hromadně nastavit novou množinu bodů, která bude následně zobrazena.

9.6 Třída MainForm

Třída představuje hlavní okno aplikace a zajišťuje propojení uživatelského rozhraní s funkcionalitou výpočtů a vizualizace.

Privátní metody:

- `void on_actionCreate_DT_triggered()` – Spustí vytvoření Delaunay triangulace.
- `void on_actionCreate_Contour_lines_triggered()` – Spustí generování vrstevnic.
- `void on_actionParameters_triggered()` – Otevře dialogové okno s parametry.
- `void on_actionAnalyze_slope_triggered()` – Spustí analýzu sklonu terénu.
- `void on_actionClear_Results_triggered()` – Vymaže výpočetní výsledky (trojúhelníky, vrstevnice, atd.).
- `void on_actionPoints_changed()` – Změna v zobrazení bodů.
- `void on_actionDT_changed()` – Změna v zobrazení triangulace.
- `void on_actionContour_Lines_changed()` – Změna v zobrazení vrstevnic.
- `void on_actionSlope_changed()` – Změna v zobrazení sklonu.
- `void on_actionExposition_changed()` – Změna v zobrazení expozice (aspektu).
- `void on_actionClear_All_triggered()` – Vymaže všechna data (body i výsledky).
- `void on_actionAnalyze_exposition_triggered()` – Spustí analýzu orientace.
- `void on_actionOpen_triggered()` – Otevře dialog pro načtení vstupního souboru s body (např. ve formátu TXT).
- `void on_actionExit_triggered()` – Ukončí aplikaci.
- `void on_actionHill_triggered()` – Vygeneruje syntetická data reprezentující kopec (kupu).
- `void on_actionValley_triggered()` – Vygeneruje syntetická data reprezentující údolí.
- `void on_actionRidge_triggered()` – Vygeneruje syntetická data reprezentující hřbet.
- `void on_actionBench_triggered()` – Vygeneruje syntetická data pro spočinek.
- `void on_actionSaddle_triggered()` – Vygeneruje syntetická data pro sedlo.
- `void on_action3D_Viewer_triggered()` – Spustí 3D vizualizaci bodů s možností otáčení a zoomu.

9.7 Třída QPoint3DF

Třída představuje rozšířený dvourozměrný bod o třetí souřadnici z . Dědí ze třídy `QPointF` a přidává výškovou hodnotu.

Veřejné metody:

- `QPoint3DF()` – Výchozí konstruktor, inicializuje bod $(0, 0, 0)$.
- `QPoint3DF(double x, double y)` – Konstruktor pro dvourozměrný bod se souřadnicemi (x, y) a $z = 0$.
- `QPoint3DF(double x, double y, double z)` – Konstruktor pro trojrozměrný bod se souřadnicemi (x, y, z) .
- `void setZ(double z)` – Nastaví hodnotu výšky z .
- `double getZ() const` – Vrátí hodnotu výšky z .

9.8 Třída Edge

Třída reprezentuje hranu mezi dvěma trojrozměrnými body typu `QPoint3DF`.

Veřejné metody:

- `Edge(const QPoint3DF &start, const QPoint3DF &end)` – Konstruktor hrany definované počátečním a koncovým bodem.
- `QPoint3DF getStart() const` – Vrátí počáteční bod hrany.
- `QPoint3DF getEnd() const` – Vrátí koncový bod hrany.
- `Edge changeOrientation() const` – Vrátí novou hranu s opačnou orientací.
- `bool operator== (const Edge &e)` – Porovná dvě hrany podle jejich počátečního a koncového bodu.

9.9 Třída Triangle

Třída reprezentuje trojúhelník definovaný třemi vrcholy v prostoru. Obsahuje navíc informace o sklonu a orientaci.

Veřejné metody:

- `Triangle()` – Výchozí konstruktor, inicializuje trojúhelník s nulovým sklonem a aspektem.
- `Triangle(QPoint3DF p1, QPoint3DF p2, QPoint3DF p3)` – Vytvoří trojúhelník bez specifikovaného sklonu a aspektu.
- `Triangle(QPoint3DF p1, QPoint3DF p2, QPoint3DF p3, double aspect, double slope)` – Vytvoří trojúhelník se zadanými hodnotami sklonu a expozice.
- `QPoint3DF getP1()` – Vrátí první bod trojúhelníku.
- `QPoint3DF getP2()` – Vrátí druhý bod trojúhelníku.
- `QPoint3DF getP3()` – Vrátí třetí bod trojúhelníku.
- `double getSlope()` – Vrátí hodnotu sklonu trojúhelníku.
- `double getAspect()` – Vrátí hodnotu expozice (aspektu) trojúhelníku.
- `void setSlope(const double &slope)` – Nastaví hodnotu sklonu trojúhelníku.
- `void setAspect(const double &aspect)` – Nastaví hodnotu orientace trojúhelníku.

9.10 Třída Terrain3DForm

Třída představuje dialogové okno pro zobrazení 3D modelu terénu. Používá `Terrain3DCanvas`, která vykresluje výškové body pomocí ortogonální projekce. Je součástí samostatného 3D zobrazení a slouží jako orientační přehled výškového rozložení.

Veřejné metody:

- `explicit Terrain3DForm(QWidget *parent = nullptr)` – Konstruktor okna. Inicializuje GUI a vytvoří instanci 3D plátna.
- `Terrain3DForm()` – Destruktor okna, zajišťuje uvolnění prostředků.
- `void setPoints(const std::vector<QPoint3DF> &points)` – Předá množinu bodů do interní komponenty `Terrain3DCanvas`, kde dojde k jejich vykreslení.

9.11 Třída Terrain3DCanvas

Třída zajišťuje základní 3D vizualizaci digitálního modelu terénu formou ortogonální projekce. Umožňuje zobrazení bodů v prostoru a interaktivní manipulaci pomocí myši (otáčení) a kolečka (zoom).

Veřejné metody:

- `explicit Terrain3DCanvas(QWidget *parent = nullptr)` – Konstruktor třídy, nastavuje výchozí úhel rotace a měřítko vizualizace.
- `void setPoints(const std::vector<QPoint3DF> &points)` – Předává objektu množinu 3D bodů, které budou zobrazeny na plátně.

Překryté události:

- `void paintEvent(QPaintEvent *event)` – Vykreslí všechny body na základě aktuální rotace a přiblížení. Používá 2D projekci s otočením podle hodnot `angleX` a `angleY`.
- `void mousePressEvent(QMouseEvent *event)` – Ukládá pozici myši při stisknutí tlačítka – používá se pro výpočet posunu při rotaci scény.
- `void mouseMoveEvent(QMouseEvent *event)` – Zajišťuje interaktivní otáčení terénu podle pohybu myši.
- `void wheelEvent(QWheelEvent *event)` – Umožňuje přiblížení nebo oddálení scény pomocí kolečka myši.

Privátní metoda:

- `QPoint projectPoint(const QPoint3DF &p) const` – Provádí převod 3D souřadnice do 2D roviny obrazovky pomocí jednoduché ortogonální projekce a aktuálních transformačních parametrů.

10 Závěr

V rámci této úlohy byla vytvořena aplikace, která umožňuje analýzu terénu. Aplikace dokáže analyzovat data ručně zadaná do aplikace, automaticky generovaná aplikací, nebo načítáním dat z formátu `.txt` nebo `.xyz`. Výsledky jsou reprezentovány ve 2D pomocí TIN, vrstevnic, sklonu a orientace terénu. Body terénu mohou být zobrazeny ve 3D prohlížeči. Vzhledem k časovému vytížení v rámci jiných předmětů, zejména projektu z kartografie, jsme nestihli zpracovat všechny bonusové úlohy, které bychom si jinak rádi splnili.

10.1 Další Možné Neřešené Problémy a Náměty na Vylepšení

- **Podpora dalších formátů pro vstup:** V současné verzi aplikace je možné načítat bodová data pouze z formátů `TXT` a `XYZ`, přičemž jako oddělovač slouží čárka nebo mezera. Do budoucna by bylo vhodné rozšířit možnosti importu o podporu volitelného nastavení formátu (včetně výběru oddělovače, hlaviček apod.) a přidat podporu běžně používaných formátů pro práci s mračenými body, jako jsou `LAS` a `LAZ`.
- **Možnost exportu dat:** Aplikace aktuálně neumožňuje export výsledků. Do budoucna by bylo vhodné implementovat podporu pro ukládání výsledků do běžně používaných GIS formátů, jako je `SHP` nebo `GeoPackage`, a zároveň umožnit export vizualizací do rastrových (např. `TIFF`) nebo vektorových (např. `SVG`) formátů.
- **Dávkové zpracování:** Dalším vylepšením by mohlo být aplikaci kromě zpracování v GUI umožnit i dávkové zpracování mračených bodů v příkazové řádce, které by mohlo vypadat například takto:
`DMT.exe -TIN mracno.txt > tin.shp`
`DMT.exe -contourLines mracno.xyz > vrstevnice.shp.`
- **Zoomování a posouvání mapy:** Aplikace v současnosti nepodporuje přibližování a pohyb v mapovém okně.
- **Povinné hrany** Možnost nastavení povinných hran.
- **Nastavení:** Přidat další možnosti nastavení, kde se budou moc nastavit vlastní parametry barevného zobrazení sklonu a orientace terénu.
- **Vizuál aplikace:** Zlepšit vizuální stránku aplikace.

Odkazy

- [1] Tomáš Bayer. *Algoritmy v digitální kartografii*. 2008. vyd. Praha: Karolinum, 2008. ISBN: 978-80-246-1499-1.
- [2] Český úřad zeměměřický a katastrální. *Geoportál ČÚZK*. <https://geoportal.cuzk.cz/>. Přístup: květen 2025. 2024.