<table>
<tr><td colspan="6">ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA STAVEBNÍ
KATEDRA GEOMATIKY</td></tr>
<tr><td colspan="6">Název předmětu
Algoritmy digitální kartografie a GIS</td></tr>
<tr><td>Úloha
U3</td><td colspan="5">Název úlohy:
Digitální model terénu</td></tr>
<tr><td>akademický rok
2024/2025</td><td>semestr
letní</td><td>studijní skupina
C102</td><td>vypracoval
Matyáš Pokorný
Tereza Černohousová</td><td>datum
17.05.2025</td><td>klasifikace</td></tr>
</table>

Technická zpráva

1 Zadání

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

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

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

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

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

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

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

2 Pracovní postup a použité algoritmy

2.1 Delaunayova triangulace

Delaunayova triangulace je základní metodou v oblasti výpočetní geometrie, která se používá pro tvorbu sítě trojúhelníků z množiny bodů v rovině nebo prostoru. Cílem triangulace je vytvořit síť tak, aby byly splněny určité geometrické podmínky, přičemž Delaunayova triangulace patří mezi nejčastěji používané typy díky svým výhodným vlastnostem.

2.1.1 Princip triangulace

Delaunayova triangulace zajišťuje, že pro každé tři body tvořící trojúhelník neexistuje žádný další bod z dané množiny, který by ležel uvnitř kružnice opsané tomuto trojúhelníku. Tato podmínka se označuje jako **pravidlo prázdného kruhu**.

2.1.2 Výpočetní postup a vysvětlení algoritmu

Výpočetní postup Delaunayovy triangulace začíná převodem jednotlivých hran na trojúhelníky. Každé tři po sobě jdoucí hrany tvoří jeden trojúhelník, přičemž pro každý z nich jsou definovány tři vrcholy, které odpovídají počátečním a koncovým bodům těchto hran. V tomto algoritmu jsou hrany reprezentovány objekty typu `Edge`, které uchovávají potřebné informace o spojovaných bodech.

Samotná Delaunayova triangulace je konstruována pomocí inkrementální metody. Nejprve je z celé množiny vstupních bodů vybrán tzv. pivot – bod s nejmenší souřadnicí x. Následně se k tomuto bodu najde nejbližší soused, čímž vznikne počáteční hrana, která dále slouží jako základ pro rozvíjení triangulace.

Pomocí této počáteční hrany se začínají hledat další body, které tvoří vhodné trojúhelníky. Pro každou aktuálně zpracovávanou hranu se hledá třetí bod, jenž spolu s touto hranou tvoří trojúhelník splňující Delaunayovu podmínku – tedy takový, že žádný jiný bod z množiny neleží uvnitř kruhu opsaného tomuto trojúhelníku.

Pokud je takový bod nalezen, vytvoří se tři nové hrany spojující vrcholy vzniklého trojúhelníku. Tyto nové hrany jsou následně testovány, zda již nejsou součástí triangulace v opačném směru. Pokud nejsou, jsou přidány k dalšímu zpracování. Pokud se již vyskytují, jsou odstraněny, čímž

se zajistí, že každá hrana je součástí právě dvou trojúhelníků – jednoho z každé strany.

Tento postup se opakuje tak dlouho, dokud nezůstanou žádné další hrany ke zpracování. Výsledkem je množina hran, která tvoří konečnou podobu Delaunayovy triangulace dané množiny bodů.

2.2 Konstrukce vrstevnic

Konstrukce vrstevnic (izolinií) je důležitou součástí analýzy digitálního modelu terénu (DTM), která umožňuje vizuálně interpretovat výškové rozdíly v terénu formou čar stejné nadmořské výšky. Každá vrstevnice spojuje body se stejnou výškou z a vzniká jako průnik vodorovné roviny o výšce z s povrchem reprezentovaným pomocí trojúhelníků vytvořených Delaunayovou triangulací.

2.2.1 Výpočetní postup a vysvětlení algoritmu

Vrstevnice jsou generovány v definovaném výškovém intervalu dz, v rozsahu od $z_{\min}$ do $z_{\max}$. Pro každou hodnotu z z tohoto intervalu procházíme všechny trojúhelníky a určujeme, zda rovina výšky z protíná daný trojúhelník. Pokud ano, určíme průsečíky roviny s jednotlivými stranami trojúhelníka a vytvoříme úsečku (hranici vrstevnice), která je do výsledné sady vrstevnic uložena.

Samotný výpočet průsečíku vrstevnice s hranou trojúhelníka probíhá pomocí lineární interpolace mezi dvěma body $\mathbf{p}_1 = (x_1, y_1, z_1)$ a $\mathbf{p}_2 = (x_2, y_2, z_2)$, kde $z_1 \neq z_2$. Výsledný bod $\mathbf{p}_b = (x_b, y_b, z)$ na výšce z spočítáme ze vzorců:

$$x_b = \left(\frac{x_2 - x_1}{z_2 - z_1}\right)(z - z_1) + x_1,$$
$$y_b = \left(\frac{y_2 - y_1}{z_2 - z_1}\right)(z - z_1) + y_1.$$

Pomocí těchto vztahů je možné určit souřadnice průsečíkového bodu vrstevnice s hranou. V rámci implementace se následně kontrolují všechny tři strany každého trojúhelníka a podle kombinace, které dvě strany rovina protíná, se vytvoří odpovídající úsečka vrstevnice.

Výsledkem celého postupu je množina úseček (hran) reprezentujících vrstevnice pro zadané výškové úrovně.

Sklon a orientace svahu

Sklon a orientace svahu (*slope* a *aspect*) jsou dvě základní morfometrické charakteristiky terénu, které lze jednoduše odvodit z digitálního modelu terénu (DTM) reprezentovaného pomocí trojúhelníků získaných z Delaunayovy triangulace.

Sklon svahu

Sklonem svahu rozumíme úhel mezi normálovým vektorem roviny trojúhelníka a svislou osou z. Body $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3$ jsou vrcholy trojúhelníka a směrové vektory dvou hran lze zapsat jako:

$$\vec{u} = \mathbf{p}_3 - \mathbf{p}_2 = (u_x, u_y, u_z),$$
$$\vec{v} = \mathbf{p}_1 - \mathbf{p}_2 = (v_x, v_y, v_z).$$

Normálový vektor $\vec{n}$ roviny definované těmito dvěma vektory pak získáme jako jejich vektorový součin:

$$n_x = u_y v_z - u_z v_y,$$
$$n_y = -(u_x v_z - u_z v_x),$$
$$n_z = u_x v_y - u_y v_x.$$

Sklon svahu θ je následně definován jako:

$$\theta = \arccos\left(\frac{n_z}{\|\vec{n}\|}\right), \quad \text{kde } \|\vec{n}\| = \sqrt{n_x^2 + n_y^2 + n_z^2}.$$

Tento výpočet se provede pro každý trojúhelník v modelu, přičemž výsledkem je hodnota sklonu v radiánech.

Orientace svahu

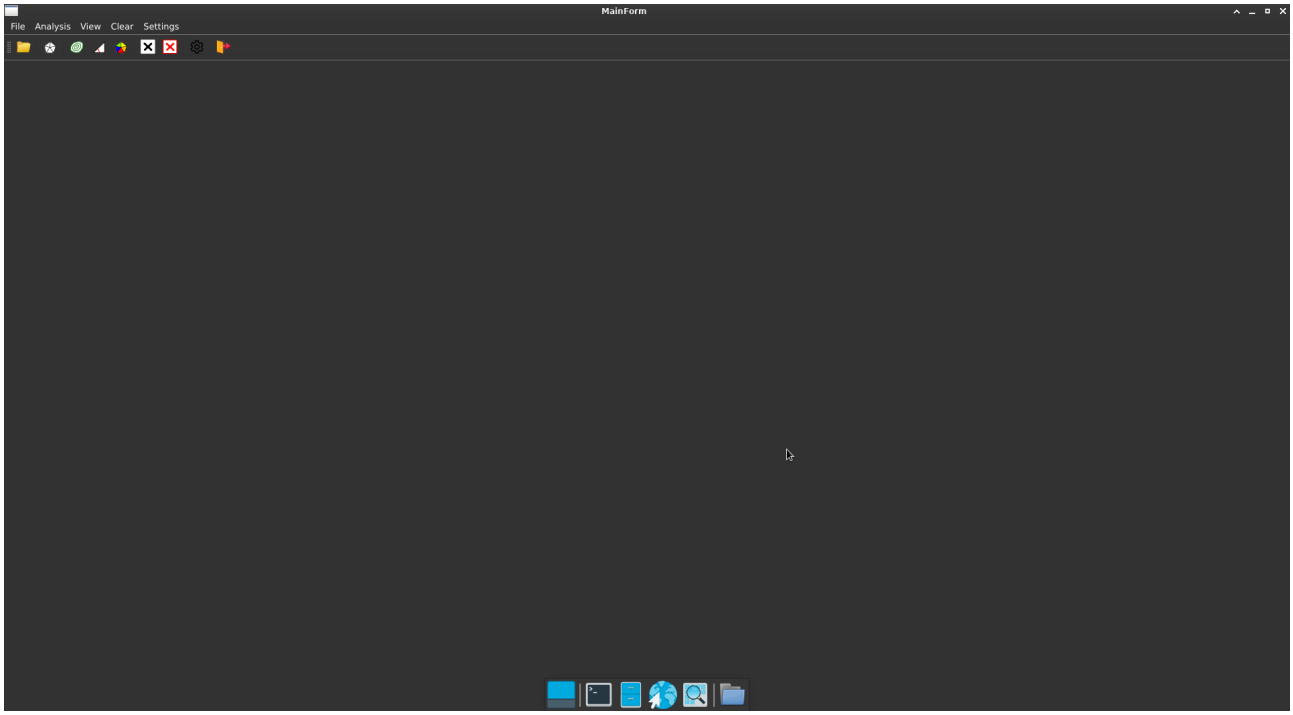
Orientace svahu (aspect) vyjadřuje směr, kterým je svah orientován, neboli azimut horizontální projekce normálového vektoru. Pro její výpočet stačí horizontální složky normály (n_x, n_y) a azimut ϕ se pak určí jako:

$$\phi = \arctan 2(n_y, n_x),$$

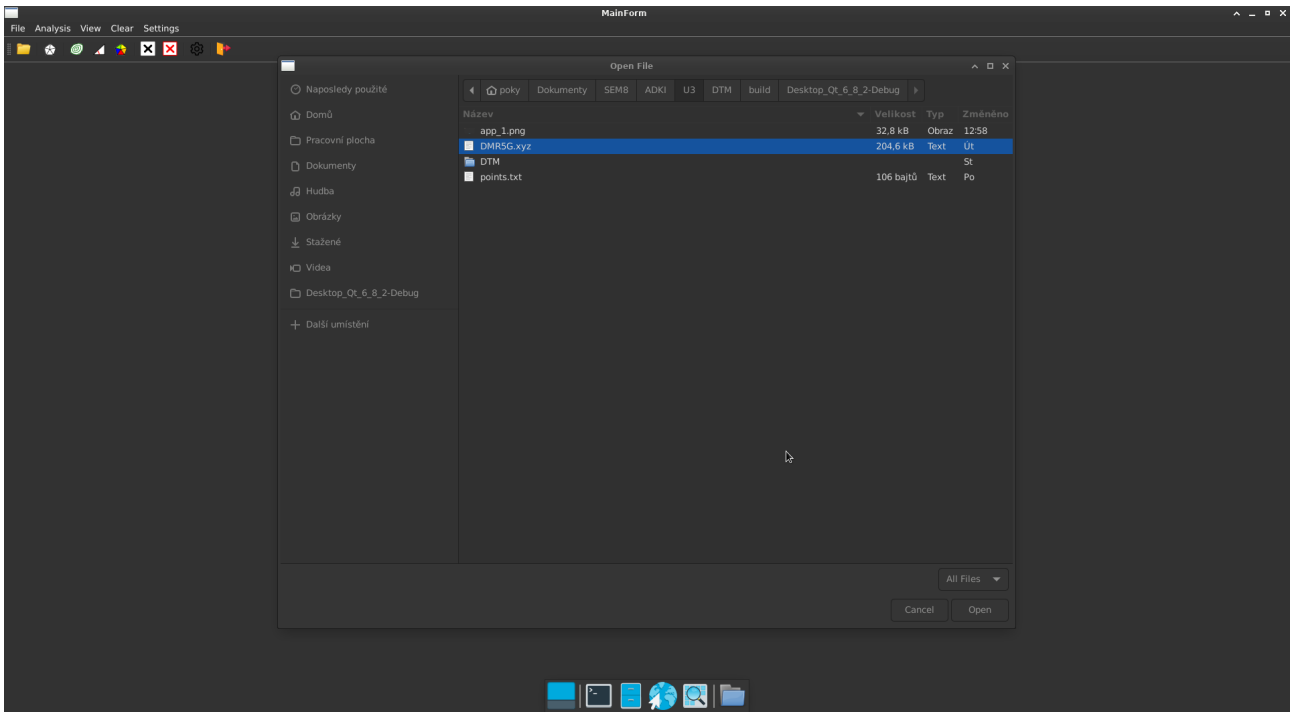
kde funkce $\arctan 2$ zajišťuje správný výsledek v celém rozsahu $\langle -\pi, \pi \rangle$.

3 Zhodnocení aplikace

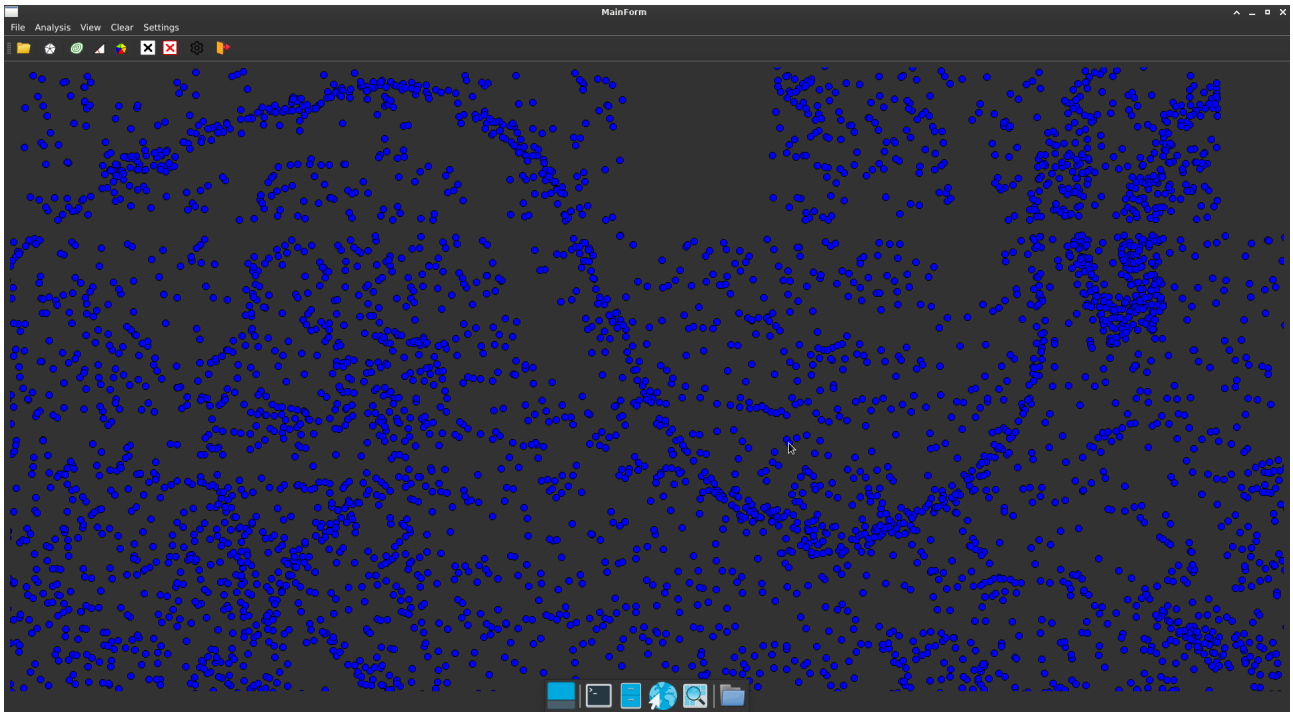
Po spuštění zdrojového kódu je otevřeno grafické okno aplikace. Aplikace se otevře prázdná. V horní části aplikace se nachází lišta s nástroji, které jsou k dispozici. Data (body), které jsou pro aplikaci a výpočty stěžejní je možné nahrát několika způsoby. Prvním je uživatelský vstup – uživatel může sám „naklikat“ body, kterým bude automaticky přidělena nějaká výška (hodnota souřadnice Z). Lze také nahrát seznam souřadnic v různých formátech. Například jako TXT nebo XYZ.



Obrázek 1: Vzhled aplikace

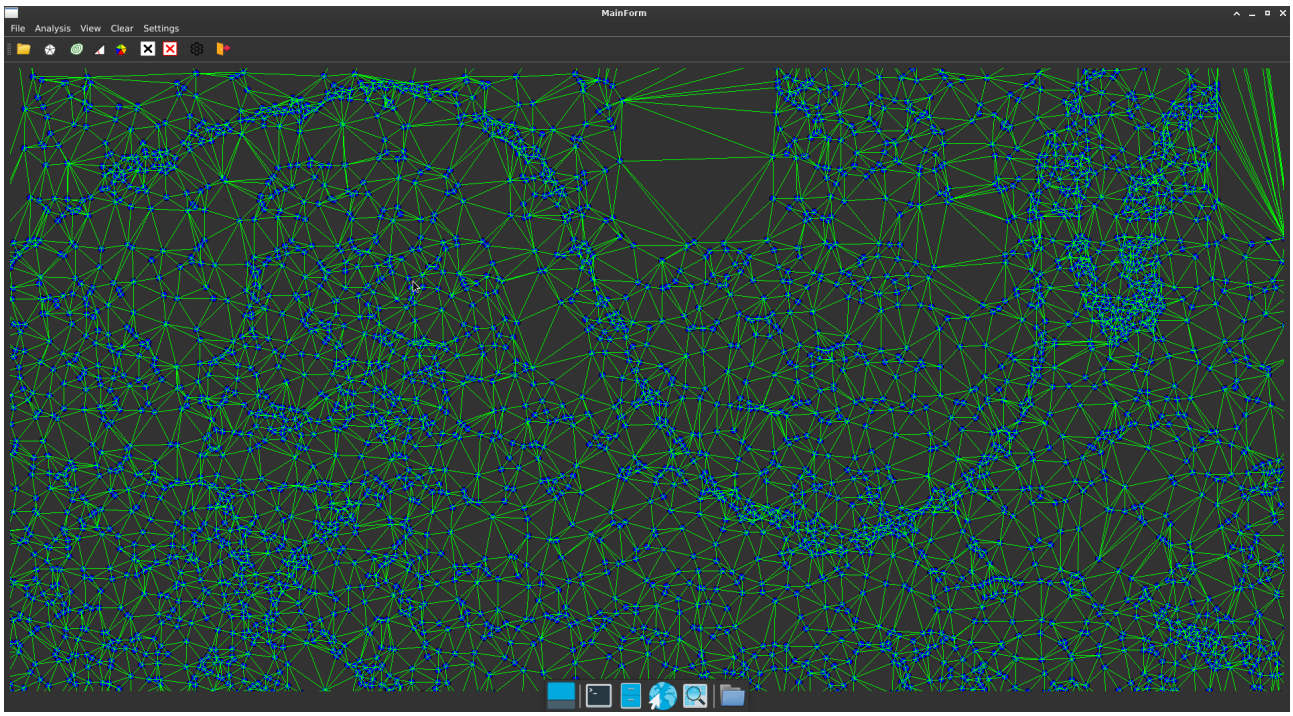


Obrázek 2: Import mračna bodů



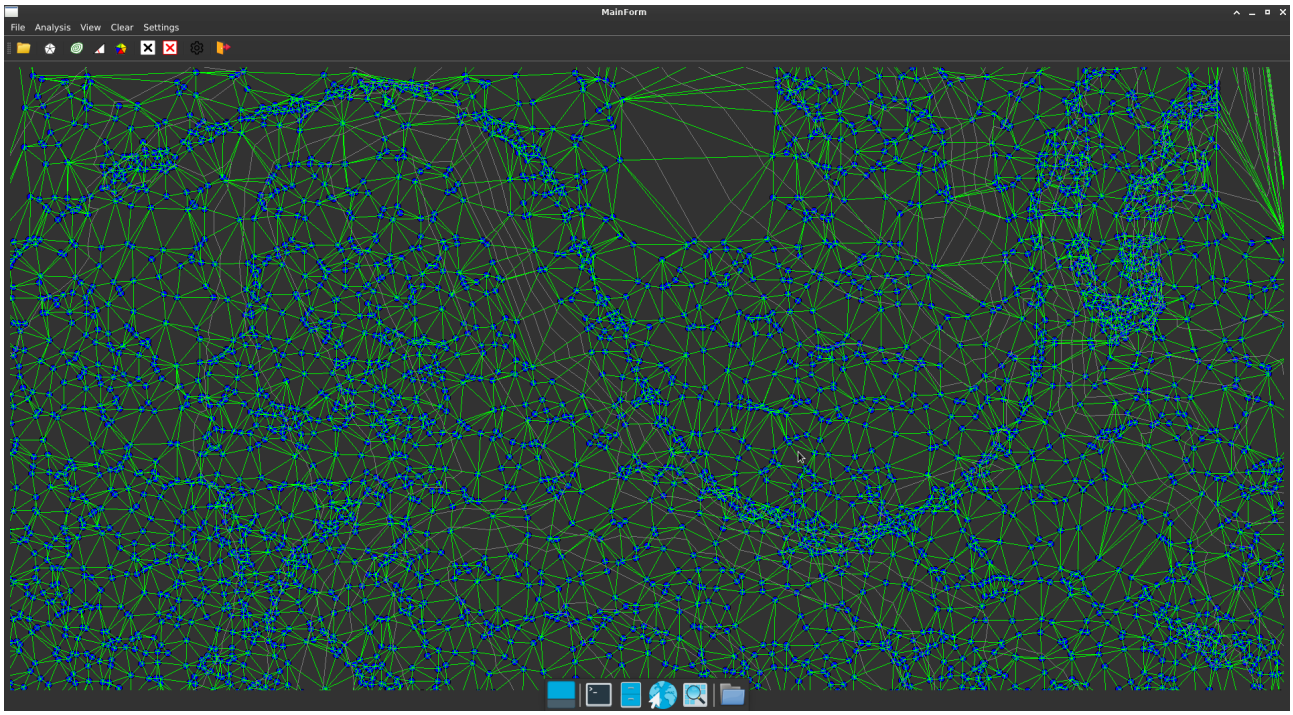
Obrázek 3: Mračno bodů

Nad bodovým mračnem lze provádět několik operací. Zaprvé je možné vytvořit trojúhelníkovou síť pomocí Delaunayho triangulace.

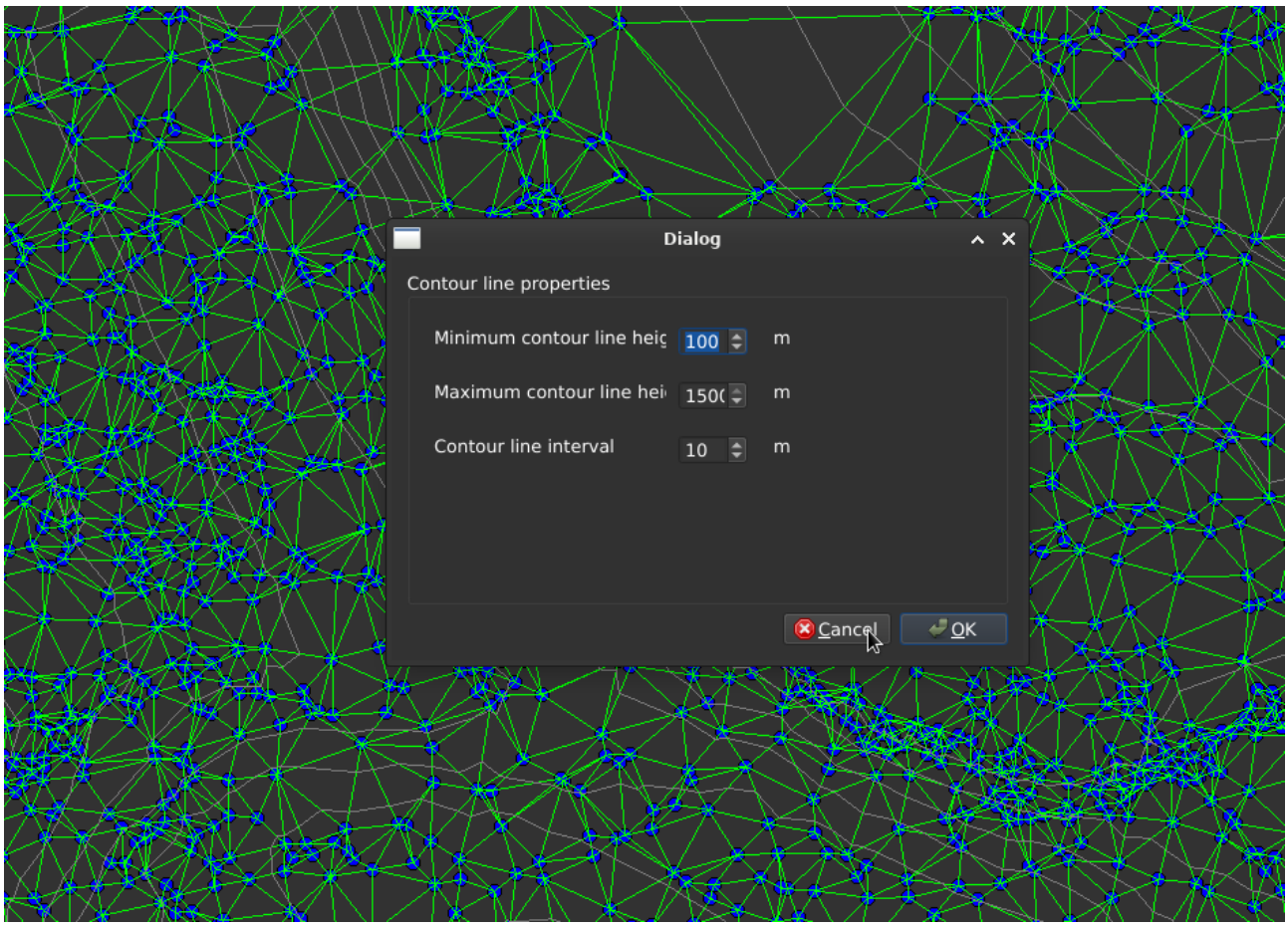


Obrázek 4: Delaunayho triangulace

Další funkcí je generování vrstevnic, které se nejprve vygenerují dle přednastavených hodnot v algoritmu. Pokud je ale potřeba tyto parametry upravit, lze spustit dialogové okno a nastavit několik vlastností pro vrstevnice. Například minimální/maximální výšku nebo interval rozestupu vrstevnic.

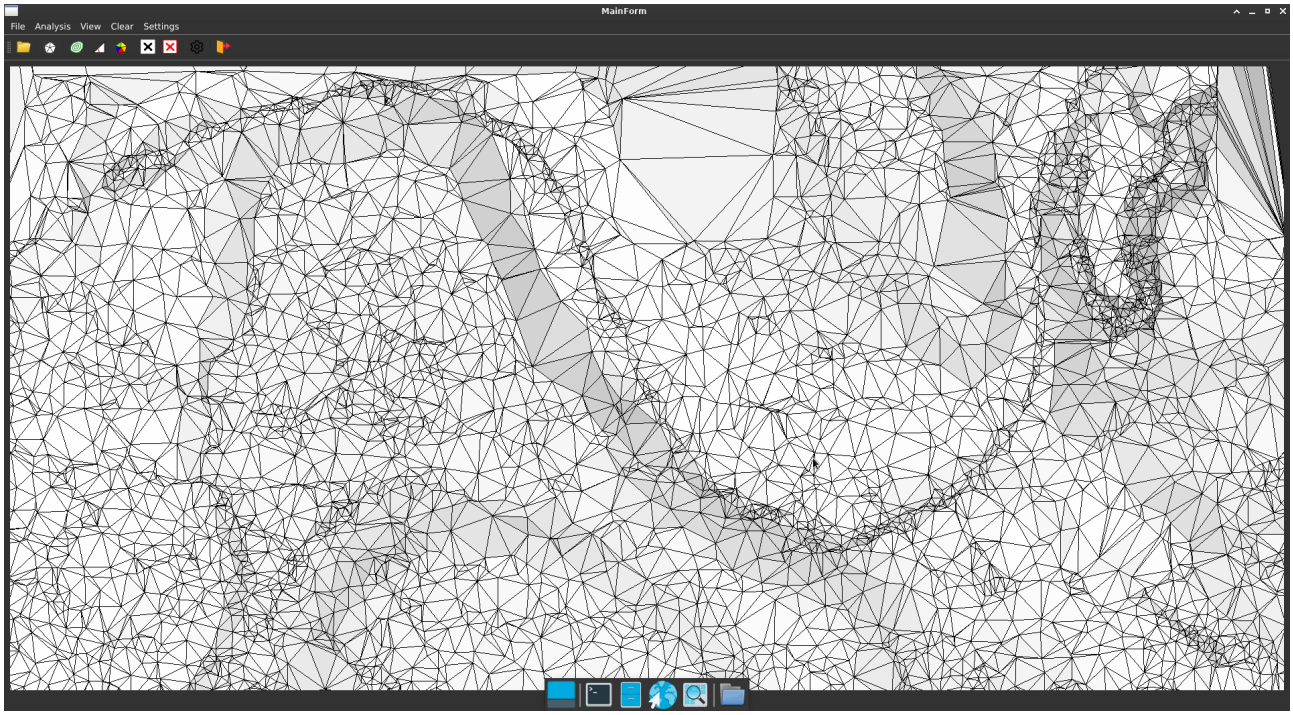


Obrázek 5: Vrstevnice



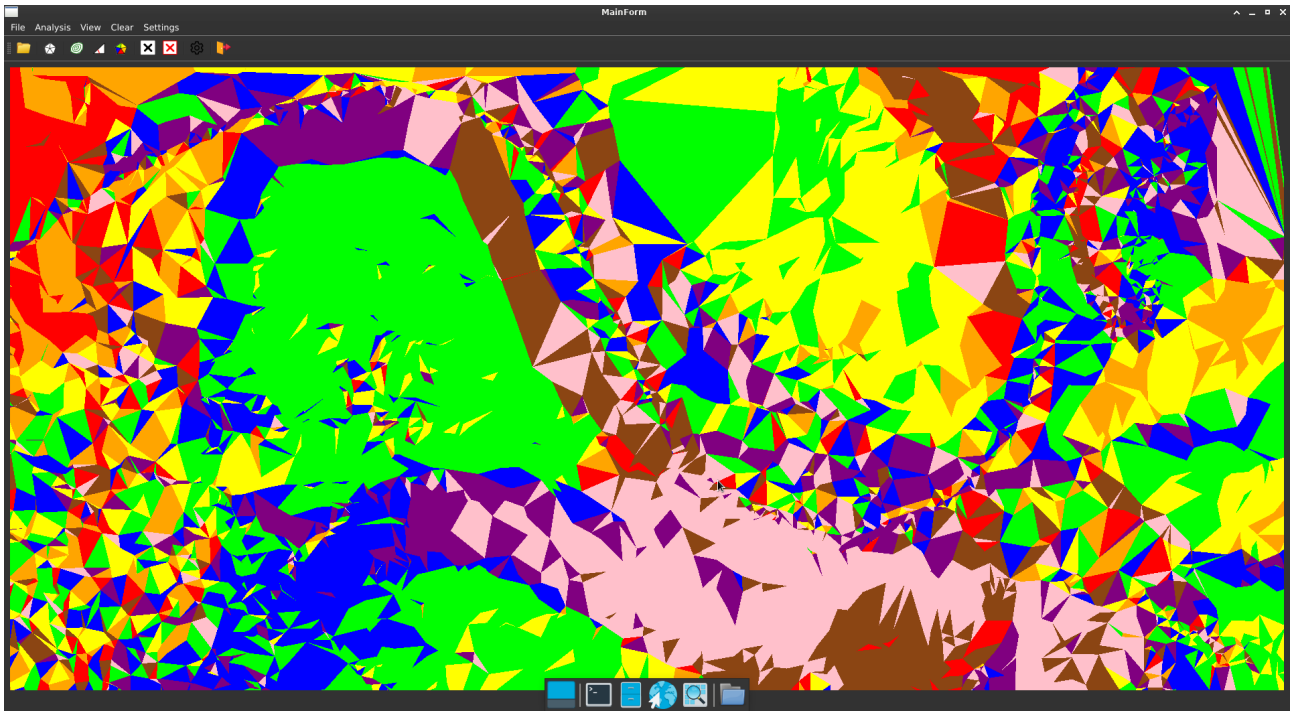
Obrázek 6: Nastavení vrstevnic

Třetí možnou funkcí je zobrazení sklonu terénu a to pomocí stupňů šedi.



Obrázek 7: Sklon terénu

Nebo je možné zobrazit orientaci svahu.

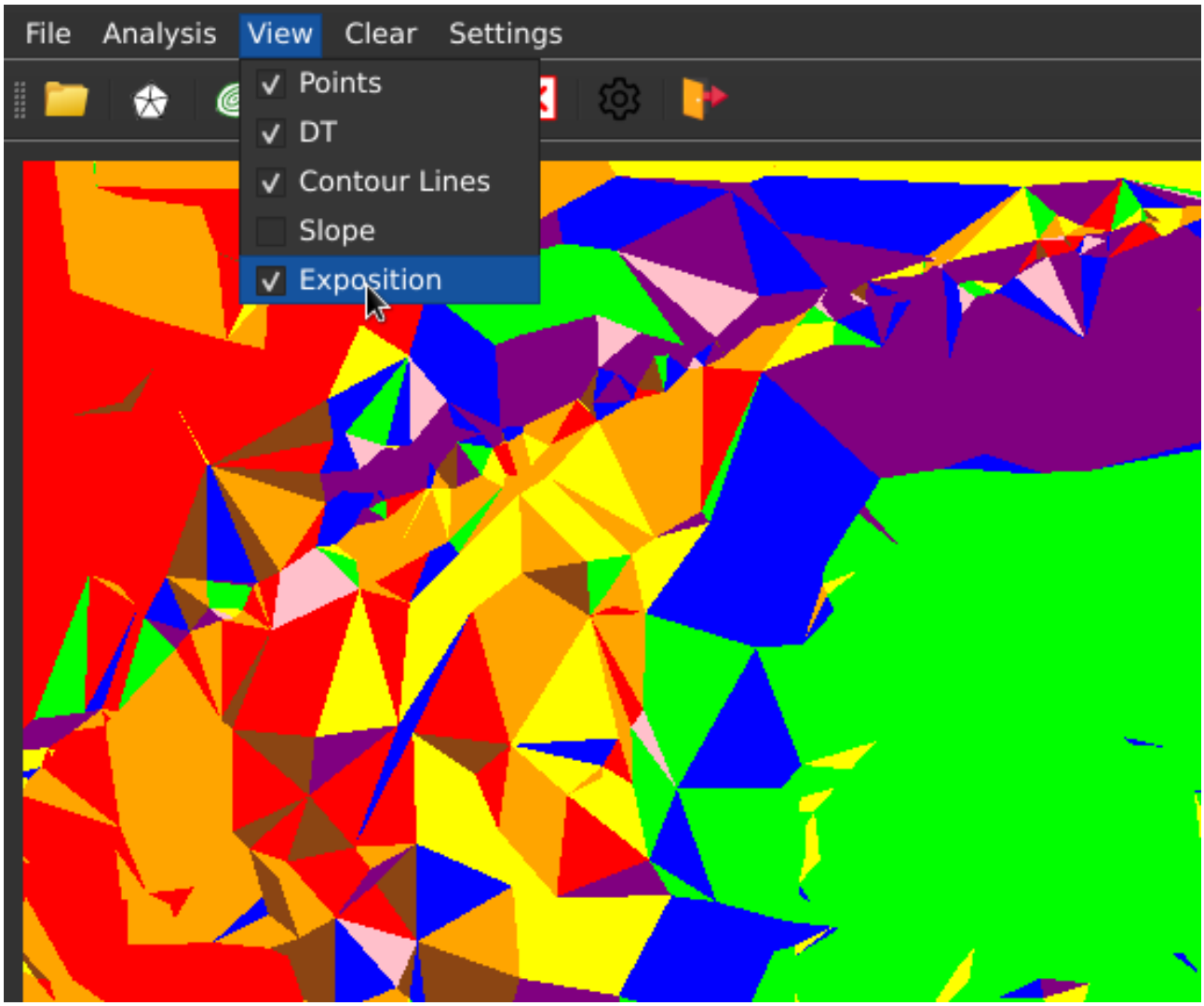


Obrázek 8: Orientace svahu

Jakou orientaci jednotlivé barvy představují znázorňuje tato tabulka.

Směr orientace	**Barva**
Sever (North)	
Severovýchod (Northeast)	
Východ (East)	
Jihovýchod (Southeast)	
Jih (South)	
Jihozápad (Southwest)	
Západ (West)	
Severozápad (Northwest)	

Aplikace nabízí i možnost přepínání toho, co je právě zobrazeno.



Obrázek 9: Možnosti zobrazení

4 Návrhy na vylepšení

Tato vytvořená aplikace je velmi jednoduchá, ale splňuje základní požadavky na výpočet a vizualizaci Delaunayho triangulace, sklonu a orientace terénu. Je ale samozřejmé, že tu je prostor na zlepšení. Zde je několik návrhů:

- Možnost „zoomovat“
- Zobrazení legendy pro barevné zobrazení sklonu a orientace svahu
- Zvýraznění některých vrstevnic
- Přidání popisu vrstevnic dle kartografických pravidel

- Zobrazení souřadic ve spoední liště aplikace nebo pod kurzorem
- Uložení výsledku jako PNG apod.
- Zvýraznění trojúhelníku, když na něj najede myš
- Pop-up po kliknutí na trojúhelník, které se zobrazí hodnoty pro sklon a orientaci svahu

5 Závěr

Byla vytvořena aplikace, která umožňuje import mračna bodů, nad kterým je možné provádět různé funkce. Nad mračnem bodů je možné vytvořit trojúhelníkovou síť podle Delaunayho triangulace. Dále je možné vypočítat a vizualizovat vrstevnice. Pro trojúhelníkovou síť je také možné zobrazit sklon a orientaci svahu pomocí barevné stupnice.

V Praze dne: 17.05. 2025

T. Černohousová

M. Pokorný

Pseudokódy

algorithms.cpp

Algorithm 1 Normalizace bodů podle středu a velikosti plátna

Vstup: Seznam bodů *points*, šířka *width*, výška *height*
Výstup: Upravené body *points*, centrované a zrcadlené podle osy y
if *points* je prázdný **then**
    vrátit
end if
$centroid \leftarrow$ **vypočtiCentroid**$(points)$
$offsetX \leftarrow \frac{width}{2.0} - centroid_x$
$offsetY \leftarrow \frac{height}{2.0} - centroid_y$
for každý bod p v *points* **do**
    $p_x \leftarrow p_x + offsetX$
    $p_y \leftarrow height - (p_y + offsetY)$
end for
vrátit *points*

Algorithm 2 Výpočet těžiště bodů

Vstup: Seznam bodů *points*
Výstup: Těžiště *centroid*
if *points* je prázdný **then**
    vrátit bod $(0, 0)$
end if
$totalX \leftarrow 0$
$totalY \leftarrow 0$
for každý bod p v *points* **do**
    $totalX \leftarrow totalX + p_x$
    $totalY \leftarrow totalY + p_y$
end for
$centroid \leftarrow \left(\frac{totalX}{|points|}, \frac{totalY}{|points|}\right)$
vrátit *centroid*

Algorithm 3 Analýza pozice bodu vzhledem k přímce

function GetPointAndLinePosition($p, p1, p2$)
    $ux \leftarrow p2.x() - p1.x()$
    $uy \leftarrow p2.y() - p1.y()$
    $vx \leftarrow p.x() - p1.x()$
    $vy \leftarrow p.y() - p1.y()$
    $t \leftarrow ux \cdot vy - uy \cdot vx$
    if $t > \epsilon$ **then**
        return 1 ▷ Bod je vlevo od přímky
    else if $t < -\epsilon$ **then**
        return 0 ▷ Bod je vpravo od přímky
    else
        return -1 ▷ Bod je na přímce
    end if
end function

Algorithm 4 Výpočet úhlu mezi dvěma přímkami

function Get2LinesAngle($p1, p2, p3, p4$)
    $ux \leftarrow p2.x() - p1.x()$
    $uy \leftarrow p2.y() - p1.y()$
    $vx \leftarrow p4.x() - p3.x()$
    $vy \leftarrow p4.y() - p3.y()$
    $dot \leftarrow ux \cdot vx + uy \cdot vy$
    $n_u \leftarrow \sqrt{ux^2 + uy^2}$
    $n_v \leftarrow \sqrt{vx^2 + vy^2}$
    return $\cos^{-1}\left(\frac{dot}{n_u \cdot n_v}\right)$
end function

Algorithm 5 Vyhledání optimálního Delaunay bodu

1: **function** FINDDELAUNAYPOINT($p1, p2, points$)
2: $idx_{max} \leftarrow -1$
3: $\omega_{max} \leftarrow 0$
4: **for** každý bod p_i v $points$ **do**
5: **if** GETPOINTANDLINEPOSITION($p_i, p1, p2$) $= 1$ **then**
6: $\omega \leftarrow$ GET2LINESANGLE($p_i, p1, p_i, p2$)
7: **if** $\omega > \omega_{max}$ **then**
8: $\omega_{max} \leftarrow \omega$
9: $idx_{max} \leftarrow i$
10: **end if**
11: **end if**
12: **end for**
13: **return** idx_{max}
14: **end function**

Algorithm 6 Výpočet 2D Eukleidovské vzdálenosti

1: **function** GET2DDISTANCE($p1, p2$)
2: $dx \leftarrow p1.x() - p2.x()$
3: $dy \leftarrow p1.y() - p2.y()$
4: **return** $\sqrt{dx^2 + dy^2}$
5: **end function**

Algorithm 7 Vyhledání nejbližšího bodu k bodu p

```
1: function FindNearestPoint(p, points)
2:     idx_min ← −1
3:     d_min ← ∞
4:     for každý bod p_i v points do
5:         if p ≠ p_i then
6:             d ← Get2DDistance(p, p_i)
7:             if d < d_min then
8:                 d_min ← d
9:                 idx_min ← i
10:            end if
11:        end if
12:    end for
13:    return idx_min
14: end function
```

Algorithm 8 Vytvoření Delaunay triangulace

```
1: function DT(points)
2:     dt ← prázdný seznam hran
3:     ael ← prázdný seznam aktivních hran
4:     p1 ← bod s nejmenší x-souřadnicí z points
5:     p2 ← FindNearestPoint(p1, points)
6:     Vytvoř hrany e1 = (p1, p2) a e2 = (p2, p1)
7:     Přidej e1 a e2 do ael
8:     while ael není prázdný do
9:         e ← poslední hrana z ael
10:        Odstraň e z ael
11:        es ← změněná orientace hrany e
12:        idx ← FindDelaunayPoint(es.start, es.end, points)
13:        if idx ≠ −1 then
14:            p_del ← points[idx]
15:            Vytvoř nové hrany e2s = (es.end, p_del) a e3s = (p_del, es.start)
16:            Přidej es, e2s, e3s do dt
17:            UpdateAEL(e2s, ael)
18:            UpdateAEL(e3s, ael)
19:        end if
20:    end while
21:    return dt
22: end function
```

Algorithm 9 Aktualizace seznamu aktivních hran (AEL)

```
1: function UpdateAEL(e, ael)
2:     es ← změněná orientace hrany e
3:     if es není v ael then
4:         Přidej e do ael
5:     else
6:         Odstraň es z ael
7:     end if
8: end function
```

Algorithm 10 Výpočet bodu na konturové čáře

1: **function** CountourLinePoint($p1, p2, z$)

2: $xb \leftarrow \frac{(p2.x()-p1.x())}{(p2.z()-p1.z())} \cdot (z - p1.z()) + p1.x()$

3: $yb \leftarrow \frac{(p2.y()-p1.y())}{(p2.z()-p1.z())} \cdot (z - p1.z()) + p1.y()$

4: **return** $QPoint3DF(xb, yb, z)$

5: **end function**

Algorithm 11 Vytvoření vrstevnic pro zadané výšky

```
1: function CreateContourLines(dt, zmin, zmax, dz)
2:     contour_lines ← prázdný seznam
3:     for z ← zmin to zmax s krokem dz do
4:         for každý trojúhelník t v dt do
5:             dz1 ← z − p1.z()
6:             dz2 ← z − p2.z()
7:             dz3 ← z − p3.z()
8:             if dz1 == 0 and dz2 == 0 and dz3 == 0 then
9:                 Pokračuj
10:            end if
11:            if dz1 == 0 and dz2 == 0 then
12:                Přidej hranu t do contour_lines
13:            end if
14:            if dz2 == 0 and dz3 == 0 then
15:                Přidej hranu t do contour_lines
16:            end if
17:            if dz3 == 0 and dz1 == 0 then
18:                Přidej hranu t do contour_lines
19:            end if
20:            if (dz1 · dz2 ≤ 0) and (dz2 · dz3 < 0) then
21:                a ← CountourLinePoint(p1, p2, z)
22:                b ← CountourLinePoint(p2, p3, z)
23:                Vytvoř novou hranu e(a, b)
24:                Přidej hranu e do contour_lines
25:            end if
26:        end for
27:    end for
28:    return contour_lines
29: end function
```

Algorithm 12 Výpočet sklonu trojúhelníku

function ComputeSlope($p1, p2, p3$)
    $ux \leftarrow p3.x() - p2.x()$
    $uy \leftarrow p3.y() - p2.y()$
    $uz \leftarrow p3.z() - p2.z()$
    $vx \leftarrow p1.x() - p2.x()$
    $vy \leftarrow p1.y() - p2.y()$
    $vz \leftarrow p1.z() - p2.z()$
    $nx \leftarrow uy \cdot vz - uz \cdot vy$
    $ny \leftarrow -(ux \cdot vz - uz \cdot vx)$
    $nz \leftarrow ux \cdot vy - uy \cdot vx$
    $n \leftarrow \sqrt{nx^2 + ny^2 + nz^2}$
    return $\cos^{-1}\left(\frac{nz}{n}\right)$
end function

Algorithm 13 Analýza sklonu terénu

function AnalyzeSlope($dt, triangles$)
    for každý trojúhelník t v $triangles$ **do**
        $slope \leftarrow$ ComputeSlope($t.getP1(), t.getP2(), t.getP3()$)
        $t.setSlope(slope)$
    end for
end function

Algorithm 14 Výpočet orientace svahu

function ComputeAspect($p1, p2, p3$)
    $ux \leftarrow p3.x() - p2.x()$
    $uy \leftarrow p3.y() - p2.y()$
    $uz \leftarrow p3.z() - p2.z()$
    $vx \leftarrow p1.x() - p2.x()$
    $vy \leftarrow p1.y() - p2.y()$
    $vz \leftarrow p1.z() - p2.z()$
    $nx \leftarrow uy \cdot vz - uz \cdot vy$
    $ny \leftarrow -(ux \cdot vz - uz \cdot vx)$
    return atan2(ny, nx) ▷ Výpočet aspektu trojúhelníku
end function

Algorithm 15 Analýza orientace svahu

```
1: function AnalyzeAspect(dt, triangles)
2:     if triangles je prázdný then
3:         EdgesToTriangles(dt, triangles)
4:     end if
5:     for každý trojúhelník t v triangles do
6:         aspect ← ComputeAspect(t.getP1(), t.getP2(), t.getP3())
7:         t.setSlope(aspect)
8:     end for
9: end function
```

Algorithm 16 Konverze Delaunay hran na trojúhelníky

```
1: function EdgesToTriangles(dt, triangles)
2:     for i ← 0 to dt.size() − 1 with step 3 do
3:         p1 ← dt[i].getStart()
4:         p2 ← dt[i + 1].getStart()
5:         p3 ← dt[i + 2].getStart()
6:         t ← nový trojúhelník (p1, p2, p3)
7:         Přidej t do triangles
8:     end for
9:     return triangles
10: end function
```

draw.cpp

Algorithm 17 Zpracování události kliknutí myší

1: **function** MOUSEPRESSEVENT(e)
2: $x \leftarrow e.x()$
3: $y \leftarrow e.y()$
4: $z \leftarrow$ náhodná hodnota z intervalu $\langle 100, 1100 \rangle$
5: $p \leftarrow$ bod (x, y, z)
6: Přidej p do seznamu *points*
7: Aktualizuj obrazovku (repaint)
8: **end function**

Algorithm 18 Zobrazení grafických prvků

```
1: function paintEvent(event)
2:     Inicializuj kreslící nástroj painter
3:     if view_points then
4:         Nastav barvu výplně na modrou, obrys na černou
5:         for all bod p v points do
6:             Vykresli kruh se středem p a poloměrem r
7:         end for
8:     end if
9:     if view_dt then
10:        Nastav barvu na zelenou
11:        for all hrana e v dt do
12:            Vykresli úsečku z e.start do e.end
13:        end for
14:    end if
15:    if view_contour_lines then
16:        Nastav barvu na tmavě šedou
17:        for all hrana e v contour_lines do
18:            Vykresli úsečku z e.start do e.end
19:        end for
20:    end if
21:    if view_aspect then
22:        for all trojúhelník t v triangles do
23:            Získej vrcholy p1, p2, p3
24:            aspect ← t.getAspect()
25:            aspect_deg ← aspect · 180/π
26:            if aspect_deg < 0 then aspect_deg ← aspect_deg + 360
27:            end if
28:            Zvol barvu podle aspect_deg (8 směrů kompasu)
29:            Nastav barvu výplně a vypni obrys
30:            Vykresli trojúhelník s příslušnou barvou
31:        end for
32:    end if
33:    if view_slope then
34:        for all trojúhelník t v triangles do
35:            Získej vrcholy p1, p2, p3
36:            slope ← t.getSlope()
37:            color ← 255 − 255/π · slope
38:            Nastav barvu QColor(color, color, color)
```

Algorithm 19 Načtení bodů ze souboru

1: **function** POINTSFROMTXT
2: Zobraz dialog pro výběr souboru
3: **if** uživatel nezvolí soubor **then return**
4: **end if**
5: $points0 \leftarrow$ načtené body ze souboru
6: **for all** p v $points0$ **do** Přidej p do $points$
7: **end for**
8: Aktualizuj obrazovku (repaint)
9: **end function**

Algorithm 20 Vymazání všech dat

1: **function** CLEAR
2: Vymaž dt, $points$, $contour_lines$, $triangles$
3: Aktualizuj obrazovku (repaint)
4: **end function**

Algorithm 21 Vymazání výsledků algoritmů

1: **function** CLEARRESULTS
2: Vymaž dt, $contour_lines$, $triangles$
3: Aktualizuj obrazovku (repaint)
4: **end function**