ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

ALGORITMY DIGITÁLNÍ KARTOGRAFIE A GIS

KATEDRA GEOMATIKY

Úloha č. 3: Digitální model terénu

Monika KŘÍŽOVÁ
Marek HOFFMANN

4.12.2021

Obsah

1 Zadání **3**

2 Údaje o bonusových úlohách **3**

3 Popis problému **3**
3.1 Formulace problému 3

4 Popisy algoritmů **4**
4.1 Delaunayova triangulace 4
4.1.1 Inkrementální konstrukce 4
4.2 Delaunayova triangulace nekonvexního polygonu 6
4.3 Konstrukce vrstevnic lineární interpolací 6
4.3.1 Lineární interpolace 6
4.3.2 Algoritmus 6
4.4 Algoritmus výpočtu sklonu trojúhelníku 7
4.5 Algoritmus výpočtu expozice trojúhelníku 8

5 Problematické situace **9**
5.1 Transformace souřadnic 9
5.1.1 Počátek 9
5.1.2 Měřítko 9
5.1.3 Transformace souřadnic 10
5.2 Automatické určování maximální a minimální hodnoty z . . 11
5.3 Automatické generování popisu vrstevnic 12

6 Vstupní data **15**
6.1 Načítání bodů 15
6.2 Načítání polygonu 15

7 Výstupní data **16**

8 Printscreen vytvořené aplikace **24**

9 Dokumentace **25**
9.1 Třída Algorithms 25
9.2 Třída Draw 28
9.3 Třída Widget 33
9.4 Třída Edge 35
9.5 Třída QPoint3D 35

9.6 Třída Triangle 35
9.7 Třída SortByX 36

10 Závěr **36**

1 Zadání

Navrhněte aplikaci s grafickým rozhraním, která nad množinou bodů vytvoří polyedrický digitální model terénu.

Jako vstupní data využijte existující data o alespoň 300 bodech. Nad nimi vytvořte Delaunayovu triangulaci metodou inkrementální konstrukce. Triangulaci využijte pro výpočet a vykreslení vrstevnic. Výslednou triangulaci doplňte o vhodnou vizualizaci sklonu jednotlivých trojúhelníků a jejich expozici.

2 Údaje o bonusových úlohách

Zpracovány byly celkem 3 bonusové úlohy ze 7 zadaných.

- Triangulace nekonvexní oblasti zadané polygonem +5b
- Výběr barevných stupnic při vizualizaci sklonu a expozice. +3b
- Automatický popis vrstevnic respektující kartografické zásady +10b - z části

3 Popis problému

Triangulace je jednou ze základních výpočetních úloh nejen v oblasti digitální kartografie. Kromě tvorby DMT své uplatnění najde i při tvorbě jakékoliv vizualizace prostorových dat ve formě 3D modelů. Dále lze triangulaci využít při detekci otisku prstů či např. modelování přírodních jevů.

3.1 Formulace problému

Mějme množinu bodů P. Triangulací se rozumí převod množiny bodů na trojúhelníkovou síť m trojúhelníků t tak, aby platilo:

- Libovolné 2 trojúhelníky mají společnou nejvýše jednu hranu.
- Sjednocení trojúhelníků je souvislá množina.
- Uvnitř žádného trojúhelníku neleží žádný další bod z P.

Triangulace by měla tvořit pravidelné trojúhelníky blížící se trojúhelníkům rovnostranným a všechny trojúhelníky by se v každém bodě měly co nejlépe přimykat terénu.

4 Popisy algoritmů

4.1 Delaunayova triangulace

Existuje několik druhů triangulace. Nejpoužívanější je však Delaunayova triangulace, která svými kritérii zajišťuje jednoznačnost, spolehlivé výsledky a má optimální časovou složitost. Tyto požadavky jsou následující:

- Uvnitř kružnice k opsanému libovolnému trojúhelníku t_j neleží žádný jiný bod množiny P.
- DT maximalizuje minimální úhel v každém t_j, ale neminimalizuje maximální úhel v t_j.
- DT je lokálně i globální optimální vůči kritériu minimálního úhlu.
- DT je jednoznačná, pokud žádné 4 body neleží na kružnici.

4.1.1 Inkrementální konstrukce

Existuje několik metod konstrukce Delaunayho triangulace. V této úloze však byla implementována metoda inkrementální konstrukce, která je založena na postupném přidávání bodů do již vytvořené triangulace.

Pro konstrukci je používána datová struktura AEL (Active Edge List), která obsahuje hrany e, ke kterým hledáme Delaunayovský bod minimalizující poloměr kružnice mezi Delaunayovskou hranou a hledaným bodem. Delaunayovská hrana je orientovaná a bod hledáme pouze vlevo od ní.

Popis Algoritmu

1. Nalezení bodu q s nejmenší x-ovou souřadnicí
$$q = min(x_i)$$
2. Nalezení nejbližšího bodu k počátečnímu bodu
$$||p_1 - q||_2 = min$$
3. Vytvoření první hrany
$$e = (q, p_1)$$
4. Hledání Delaunayho bodu
$$\underline{p} = argmin_{\forall p_i \in \sigma_L(e)} r'(k_i); k_i = (a, b, p_i); e = (a, b)$$

5. Při neexistenci Delaunayho bodu změna orientace hrany a opakování předchozího kroku

$$\nexists \underline{p} : e \leftarrow (p_1, q);$$

6. Vytvoření zbývajících hran trojúhelníka

$$e_2 = (p_1, \underline{p}); e_3 = (\underline{p}, q)$$

7. Přidání hran do AEL

$$AEL \leftarrow e; AEL \leftarrow e_2; AEL \leftarrow e_3$$

8. Přidání hran do DT

$$DT \leftarrow e; DT \leftarrow e_2; DT \leftarrow e_3$$

9. Dokud není vektor AEL prázdný:

   Změna orientace první hrany z AEL

$$e = (p_1, q)$$

   Hledání Delaunayho bodu k e

$$\underline{p} = argmin_{\forall p_i \in \sigma_L(e)} r'(k_i); k_i = (a, b, p_i); e = (a, b)$$

   Pokud bod existuje

$$if \exists \underline{p}$$

   Tvorba zbývajících hran trojúhelníku

$$e_2 = (p_2, \underline{p}); e_3 = (\underline{p}, q)$$

   Přidání hran do DT

$$DT \leftarrow e; DT \leftarrow e_2; DT \leftarrow e_3$$

   Přidání hrany do AEL, pokud již neexistuje s opačnou orientací.

$$AELif \ \nexists \leftarrow e_2; AEL \leftarrow e_3$$

4.2 Delaunayova triangulace nekonvexního polygonu

Konstrukce Delaunayovy triangulace nekonvexního polygonu je téměř totožná s triangulací tachymetrie. Jedinou výjimkou je nutnost odebrání trojúhelníků s těžištěm ležícím mimo polygon.

Těžiště trojúhelníku se vypočte jako průměr souřadnic vrcholů trojhúhelníku. Pro zjištění, zda leží bod (těžiště) uvnitř polygonu byla použita metoda Winding number algoritmu, jež byla konstruována v první úloze tohoto předmětu, v jejíž dokumentaci je algoritmus podrobně vysvětlen.

4.3 Konstrukce vrstevnic lineární interpolací

Vrstevnice je křivka, která spojuje body se stejnou nadmořskou výškou. Zpravidla bývají vrstevnice vykreslovány s pravidelným intervalem.

Vrstevnice dělíme na základní a hlavní. Základní vrstevnice tvořené s pravidelným základním intervalem bývají vyznačeny tenkou, nepřerušovanou linií a jsou uzavřené. Hlavní vrstevnice je každá pátá základní vrstevnice. V mapách bývá značena tlustou, nepřerušovanou, uzavřenou linií a bývají doplněny popisem.

Dále existují vrstevnice doplňkové, které doplňují základní vrstevnice v těch místech, kde dostatečně nevystihují zejména sklonitý terén, a vrstevnice pomocné, které se používají pro místa s nestálým reliéfem jako např. v oblastech povrchové těžby. Doplňkové ani pomocné vrstevnice nebyly v této aplikaci implementovány.

4.3.1 Lineární interpolace

Lineární interpolace předpokládá:

1. Spád terénu mezi dvěma body, mezi kterými provádíme interpolaci, je konstantní.
2. Rozestup vrstevnic mezi dvěma body je konstantní.

Na vstupu máme trojúhelník t a vodorovnou rovinu ρ o výšce h. Lineární interpolace je založena na analytické geometrii. Hledáme tedy průsečnici roviny T určené trojúhelníkem t a vodorovnou rovinou. Tento výpočet provedeme nad každým trojúhelníkem t

4.3.2 Algoritmus

1. Nalezení průsečíků všech horizontálních rovin s jednotlivými hranami.

Varianty vzájemné polohy ρ a T:

(a) Nemají žádný společný bod - neřešíme.

(b) Průsečnice tvoří jeden bod - neřešíme.

(c) Průsečnice je úsečka:

- Hrana leží v rovině ρ - vrstevnice je přímo v hraně.
- Průsečnice roviny s trojúhelníkem vytvářející nové průsečíky mezi vrcholy trojúhelníku - vrstevnice je vykreslena mezi body vypočtenými následujícími rovnicemi:

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

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

(d) Všechny hrany trojúhelníku leží v rovině ρ - neřešíme.

4.4 Algoritmus výpočtu sklonu trojúhelníku

Sklon představuje jednu z analytických úloh realizovaných nad DMT. Určuje, zdali terén stoupá, klesá nebo je rovinný.

Sklon představuje úhel φ mezi svislicí a normálovým vektorem trojúhelníku. Vypočte se následujícím způsobem.

$$u = (u_x, u_y, u_z)$$

$$v = (v_x, v_y, v_z)$$

$$n = \vec{u} \times \vec{v} = (n_x, n_y, n_z)$$

$$\varphi = arccos\frac{n_z}{|n|}$$

Barevná vizualizace sklonu terénu je vytvořena pomocí interpolace jednotlivých složek barevného schématu RGB. Při tvorbě je vždy nejprve vypočten podíl sklonu trojúhelníku a sklonu maximálního. Tento podíl je následně násoben hodnotou 255 a poté je vždy buď uložen do složky barvy, nebo odečten od hodnoty 255, má-li klesat podíl této složky.

Při volbě vizualizace sklonu (s) v barvách mezi zelenou a červenou barvou je výpočet následující. Zelená barva má složky (0,255,0) a červená (255,0,255), proto je nutné, aby podíl zelené složky s rostoucím sklonem

klesal a podíl červené složky naopak stoupal, podíl modré složky je vždy nulový. Výpočet tedy bude proveden následovně:

$$x = \frac{s}{s_{max}}$$

$$red = x * 255$$

$$green = 255 - x * 255$$

$$blue = 0$$

4.5 Algoritmus výpočtu expozice trojúhelníku

Expozice představuje orientaci terénu vzhledem ke Slunci.

Orientace v bodě je definován jako azimut průmětu normálového vektoru roviny trojúhelníku do roviny x,y.

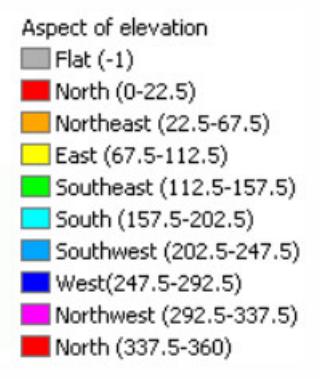
$$u = (u_x, u_y, u_z)$$

$$v = (v_x, v_y, v_z)$$

$$n_t = \vec{u} \times \vec{v} = (n_x, n_y, n_z)$$

$$A = atan2(\frac{n_x}{n_y})$$

Expozice je v případě barevného schématu zobrazována v následujícím barevném schématu.



Obrázek 1: vizualizace expozice terénu

5 Problematické situace

5.1 Transformace souřadnic

Abychom mohli v oknu aplikace zobrazovat budovy, jejíž lomové body jsou určeny v souřadnicovém systému S-JTSK, bylo nutné souřadnic převést do souřadnicového systému widgetu.

5.1.1 Počátek

Pro určení souřadnic počátku bylo nejdříve nutno zjistit maximální hodnotu souřadnice Y a minimální hodnotu souřadnice X v souřadnicovém systému S-JTSK.

```
QString line = in.readLine();
//int id = line.split(" ")[0].toInt();
double y = line.split(" ")[1].toDouble();
double x = line.split(" ")[2].toDouble();
double z = line.split(" ")[3].toDouble();

//Add vertice to the end of the QPoint3D vector
point.setX(x);
point.setY(y);
point.setZ(z);

//Find maximum and minimum coordinates
if (y > y_max)
    y_max = y;
else if (y < y_min)
    y_min = y;
if (x < x_min)
    x_min = x;
else if (x > x_max)
    x_max = x;
if (z < z_min)
    z_min = z;
else if (z > z_max)
    z_max = z;

//Save point to the vector of points
points.push_back(point);
```

Obrázek 2: Výpočet souřadnic počátku

5.1.2 Měřítko

Abychom zobrazili pouze území, kde se polygony nachází, bylo nutné také zavést měřítko.

Měřítko jsme definovali jako podíl šířky/výšky widgetu ku rozdílu maximální a minimální souřadnici v dané ose. Bylo tedy nutné zjistit také minimální hodnotu souřadnice Y a maximální hodnotu souřadnice X.

Měřítko se následně vypočetlo v obou osách, a abychom zabránili tomu, že transformované souřadnice budou mít vyšší hodnotu, než je rozměr widgetu, použilo se měřítko s menším maximálním souřadnicovým rozdílem.

```
//Compute scales to zoom in in canvas
double canvas_weight = 1061.0;
double canvas_height = 777.0;

double dy = fabs(y_max-y_min);
double dx = fabs(x_max-x_min);

double k;
if (dy > dx)
    k = canvas_weight/dy;
else
    k = canvas_height/dx;
```

Obrázek 3: Výpočet měřítka

5.1.3 Transformace souřadnic

Souřadnice v souřadnicovém systému widgetu byly definovány následujícími vzorci.

$$X_{W_i} = -k * (Y_{S-JTSK_i} - Y_{S-JTSK_{\max}})$$

$$Y_{W_i} = k * (X_{S-JTSK_i} - X_{S-JTSK_{\min}})$$

```
//Transform coordinates from JTSK to canvas
for (int unsigned i = 0; i < points.size(); i++)
{
    QPoint3D pol = points[i];

    double temp = points[i].x();
    points[i].setX(-k*(points[i].y()-y_max));
    points[i].setY(k*(temp-x_min));
}
```

Obrázek 4: Transformace souřadnic

5.2 Automatické určování maximální a minimální hodnoty z

Aby nemusel uživatel ručně zadávat maximální a minimální hodnotu z, byla napsána funkce round2num(), která má na vstupu 3 parametry; zaokrouhlované číslo, násobek a směr.

Zaokrouhlované číslo představuje číslo, které chceme zaokrouhlit. Násobek je číslo, na jehož násobek chceme zaokrouhlované číslo zaokrouhlit. Směr je proměnná typu bool. Pokud je rovna true, zaokrouhlujeme nahoru, pokud je rovna false, zaokrouhlujeme dolů.

```
int Draw::round2num(int &numToRound, int &multiple, bool &dir)
{
    //Round to the closest multiple
    //dir == true -> up
    //dir == false -> down

    int remainder = numToRound % multiple;
    if (remainder == 0)
        return numToRound;

    if (dir == true) //round up
    {
        return numToRound + multiple - remainder;
    }
    else    //round down
        return numToRound - remainder;

}
```

Obrázek 5: Funkce round2num()

Ve funkci LoadData() tak nejdříve zjistíme extrémní hodnoty y, poté zaokrouhlíme na celá čísla a poté provedeme funkci round2num() jako zobrazuje následující obrázek.

```
//Round number to closest multiple by 5
z_max = round(z_max);
z_min = round(z_min);

int round = 5; bool up = true; bool down = false;

int z1_max = (int)z_max;
int z1_min = (int)z_min;

z_max = round2num(z1_max, round, up);
z_min = round2num(z1_min, round, down);
```

Obrázek 6: Funkce aplikace round2num() v LoadData()

Jelikož chceme, aby se vrstevnice zobrazovaly ve výškách po pěti metrech, rovná se násobek pěti.

5.3 Automatické generování popisu vrstevnic

Výškovou kótou bývají popisovány hlavní vrstevnice. I popis by však měl dodržovat určité kartografické zásady.

- Kóty se umisťují tak, aby byly čitelné při pohledu ve směru stoupání.
- Kóty se umisťují nepravidelně a rovnoměrně po celé ploše zobrazovaného území.

Nicméně, automatický popis vrstevnic respektující kartografické zásady byl v kódu implementován pouze z části.

Ve třídě Algorithms se vytvořila funkce calculateLabelPoints(), která se volá po stisknutí tlačítka *Create contours* se zaškrtnutým checkboxem *Show contour labels*.

Na vstupu jsou vektor hran hlavních vrstevnic a množina bodů P. Výstupem funkce jsou vektor 3D bodů a vektor úhlů s datovým typem double. Funkce počítá souřadnice x a y bodu uprostřed vrstevnice v každém patnáctém trojúhelníku, čímž je zajištěné nepravidelné a náhodné umístění výškových popisů. Z-ová souřadnice je převzána z počátečního bodu segmentu, neboť jejíž hodnota je pro celou vrstevnici konstantní. Ze souřadnicových rozdílů počátečního a koncového bodu každého segmentu vrstevnice je dále vypočtena směrnice přímky, podle které se následně budou orientovat popisy tak, aby byly rovnoběžné s vrstevnicemi.

$$lp_x = (\frac{s_x + e_x}{2}); lp_y = (\frac{s_y + e_y}{2})$$

$$\epsilon = atan2(\frac{dy}{dx})$$

```
//Compute direction
double dx = e1_point.x() - s1_point.x();
double dy = e1_point.y() - s1_point.y();
double rot = atan2(dy,dx);

//Determine location for label
QPoint3D label_point;
label_point.setX((s1_point.x()+e1_point.x())/2);
label_point.setY((s1_point.y()+e1_point.y())/2);
label_point.setZ(s1_point.getZ());

label_points.push_back(label_point);
```

Obrázek 7: výpočet souřadnic a směrnice bodu na popis průmky

Ve třídě Draw pak byly nejdříve vygenerovány obdélníky v barvě plátna, aby mohly být popisy umístěny přímo na vrstevnice a aby nebyly popisy vrstevnicemi rušeny.

Abychom mohli vykreslit obdélníky ve směru vrstevnice, musíme nejdříve celý souřadnicový systém posunout do bodu vypočteného funkcí getContourLines() a pootočit o směrnici vypočtenou ze stejné funkce. V transformovaném souřadnicovém systému následně vykreslíme obdélník s vhodnými rozměry a souřadnicový systém transformujeme zpět do původního souřadnicového systému plátna.

```
//Draw squares bellow labels nad contours but above triangulation
for (int unsigned i = 0; i < label_points.size(); i++)
{
        if (labels == true)
        {
            //Translate and rorate coordinate system
            qp.translate(label_points[i]);
            qp.rotate(directions[i]*180/M_PI);
            qp.translate(-label_points[i]);

            //Color of widget window
            QColor color (240, 240, 240 ,255);

            //Draw rectangle bellow number
            qp.setBrush(color);
            QPen fill_pen(color);
            qp.setPen(fill_pen);

            qp.drawRect(label_points[i].x()-font.pixelSize()/2, label_points[i].y()-font.pixelSize()/2,20, 20)

            //Translate and rorate coordinate system back
            qp.translate(label_points[i]);
            qp.rotate(-directions[i]*180/M_PI);
            qp.translate(-label_points[i]);
        }
}
```

Obrázek 8: Vykreslování obdélníku pod popisy vrstevnic

Následně se po vykreslení triangulace začnou vykreslovat popisy vrstevnic.

Algoritmus transformace SS je stejný jako u vykreslování obdélníků, pouze je zde ve vypočteném bodě zobrazen popis vrstevnice.

V kódu byla snaha i o otáčení popisů vrstevnic tak, aby bylo možné je číst po směru spádu. Záměr byl takový, že se pro každý segment vrstevnice vypočte směrnice. Dále se vypočtou souřadnice bodu, kde chceme vykreslovat popis vrstevnice ($label_point$. Následně se funkcí getNearestPoint() vypočte nejbližší bod qn ze vstupní množiny bodů P ke zjištěnému $label_point$. Dále se vypočte směrnice přímky mezi qn a $label_point$ a vypočte se výškový rozdíl dz těchto bodů. Následně se na základě rozdílu vypočtených směrnic a výškového rozdílu dz otočí orientace popisu vrstevnic o 180 stupňů.

Bohužel, jeli úvaha správná, nepodařilo se nám ji do funkce Algorithms::calculateLabelPoints() zaimplementovat.

```
//Draw squares bellow labels nad contours but above triangulation
for (int unsigned i = 0; i < label_points.size(); i++)
{
        if (labels == true)
        {
            //Translate and rorate coordinate system
            qp.translate(label_points[i]);
            qp.rotate(directions[i]*180/M_PI);
            qp.translate(-label_points[i]);

            //Color of widget window
            QColor color (240, 240, 240 ,255);

            //Draw rectangle bellow number
            qp.setBrush(color);
            QPen fill_pen(color);
            qp.setPen(fill_pen);

            qp.drawRect(label_points[i].x()-font.pixelSize()/2, label_points[i].y()-font.pixelSize()/2,20, 20)

            //Translate and rorate coordinate system back
            qp.translate(label_points[i]);
            qp.rotate(-directions[i]*180/M_PI);
            qp.translate(-label_points[i]);
        }
}
```

Obrázek 9: Vykreslování popisů vrstevnic

```
std::tuple<std::vector<QPoint3D>,std::vector<double>>
Algorithms::calculateLabelPoints(std::vector<Edge> &main_contours, std::vector<QPoint3D> &points)
{
    std::vector<QPoint3D> label_points;
    std::vector<double> rotations;

    //Draw labels
    for (int unsigned i = 0; i < main_contours.size(); i+=15)
    {
        Edge c = main_contours[i];

        QPoint3D s1_point = c.getStart();
        QPoint3D e1_point = c.getEnd();

        //Compute direction
        double dx = e1_point.x() - s1_point.x();
        double dy = e1_point.y() - s1_point.y();
        double rot = atan2(dy,dx);

        //Determine location for label
        QPoint3D label_point;
        label_point.setX((s1_point.x()+e1_point.x())/2);
        label_point.setY((s1_point.y()+e1_point.y())/2);
        label_point.setZ(s1_point.getZ());

        label_points.push_back(label_point);

        int i_nearest = getNearestPoint(label_point, points);
        QPoint3D qn = points[i_nearest];

        double dx2 = qn.x()-label_point.x();
        double dy2 = qn.y()-label_point.y();
        double rot2 = atan2(dy2,dx2);

        double dz = qn.getZ() - label_point.getZ();

        if ((rot2 - rot > 0) && (rot2 - rot < M_PI) && (dz < 0))
            rot += M_PI;

        rotations.push_back(rot);
    }

    return {label_points, rotations};

}
```

Obrázek 10: Funkce calculateLabelPoints() pro změnu orientace textu podle kartografických zásad

6 Vstupní data

6.1 Načítání bodů

Pro účel aplikace se použila lidarová data DMR 5G. Jelikož byla surová data příliš hustá, byla využita filtrace, která vzala každý desátý bod měření.

Soubor bodů se souřadnicemi x, y, z se do projektu nahrávají po stisknutí tlačítka *Load points*. Souřadnice x, y jsou v souřadnicovém systému S-JTSK a souřadnice z je ve výškovém systému Bpv.

Body jsou načítány postupně řádek po řádku, přičemž musí v nahrávaném souboru platit následující pravidla:

- na řádku je pořadí proměnných id » y (S-JTSK) » x (S-JTSK) » z (Bpv), hodnoty jsou od sebe odděleny jednou mezerou,
- id je identifikátor jednotlivých vrcholů polygonu, x, y, z jsou prostorové souřadnice bodů

```
1 625127.816 1088376.197 468.231
2 625134.269 1088410.866 470.262
3 625128.565 1088393.878 468.519
4 625127.938 1088257.727 479.721
5 625155.485 1088251.854 487.004
6 625133.559 1088246.975 481.967
7 625132.382 1088239.795 478.641
8 625132.881 1088245.76 481.222
9 625133.913 1088244.533 481.482
10 625133.441 1088243.577 480.828
```

Obrázek 11: Špagetový model vstupního souboru bodů

6.2 Načítání polygonu

Polygon, nad kterým lze následně zobrazit triangulaci, se načítá stisknutím tlačítka *Load polygon*.

Každý bod polygonu má ID a rovinné souřadnice x,y v souřadnicovém systému S-JTSK.

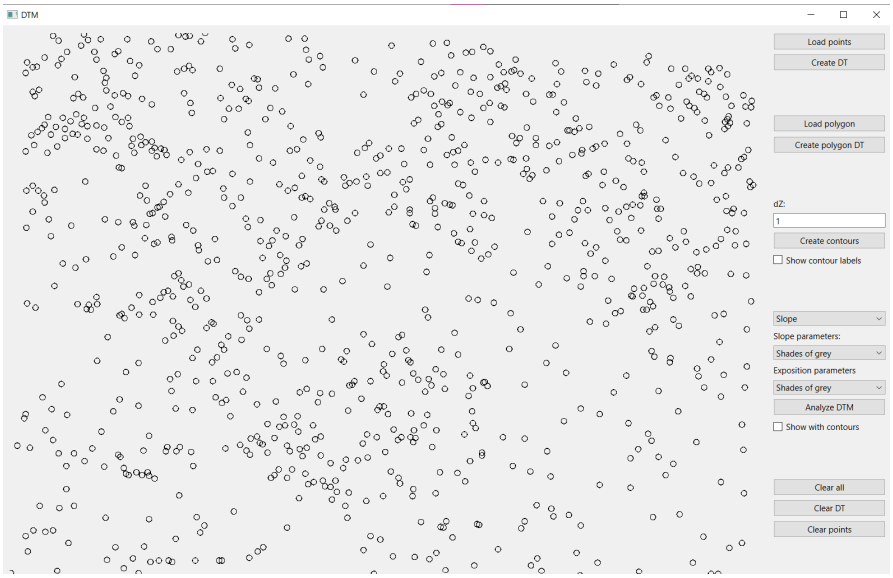
Body jsou načítány postupně řádek po řádku, přičemž musí v nahrávaném souboru platit následující pravidla:

- na řádku je pořadí proměnných id » y (S-JTSK) » x (S-JTSK) hodnoty jsou od sebe odděleny jednou mezerou,
- id je identifikátor jednotlivých vrcholů polygonu, x, y jsou rovinné souřadnice bodů v S-JTSK

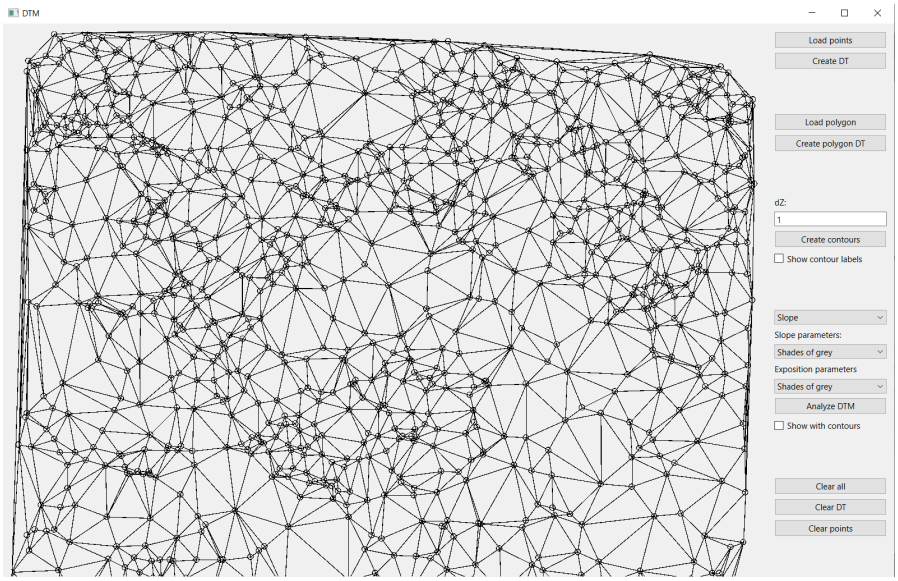
```
1 743001.84 1039322.30
2 743003.27 1039321.21
3 743004.13 1039322.31
4 743003.91 1039322.48
5 743005.24 1039324.19
6 743007.20 1039326.72
7 743010.25 1039324.34
8 743010.50 1039324.36
9 743010.53 1039324.11
10 743013.48 1039321.84
```

Obrázek 12: Špagetový model vstupního souboru polygonu

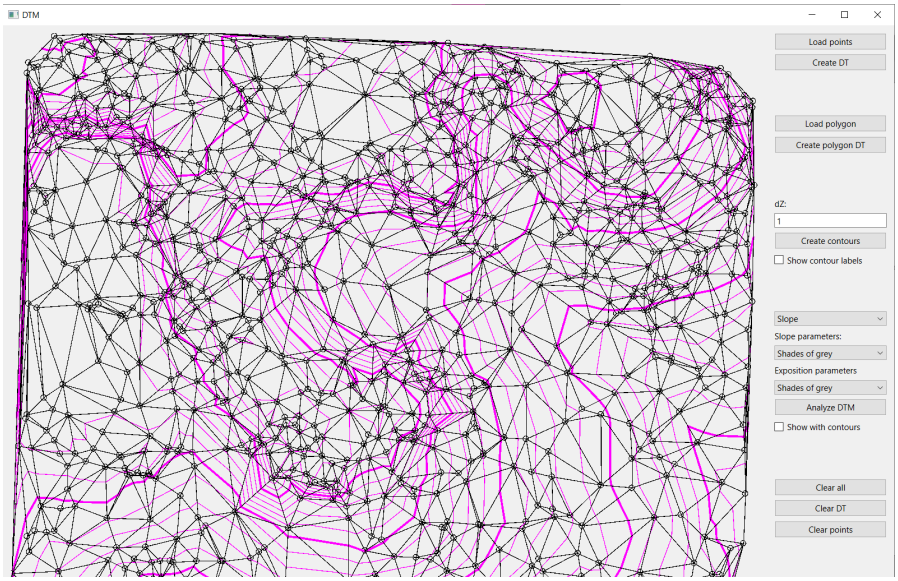
7 Výstupní data



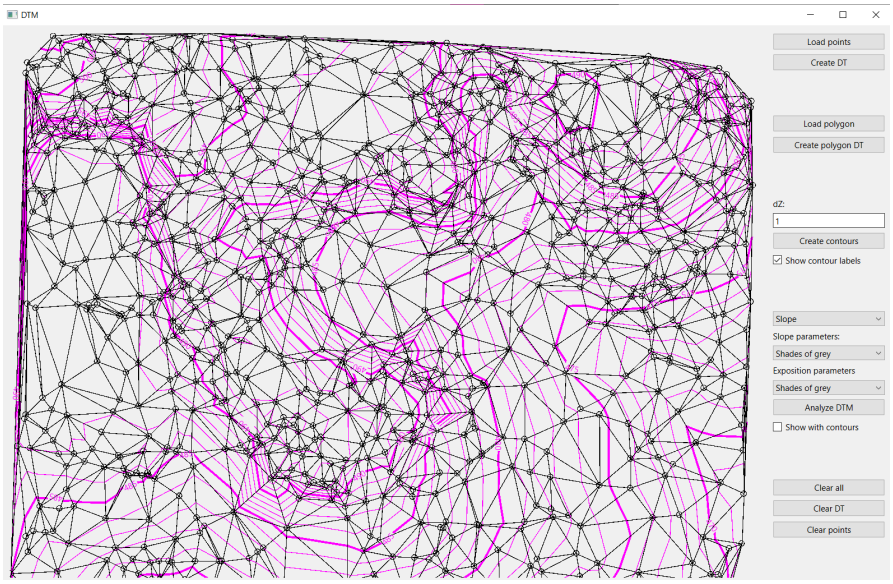
Obrázek 13: Aplikace po načtení bodů z textového souboru



Obrázek 14: Aplikace po vytvoření Delaunayho triangulace nad množinou bodů

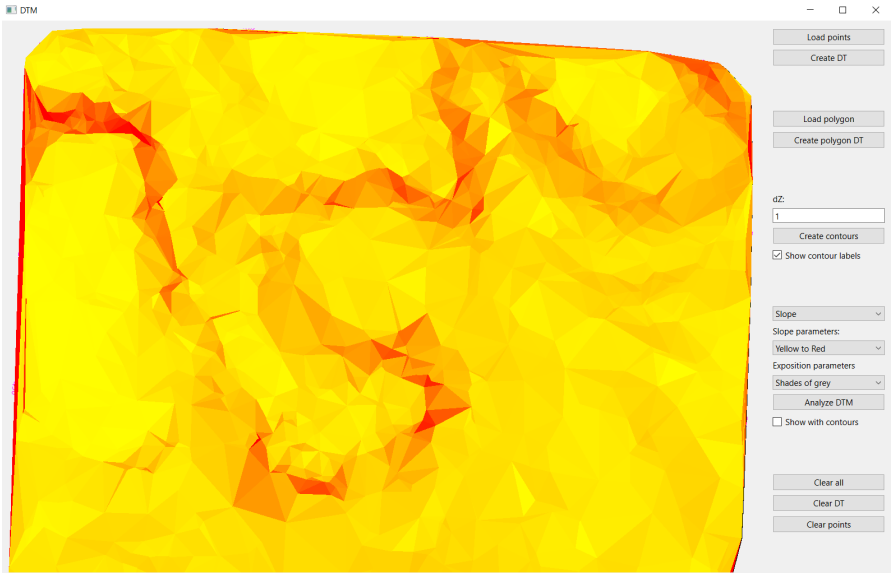


Obrázek 15: Aplikace po vykreslení vrstevnic

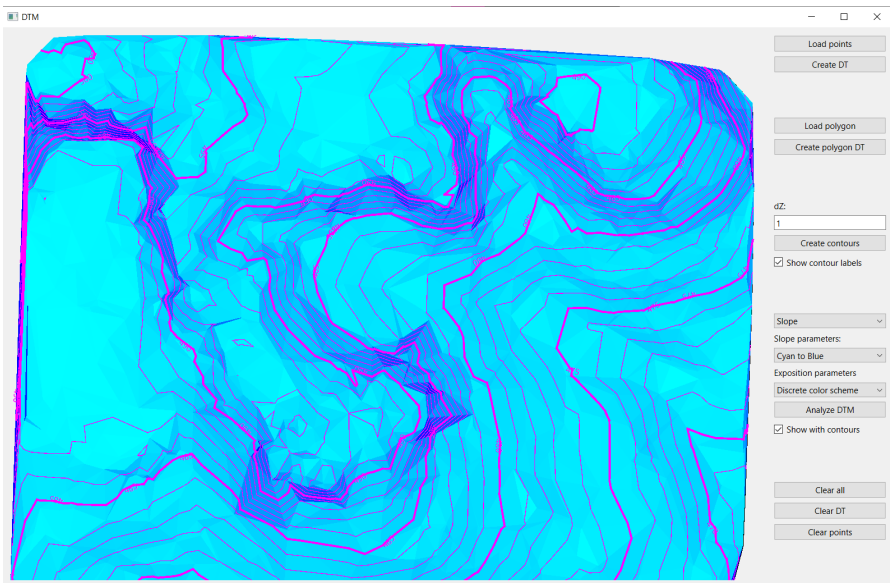


Obrázek 16: Aplikace po vykreslení popisů vrstevnic

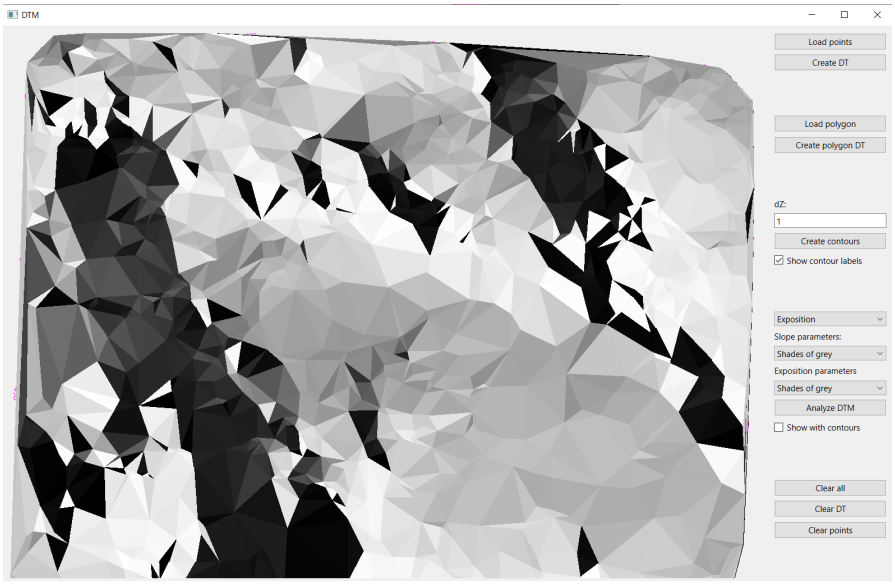
Obrázek 17: Aplikace po vykreslení sklonu terénu v odstínech šedi



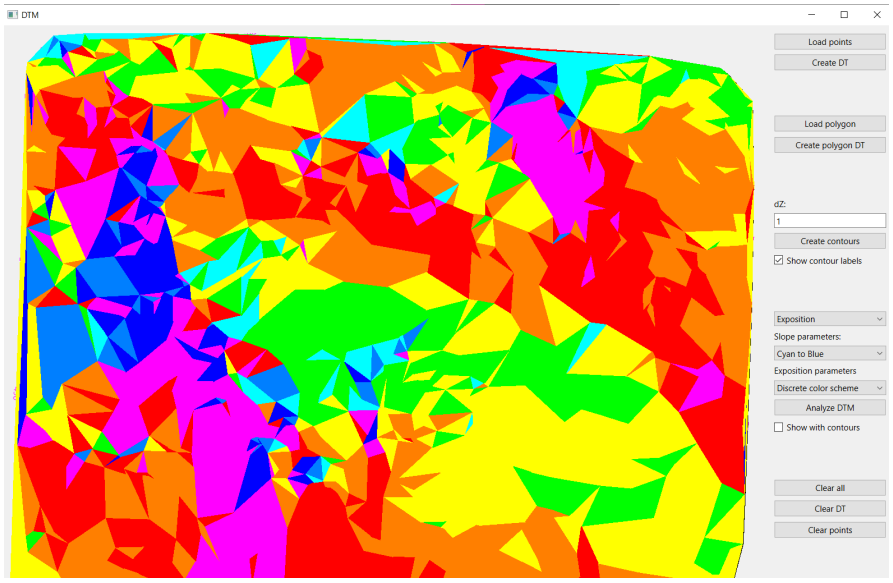
Obrázek 18: Aplikace po vykreslení sklonu terénu v odstínech od žluté po červenou



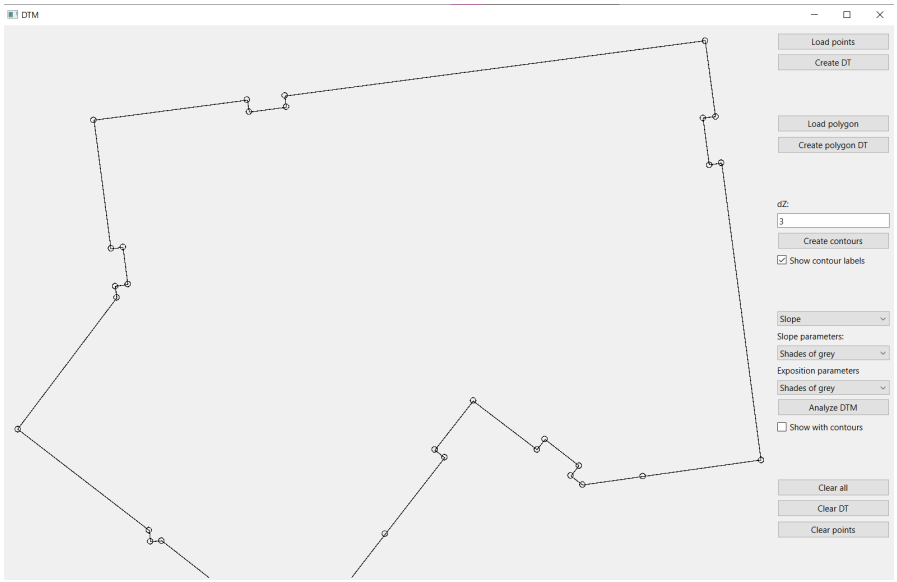
Obrázek 19: Aplikace po vykreslení sklonu terénu v odstínech od tyrkysové po modrou s vrstevnicemi a popisy vrstevnic



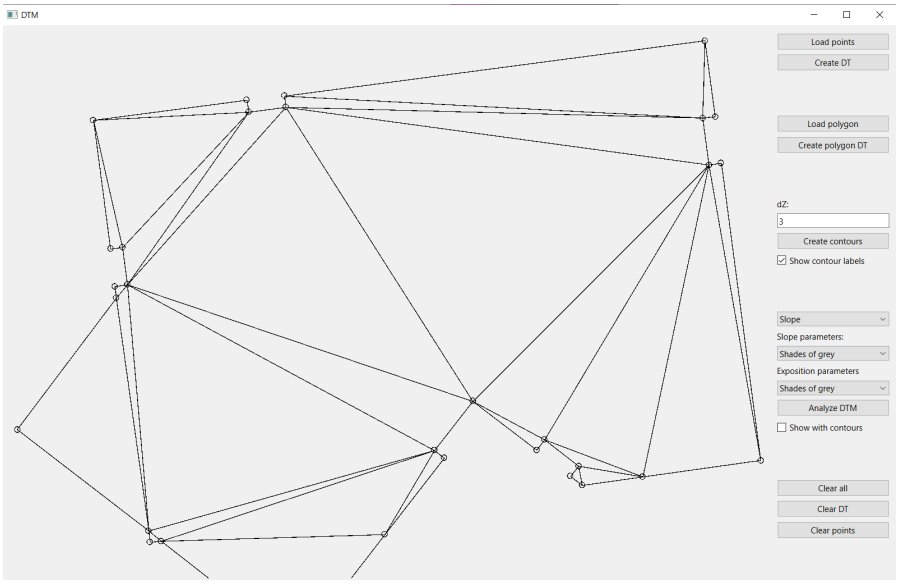
Obrázek 20: Aplikace po vykreslení expozice terénu v odstínech šedi



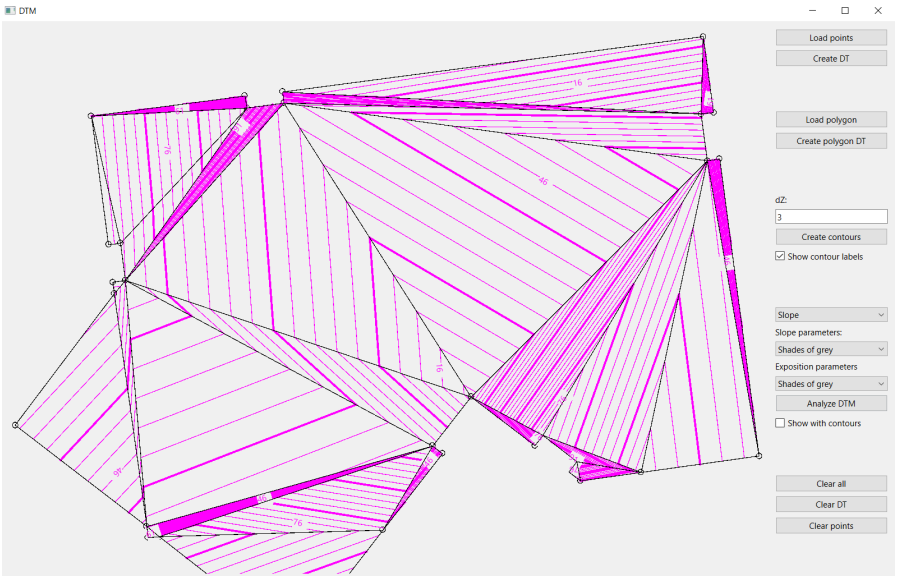
Obrázek 21: Aplikace po vykreslení expozice terénu v barvách podle tříd z obrázku 1



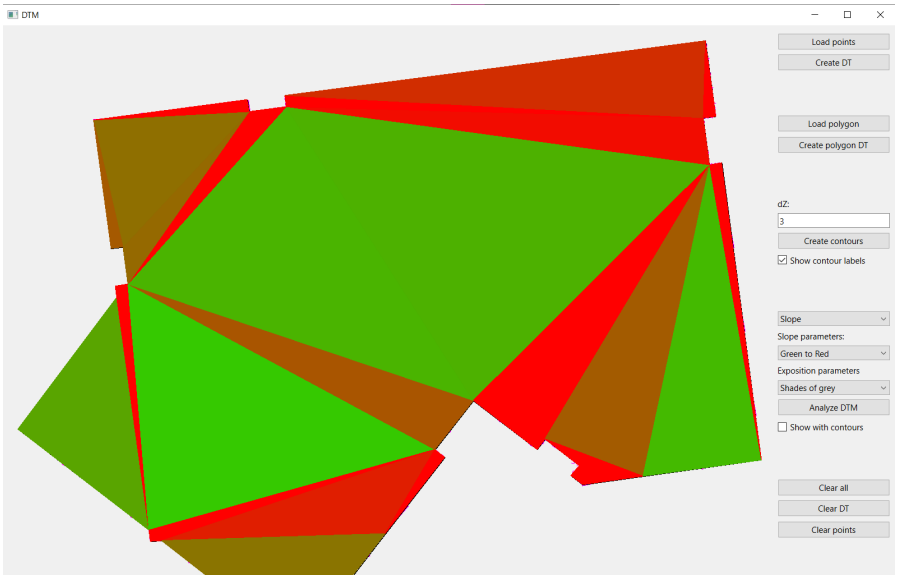
Obrázek 22: Aplikace po načtení polygonu z textového souboru



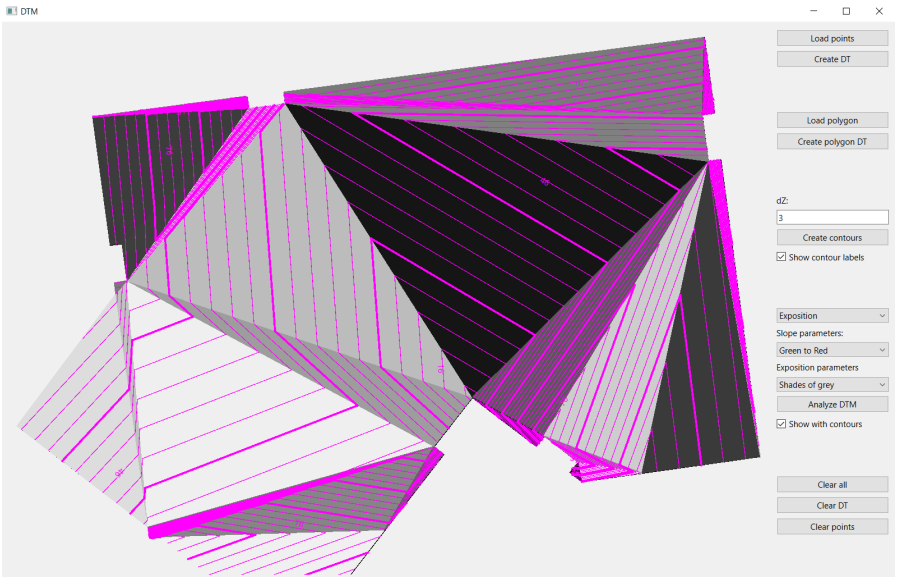
Obrázek 23: Aplikace po vytvoření Delaunayho triangulace nad nekonvexním polygonem



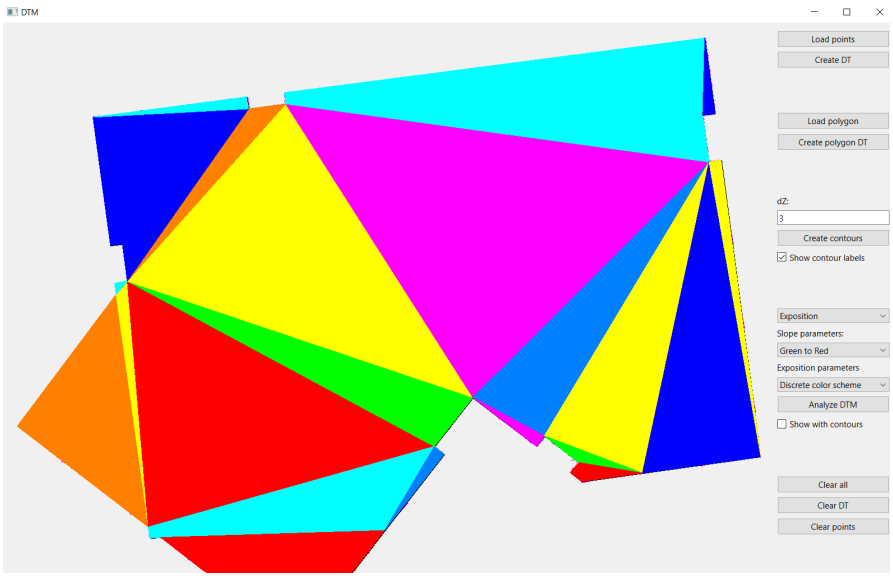
Obrázek 24: Aplikace po vykreslení vrstevnic s popisy nad DT nekonvexního polygonu



Obrázek 25: Aplikace po vykreslení sklonu terénu nad nekonvexním polygonem v odstínech od zelené po červenou

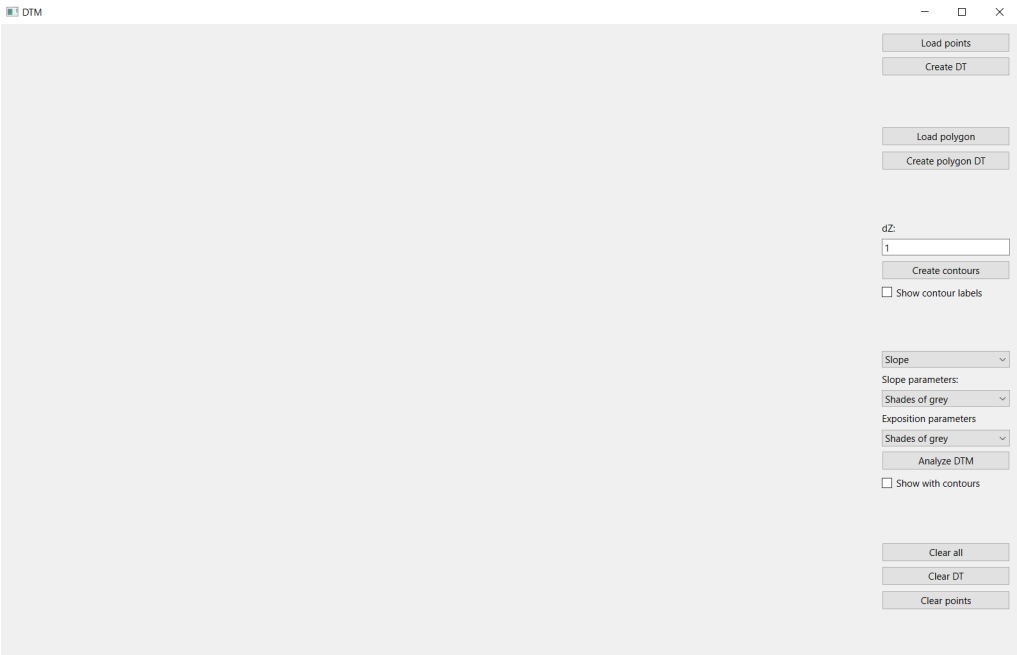


Obrázek 26: Aplikace po vykreslení expozice nekonvexního polygonu v odstínech šedi s vykreslenými vrstevnicemi a jejich popisy



Obrázek 27: Aplikace po vykreslení expozice nekonvexního polygonu v barvách podle tříd z obrázku 1

8 Printscreen vytvořené aplikace



Obrázek 28: Úvodní okno aplikace

9 Dokumentace

Kód zahrnuje 3 třídy – Draw, Algorithms a Widget, které budou následně detailněji popsány.

9.1 Třída Algorithms

Třída Algorithms obsahuje 14 funkcí:

- 0 v případě, že bod leží v pravé polorovině,
- 1 v případě, že bod leží v levé polorovině,
- -1 v případě, že bod leží na linii.
- double get2LinesAngle(QPoint &p1, QPoint &p2, QPoint &p3, QPoint &p4);
- std::tuple<QPoint3D,double> getCircleCenterAndRadius(QPoint3D &p1,QPoint3D &p2,QPoint3D &p3);
- int getDelaunayPoint(QPoint3D &s,QPoint3D &e,std::vector<QPoint3D> &points);
- int getNearestPoint(QPoint3D &p, std::vector<QPoint3D> &points);
- std::vector<Edge> dT(std::vector<QPoint3D> &points);
- std::vector<Edge> dTPolygon(std::vector<QPoint3D> &points);
- void updateAEL(Edge &e, std::list<Edge> &ael);
- QPoint3D getContourPoint(QPoint3D &p1, QPoint3D &p2, double z);
- std::vector<Edge> getContourLines(std::vector<Edge> &dt, double zmin, double zmax, double dz);
- double getSlope(QPoint3D &p1, QPoint3D &p2, QPoint3D &p3);
- double getExposition(QPoint3D &p1, QPoint3D &p2, QPoint3D &p3);
- std::vector<Triangle> analyzeDTM(std::vector<Edge> &dt);
- QPoint3D getCentreOfMass(QPoint3D &p1, QPoint3D &p2, QPoint3D &p3);

- int getPositionWinding(QPoint3D &q, std::vector<QPoint3D> &pol);
- int getPointLinePosition(QPoint &a, QPoint &p1, QPoint &p2);

double get2LinesAngle(QPoint &p1, QPoint &p2, QPoint &p3, QPoint &p4); Počítá úhel mezi dvěma liniemi. Vstupními argumenty jsou body určující linie, tzn. vrcholy polygonu. Návratovou hodnotou funkce je double – desetinné číslo s velikostí úhlu mezi těmito přímkami.

std::tuple<QPoint3D,double> getCircleCenterAndRadius(QPoint3D &p1,QPoint3D &p2,QPoint3D &p3); Funkce počítá poloměr a střed vepsaného trojúhelníku. Vstupními argumenty jsou vrcholy trojúhelníku uložené v datovém typu QPoint3D, výstupním je střed trojúhelníku (QPoitn3D) a hodnota poloměru trojúhelníku (double).

int getDelaunayPoint(QPoint3D &s,QPoint3D &e,std::vector<QPoint3D> &points); Funkce hledá Delaunay bod. Vstupními argumenty jsou 2 body Delaunay triangulace a vektor QPoint3D bodů, z nichž je hledán Delaunay bod. Výstupním argumentem funkce je index nalezeného bodu uložený jako integer.

int getNearestPoint(QPoint3D &p, std::vector<QPoint3D> &points); Funkce vyhledává k zadanému bodu bod nejbližší ze zadané množiny. Vstupním argumentem je bod QPoint3D, k němuž hledáme bod nebližší vektor bodů QPoint3D, výsledným je nejbližší nalezený bod – QPoint3D.

std::vector<Edge> dT(std::vector<QPoint3D> &points); Funkce počítá Delaunay triangulaci. Vstupním argumentem funkce je vektor bodů obsahující souřadnice x, y, z. Výstupním argumentem je vektor hran (Edge) vytvořené Delaunay triangulace.

std::vector<Edge> dTPolygon(std::vector<QPoint3D> &points); Funkce počítá Delaunay triangulaci nekonvexního polygonu. Vstupním argumentem funkce je vektor uzlů polygonu obsahující souřadnice x, y, z. Výstupním argumentem je vektor hran (Edge) vytvořené Delaunay triangulace.

void updateAEL(Edge &e, std::list<Edge> &ael);

QPoint3D getContourPoint(QPoint3D &p1, QPoint3D &p2, double z); Funkce hledá mezi dvěma zadanými body lineární interpolací bod v zadané výšce. Vstupními argumenty jsou 2 body QPoint3D a výška, jejíž hodnotu má výsledná vrstevnice mít. Výstupní hodnotou je nalezený průsečík uložený v datovém typu QPoint3D.

std::vector<Edge> getContourLines(std::vector<Edge> &dt, double zmin, double zmax, double dz); Funkce vytváří vrstevnice o zadaném základním intervalu. Vstupním argumentem je Delaunay triangulace, maximální a minimální hodnota souřadnice z a dz, což je základní interval vrstevnic. Výstupním argumentem je vektor hran s vytvořenými vrstevnicemi.

double getSlope(QPoint3D &p1, QPoint3D &p2, QPoint3D &p3); Funkce počítá hodnotu sklonu trojúhelníku Delaunay triangulace. Vstupními argumenty jsou vrcholy trojúhelníku uložené v datovém typu QPoint3D, výstupním je hodnota sklonu.

double getExposition(QPoint3D &p1, QPoint3D &p2, QPoint3D &p3); Funkce počítá hodnotu expozice trojúhelníku Delaunay triangulace. Vstupními argumenty jsou vrcholy trojúhelníku uložené v datovém typu QPoint3D, výstupním je hodnota expozice.

std::vector<Triangle> analyzeDTM(std::vector<Edge> &dt); Funkce analyzuje všechny trojúhelníky vytvořené Delaunay triangulací – počítá jejich sklon, expozici. Vstupním argumentem je vektor hran Delaunay triangulace, výstupním je vektor Triangle, v němž jsou uložené hrany jednotlivých trojúhelníků, jejich sklony a expozice.

QPoint3D getCentreOfMass(QPoint3D &p1, QPoint3D &p2, QPoint3D &p3); Funkce počítá těžiště trojúhelníku. Vstupními argumenty jsou vrcholy trojúhelníku uložené v datovém typu QPoint3D, výstupním je bod těžiště trojúhelníku v datovém typu QPoint3D.

int getPositionWinding(QPoint &q, std::vector<QPoint> &pol);
Funkce analyzuje polohu bodu metodou Winding Number, jež byla podrobně vysvětlena v kapitole 3. Vstupními argumenty jsou souřadnice bodu q (jako QPoint) a vektor vrcholů polygonu (vektor naplněný prvky QPoint). Funkce vrací integer, jež může nabývat 3 hodnot:

- 1, leží-li analyzovaný bod uvnitř polygonu
- -1 leží-li na linii polygonu
- a 0, leží-li mimo kontrolovaný polygon.

int getPointLinePosition(QPoint &a, QPoint &p1, QPoint &p2); Analyzuje vzájemnou polohu mezi bodem a linií polygonu, resp. v jaké polorovině vůči linii se bod nachází. Vstupními argumenty funkce jsou souřadnice určovaného bodu (jako QPoint) a souřadnice 2 bodů určujících polohu linie (vrcholy polygonu taktéž jako QPoint). Funkce vrací vždy hodnotu 1, 0 nebo -1 dle následujících pravidel:

- 0 v případě, že bod leží v pravé polorovině,
- 1 v případě, že bod leží v levé polorovině,
- -1 v případě, že bod leží na linii.

9.2 Třída Draw

Třída Draw obsahuje následující funkce, které budou dále podrobně popsány:

- void paintEvent(QPaintEvent *event);
- void mousePressEvent(QMouseEvent *event);
- void clear();
- explicit Draw(QWidget *parent = nullptr);
- void loadData(QString &file_name);
- void loadPolygon(QString &file_name);
- std::vector<QPoint3D> getPoints();
- double getZmin();
- double getZmax();
- void setDT(std::vector<Edge> &dt_);
- void setDZ(int &dz_);

- void setMinSlope(double &minsl_);
- void setMaxSlope(double &maxsl_);
- void setDirections(std::vector<double> directions_);
- std::vector<double> getDirections();
- void setLabelPoints(std::vector<QPoint3D> &label_points_);
- std::vector<QPoint3D> getLabelPoints();
- std::vector<Edge> getDT();
- void setContours(std::vector<Edge> &contours_);
- void setMainContours(std::vector<Edge> &main_contours_);
- std::vector<Edge> getContours();
- std::vector<Edge> getMainContours();
- std::vector<Triangle> getTriangles();
- void setTriangles(std::vector<Triangle> &triangles_);
- int round2num(int &numToRound, int &multiple, bool &dir);
- void clearDT();
- void setSlopeParameters(int slope_param_);
- void setExpositionParameters(int expos_param_);
- void loadData(QString &file_name);

a následující proměnné:

- int dz - základní interval vrstevnic
- int slope_param - způsob vykreslování sklonu
- int expos_param - způsob vykreslování expozice
- double max_slope - maximální sklon
- min_slope - minimální sklon

- double y_max - maximální y
- double x_min - minimální x
- double y_min - minimální y
- double x_max - maximální x
- double z_min - minimální z
- double z_max - maximální z
- QPolygonF polygon - nekonvexní polygon
- std::vector<QPoint3D> points - body tachymetrie
- std::vector<Edge> dt - Delaunay triangulace
- std::vector<Edge> contours, main_contours - hlavní vrstevnice
- std::vector<Triangle> triangles - trojúhelníky Delaunay triangulace
- std::vector<QPoint3D> label_points - popisy vrstevnic
- std::vector<double> directions - směry popisů vrstevnic

void paintEvent(QPaintEvent *event);
Funkce nastavuje grafické atributy vykreslovaných objektů a kreslí na plátno vyhodnocovaný bod a zadané polygony a znázorňuje polygon incidující s bodem.

void mousePressEvent(QMouseEvent *event);
Metoda získává souřadnice myši a ukládá je do vektoru QPoint3D, tedy jako souřadnice analyzovaných bodů.

void clear();
Funkce smaže veškeré objekty, jež jsou vykresleny na canvasu, funkce nemá vstupní argumenty.

void loadData(QString &file_name);
Funkce načítá data z textového souboru a ukládá je do vektoru QPoint3D. Vstupní argumentem je cesta k souboru, jejž chceme načíst do aplikace.

void loadPolygon(QString &file_name); Funkce načítá souřadnice vrcholů nekonvexního polygonu z textového souboru a ukládá je do vektoru QPoint3D. Vstupní argumentem je cesta k souboru, jejž chceme načíst do aplikace.

std::vector<QPoint3D> getPoints(); Funkce pro získání vektoru bodů měřené tachymetrie ze třídy Draw. Vstupní argument funkce nemá, výstupním argumentem je vektor QPoint3D.

double getZmin(); Funkce pro získání minimální souřadnice Z ze třídy Draw. Vstupní argument funkce nemá, výstupním argumentem je hodnota nejmenší souřadnice z.

double getZmax(); Funkce pro získání maximální souřadnice Z ze třídy Draw. Vstupní argument funkce nemá, výstupním argumentem je hodnota největší souřadnice z.

void setDT(std::vector<Edge> &dt_); Ukládá vybraný Delaunay triangulaci do třídy Widget - do proměnné dt.

void setDZ(int &dz_); Ukládá základní interval vrstevnic do třídy Widget - do proměnné dz.

void setMinSlope(double &minsl_); Ukládá základní minimální hodnotu sklonu do třídy Widget - do proměnné min_slope.

void setMaxSlope(double &maxsl_); Ukládá základní maximální hodnotu sklonu do třídy Widget - do proměnné max_slope.

void setDirections(std::vector<double> directions_); Ukládá směry natočení popisků vrstevnic do třídy Widget - do proměnné directions.

std::vector<double> getDirections(); Funkce pro získání směrů natočení popisků vrstevnic ze třídy Draw. Vstupní argument funkce nemá, výstupním argumentem je hodnota vektor hodnot natočení popisků.

void setLabelPoints(std::vector<QPoint3D> &label_points_);

std::vector<QPoint3D> getLabelPoints();

std::vector<Edge> getDT(); Funkce pro získání Delaunay triangulace ze třídy Draw. Vstupní argument funkce nemá, výstupním argumentem je hodnota vektor Edge Delaunay triangulace.

void setContours(std::vector<Edge> &contours_); Ukládá vrstevnice uložené jako vektor Edge do třídy Widget - do proměnné contours.

void setMainContours(std::vector<Edge> &main_contours_); Ukládá hlavní vrstevnice uložené jako vektor Edge do třídy Widget - do proměnné main_contours.

std::vector<Edge> getContours(); Funkce pro získání vrstevnic ze třídy Draw. Vstupní argument funkce nemá, výstupním argumentem je vektor vrstevnic.

std::vector<Edge> getMainContours(); Funkce pro získání hlavních vrstevnic ze třídy Draw. Vstupní argument funkce nemá, výstupním argumentem je vektor hlavních vrstevnic.

std::vector<Triangle> getTriangles(); Funkce pro získání trojúhelníků Delunay triangulace ze třídy Draw. Vstupní argument funkce nemá, výstupním argumentem je hodnota vektor trojúhelníků.

void setTriangles(std::vector<Triangle> &triangles_); Ukládá trojúhelníky dt uložené jako vektor Triangle do třídy Widget - do proměnné triangles.

int round2num(int &numToRound, int &multiple, bool &dir); Zaokrouhluje číslo na požadovaný násobek a to buď směrem nahoru nebo dolu. Vstupem jsou zaokrouhlované číslo typu int, násobek typu int a směr datového typu bool. Pokud se dir = true, zaokrouhlujeme na nejbližší násobek směrem nahoru, pokud se dir = false, zaokrouhlujeme dolů. Výstupem je zaokrouhlené číslo typu int. Více viz. kapitola 5.2.

void clearDT(); Maže zobrazenou Delaunay triangulaci z canvasu.

void setSlopeParameters(int slope_param_); Ukládá vybraný typ vizualizace sklonu terénu ze třídy Widget a ukládá jej do proměnné slope_param podle následujících hodnot se následně vykresluje sklon terénu:

- 0 - vizualizace ve stupních šedé barvy
- 1 - vizualizace v barvách od zelené do červené
- 2 - vizualizace v barvách od žluté do červenou
- 3 - vizualizace v barvách od tyrkysové po modrou

void setExpositionParameters(int expos_param_); Ukládá vybraný typ vizualizace expozice terénu ze třídy Widget a ukládá jej do proměnné s expos_param podle následujících hodnot se následně vykresluje sklon terénu:

- 0 - vizualizace ze stupních šedé barvy
- 1 - vizualizace v diskrétním barevném schématu
- 2 - vizualizace ve spojitém barevném schématu

9.3 Třída Widget

Třída Widget obsahuje 13 metod:

- void on_pushButton_ClearAll_clicked();
- void on_pushButton_CreateDT_clicked();
- void on_pushButton_cleardt_clicked();
- void on_lineEdit_3_editingFinished();
- void on_pushButton_CreateContours_clicked();
- void on_pushButton_AnalyzeDTM_clicked();
- void on_pushButton_LoadPoints_clicked();
- void on_pushButton_LoadPolygon_clicked();
- void on_pushButton_CreatePolDT_clicked();

- void on_pushButton_contour_labels_clicked();
- void on_checkBox_stateChanged(int arg1);
- void on_pushButton_ClearPoints_clicked();
- void on_checkBox_ShowContours_stateChanged(int arg1);

void on_pushButton_ClearAll_clicked(); Funkce se volá po stisknutí tlačítka *Clear all*, slouží k volání funkce clear() ze třídy Draw, tato funkce následně smaže všechny prvky z canvasu.

void on_pushButton_CreateDT_clicked(); Funkce se volá po stisknutí tlačítka *Create DT*, slouží k vytvoření Delaunay triangulace pomocí metody dT ze třídy Algorithms.

void on_pushButton_cleardt_clicked(); Funkce se volá po stisknutí tlačítka *Clear DT*, slouží k volání funkce clearDT() ze třídy Draw, tato funkce následně smaže zobrazenou Delaunay triangulaci, vrstevnice a vizualizaci sklonu/expozice.

void on_lineEdit_3_editingFinished(); Funkce se volá při změně parametrů základního intervalu vrstevnic.

void on_pushButton_CreateContours_clicked(); Funkce se volá po stisknutí tlačítka *Create Contours*, slouží k vytvoření vrstevnic pomocí metody getContourLines ze třídy Algorithms.

void on_pushButton_AnalyzeDTM_clicked(); Funkce se volá po stisknutí tlačítka *Analyze DTM*, slouží k výpočtu sklonu terénu a expozice pomocí metody analyzeDTM ze třídy Algorithms.

void on_pushButton_LoadPoints_clicked(); Funkce se volá po stisknutí tlačítka *Load points*, slouží k načtení bodů tachymetrie z textového souboru, volá metodu loadData ze třídy Draw.

void on_pushButton_LoadPolygon_clicked(); Funkce se volá po stisknutí tlačítka *Load Polygon*, slouží k načtení vrcholů polygonu z textového souboru, volá metodu loadPolygon ze třídy Draw.

void on_pushButton_CreatePolDT_clicked(); Funkce se volá po stisknutí tlačítka *Create Polygon DT*, slouží k vytvoření Delaunay triangulace nekonvexního polygonu z textového souboru, volá metodu dTPolygon třídy Algorithms.

void on_pushButton_contour_labels_clicked(); Funkce se volá v případě zaškrtnutí možnosti *Show contours labels*, v případě zaškrtnutí umožňuje vykreslení vrstenic s popisy. Volá metodu getMainContours třídy Draw.

void on_checkBox_stateChanged(int arg1); Pokud se zaškrtne checkbox *Show contour labels*, změní se hodnota labels ze třídy Draw na true a opačně.

void on_pushButton_ClearPoints_clicked(); Funkce se volá po kliknutí na tlačítko *Clear points* a odstraňuje množinu načtených bodů z textového souboru.

void on_checkBox_ShowContours_stateChanged(int arg1); Funkce se volá v případě zaškrtnutí možnosti *Show with contours*, v případě zaškrtnutí umožňuje vykreslení vrstevnic nad zobrazovaným sklonem terénu nebo expozicí.

9.4 Třída Edge

Ve třídě Edge je definován datový typ Edge, jenž definuje hranu (DT), je definován počátečním a koncovým bodem uloženým v datovém typu QPoint3D.

9.5 Třída QPoint3D

Třída definuje datový typ QPoint3D, jež je dědičným datovým typem QPointF. Umožňuje uložení bodu se souřadnicemi x, y a z s předností float.

9.6 Třída Triangle

Třída definuje datový typ Triangle, v němž je možné uložit jednotlivé trojúhelníky Delaunay triangulace. Ukládá souřadnice 3 vrcholů trojúhelníku uložených v datovém typu QPoint3D a sklon a expozici v datových typech double.

9.7 Třída SortByX

Třída definuje operátor přetížení, který řadí body podle x-ové souřadnice.

10 Závěr

Vytvořená aplikace vizualizuje polygonovou mapu uloženou ve výše popsaném formátu, po přidání bodu taktéž analyzuje polohu tohoto bodu vůči jednotlivým polygonům mapy. Po analýze bodu se vypíše výsledek – tedy zda bod leží uvnitř nebo vně polygonové mapy, v případě umístění bodu uvnitř polygonu se polygon obsahující tento bod graficky zvýrazní.

V aplikaci jsou pro vyhodnocení polohy bodu implementovány 2 algoritmy – Winding number a Ray Crossing, mezi nimiž je možné přepínat pomocí comboboxu umístěném v pravé části okna. Princip fungování obou algoritmů je popsán v kapitole 3.

Možné zlepšení kódu programu by bylo v řešení singularit polohy bodu q, tedy případu, kdy bod q leží na vrcholu jednoho či více polygonů a dále přidání analýzy případu, kdy bod q leží na linii polygonu pomocí Ray Crossing algoritmu. Dalším možným krokem pro zlepšení funkčnosti by mohlo být přidání bodu q pomocí vepsání souřadnic, nikoliv pouze odečtením polohy myši.