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

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 trojúhelnikovou 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 výslekdy 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 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.

4.2.1 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

4.2.2 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.

4.3 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.

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

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

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

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

4.4 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.

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

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

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

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

4.5 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.

Ve třídě Algorithms se vytvořila funkce calculateLabelPoints(), která se volá po stisknutí tlačítka *Create contours* se začkrtnutým 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 1: 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 nebyly jimy rušeny.

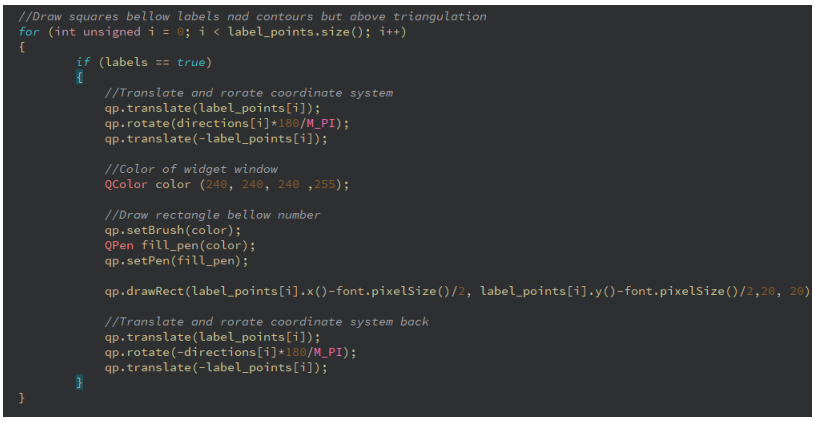
Abychom mohli vykreslit obdélníky ve směru vrstevnice, musíme nejdříve celý souřadnicový systém posunout do bodu vypočteného funkcí getContourLines() a pootočit o směrnici vypočtenou ze stejné funkce. V transformovaném souřadnicovém systému následně vykreslíme obdélník s vhodnými rozměry a souřadnicový 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 2: 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.



```
//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 3: Vykreslování popisů vrstevnic

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 4: 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 5: 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 6: 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 7: 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 8: Funkce aplikace round2num() v LoadData()

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

5.3 Generování popisu vrstevnice ve směru spádu

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 9: 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:

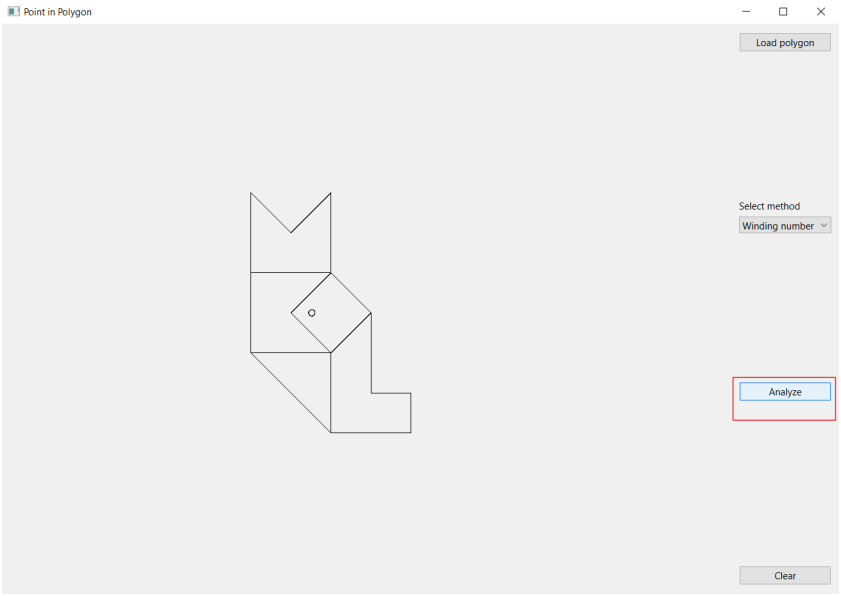
- na řádku je pořadí proměnných id » y (S-JTSK) » x (S-JTSK) hodnoty jsou od sebe odděleny jednou mezerou,
- id je identifikátor jednotlivých vrcholů polygonu, x, y jsou rovinné souřadnice bodů v S-JTSK

```
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 10: Špagetový model vstupního souboru polygonu

7 Výstupní data

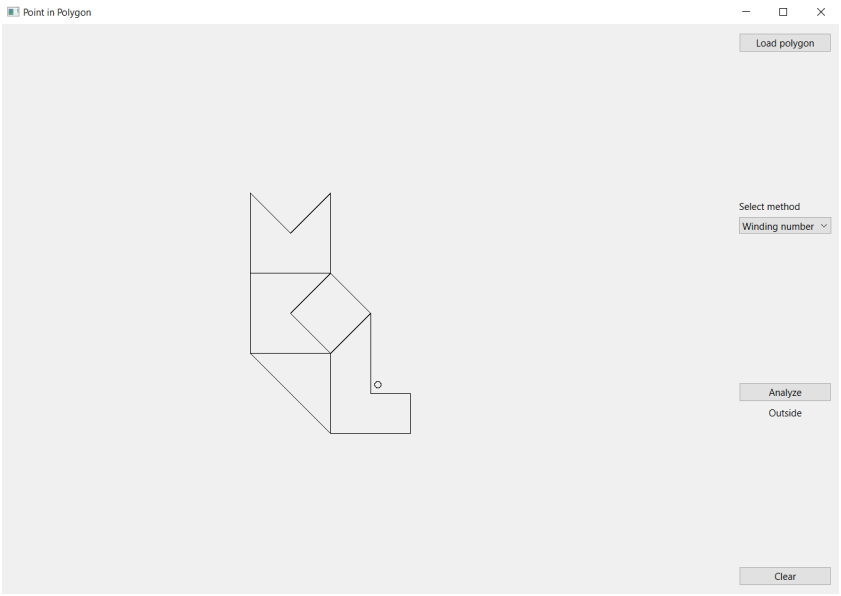
Po načtení souboru s polygonovou mapou, přidání bodu q a stisknutí tlačítka Analyze se v místě vyznačeném na obrázku 11 vytiskne výsledek analýzy polohy bodu.



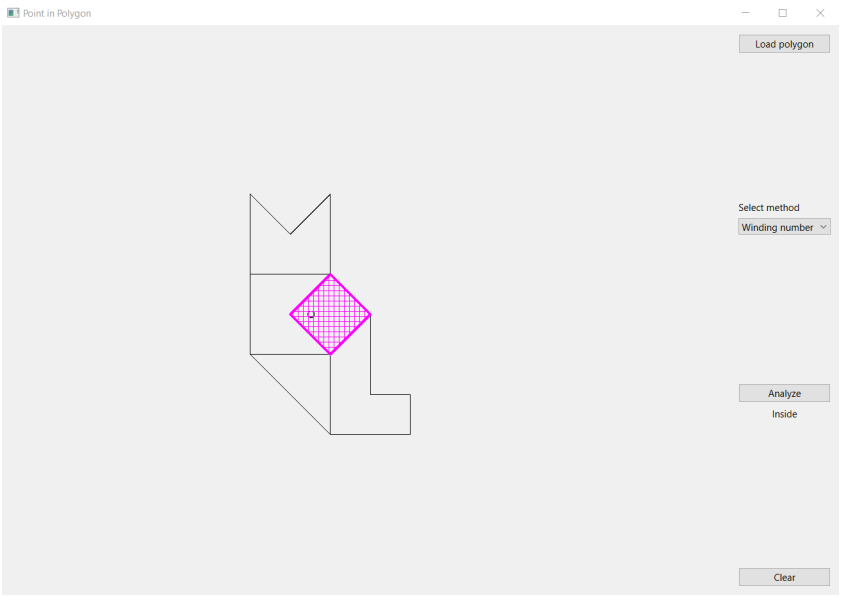
Obrázek 11: Aplikace s načteným polygonem

Výsledek dotazu může mít 3 různé hodnoty:

- je-li bod mimo polygonovou mapu, vypíše se zpráva „Outside“,
- je-li bod uvnitř polygonu, vypíše se zpráva „Inside“ a polygon obsahující zadaný bod se graficky zvýrazní,
- je-li bod na hranici polygonu, vypíše se „Point is on the line“ a graficky se zvýrazní polygon, na jehož linii bod leží.

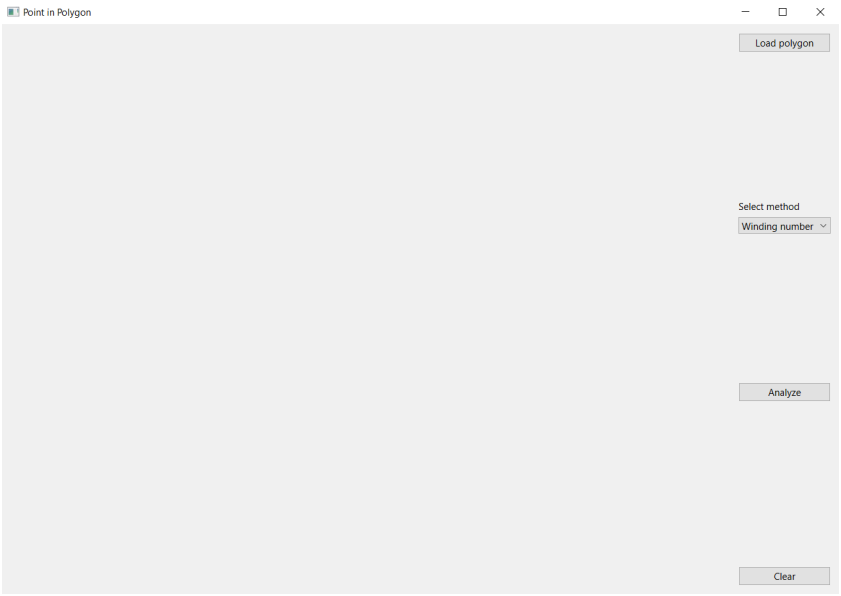


Obrázek 12: Bod vně polygonu



Obrázek 13: Bod uvnitř polygonu

8 Printscreen vytvořené aplikace



Obrázek 14: 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 4 funkce :

- int getPointLinePosition(QPoint &a, QPoint &p1, QPoint &p2);
- double get2LinesAngle(QPoint &p1, QPoint &p2, QPoint &p3, QPoint &p4);
- int getPositionWinding(QPoint &q, std::vector<QPoint> &pol);
- int getPositionRayCrossing(QPoint &q, std::vector<QPoint> &pol);

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.

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

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 getPositionRayCrossing(QPoint &q, std::vector<QPoint> &pol);
Funkce analyzuje polohu bodu metodou Ray Crossing, 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, stejně jako funkce předchozí, celé číslo, jež může nabývat následujících hodnot:

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

Ošetření případu, kdy bod leží na linii nebylo do kódu implementováno, funkce tedy nevrací hodnotu -1, leží-li bod 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();
- QPoint getPoint()return q;
- std::vector<QPolygon> getPolygon(){return polygons;}
- void fillPolygon(int result);
- void loadData(QString &file_name);

a následující proměnné:

- std::vector<QPolygon> polygons;
- QPoint q; - analyzovaný bod
- int highlighted_polygon=-99; - id polygonu, který obsahuje analyzovaný bod

• 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 proměnné QPoint q, tedy jako souřadnice analyzovaného bodu q.

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

QPoint getPoint(){return q;}
Funkce vrací souřadnice bodu q, návratový typ funkce je QPoint, funkce nemá vstupní argumenty.

std::vector<QPolygon> getPolygon(){return polygons;}
Funkce vrací vektor polygonů načtených z textového souboru – návratový typ je std::vector<QPolygon>, funkce nemá vstupní argumenty.

void fillPolygon(int result);
Funkce ukládá identifikátor polygonu obsahujícího analyzovaný bod, pomocí identifikátoru se tento polygon následně ve funkci paintEvent() zvýrazní. Vstupním argumentem je int result, jež stanovuje polohu bodu vůči linii (má hodnoty 0,1,-1,dle definice výše).

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

9.3 Třída Widget

Třída Widget obsahuje 3 metody:

- void on_pushButtonClear_clicked();
- void on_pushButtonAnalyze_clicked();
- void on_pushButtonLoad_clicked();

void on_pushButtonClear_clicked();
Funkce se volá při stisknutí tlačítka Clear, volá funkci clear(), jež je definovaná v třídě Draw.

void on_pushButtonAnalyze_clicked();
Funkce se volá při stisknutí tlačítka Analyze. Ve funkci nejprve dojde k uložení jednotlivých polygonů do vektoru vrcholů polygonu, následně je zavolána funkce getPositionRayCrossing ze třídy Algorithms nebo getPositionWinding dle výběru v comboboxu.

void on_pushButtonLoad_clicked();
Funkce slouží k inicializaci načítání souboru, spustí se po stisknutí tlačítka Load, čímž zobrazí dialog pro výběr souboru obsahujícího data, jež mají být načtena do aplikace. Po výběru souboru se zavolá funkce loadData ze třídy Draw, která přečte data ze souboru a uloží je do proměnné.

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.