shipout/backgroundshipout/foreground

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

ALGORITMY DIGITÁLNÍ KARTOGRAFIE A GIS

KATEDRA GEOMATIKY

Úloha č. 2: Konvexní obálky

Monika KŘÍŽOVÁ
Marek HOFFMANN

06.11.2021

Obsah

1 Zadání

Vytvořte aplikaci s grafickým uživatelským rozhraním, která bude ze souboru načítat polygony budov a následně je bude generalizovat na obdélníky s plochou odpovídající generalizované budově a bude vytvářet konvexní obálky. Hlavní směry budov budou určeny metodami Minimum Area Enclosing Rectangle a Wall Average.

Přesné zadání je vloženo na následující stranu.

2 Údaje o bonusových úlohách

Zpracovány byly celkem 3 bonusové úlohy ze 4 zadaných.

- Generalizace budov metodou Longest Edge +5b
- Generalizace budov metodou Weighted Bisector +8b
- Implementace další metody konstrukce konvexní obálky. +5b

Generalizace budov metodou Longest Edge byla implementována ve třídě Algorithms, je možné ji spustit pomocí tlačítka Building simplify po předchozím výběru v comboBoxu umožňujícího výběr metody generalizace polygonu.

Generalizace budov metodou Weighted Bisector byla stejně jako metoda Longest Edge implementována ve třídě Algorithms a taktéž je ji možné spustit pomocí tlačítka Building simplify po předchozím výběru v comboBoxu umožňujícího výběr metody generalizace polygonu.

Pro implementaci další metody konstrukce konvexní obálky byla vybrána metoda Quick Hull, jež byla přidána do třídy Algorithms. Pro její spuštění je nutné vybrat tuto metodu v náležitém comboBoxu a následně stisknout tlačítko Create convex hull.

3 Popis problému

Generalizace je v kartografii velmi důležitý proces, který je používán například při změně měřítka mapy, změně jejího účelu nebo pro zlepšení grafické přehlednosti. Generalizace může být provedena několika různými metodami, v řešené úloze byla aplikována tzv. geometrická generalizace, jež spočívá ve zjednodušení tvaru objektu. Objekty (v této úloze budovy) byly generalizovány na obdélníky s plochou odpovídající ploše původních objektů.

Jednotlivé generalizační algoritmy hledají hlavní směr natočení budovy, po nalezení tohoto směru se vytvoří opsaný obdélník s minimálním obsahem, jež je natočený do tohoto směru (dále pod zkratkou MMB). MMB se následně zmenší tak, aby byla jeho plocha totožná s plochou generalizovaného objektu.

4 Popisy algoritmů

4.1 Tvorba konvexní obálky

4.1.1 Jarvis scan

Prvním krokem při tvorbě konvexní obálky $\mathcal{H}$ metodou Jarvis scan je nalezení pivotu q. Tento bod je bod s nejmenší souřadnicí y v množině bodů S. Dalším krokem je nalezení bodu p, jež svírá největší úhel s rovnoběžkou s osou x procházející bodem q. Nalezený bod se přidá do $\mathcal{H}$ a opět se hledá bod, jež bude svírat s předchozím bodem největší úhel. Tímto způsobem se do $\mathcal{H}$ přidávají body do okamžiku, kdy by byl přidávaným bodem bod q. Tato metoda je velmi jednouchá, ale kvůli časové náročnosti nevhodná pro velké množiny bodů.

4.1.2 Quick Hull

V algoritmu Quick Hull je nejprve nutné seřadit množinu bodů S podle souřadnice x a následně ze setříděného souboru vybrat dva body, tzv. pivoty. Bod q_1 je bodem s nejmenší souřadnicí x, bod q_3 má naopak souřadnici x ze souboru bodů S největší. Konvexní obálka $\mathcal{H}$ se vytváří ze dvou částí – horní ($\mathcal{H}_{\mathcal{U}}$) a spodní $\mathcal{H}_{\mathcal{L}}$ konvexní obálky. Do $\mathcal{H}_{\mathcal{U}}$ se přidávají body ležící nalevo od spojnice pivotů q_1 a q_3, do $\mathcal{H}_{\mathcal{L}}$ naopak body ležící na pravé straně od spojnice týchž bodů.

Posledním krokem je spojení obou částí konvexní obálky do výsledné množiny $\mathcal{H}$. Nejprve se do $\mathcal{H}$ přidá bod q_3, poté se provede rekurze $\mathcal{H}_{\mathcal{U}}$, pomocí níž se do $\mathcal{H}$ přidají body z horní části konvexní obálky, poté se přidá bod q_1 a posledním krokem je provedení rekurze $\mathcal{H}_{\mathcal{L}}$, díky níž se do konvexní obálky přidají i body ze spodní části konvexní obálky.

4.2 Generalizace budov

4.2.1 Minimum Enlosing Rectangle

Algoritmus se snaží vyhledat takovou hranu konvexní obálky $\mathcal{H}$, aby po vytvoření opsaného obdélníka s jednou stranou kolineární s touto hranou měl takto vytvořený obdélník minimální plochu.

Algoritmus využívá již zkonstruovanou konvexní obálku $\mathcal{H}$, pro jejíž hrany jsou vypočítávány směrnice σ hrany e:

$$\tan \sigma = \frac{d_x}{d_y}, \tag{1}$$

kde d_x a d_y jsou souřadnicové rozdíly počátečního a koncového bodu hrany konvexní obálky $\mathcal{H}$. Všechny body množiny S se následně pomocí matice rotace R otočí o úhel $-\sigma$:

$$S_R = R(-\sigma)S \tag{2}$$

Pro otočené body množiny se vytvoří MMB s následujícími souřadnicemi vrcholů:

$$V_1 = [\underline{x}, \underline{y}], V_2 = [\overline{x}, \underline{y}], V_3 = [\overline{x}, \overline{y}], V_4 = [\underline{x}, \overline{y}], \tag{3}$$

kde $\underline{x}, \overline{x}, \underline{y}, \overline{x}$ jsou minimální a maximální souřadnice natočených bodů množiny S_R. Vypočte se plocha vytvořeného *MMB*:

$$A = (\overline{x} - \underline{x})(\overline{y} - \underline{y}), \tag{4}$$

jež se následně porovná s minimální uloženou plochou. Je-li vypočtená plocha A menší než je plocha minimální, uloží se jako nové minimum. Dále se uloží pro tuto plochu uloží úhel σ_{min} a MMB_{min}. Po ukončení výpočtu totoho cyklu je již vypočten výsledný úhel natočení budovy σ_{min}. Následně je nutné *MMB* natočit o tento úhel a zmenšit tak, aby byla plocha *MMB* rovna ploše generalizovaného polygonu. Tento postup bude používán i pro další metody.

MMB_{min} se otočí o úhel σ_{min}:

$$\mathcal{R} = R(\sigma_{min})MMB_{min}, \tag{5}$$

následně se jeho plocha proporcionálně zmenší vůči těžišti tak, aby byla rovna ploše generalizovaného polygonu. Nejprve je nutné vypočítat poměr k plochy A_b generalizované budovy a plochy A *MMB*

$$k = \frac{A_b}{A}. \tag{6}$$

Souřadnice těžiště T jsou aritmetickým průměrem souřadnic vrcholů $\mathcal{R}$. Vypočítají se nové vrcholy V obdélníku $\mathcal{R}_r$:

$$V_i = T + \sqrt{k}u_i, \tag{7}$$

kde u_i jsou směrové vektory vrcholů a těžiště odvozené z Pythagorovy věty:

$$u_i = V_i - T \tag{8}$$

4.2.2 Metoda Wall Average

Metoda je velmi komplexní a citlivá na nepravé úhly. Pro každou hranu je pro výsledky operace mod() vypočítáván vážený průměr, v němž je vahou délka hrany.

Pro každou hranu budovy se zredukuje směrnice σ:

$$\Delta\sigma = \sigma_i - \sigma', \tag{9}$$

kde σ_i je směrnice hrany vypočtená dle vztahu (1) a σ' je směrnice první hrany vypočtená dle vztahu (1).

V dalším kroku se vypočítá zaokrouhlený podíl:

$$k_i = \left[\frac{2\Delta\sigma_i}{\pi}\right] \tag{10}$$

a následně se dopočítá zbytek po dělení:

$$r_i = \Delta\sigma_i - k_i\frac{\pi}{2}. \tag{11}$$

Výsledný směr natočení budovy roven:

$$\sigma = \sigma' + \sum\frac{r_i s_i}{s_i}. \tag{12}$$

Následně se vytvoří MMB, který se otočí o úhel σ a zmenší do požadované plochy. Tento postup byl již shrnut v kapitole 4.2.1.

4.2.3 Metoda Longest Edge

Hlavní směrem budovy se dle této metody rozumí směrnice nejdelší hrany budovy, druhý směr je kolmý na směr hlavní. Pomocí cyklu se postupně vypočítává délka d všech hran generalizované budovy, postupně se zjišťuje, je-li vypočtená délka větší než je dosavadně největší uložená délka.

$$d = \sqrt{d_x^2 + d_y^2} \tag{13}$$

Následně se vytvoří MMB, který se otočí o úhel σ a zmenší do požadované plochy. Tento postup byl již shrnut v kapitole 4.2.1.

4.2.4 Metoda Weighted Bisector

Algoritmus hledá dvě nejdelší úhlopříčky generalizované budovy, pro tyto dvě úhlopříčky se následně vypočítají směrnice (rovnice (1)) a délky hran (rovnice (13)). Výsledný směr se získá z váženého průměru:

$$\sigma = \frac{s_1\sigma_1 + s_2\sigma_2}{s_1 + s_2}. \tag{14}$$

Následně se vytvoří MMB, který se otočí o úhel σ a zmenší do požadované plochy. Tento postup byl již shrnut v kapitole 4.2.1.

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.

```
//Find max Y and min X to determine origin of local system
QPointF pymax = *std::max_element(polygon.begin(), polygon.end(), sortByY());
double yy = pymax.y();
QPointF pxmin = *std::min_element(polygon.begin(), polygon.end(), sortByX());
double xx = pxmin.x();

if (yy > y_max)
    y_max = yy;
if (xx < x_min)
    x_min = xx;
```

Obrázek 1: 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.

```
//Additionaly find min Y and max X to determine the scale in both directions
QPointF pymin = *std::min_element(polygon.begin(), polygon.end(), sortByY());
double yy_ = pymin.y();
QPointF pxmax = *std::max_element(polygon.begin(), polygon.end(), sortByX());
double xx_ = pxmax.x();


if (yy_ < y_min)
    y_min = yy_;
if (xx_ > x_max)
    x_max = xx_;
```

Obrázek 2: Výpočet Ymin a Xmax

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 = 952.0;
double canvas_height = 748.0;

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

Souřadnice v souřadnicovém systému widgetu byly definovány následujícími vzorci.

$$X_{W_i} = -k * (Y_{S-JTSK_i} - Y_{S-JTSK_{\max}}) \tag{15}$$

$$Y_{W_i} = k * (X_{S-JTSK_i} - X_{S-JTSK_{\min}}) \tag{16}$$

```
//Transform coordinates from JTSK to canvas
for (int unsigned i = 0; i < buildings_.size(); i++)
{
    QPolygonF pol = buildings_[i];
    for (int j = 0; j < pol.size(); j++)
    {
        double temp = pol[j].x();
        pol[j].setX(-k*(pol[j].y()-y_max));
        pol[j].setY(k*(temp-x_min));

    }

    buildings.push_back(pol);
```

Obrázek 4: Transformace souřadnic

5.2 Načítání souřadnic s reálným číslem

V průběhu práce jsme narazili na problém, kdy ačkoliv jsme souřadnice přetypovali ze stringu na double, stála se nám načítaly souřadnice typu integer. Zaokrouhlení souřadnic na metry následně vedlo k viditelné destrukci načtených polygonů. Problém byl v tom, že souřadnice QPointu jsou definovány typem integer. Proto byly všechny objekty datového typu QPoint nahrazeny typem QpointF, jehož souřadnice jsou typu float.

6 Vstupní data

Polygonová mapa se do projektu nahrává z textového souboru, ve kterém jsou uloženy souřadnice lomových bodů budov v souřadnicovém systému S-JTSK s centimetrovou přesností. Data byla získána ze systému ZABAGED, pomocí QGISu se vyexportovaly souřadnice do csv souboru, jehož formát se pak pomocí jednoduchých funkcí v MS Excel upravil do špagetového modelu. Pro úspěšné načtení souboru musí data splňovat specifický formát, bez něhož nebude zajištěno správné načítání dat.

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

- každý vrchol polygonu je definován na jednotlivém řádku,
- na řádku je pořadí proměnných id » y » x , hodnoty jsou od sebe odděleny jednou mezerou,
- id je identifikátor jednotlivých vrcholů polygonu, x a y jsou souřadnice vrcholů polygonu,
- nový polygon má vždy hodnotu id prvního vrcholu rovnu 1, od této hodnoty se postupně načítají body, a to až do okamžiku, kdy program dojde k dalšímu identifikátoru s označením 1.

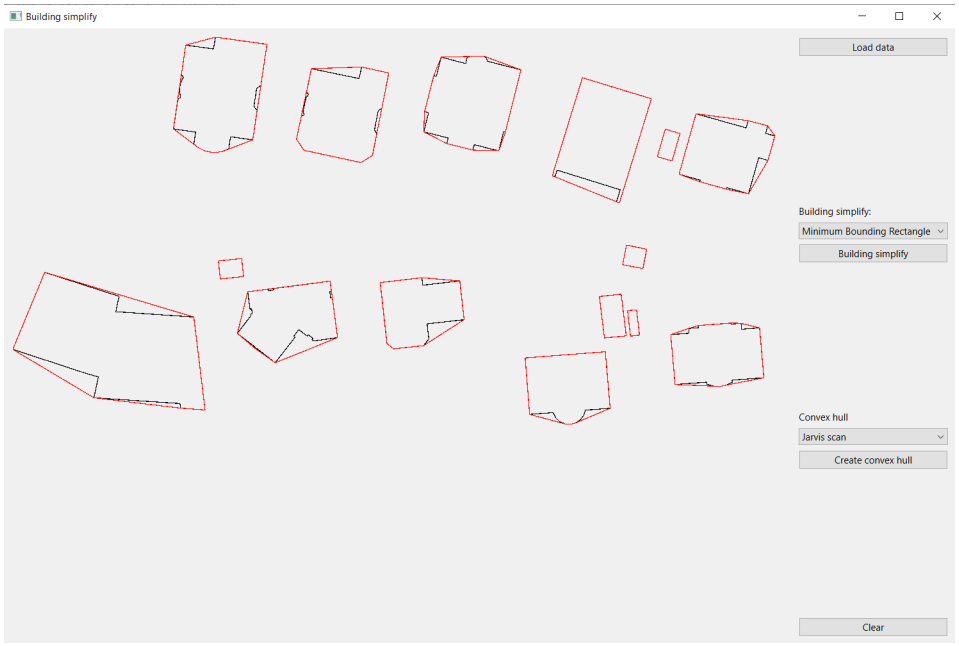
```
1 742946.03 1039311.05
2 742949.40 1039310.41
3 742948.75 1039307.15
4 742945.39 1039307.84
5 742946.03 1039311.05
1 742942.31 1039287.88
2 742939.85 1039288.61
3 742941.18 1039293.16
4 742943.61 1039292.46
5 742942.31 1039287.88
```

Obrázek 5: Špagetový model vstupního souboru

Souřadnice byly následně transformovány z S-JTSK do souřadnicového systému widgetu a změněno bylo měřitko tak, aby se zobrazovala oblast se zobrazovanými polygony s maximálním přiblížením. Tento postup byl již popsán v kapitole 4.

7 Výstupní data

Po načtení souboru lze nad polygony provádět dva druhy operací. Po stisknutí tlačítka *Create convex hull* se vytvoří konvexní obálky nad jednotlivými polygony. V combo boxu lze vybrat, zdali se konvexní obálka vytvoří metodou *Jarvis scan* nebo *Quick hull*. Výsledek obou metod je stejný a zobrazuje ho následující obrázek 6.



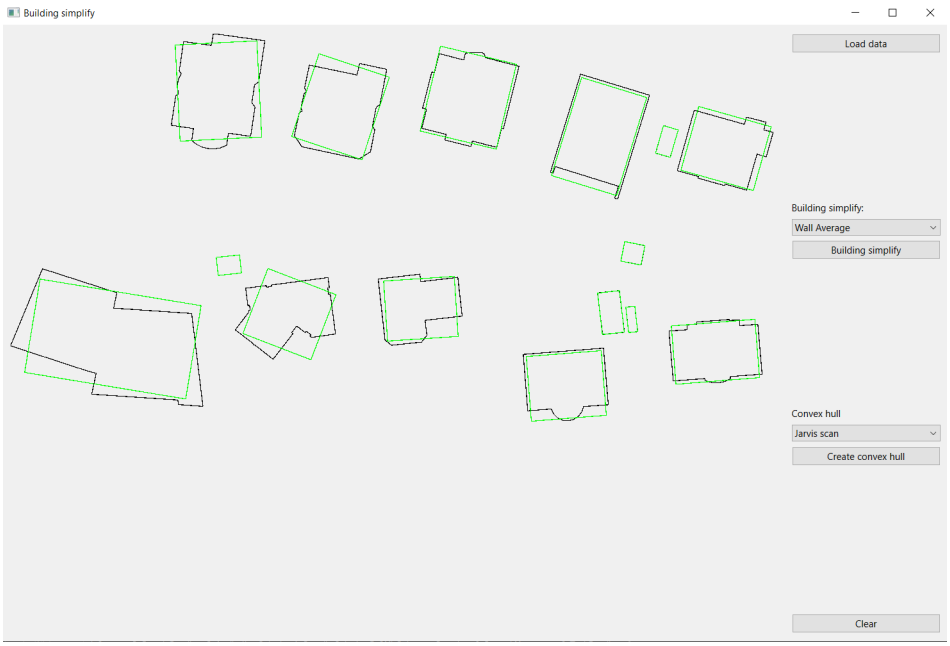
Obrázek 6: Konvexní obálka nad polygony

Druhým typem operací, které lze v aplikaci provádět, je generalizace budov. Po stisknutí tlačítka *Building simplify* je možné provést čtyři následující metody pro výpočet hlavního směru orientace generalizované budovy.

- Minimum Area Enclosing Rectangle
- Wall Avarage
- Longest Edge
- Weighted Bisector

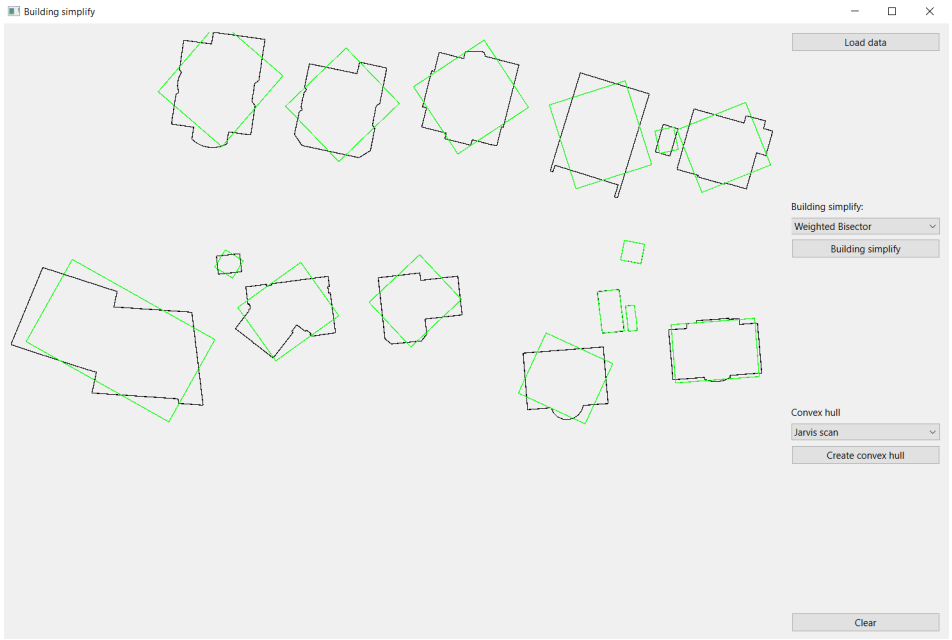
Výsledky jednotlivých metod zobrazují obrázky 7, 8, 9, 10.

Obrázek 7: Minimum Area Enclosing Rectangle



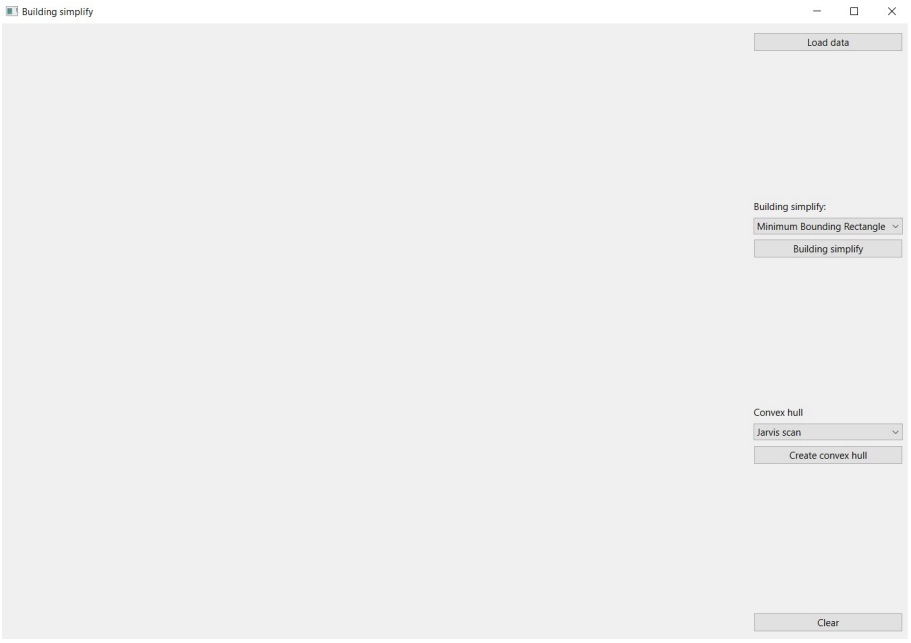
Obrázek 8: Wall Avarage

Obrázek 9: Longest Edge



Obrázek 10: Weighted Bisector

8 Printscreen vytvořené aplikace



Obrázek 11: Úvodní okno aplikace

9 Dokumentace

Kód zahrnuje 5 tříd – Algorithms, Draw, SortByX, SortByY a Widget, které budou následně detailněji popsány.

9.1 Třída Algorithms

Třída Algorithms obsahuje 4 funkce :

- double get2LinesAngle(QPoint &p1, QPoint &p2, QPoint &p3, QPoint &p4);
- int getPointLinePosition(QPoint &a, QPoint &p1, QPoint &p2);
- double getPointLineDistance(QPointF &a, QPointF &p1, QPointF &p2);
- QPolygonF cHull (std::vector <QPointF> &points);

- QPolygonF qHull (std::vector <QPointF> &points);
- void qHullRecursive(int r, int s, std::vector<QPointF> &points, QPolygonF &ch);
- std::vector <QPointF> rotate(std::vector <QPointF> &points, double sigma);
- std::tuple<std::vector<QPointF>, double> minMaxBox(std::vector <QPointF> &points);
- QPolygonF minAreaEnclosingRectangle(std::vector <QPointF> &points);
- QPolygonF wallAverage(std::vector <QPointF> &points);
- QPolygonF longestEdge(std::vector <QPointF> &points);
- QPolygonF weightedBisector(std::vector <QPointF> &points);
- double LH(std::vector <QPointF> &points);
- std::vector <QPointF> resizeRectangle(std::vector <QPointF> &points, std::vector <QPointF> &er);

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 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 getPointLineDistance(QPointF &a, QPointF &p1, QPointF &p2);
Počítá vzdálenost mezi bodem a přímkou definovanou dvěma vrcholy polygonu. Vstupní hodnoty jsou 3 body datového typu QpointF.

9.2 Třída Algorithms

QPolygonF cHull (std::vector <QPointF> &points)
Funkce vytváří konvexní obálku metodou Jarvis scan popsanou v kapitole 4.1.1. Vstupním argumentem funkce je vektor obsahující body množiny S, body jsou uloženy jako *QPointF*. Návratovým typem funkce je *QPolygonF*, tedy polygon se souřadnicemi uloženými s přesností float obsahující souřadnice vrcholů vypočtené konvexní obálky.

QPolygonF qHull (std::vector <QPointF> &points)
Funkce vytváří konvexní obálku metodou Quick Hull popsanou v kapitole 4.1.2. Vstupním argumentem funkce je vektor obsahující body množiny S, které jsou uloženy jako *QPointF*. Návratovým typem funkce je *QPolygonF*, tedy polygon se souřadnicemi uloženými s přesností float, který obsahuje souřadnice vrcholů vypočtené konvexní obálky.

void qHullRecursive(int r, int s, std::vector<QPointF> &points, QPolygonF &ch)
Funkce je rekurzivní funkcí k výše uvedené funkci qHull. Touto funkcí se přidávají části konvexní obálky $\mathcal{H}_{\mathcal{U}}$ a $\mathcal{H}_{\mathcal{L}}$ do výsledné konvexní obálky $\mathcal{H}$. Vstupními argumenty jsou integery r a s, jež označují směr linie mezi pivoty q_1 a q_3. Tyto body jsou uloženy ve vektoru *points* (popsán níže) na prvních dvou místech, volá-li se tedy funkce například s těmito parametry: qHullRecursive(1,0,su,qh), bude počátečním bodem linie pivot q_3 a konečným bodem pivot q_1. Třetím vstupním argumentem je vektor bodů *QPointF points*, do této proměnné se ukládá $\mathcal{H}_{\mathcal{U}}$ nebo $\mathcal{H}_{\mathcal{L}}$. Posledním vstupním argumentem je *QPolygonF*, do nějž se bude ukládat horní/spodní část konvexní obálky. Návratový typ funkce je void, funkce tedy nevrací žádnou proměnnou, pouze ukládá body do předem vytvořené proměnné.

std::vector <QPointF> rotate(std::vector <QPointF> &points, double sigma);
Otáčí množinu bodů u úhel sigma. Vstupními hodnotami jsou vektor bodů datového typu QPointF a úhel sigma, která vstupuje do transformační rovnice. Funkce opět vrací vektor bodů datového typu QPointF.

std::tuple<std::vector<QPointF>, double> minMaxBox(std::vector <QPointF> &points); Vytváří kolem polygonu minimální ohraničující obdélník s vrcholy s extrémními souřadnicemi polygonu, nad kterým se obdélník vytváří. Vstupní hodnotou je vektor datového typu QPointF. Funkce vrací minimální ohraničující obdélník ve vektoru QPointF a jeho plochu s datovým typem double. Aby funkce mohla vracet dva objekty, je nutné je vracet jako tuple.

QPolygonF minAreaEnclosingRectangle(std::vector <QPointF> &points); Generalizuje polygon, jehož hlavní směr je určen metodou Minimum Area Enclosing Rectangle. Hlavní směr je zde určen stranou konvexní obálky, která minimalizuje plochu minimální ohraničujícího obdélníku otočeného o její směrnici. Do funkce vstupuje vektor datového typu QPointF a algoritmus vrací natočený obdélník, který je uložen v datovém typu QPolygonF.

QPolygonF wallAverage(std::vector <QPointF> &points); Generalizuje polygon, jehož hlavní směr je určen metodou Wall Avarage, kde je hlavní směr dán váženým průměrem, do kterého vstupují směrnice první strany polygonu, rezidua po dělení směrnic hodnotou $\pi/2$ a délka strany. Do funkce vstupuje vektor datového typu QPointF a algoritmus vrací natočený obdélník, který je uložen v datovém typu QPolygonF.

QPolygonF longestEdge(std::vector <QPointF> &points); Generalizuje polygon, jehož hlavní směr je určen metodou Longest Edge. Hlavní směr je zde představován nejdelší stranou budovy. Do funkce vstupuje vektor datového typu QPointF a algoritmus vrací natočený obdélník, který je uložen v datovém typu QPolygonF.

QPolygonF weightedBisector(std::vector <QPointF> &points); Generalizuje polygon, jehož hlavní směr je určen metodou Weighted Bisector, která hlavní směr určuje váženým průměrem směrů dvou nejdelších úhlopříček. Do funkce vstupuje vektor datového typu QPointF a algoritmus vrací natočený obdélník, který je uložen v datovém typu QPolygonF.

double LH(std::vector <QPointF> &points); Počítá plochu budovy pomocí L'Hullierova vzorce. Vstupní hodnotou je vektor datového typu QPointF, výstupní hodnotou je plocha budovy typu double.

std::vector <QPointF> resizeRectangle(std::vector <QPointF> &points, std::vector <QPointF> &er); Mění velikost generalizované budovy tak, aby se její plocha shodovala s plochou původní budovy. Vstupními hodnotami jsou vektor bodů datového typu QPointF představující původní budovu a vektor bodů datového typu QPointF představující generalizovanou budovu. Funkce vrací opět vektor bodů datového typu QPointF, který představuje generalizovanou budovu s plochou odpovídající původní budově.s

9.3 Třída Draw

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

- void paintEvent(QPaintEvent *event);
- void clear();
- QPoint getPoint()return q;
- std::vector<QPolygonF> getPolygons(){return buildings;}
- void setCh(std::vector <QPolygonF> &ch_{chs = ch_;}
- void setEr(std::vector <QPolygonF> &er_){ers = er_;}
- void loadData(QString &file_name);

a následující proměnné:

- std::vector<QPointF> points;
- std::vector <QPolygonF> ers, chs;
- std::vector <QPolygonF> buildings;
- QPointF p;
- double y_max = 0, x_min = 999999999;
- double y_min = 999999999, x_max = 0;

• void paintEvent(QPaintEvent *event);
Funkce nastavuje grafické atributy vykreslovaných objektů a kreslí na plátno zadané polygony, jejich konvexní obálku a jejich generalizaci.

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

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

void setCh(std::vector <QPolygonF> &ch_{chs = ch_;}

void setEr(std::vector <QPolygonF> &er_){ers = er_;}

void loadData(QString &file_name);
Funkce načítá data z textového souboru a ukládá je do vektoru QPolygonF. Funkce také rovnou transformuje souřadnice s S-JTSK do souřadnicového systému plátna. Vstupní argumentem je cesta k souboru, jejž chceme načíst do aplikace.

9.4 Třída SortByX

Řadí body podle xové souřadnice.

9.5 Třída SortByY

Řadí body podle y-ové souřadnice.

9.6 Třída Widget

Třída Widget obsahuje 3 metody:

- void on_pushButtonClear_clicked();
- void on_pushButtonSimplify_clicked();
- void on_pushButtonLoadData_clicked();
- void on_pushButtonConvexHull_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 Building simplify a provádí jednotlivé metody generalizace budov na základně indexu 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