ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

FAKULTA STAVEBNÍ, OBOR GEODÉZIE A KARTOGRAFIE

KATEDRA GEOMATIKY

název předmětu

Algoritmy digitální kartografie a GIS

název úlohy

Generalizace budov

školní rok	studijní skup.	číslo zadání	zpracoval, email	datum	klasifikace
2021/22	60	-	Jakub Šimek, Jan Kučera	7.11.21	

Obsah

1. Zadání

Úloha č. 2: Generalizace budov

Vstup: množina budov $B = \{B_i\}_{i=}^{n}$, *budova* $B_i = \{P_{i,j}\}_{j=1}^{m}$.

Výstup: $G(B_i)$.

Ze souboru načtěte vstupní data představovaná lomovými body budov. Pro tyto účely použijte vhodnou datovou sadu, např. ZABAGED.

Pro každou budovu určete její hlavní směry metodami:

- Minimum Area Enclosing Rectangle,
- Wall Average.

U první metody použijte některý z algoritmů pro konstrukci konvexní obálky. Budovu nahraďte obdélníkem se středem v jejím těžišti orientovaným v obou hlavních směrech, jeho plocha bude stejná jako plocha budovy. Výsledky generalizace vhodně vizualizujte.

Odhadněte efektivitu obou metod, vzájemně je porovnejte a zhodnoťte. Pokuste se identifikovat, pro které tvary budov dávají metody nevhodné výsledky, a pro které naopak poskytují vhodnou aproximaci.

Hodnocení:

Krok	**Hodnocení**
Generalizace budov metodami Minimum Area Enclosing Rectangle a Wall Average	15b
Generalizace budov metodou Longest Edge	*+5b*
Generalizace budov metodou Weighted Bisector	*+8b*
Implementace další metody konstrukce konvexní obálky.	*+5b*
Ošetření singulárního případu u při generování konvexní obálky	*+2b*
Max celkem:	**35b**

Čas zpracování: 3 týdny.

Obrázek 1 - zadání

2. Bonusové úlohy

Ke standartnímu zadání byly zpracovány tyto bonusové úlohy:

1. Generalizace budov metodou Longest Edge
2. Generalizace budov metodou Weighted Bisector
3. Implementace další metody konstrukce konvexní obálky – *Graham Scan*

3. Popis a rozbor problému

Na vstupu máme množinu budov $B = \{B_i\}_{i=1}^{n}$, kde budova $B_i = \{p[x,y]_{i,j}\}_{j=1}^{m}$.
Pro každou budovu pak hledáme její generalizaci $G(B_i)$.
Generalizací se rozumí zjednodušení tvaru budovy za účelem redukce objemu dat.

4. Popis algoritmů

4.1 Hledání konvexní obálky

Algoritmus Jarvis Scan

První metodou pro generování konvexní obálky je metoda Jarvis Scan.
V prvním kroku je potřeba nejprve nalézt pivota q:

$$q = min_{\forall P_j \in B_i}(y_j).$$

Bod q je přidán do konvexní obálky H.
Dále je vybrán bod p_{j-1} tak, aby přímka dána body q a p_{j-1} byla rovnoběžná s osou x. Pak je vždy v cyklu přidáván do konvexní obálky bod s maximálním úhlem $\measuredangle(p_{j-1}, p, p_{j+1})$.

Implementace algoritmu Jarvis Scan

1. Nalezení pivota $q = \min(y_j)$, $q \gg$ H
2. Inicializuj: $p_{j-1} \in X, p_j = q, p_{j+1} = p_{j-1}$
3. Opakuj, dokud $p_{j+1} != q$:
4. $\quad p_{j+1} = argmax\measuredangle(p_{j-1}, p_j, p_i)$, $p_{j+1} \gg$ H
5. $\quad p_{j-1} = p_j, p_j = p_{j+1}$

Algoritmus Graham Scan

Nejdříve najdeme pivota q. Bod q je prvním bodem konvexní obálky. Pro všechny vstupní body spočteme úhel ω_j měřený od rovnoběžky s osou x s vrcholem v bodě q. Množinu vstupních bodů pak setřídíme podle velikosti ω. Tím vznikne star-shaped polygon.
Do zásobníku přidáme bod q a první bod ze star-shaped polygonu. Dále je testován další bod v pořadí. Za předpokladu, že je splněna podmínka levotočivosti, je bod přidán do zásobníku. Pak je testován 4. bod. Jestliže je splněna podmínka levotočivosti, je také přidán do zásobníku. Jestliže není podmínka splněna, je poslední bod smazán ze zásobníku a testuje čtvrtý bod z bodu druhého. Po vykonání cyklu je v zásobníku konvexní obálka.

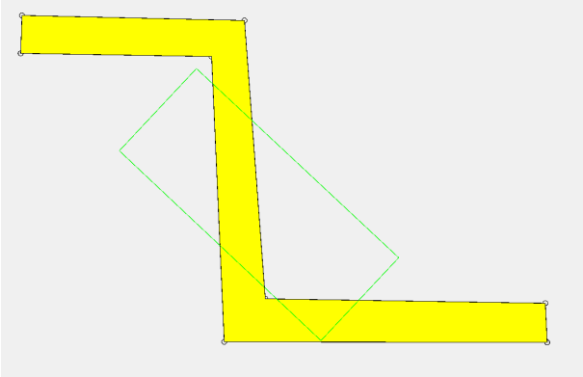
Implementace algoritmu Graham Scan

1. Nalezení pivota $q = \min(y_j)$, $q \gg$ H
2. Setřídění $\forall p_j$ dle $\omega_j \measuredangle < p_j, q, x >$. Index k odpovídá setříděnému pořadí.
3. Nalezení bodů se stejným ω a vymazání bodu bližšího ke q.
4. Inicializuj $k = 2, S = \emptyset$
5. $S \ll q, S \ll p_1$, (indexy posledních dvou prvků p_t, p_{t-1})
6. Opakuj pro $k < m$:
7. $\quad$ Pokud p_k vlevo od p_{t-1}, p_t:
8. $\quad\quad S \ll p_k$
9. $\quad\quad j++$
10. $\quad$ Jinak $S.pop()$

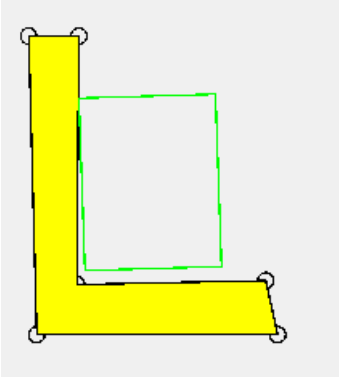
4.2 Generalizace budov

Metoda Minimum Area Enclosing Rectangle

Tato metoda slouží k zjištění hlavního směru budovy. Tento směr je určen směrem delší strany ohraničujícího obdélníku, který má minimální plochu.
U tohoto algoritmu nastávají problémy u budov tvaru L a Z, jak je vidět na obrázku.



Obrázek 2



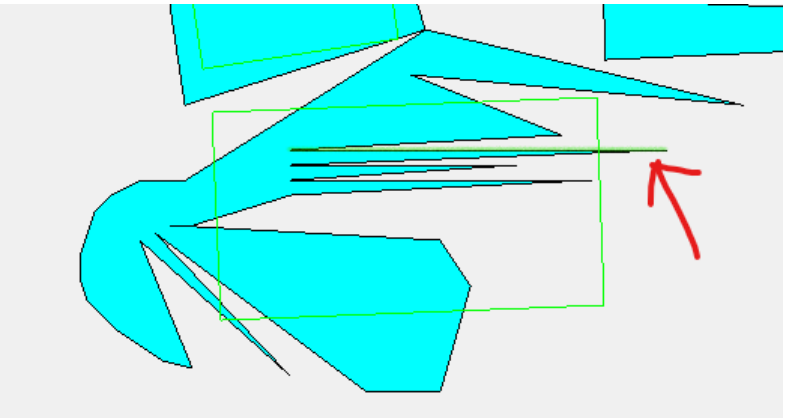
Obrázek 3

Implementace algoritmu Minimum Area Enclosing Rectangle

1. Nalezení H $= CH(S)$
2. Inicializuj $R = MMB(S), \underline{A} = A(MMB(S))$
3. Opakuj pro každou hranu e obálky H:
4. Spočti směrnici σ hrany e
5. Otoč S o $-\sigma$: $S_r = R_z(-\sigma)S$
6. Najdi $MMB(S_r)$ a urči $A(MMB(S_r))$
7. Pokud $A < \underline{A}$:
8. $\underline{A} = A, \underline{MMB} = MMB, \underline{\sigma} = \sigma$
9. $R = R_z(\sigma)MMB$

Metoda Longest Edge

Algoritmus pro detekci hlavního směru budovy. Hlavním směrem budovy je jednoduše nejdelší hrana polygonu.
Tato metoda má nevýhodu, že nejdelší hrana nemusí ve skutečnosti reprezentovat hlavní směr budovy (viz obrázek). Proto není vhodná pro atypické tvary budov.



Obrázek 4

Implementace metody Longest Edge

1. Inicializace vektoru dvojic délka hrany/směrnice přímky
2. Opakuj pro všechny body budovy:
3. $$s_j = \sqrt{(x_{j+1} - x_j)^2 + (y_{j+1} - y_j)^2}$$
4. $$\sigma_j = atan2(\frac{y_{j+1} - y_j}{x_{j+1} - x_j})$$
5. Seřazení vektoru dvojic podle velikosti s_j
6. Uložení σ_{smax} – poslední prvek ve vektoru dvojic
7. Vytvoření *enclosing rectangle*

Wall Average

Pro každou stranu budovy je spočtena směrnice. Na ní je pak aplikována metoda $mod(\frac{\pi}{2})$, která slouží k nalezení zbytku po dělení. Výsledný směr natočení budovy je pak dán váženým průměrem s váhou rovnou délce příslušné strany.

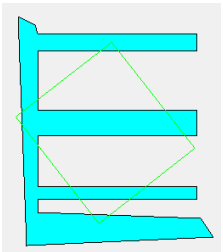
Implementace metody Wall Average

1. Inicializace
   $\sigma = 0$ – směr natočení budovy
   $sumS_i = 0$ – obvod budovy
   $\sigma\_$ - směrnice první hrany budovy
2. Cyklus přes všechny body budovy:
3. $s_j = \sqrt{(x_{j+1} - x_j)^2 + (y_{j+1} - y_j)^2}$
4. $\sigma_j = atan2(\frac{y_{j+1} - y_j}{x_{j+1} - x_j})$
5. $d\sigma_i = \sigma_i - \sigma\_$ - rozdíl směrnic
6. $k_i = round(\frac{2 * d\sigma_i}{\pi})$
7. $r_i = d\sigma_i - k_i * \frac{\pi}{2}$
8. $\sigma += r_i * S_i$
9. $sumS_i += S_i$
10. $\sigma = \sigma\_ + \frac{\sigma}{sumS_i}$ – vážený průměr
11. Vytvoření *enclosing rectangle*

Metoda Weighted Bisector

Tento algoritmus slouží pro detekci hlavního směru budovy. Hlavním směrem budovy je vážený průměr směrníků dvou nejdelších úhlopříček v polygonu. Váhami jsou délky úhlopříček.

Obecně dává dobré výsledky. Přesto může někdy špatně odhadnout celkovou orientaci budovy (viz obrázek).



Obrázek 5

Implementace budovy Weighted Bisector

Body 1 až 5 jsou stejné jako u metody *Longest Edge*.

6. Uložení posledních dvou dvojic délka/směrnice ze srovnaného vektoru podle velikosti hrany do proměnných $S_1, \sigma_1, S_2, \sigma_2$
7. Vážený průměr směru podle vzorce

$$\sigma = \frac{s_1 \cdot \sigma_1 + s_2 \cdot \sigma_2}{s_1 + s_2}$$

8. Vytvoření *enclosing area*

5. Problematické situace a jejich rozbor

Jedním z velkých problémů je souřadnicový systém načítaných dat. Widget *Canvas* implementovaný v aplikaci má souřadnice pouze v 1. kvadrantu a jedná se o pixelové souřadnice. Klasický monitor je schopný zobrazit objekty v přibližném rozmezí souřadnic *x* a *y* cca [0, 900]. Widget je samozřejmě natahovací, ale objekty např. v souřadnicích S-JTSK tu rozhodně bez transformace nebudou vidět. Nemluvě o souřadnicích, které nebudou v pravoúhlém XY systému. Pro objekty z Jihlavy byla v kódu zavedena primitivní transformace jen pro účely funkčností načítací nadstavby reálných objektů.

```
//If there is x coor to read
if (pairIterator == 0)
{
        //Convert string value to integer value
        x = (std::stod(coordinate)-668000)/2;
        x = (std::stod(coordinate));
        pairIterator++;
}
//If there is y coor to read
else if (pairIterator == 1)
{
        //Convert string value to integer value
        y = (std::stod(coordinate)-1130000)/2 +200;
        y = (std::stod(coordinate));
        pairIterator++;
}
```

Obrázek 6

Ale je jasné, že takový způsob je nesmyslný aplikovat. Proto je zde uveden jen na ukázku. Takový výstup by byl na následujícím obrázku (část Jihlavy).

Obrázek 7

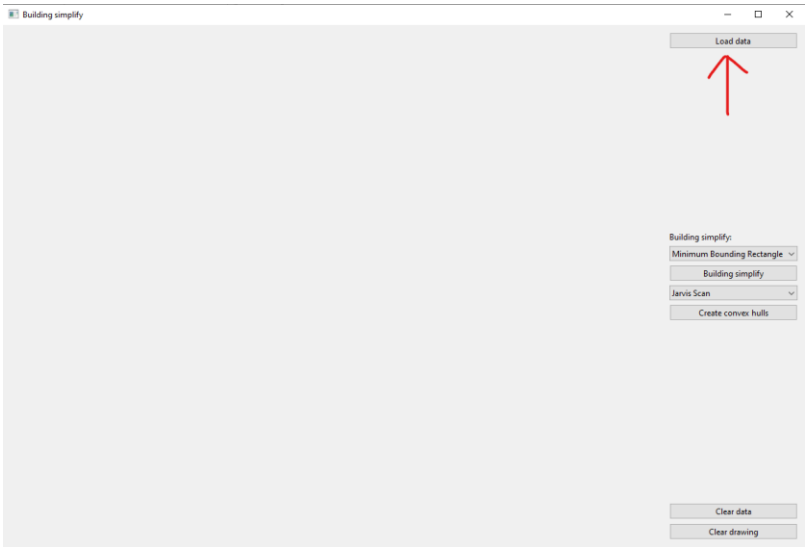
Dalšími problémy jsou čistota souřadnic exportovaných dat. Je potřeba, aby načtená data splňovala kritéria Simple features. I když jsou splněna, tak po transformaci do *Canvas* mohou opět vzniknout problémy narušující tato kritéria. Tyto problémy jsou pak promítnuté do nefunkčnosti algoritmů (hlavně Graham scan).

6. Vstupní data

Data do aplikace můžeme importovat dvěma způsoby. První způsob je zadání polygonu uživatelem klikáním do grafického okna. Druhým způsobem je pak import polygonů z textového souboru, ve kterém jsou uloženy polygony ve špagetové reprezentaci.

Import textového souboru

Pro načítaní dat bylo do widgetu vloženo tlačítko *Load data*.



Obrázek 8

Po rozkliknutí tlačítka se objeví Windows dialog pro výběr souborů.
Formát vstupních dat nebyl předem zadán. Implementace načítání souborem je provedena s uvážením špagetového modelu exportovaného ze softwaru QGIS formou WKT. Jedná se o CSV soubor. Program předpokládá, že se na každém řádku nachází WKT *Multipolygon* obsahující sled jednotlivých jeho bodů. Příklad je na následujícím obrázku.

```
1  WKT,
2  "MULTIPOLYGON (((110.1 100,120 150,150 180,170 130,125 125)))"
3  "MULTIPOLYGON (((200 200,230 400,390 350,350 220, 275 275)))"
4  "MULTIPOLYGON (((390 350,230 450,200 450,190 455,180 460,170 470,...
5  "MULTIPOLYGON (((500 100,510 370,730 360,715 338,523 331,520 110)))"
```

Obrázek 9

Řádky se mohou lišit v souřadnicích a v jejich množství. Nicméně struktura musí zůstat stejná. Tedy na prvním řádku je z QGIS vypsaná hlavička a pak následují řádky se souřadnicemi ve WKT. Souřadnicový řádek musí obsahovat klíčové slovo *Multipolygon*, následovat musí 3 kulaté závorky, poté sled souřadnic *(x1 y1, x2 y2, x3 y3, ...)* a ty jsou opět zakončeny trojicí kulatých závorek. Celý řádek je v uvozovkách.
Po kliknutí na tlačítko *Load* data se do do datové struktury QString uloží řetězec obsahující cestu k souboru vstupních dat získanou z dialogu Windows průzkumníka.

```
void Widget::on_pushButton_load_clicked()
{
        //Open dialog to choose data file and store its path to QString
        QString path(QFileDialog::getOpenFileName(this, tr("Open file with polygons"), "../", tr("CSV Files (*.csv)")));

        //Convert QString path to string path
        std::string filename = path.toStdString();

        //Read the file with chosen path
        std::vector<QPolygon> polygon_vector = CSV::read_csv(filename);

        //Draw polygons
        ui->Canvas->drawPolygons(polygon_vector);
}
```

Obrázek 10

Čtení dat bylo realizováno statickou metodou v třídě CSV, jejíž vstupem je do klasického stringu převedená cesta k souboru s uvedeného QStringu.

```
class CSV
{
public:
        CSV();

        static std::vector<QPolygon> read_csv(std::string &filename);
};
```

Obrázek 11

Metoda nejdříve zkontroluje, zdali je soubor správně načtený.

```
// Make sure the file is open
if(!myFile.is_open()) throw std::runtime_error("Could not open file");
```

Obrázek 12

Poté přečte hlavičku souboru (pouze 1. řádek).

```
// Read the column names
if(myFile.good())
{
    // Extract the first line in the file
    std::getline(myFile, line);

    // Create a stringstream from line
    std::stringstream ss(line);

    // Extract each column name
    while(std::getline(ss, colname, ',')){
    }
}
```

Obrázek 13

Poté metoda bude projíždět každý řádek zvlášť a bude ukládat do struktury QPolygon všechny body z aktuálního řádku. Z každého řádku odstraní metodami *erase* a *getline* veškeré nečíselné znaky a pomocí *pair iterátoru* postupně z řádku uloží souřadnice každého bodu. Zde je hlavní část načítacího kódu.

```
        // Help vars
        int pairIterator = 0;
        int x, y;
        std::string coordinate_pair, coordinate;
        QPolygon polygon;

        //Go through every point in polygon
        while(std::getline(ss1, coordinate_pair, ','))
        {
                std::stringstream ss2(coordinate_pair);

                //Go through every coordinate in point
                while(std::getline(ss2, coordinate, ' '))
                {
                        if (coordinate == "")
                                continue;

                        //If coordinate pair incomplete (must include both x and y)
                        if (pairIterator < 2)
                        {
                                //If there is x coor to read
                                if (pairIterator == 0)
                                {
                                        //Convert string value to integer value
//                                      x = (std::stod(coordinate)-668000)/2;
                                        x = (std::stod(coordinate));
                                        pairIterator++;
                                }
                                //If there is y coor to read
                                else if (pairIterator == 1)
                                {
                                        //Convert string value to integer value
//                                      y = (std::stod(coordinate)-1130000)/2 +200;
                                        y = (std::stod(coordinate));
                                        pairIterator++;
                                }
                        }
                }

                //Store the pair of coordinates and reset pair iterator
                if (pairIterator == 2)
                {
                        QPoint p(x,y);
                        polygon << p;
                        pairIterator = 0;
                }
            // Increment the column index
            colId++;
        }
        //Store the object of a line
        result.push_back(polygon);
    }
```

Obrázek 14

Nakonec se postupně všechny polygony vykreslí na *Canvas* metodou *drawPolygons* ze třídy *Draw*.

Ukázka načteného souboru dat.

Obrázek 15

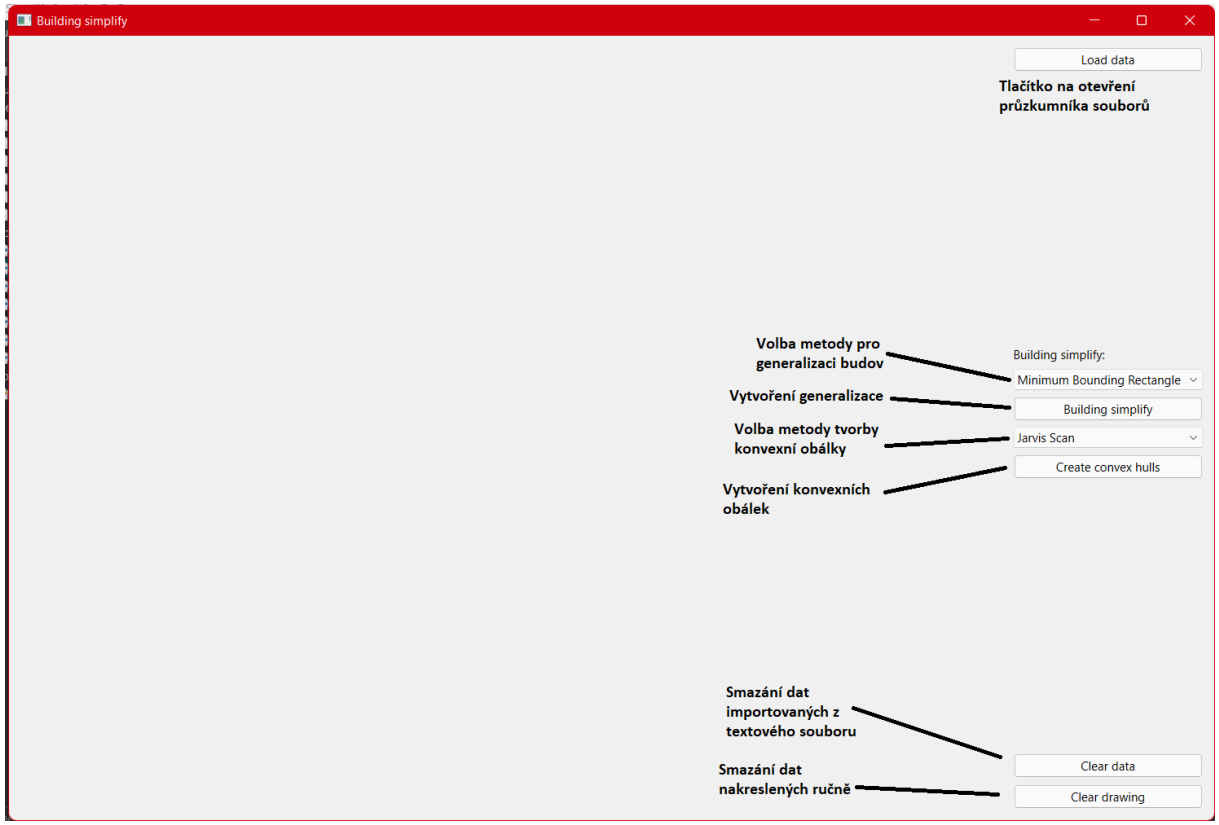
7. Výstupní data

Veškeré výstupy jsou vykreslovány do okna *Canvas*.

8. Vzhled aplikace

Po spuštění aplikace se otevře okno *Building simplify*. Úvodní plocha obsahuje 5 tlačítek třídy *QPushButton*, a to: *pushButton_load*, *pushButton*, *pushButton_createHulls*, *pushButton_clear_data* a *pushButton_2*. Tlačítka jsou uvedena postupně odshora dolů, podle obrázku.

Dále jsou na úvodní obrazovce dvě rolovací menu třídy *QComboBox*, a to: *comboBox* a *comboBox_HullsMethods*.



Obrázek 16

9. Dokumentace

1. Třída *Algorithms*

a) *double get2LinesAngle(QPoint &p1, QPoint &p2, QPoint &p3, QPoint &p4)*
- spočte úhel mezi dvěma vektory

b) *QPolygon cHull (std::vector <QPoint> &points)*
- Vytvoří konvexní obálku pomocí metody *Jarvis Scan*

c) *std::vector <QPoint> rotate(std::vector <QPoint> &points, double sigma)*
- rotace vektoru bodů o úhel *sigma* kolem osy *z*

d) *std::tuple<std::vector<QPoint>, double> minMaxBox(std::vector <QPoint> &points)*
- nalezne nejmenší ohraničující obdélník množiny bodů se stranami rovnoběžnými s osami *x* a *y*

e) *QPolygon minAreaEnclosingRectangle(std::vector <QPoint> &points)*
- Vytvoří ohraničující obdélník množiny bodů s nejmenší plochou metodou *Minimum Area Enclosing Rectangle*

f) *QPolygon wallAverage(std::vector <QPoint> &points)*
- Vytvoří ohraničující obdélník množiny bodů s nejmenší plochou metodou *Wall Average*

g) *double LH(std::vector <QPoint> &points)*
- určí plochu polygonu pomocí L'Huillierových vzorců

h) *std::vector <QPoint> resizeRectangle(std::vector <QPoint> &points, std::vector <QPoint> &er)*
- změní velikost vstupního obdélníku

i) *QPolygon weightedBisector(std::vector <QPoint> &points)*
- Vytvoří ohraničující obdélník množiny bodů s nejmenší plochou metodou *Weighted Bisector*

j) *QPolygon longestEdge(std::vector <QPoint> &points)*
- Vytvoří ohraničující obdélník množiny bodů s nejmenší plochou metodou *Longest Edge*

k) *int getPointLinePosition(QPoint &a,QPoint &p1,QPoint &p2)*
- analyzuje vzájemný vztah bodu a přímky

l) *QPolygon Algorithms::cHullGraham(std::vector <QPoint> &points)*
- vrací konvexní obálku určenou metodou *Graham Scan*

m) *double distance(QPoint p1, QPoint p2)*
- *vrací vzdálenost dvou bodů*

2. Třída *CSV*

a) *vector<QPolygon> read_csv(string &filename)*
- čte vstupní *csv* soubor a ukládá polygon do struktury *vector<QPolygon>*

3. Třída *Draw*

a) *void paintEvent(QPaintEvent *event)*
- vykreslí polygony na *Canvas*

b) *void mousePressEvent(QMouseEvent *event)*
- vrátí souřadnice kurzoru po kliknutí na *Canvas*

c) *void clearDrawing()*
- vyčistí *Canvas*

d) *void drawPolygons(vector<QPolygon> &pols)*
- vykreslí polygony na *Canvas*

e) *void clearData()*
- vyčistí *Canvas*

4. Třída *Widget*
    a) *void on_pushButton_2_clicked()*
        - vyčistí *Canvas*
    b) *void on_pushButton_clear_data_clicked()*
        - vyčistí *Canvas*
    c) *void on_processPoints()*
        - vytvoří ohraničující obdélník
    d) *void on_pushButton_load_clicked()*
        - načte data ze souboru
    e) *void createHulls(std::vector <QPoint> &points)*
        - vytvoří konvexní obálku
    f) *void on_pushButton_createHulls_clicked()*
        - vytvoří konvexní obálky

10. Závěr

Ošetřování záležitostí Simple features a souřadnicových systémů by zabralo moc času a není to hlavní úlohou této práce, proto na tyto úkony nebyla věnována dostatečná pozornost. Je tedy doporučeno nahrávat data v pixelových souřadnicích v rozmezí *x* a *y* cca [0,900] v závislosti na velikosti okna. Jinak se objekty nezobrazí.

Výsledkem je aplikace *Building Simplify* s grafickým uživatelským rozhraním, která je uvolněna pod licencí GNU GPL. Aplikace je psána v jazyce C++. Pro grafické uživatelské rozhraní byl použit framework QT.

Aplikace umožňuje nahrát polygony z textového souboru, případně si naklikat vlastní polygon. Aplikace má implementovány dvě metody pro nalezení konvexní obálky: *Graham* a *Javis Scan*.

Pro nalezení generalizace budovy byly implementovány 4 metody:

1. Minimum Area Enclosing Rectangle
2. Wall Average
3. Longest Edge
4. Weighted Bisector

11. Zdroje

[1] Konvexní obálka množiny bodů [online]. [cit. 2021-11-05]. Dostupné z: https://web.natur.cuni.cz/~bayertom/images/courses/Adk/adk4.pdf