České vysoké učení technické v Praze
Fakulta stavební
Katedra geomatiky

Technická zpráva

Algoritmy v digitální kartografii

Úloha č. 2: Generalizace budov

Bc. Pane Kuzmanov
Bc. František Mužík

Studijní program: Geodézie a kartografie
Specializace: Geomatika

Praha 2021

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.

1 Údaje o bonusových úlohách

1.1 Generalizace budov metodou Longest Edge (+5b)

Bylo implementováno určení hlavního směru metodou Longest Edge. Matematický popis algoritmu je sepsán v kapitole 3.3. Grafické znázornění fuknčního implementace algoritmu je na obr. 5.

1.2 Generalizace budov metodou Weighted Bisector (+8b)

Byl implementován algoritmus Weighted Bisector, jehož popis včetně vzorců, je uveden v kapitole 3.4. Grafické znázornění funkčnosti algoritmu je vyobrazeno na obr. 6.

1.3 Implementace další metody konstrukce konvexní obálky (+5b)

Byla implementována metoda vytvoření konvexní obálky s názvem Quick Hull. Výsledek jejího použití je na obr. 5 a obr. 6. Matematický popis řešení konstrukce konvexní obálky je součástí kapitoly 3.6.

1.4 Ošetření singulárního případu při generování konvexní obálky (+2b)

Tato bonusová úloha nebyla řešena.

2 Popis a rozbor problému

Nechť existuje libovolný polygon (tedy mnohoúhelník), který může v praktickém případě představovat hrany budovy na mapě. Cílem této úlohy je navrhnutí několika generalizačních algoritmů, které vypočítají hlavní směr budovy a vykreslí obdélník, který však zachovává rozlohu původní budovy a platí pro něj, že jeho střed je shodný s těžištěm budovy. Následně je vypočtena a vykreslena konvexní obálka budovy, která je určena jednou ze dvou užitých metod. Popis všech algoritmů následuje v kapitole 3.

3 Popisy algoritmů formálním jazykem

3.1 Minimum Area Enclosing Rectangle

První hlavní směr polygonu množiny S je zde představován delší ze stran minmax boxu (MMB).
Zadáno: Lomové body polygonu
Určováno: Hlavní směr polygonu
Implementace algoritmu:

1. Hledání konvexní obálky $H = CH(S)$,
2. inicializace obdélníku s minimální plochou $R = MMB(S)$ a jeho plochy A_{min},

3. opakování pro každou hranu e obálky H,
4. výpočet směrnice σ hrany e,
5. otočení S o úhel $-\sigma : S_r = R(-\sigma)S$,
6. nalezení $MMB(S_r)$ a určení $A(MMB(S_r))$,
7. pokud $A < A_{min}$,
8. pak $A_{min} = A, MMB_{min} = MMB, \sigma_{min} = \sigma$,
9. $R = R(\sigma)MMB$.

3.2 Wall Average

Na každou hranu budovy je aplikována operace $mod(\frac{pi}{2})$. Ze "zbytků"po dělení je vypočten vážený průměr, přičemž váhou je délka strany.
Zadáno: Lomové body polygonu
Určováno: Hlavní směr polygonu
Implementace algoritmu:

1. Určení směrnice σ_0 pro první stranu polygonu,
2. výpočet souřadnicových rozdílů $dx_i = X_{i+1} - X_i, dy_i = Y_{i+1} - Y_i$ pro sousední strany načteného polygonu,
3. určení směrnice σ_i všech stran polygonu: $\sigma_i = atan2\frac{dy_i}{dx_i}$,
4. určení délek všech hran $l_i = \sqrt{dx^2 + dy^2}$,
5. aplikace operace $mod(\frac{pi}{2})$, pro každou hranu budovy jsou redukovány směrnice: $\triangle\sigma_i = \sigma_i - \sigma^{'}$,
6. výpočet zaokrouhleného podílu: $k_i = \lfloor\frac{2\triangle\sigma_i}{pi}\rfloor$,
7. výpočet zbytku, odchylky od $0 \pm k\pi : r_i = \triangle\sigma_i - k\frac{pi}{2}$,
8. směr natočení budovy dán váženým průměrem $\sigma = \sigma^{'} + \sum_{i=1}^{n} \frac{r_i s_i}{s_i}$.

3.3 Longest Edge

První hlavní směr budovy je představován nejdelší stranou v budově, druhý hlavní směr je na něj kolmý.
Zadáno: Lomové body polygonu
Určováno: Hlavní směr polygonu
Implementace algoritmu:

1. Výpočet souřadnicových rozdílů $dx_i = X_{i+1} - X_i, dy_i = Y_{i+1} - Y_i$ pro sousední strany načteného polygonu,
2. určení délek všech hran $l_i = \sqrt{dx^2 + dy^2}$,
3. nalezení nejdelší hrany, pokud $l_i > l_{max}$,
4. pak $l_{max} = l_i, dx_{max} = dxi, dy_{max} = dyi$,
5. směr natočení budovy: $\sigma_i = atan2\frac{dy_{max}}{dx_{max}}$.

3.4 Weighted Bisector

Hledány dvě nejdelší úhlopříčky, jejich směrnice σ_1, σ_2 a jejich délky s_1, s_2. Hlavní směr σ je dán váženým průměrem.
Zadáno: Lomové body polygonu
Určováno: Hlavní směr polygonu
Implementace algoritmu:

1. Určení souřadnicových rozdílů pro každou úhlopříčku: $dx_{ij} = X_j - X_i, dy_{ij} = Y_j - Y_i$,
2. opakovaný výpočet délek všech úhlopříček: $s_i = \sqrt{dx^2 + dy^2}$,
3. nalezení dvou nejdelších úhlopříček pokud $s_i > s_1$,
4. pak $s_2 = s_1, s_1 = s_i$ (obdobně se uloží souřadnicové rozdíly),
5. výpočet směrnice úhlopříček s_1, s_2: $\sigma_1 = atan2\frac{dy_1}{dx_1}$ a $\sigma_2 = atan2\frac{dy_2}{dx_2}$,
6. výsledné určení hlavního směru váženým průměrem: $\sigma = \frac{s_1\sigma_1 + s_2\sigma_2}{s_1 + s_2}$.

3.5 Jarvis Scan

Za předpokladu, že v načtené množině bodů nejsou tři kolineární body, vytvoří konvexní obálku.
Zadáno: Lomové body polygonu
Určováno: Konvexní obálka
Implementace algoritmu:

1. Nalezení pivota q, q= $min(y_i)$,
2. přidáni q do polygonu konvexní obálky CH,
3. inicializace $p_{j-1} \in X, p_j = q, p_{j+1} = p_{i-1}$,
4. opakování, dokud $p_{j+1} \neq q$,
5. nalezení $p_{j+1} = arg\ max_{\forall p_i \in P} \angle(p_{j-1}, p_j, p_i)$,
6. přidání p_{j+1} do konvexní obálky CH,
7. $p_{j-1} = p_j, p_j = p_{j+1}$.

3.6 Quick Hull

Vytvoření konvexní obálky na základě setřídění bodů dle jejich x – ové souřadnice a následného rozdělení množiny bodů S na horní S_U a dolní část S_L vzhledem k bodům s nejmenší a největší x – ovou souřadnicí q_1, q_3. Konvexní obálka CH je dále konstruována ze 2 částí. Součástí algoritmu je rekurzivní volání lokální procedury Quick Hull.
Zadáno: Lomové body polygonu
Určováno: Konvexní obálka
Implementace algoritmu:
Globální procedura:

1. Nalezení bodů s minimální a maximální x – ovou souřadnicí $q_1 = min(x_i), q_3 = max(x_i)$,
2. přidání bodů q_1, q_3 do $S_U a S_L$,
3. pro $\forall p_i \in S$,
4. pokud $(p_i \in \sigma_l(q_1, q_3))$,
5. pak p_i je přidán do S_U.
6. jinak p_i je přidán do S_L,
7. bod q_3 je přidán do CH,
8. výpočet horního segmentu $QuickHullLokalni(1, 0, S_U, CH)$,
9. bod q_1 je přidán do CH,
10. výpočet dolního segmentu $QuickHullLokalni(0, 1, S_L, CH)$.

Lokální procedura:

1. Nalezení bodu $p' = arg\ max_{\forall p_i \in S} \|p_i - (p_s, p_e)\|, p' \in \sigma_r(p_s, p_e)$,
2. pokud existuje bod vpravo od hrany $p' \neq \emptyset$,
3. pak je vypočten horní segment $QuickHullLokalni(s, i', S, CH)$,
4. přidání bodu p' do CH,
5. výpočet dolního segmentu $QuickHullLokalni(i', e, S, CH)$.

4 Problematické situace a jejich rozbor

Aplikace má problém při řešení algoritmu Minimum Area Enclosing Rectangle, který spočívá v dělení nulou pro případ, jestliže jsou na vstupu dva body se stejnou x – ovou souřadnicí. Konkrétně se jedná o výpočet úhlu sigma – viz níže (ukázka části kódu):

```
QPolygon Algorithms::minAreaEnclosingRectangle(std::vector<QPoint> &points)
{
...
        //Coordinate differences
        double dx = ch[(i+1)%n].x() - ch[i].x();
        double dy = ch[(i+1)%n].y() - ch[i].y();

        double sigma = atan2(dy, dx);
...
}
```

5 Vstupní data

Vstupními daty jsou souřadnice jednotlivých polygonů, které se načítají do aplikace z textového souboru. Jedná se o část budov vyexportovanou z databáze ZABAGED [1] s využitím softwaru QGIS [2]. Polygony jsou v souřadnicovém systém JTSK.

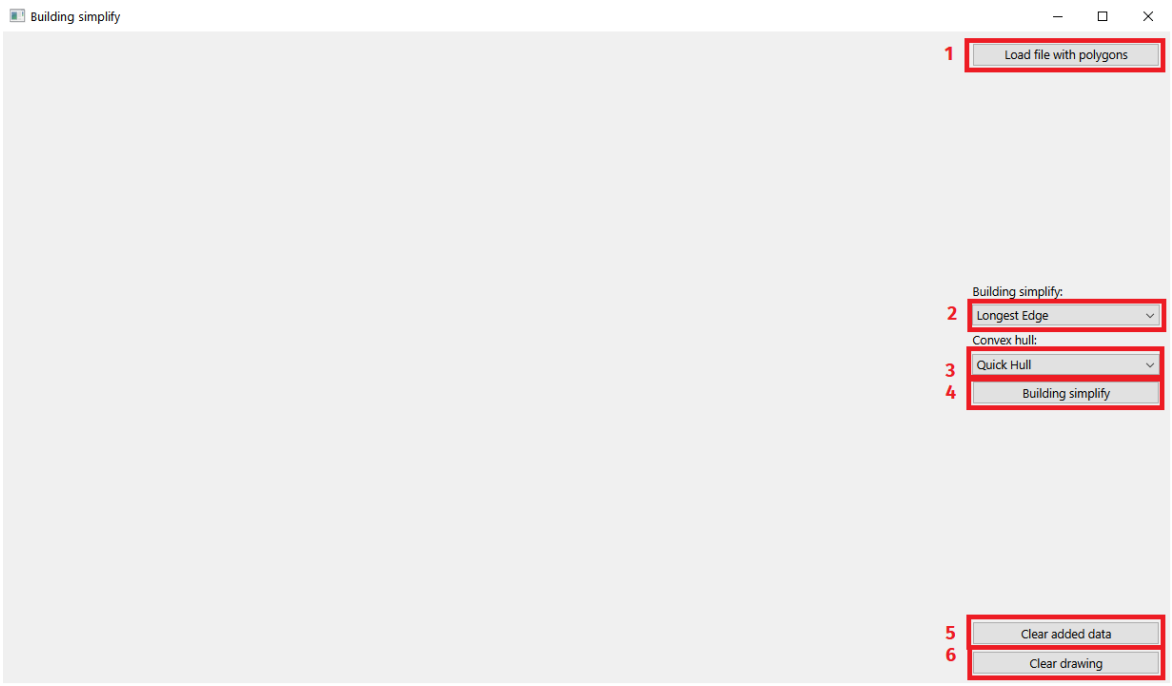
6 Výstupní data

Za výstup je považována grafický výstup vytvořené aplikace. Jeho součástí je implementace všech algoritmů zmíněných v této technické zprávě.

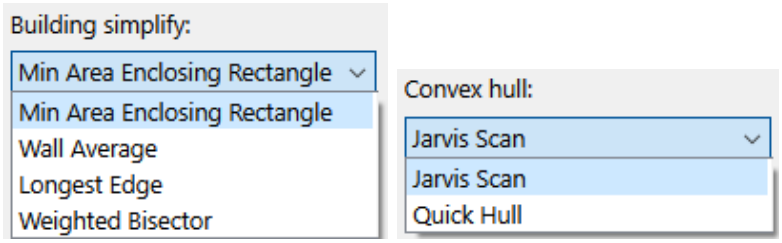
7 Snímky obrazovky vytvořené aplikace a její popis

Po spuštění aplikace se otevře prázdné okno, do kterého je potřeba nahrát polygony z textového souboru a vybrat výpočetní algoritmy (obr. 2). Ovládání aplikace je popsáno níže. S textem souvisí obr. 1.

1. **Load file with polygons**: Načtení polygonů z textového souboru.
2. **Building simplify – výběr algoritmu**: Rozevírací nabídka pro volbu algoritmu určující hlavní směr budovy.
3. **Convex Hull – výběr algoritmu**: Rozevírací nabídka pro volbu metody konstrukce konvexní obálky.
4. **Building simplify**: Vykreslení generalizujícího obdélníka vypočteného s užitím zvoleného algoritmu pro určení hlavního směru budovy a vykreslení konvexní obálky.
5. **Clear added data**: Vymazání přidaných dat z obrazovky.
6. **Clear drawing**: Vymazání ručně nakreslených polygonů.

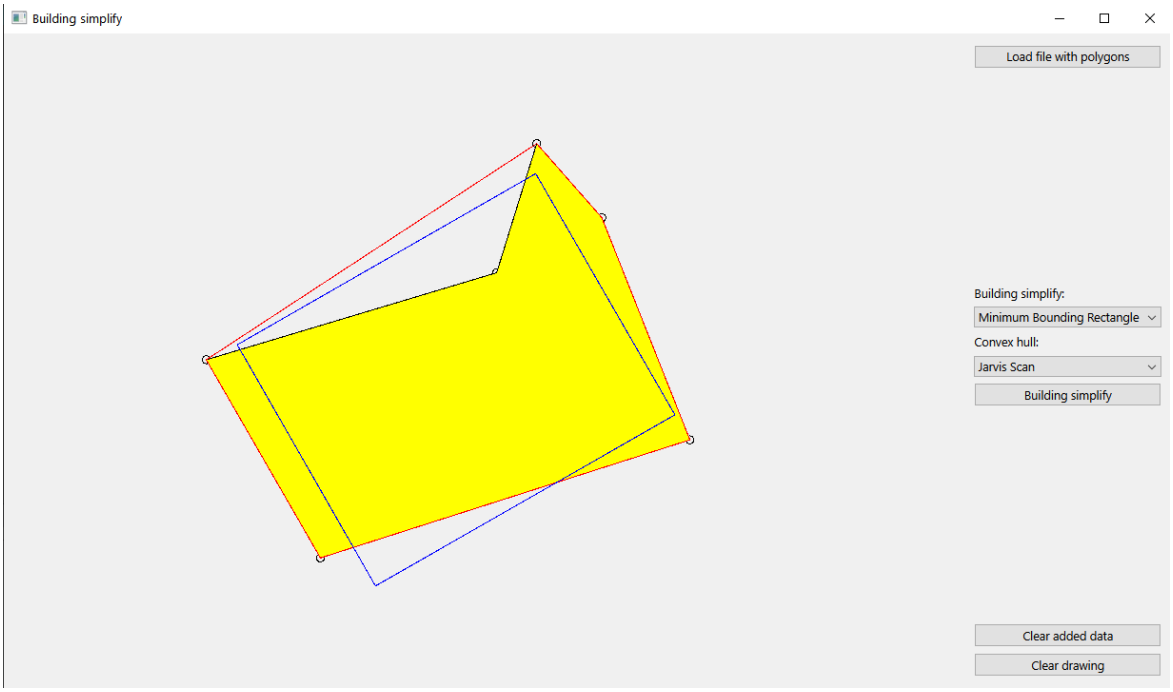


Obrázek 1: Popis uživatelského rozhraní aplikace.



Obrázek 2: Ukázka rozevíracích nabídek s algoritmy.

Ukázka generalizace polygonů všemi použitými algoritmy (modře) s konvexními obálkami (červeně):



Obrázek 3: Určení hlavního směru budov s využitím algoritmu Minimum Area Enclosing Rectangle.

Obrázek 4: Určení hlavního směru budov s využitím algoritmu Wall Average.

Obrázek 5: Určení hlavního směru budov s využitím algoritmu Longest Edge.

Obrázek 6: Určení hlavního směru budov s využitím algoritmu Weighted Bisector.

8 Dokumentace

8.1 Třída Algorithms

Tato třída obsahuje výpočetní vzorce pro použité algoritmy.
Třída algorithms obsahuje následující veřejné metody:

`double get2LinesAngle(QPoint &p1, QPoint &p2, QPoint &p3, QPoint &p4)`

Metoda vypočte velikost úhlu, který svírají dvě přímky. První přímku tvoří body p_1, p_2 a druhou přímku body p_3, p_4.

`int getPointLinePosition(QPoint &a,QPoint &p1,QPoint &p2)`

Metoda určuje, zda – li bod leží v levé či v pravé polorovině od přímky (hrany polygonu). Vstupními parametry jsou určovaný bod a a body p_1, p_2, které tvoří přímku.

`double getPointLineDistance(QPoint &a, QPoint &p1, QPoint &p2)`

Metoda určuje vzdálenost bodu a od přímky tvořenou body p_1, p_2.

`QPolygon cHullJarvisScan(std::vector <QPoint> &points)`

Výpočet konvexní obálky načtených bodů na základě algoritmu Jarvis Scan (kapitola 3.5).

`std::vector<QPoint> rotate(std::vector<QPoint> &points, double sigma)`

Otočení načtených bodů o zadaný úhel *sigma*.

`std::tuple<std::vector<QPoint>, double> minMaxBox(std::vector<QPoint> &points)`

Proměnná vracející min – max box, tedy vrcholy obdélníka a jeho výměru. Vstupem jsou načtené body.

`QPolygon minAreaEnclosingRectangle(std::vector<QPoint> &points)`

Určení hlavního směru polygonu s využitím algoritmu Minimum Area Eclosing Rectangle (kapitola 3.1). Na vstupu jsou body polygonu. Výstupem je generalizovaný a pootočený polygon.

`QPolygon wallAverage(std::vector<QPoint> &points)`

Určení hlavního směru polygonu s využitím algoritmu Wall Average (kapitola 3.2). Na vstupu jsou body polygonu. Výstupem je generalizovaný a pootočený polygon.

`double LH(std::vector<QPoint> &points)`

Výpočet plochy obecného mnohoúhelníků pomocí LH vzorce.

`std::vector<QPoint> resizeRectangle(std::vector<QPoint> &points, std::vector<QPoint`

Přepočet velikosti ohraničujícího obdélníku na základě výměry původního polygonu.

```
QPolygon longestEdge(std::vector<QPoint> &points)
```

Určení hlavního směru polygonu s využitím algoritmu Longest Edge (kapitola 3.3). Na vstupu jsou body polygonu. Výstupem je generalizovaný a pootočený polygon.

```
QPolygon weightedBisector(std::vector<QPoint> &points)
```

Určení hlavního směru polygonu s využitím algoritmu Weighted Bisector (kapitola 3.4). Na vstupu jsou body polygonu. Výstupem je generalizovaný a pootočený polygon.

```
QPolygon cHullQuickHull(std::vector <QPoint> &points)
```

Výpočet konvexní obálky načtených bodů na základě algoritmu Quick Hull (kapitola 3.6).

```
void quickHullLocal(int ps, int pe, std::vector<QPoint> &points, QPolygon &ch)
```

Metoda pro pomocný výpočet rekurentního určení konvexní obálky bodů algoritmem Quick Hull, lokální procedura (kapitola 3.6). Vstupními parametry jsou indexy počátečního a koncového bodu *ps* a *pe* vstupní hrany, načtené body a konvexní obálka.

8.2 Třída Draw

Tato třída umožňuje vykreslování bodu a polygonů.

Třída draw obsahuje následující privátní metody a proměnné:

```
std::vector<QPoint> points
```

Proměnná se souřadnicemi načtených bodů.

```
QPolygon ch, er;
```

Polygony pro ukládání konvexní obálky a ohraničujícího obdélníku.

```
std::vector<QPolygon> polygons, chs, ers
```

Proměnná uchovávající pomocné polygony v průběhu výpočtu a pro vykreslení.

Třída draw obsahuje následující veřejné metody a proměnné:

```
explicit Draw(QWidget *parent = nullptr)
```

Prvotní vykreslení bodu mimo okno aplikace.

`void paintEvent(QPaintEvent *event)`

Metoda, která vykresluje bod či polygony.

`void mousePressEvent(QMouseEvent *event)`

Metoda určující souřadnice určeného bodu.

`void clearAddedData()`

Metoda mazající přidaná data z obrazovky.

`void clearDrawing()`

Metoda mazající nakreslený polygon z obrazovky.

`std::vector<QPoint> getPoints() {return points;}`

Vrací souřadnice lomových bodů polygonu.

`std::vector<QPolygon> getPolygons(){return polygons;}`

Vrací souřadnice polygonů.

`void setCh(QPolygon &ch_) {chs.push_back(ch_);}`

Nastavení polygonu konvexní obálky.

`void setEr(QPolygon &er_) {ers.push_back(er_);}`

Nastavení polygonu ohraničujícího obdélníku.

`void clearChs(){chs.clear();}`

Smazání polygonu konvexní obálky.

`void clearErs(){ers.clear();}`

Smazání polygonu ohraničujícího obdélníku.

`void drawPolygons(std::vector<QPolygon> &data);`

Vykreslení polygonů načtených z textového souboru.

8.3 Třída Load

Třída draw obsahuje následující veřejnou proměnnou:

`static std::vector<QPolygon> load_file(std::string &filename)`

Umožňuje načítání polygonů z textového souboru.

8.4 Třída SortByX

Třída draw obsahuje následující veřejnou proměnnou:

`bool operator() (QPoint &p1, QPoint &p2)`

Ražení bodů dle jejich x – ové souřadnice.

8.5 Třída SortByY

Třída draw obsahuje následující veřejnou proměnnou:

`bool operator() (QPoint &p1, QPoint &p2)`

Ražení bodů dle jejich y – ové souřadnice.

8.6 Třída Widget

Tato třída propojuje uživatelské rozhraní aplikace s kódem. Je vytvořena v sekci *Design*.

Třída widget obsahuje následující privátní metody a proměnné:

`void on_pushButtonClear_clicked()`

Vymazání kresby. Propojení s tlačítkem *Clear drawing*.

`void on_pushButtonLoad_clicked()`

Načtení lomových bodů polygonů z textového souboru. Propojení s tlačítkem *Load file with polygon*.

`void on_pushButton_clicked()`

Spuštění funkce Building Simplify s vybraným algoritmem. Propojení s tlačítkem *Building Simplify*.

`void on_pushButtonClearData_clicked()`

Vymazání přidaných dat. Propojení s tlačítkem *Clear added data*.

`void processPoints(std::vector<QPoint> &points)`

Volba použitého algoritmu na základě výběru z combo boxů.

9 Závěr

Byla vytvořena desktopová aplikace, která umožňuje načtení vrcholů polygonu z textového souboru na disku. Funkcí této aplikace je vytvoření ohraničujícího polygonu, který generalizuje polygony (budovy) s využitím jednoho ze čtyř algoritmů popsaných v kapitole 3. Jedná se o algoritmy Minimum Area Enclosing Rectangle, Wall Average, Longest Edge a Weighted Bisector. Ohraničující obdélník má sklon hlavního směru určený vybraným algoritmem, výměru totožnou s původním polygonem a střed shodný s těžištěm polygonu. Aplikace zároveň vykresluje konvexní obálku jedním ze dvou vybraných výpočtů popsaných taktéž v kapitole 3. Těmito zvolenými výpočty jsou Jarvis Scan a Quick Hull.

9.1 Odhad efektivity metod, vzájemné porovnání

Nejefektivnější metodou je, z pohledu naší skupiny, Longest Edge. Jedná se o poměrně jednoduchý algoritmus, který sice nedosahuje nejpřesnějších výsledků, ale může sloužit jako jednoduchý generalizační nástroj například pro prvotní náhled nebo v případech, kdy není vyžadována přesnější metoda.

Minimum Area Enclosing Rectangle se jeví jako nejproblematičtější z několika důvodů. Prvním je nutnost tvorby konvexní obálky, což není v této úloze problém, avšak pro samostatné řešení se jedná o překážku. Algoritmus vrací špatné výsledky pro budovy o tvaru L či Z.

Nejkomplexnější metodou je, z hlediska výpočtu, Wall Average. Tato metoda dává dobré výsledky, avšak je citlivá na polygony s ostrými úhly.

Weighted Bisector dává dobré výsledky, nicméně pro složitější polygony může vytvořit nežádoucí naklonění ohraničujícího obdélníku, z důvodu výpočtu hlavního úhlu sigma přes úhlopříčky, jejichž délka a vzájemný úhel můžou být nevyzpytatelné.

9.2 Možné či neřešené problémy

Nebyla řešena poslední bonusová úloha. Aplikace má problém při řešení algoritmu Minimum Area Enclosing Rctangle (viz 4).

9.3 Náměty na vylepšení

K vylepšení se nabízí případ dělení nulou z předchozí kapitoly 9.2. Dále by bylo možné vizuálně upravit grafické prostředí aplikace. Vykreslování konvexní obálky s užitím samostatného tlačítka bylo taktéž bráno v potaz při psaní aplikace, avšak finální implementované řešení bylo zvoleno jako uživatelsky nejjednodušší. Zároveň by bylo možné do každé rozevírací nabídky přidat možnost nevykreslení ohraničujícího obdélníku či konvexní obálky, pokud by chtěl uživatel vykreslit pouze jednu z možností.

V Praze 7.11.2021

Bc. Pane Kuzmanov
Bc. František Mužík

Použitá literatura

[1] Český úřad zeměměřický a katastrální. Základní báze geografických dat České republiky (ZABAGED). https://www.cuzk.cz/ [cit. 2020-11-05].

[2] QGIS. A Free and Open Source Geographic Information System. https://qgis.org/en/site/index.html [cit. 2020-11-05].