Úloha 2 - Generalizace budov

Algoritmy počítačové kartografie

Tomáš Hřebec, Kateřina Obrazová

Praha 2022

1 Zadání úlohy

1.1 Povinná část

Vstup: Množina budov $B = \{B_i\}_{i=1}^{n}$, *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 orientovaných 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.

1.2 Volitelná část

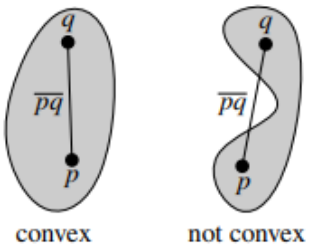
Generalizace budov metodou Longest Edge.

Implementace další metody konstrukce konvexní obálky.

Ošetření singulárního případu při generování konvexní obálky.

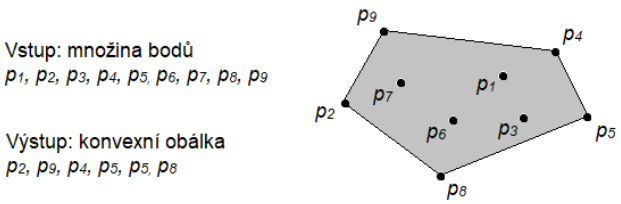
2 Konvexní obálka

Podmnožina S roviny se nazývá konvexní jen tehdy, když pro libovolnou dvojici bodů $p, q \in S$ je úsečka $\overline{pq}$ zcela obsažena v S. (Obr. 1) Konvexní obálka $\mathcal{CH}(S)$ množiny S je nejmenší konvexní množina, která obsahuje S. Přesněji řečeno, je to průnik všech konvexních množin, které obsahují S. (de Berg a kol. 2008)



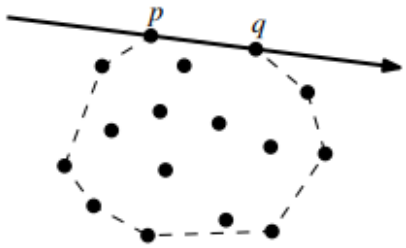
Obr. 1 - Konvexní a nekonvexní podmnožina S

Konvexní obálka P je konvexním mnohoúhelníkem. Způsob, jak znázornit mnohoúhelník, je vypsat jeho vrcholy ve směru hodinových ručiček s libovolným počátkem. Chceme tedy vyřešit problém: dáme-li množinu $P = \{p_1, p_2..., p_n\}$ bodů v rovině, vypočtěte seznam, který obsahuje ty body z P, které jsou vrcholy $\mathcal{CH}(P)$, uvedené ve směru hodinových ručiček.(de Berg a kol. 2008)



Obr. 2 - Výpočet konvexní obálky

Při návrhu algoritmu pro výpočet konvexní obálky, se hovoří o průniku všech konvexních množin obsahujících P, kterých je nekonečně mnoho. Užitečnější je pozorování, že $\mathcal{CH}(P)$ je konvexní mnohoúhelník. Dále chceme zjistit, jaké jsou okraje $\mathcal{CH}(P)$. Oba koncové body p a q takové hrany jsou body P, a pokud nasměrujeme přímku skrz p a q tak, že $\mathcal{CH}(P)$ leží vpravo, pak všechny body P musí ležet vpravo od této přímky. (Obr. 3) Platí to i obráceně: pokud všechny body $P \setminus \{p, q\}$ leží napravo od nasměrované přímky skrz p a q, pak $\overline{pq}$ je okraj $\mathcal{CH}(P)$. (de Berg a kol. 2008)



Obr. 3 - Určení hran množiny bodů

Definice 1: *„Konvexní obálka množiny bodů S je množina všech konvexních kombinací bodů bodu S.“*

Definice 2: *„Konvexní obálka množiny bodů S je průsečíkem všech konvexních množin, které obsahují S.“*

Definice 3: *„Konvexní obálka konečné množiny S v rovině je nejmenším konvexním mnohoúhelníkem P, který obklopuje S. Nejmenší v tom smyslu, že neexistuje žádný další mnohoúhelník P' takový, že $P \supset P' \supseteq S$.“*

Definice 4: *„Konvexní obálka konečné množiny bodů S v rovině je uzavřený konvexní mnohoúhelník P s nejmenší plochou a nejmenším obvodem.“*

Definice 5: *„Konvexní obálka množiny bodů S v rovině je sjednocením všech trojúhelníků určených body v S.“(O'Rourke 1998, str. 65)*

2.1 Metody konstrukce konvexní obálky

Pro konstrukci konvexní obálky se nejčastěji používají:

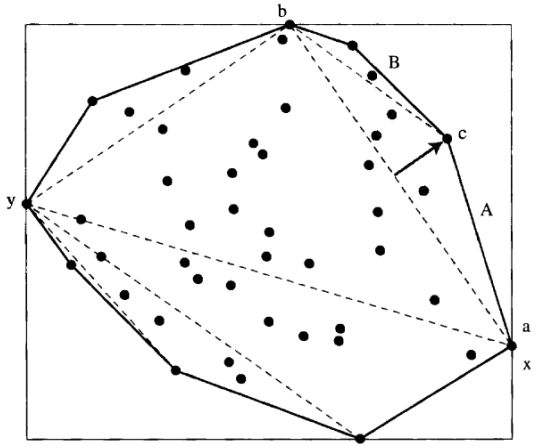
- Inkrementální algoritmy,
- Jarvis Scan (Gift Wrapping Algorithm),
- Graham Scan,
- Quick Hull,
- Sweep line (zametací přímka) a plane-sweep,
- Divide and Conquer.

Některé z těchto algoritmů lze použít pouze pro 2D (Graham Scan), další však i pro více dimenzí (Quick Hull). Algoritmy pro vytvoření 2D konvexní obálky mají vždy jako výstup uspořádaný seznam vrcholů tvořících hranici konvexní obálky seřazený ve stejném pořadí jako dříve. V naší úloze budou použity algoritmy Jarvis Scan a Graham Scan, které budou podrobněji rozebrány v kapitole 2.2 a 2.3.

U inkrementálních algoritmů vytvoříme konečný výsledek tak, že postupně vkládáme každý objekt vstupu. Při každém vložení tohoto objektu vypočítáme mezivýsledek. Tyto algoritmy nejsou ve 2D optimální, ovšem v praxi mohou být rychlejší než optimální algoritmy. Ve 3D jsou tyto algoritmy optimální. (Bærentzen a kol. 2012)

Základní myšlenkou algoritmu Quick Hull je: *„Pro „většinu“ sad bodů je snadné odhodit mnoho bodů jako definitivně uvnitř obálky, a pak se soustředit na ty, které jsou nejblíže hranici obálky.“ (O'Rourke 1998, str. 70)*

Postup algoritmu Quick Hull je nalézt dva odlišné extrémní body, kdy použijeme nejvíce vpravo nejnižší body a nejvíce vlevo nejvyšší body x a y, které jsou na první pohled zřetelné. Celá obálka se skládá z horní obálky (upper hull) nad xy a z dolní obálky (lower hull) pod xy. (Obr. 4)



Obr. 4 - Quick Hull algoritmus

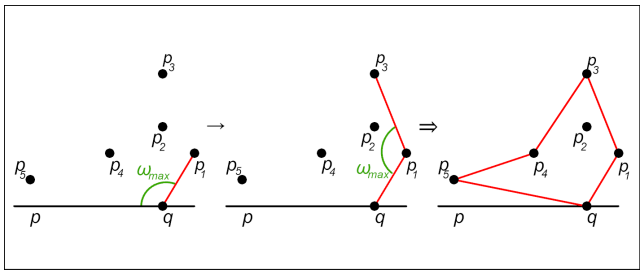
Obecně funguje Sweep line a plane-sweep tak, že se rozloží vstup na svislé pruhy tak, aby se informace potřebné k vyřešení problému nacházely ve svislých čarách vymezující tyto svislé pruhy. Pro tvorbu konvexní obálky ve 2D existuje algoritmus Sweep line, neboli zametací přímka. (Bærentzen a kol. 2012)

Algoritmus Divide and Conquer je optimální v jakékoli dimenzi. *„Při rekurzivním přístupu se rozdělí rekurzivně vstup, dokud nebude možné problém vyřešit triviálně pro danou velikost vstupu. Poté se sloučí řešení dílčích problémů.“ (Bærentzen a kol. 2012, str. 230)*

2.2 Jarvis Scan

Dalším algoritmem pro tvorbu konvexní obálky je Jarvis Scan, jenž je nazýván také jako Gift Wrapping algorithm či Jarvis March. Důvodem, proč se tento algoritmus nazývá Gift Wrapping, je následující: *„Lze na to pohlížet jako na omotávání sady bodů provázkem, který ohýbá minimální úhel od předchozí hrany obálky, dokud není sada zasažena.“ (O'Rourke 1998, str. 69)* Algoritmus byl na počátku navržen pro hledání obálek v libovolných rozměrech. Po nějaké době se však přišlo na to, že jej lze implementovat i ve 2D. Předpokladem je, že žádné tři body v S nejsou kolineární. Algoritmus je citlivý na velikost vstupních dat, tedy čím menší vstupní data, tím rychleji algoritmus pracuje. Jeho složitost je $O(nh)$, pokud má obálka h hran. (O'Rourke 1998)

Na Obrázku 5 vidíme průběh tvorby konvexní obálky P pomocí Jarvis Scan algoritmu. Začíná se počátečním bodem (pivotním bodem) $q \in P$, který má minimální souřadnici na ose y. Dále vedeme přímku p, která je rovnoběžná s osou x a která prochází bodem q ve směru vlevo. Z bodu q hledáme bod svírající maximální úhel ω_{max} (ve směru hodinových ručiček od p). Tomu odpovídá bod p_1, tudíž $p_1 \in P$. Z bodu p_1 hledáme ω_{max} měřený od $\overline{qp_1}$. Tomu odpovídá bod p_3, tudíž $p_3 \in P$. Tento postup opakujeme, dokud se konvexní obálka neuzavře v počátečním bodě q.

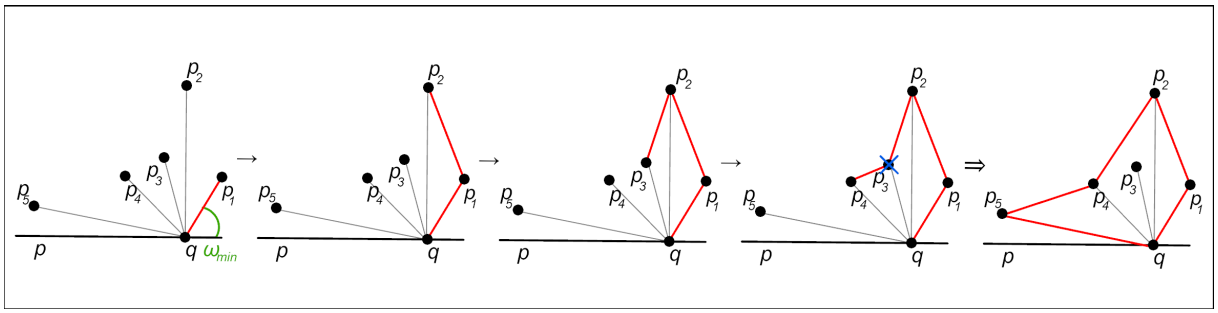


Obr. 5 - Jarvis Scan algoritmus

2.3 Graham Scan

Graham Scan je algoritmus sloužící k nalezení obálky bodů ve dvou rozměrech v čase $O(n.log(n))$. Již na počátku algoritmus vyžaduje znát všechny body na vstupu. Začíná se počátečním bodem (pivotním bodem) $q \in P$, který je konstrukcí na hranici konvexní obálky P, tedy minimální souřadnicí na ose y. (Obr. 6) Dále se seřadí body podle úhlu vzestupně (při stejném úhlu dvou bodů se bod s menší vzdálenosti od bodu q zanedbává). Z bodu q vedeme přímku do bodu p_1, se kterým svírá nejmenší úhel ω_{min} (proti směru hodinových ručiček). Bod $p_1 \in P$. Nyní spojíme bod p_1 s bodem p_2. Spojení (q, p_1, p_2) je levotočivé, bod

$p_2 \in P$. Bod p_2 spojíme s bodem p_3. Spojení (p_1, p_2, p_3) je levotočivé, bod $p_3 \in P$. Dále spojíme bod p_3 s bodem p_4. Spojení (p_2, p_3, p_4) je pravotočivé, tudíž bod $p_3 \notin P$ a z konvexní obálky je vymazán. Vracíme se opět k bodu p_2, který spojíme s bodem p_4. Spojení (p_1, p_2, p_4) je levotočivé, bod $p_4 \in P$. Takto se postupuje dokud se konvexní obálka neuzavře. V případě, kdy spojnice mezi body je kolineární, je bod automaticky přidám do konvexní obálky.



Obr. 6 - Graham Scan algoritmus

3 Algoritmy pro generalizaci budov

V této úloze byly použity pro generalizaci budov algoritmy Minimum Area Enclosing Rectangle, Wall Average a Longest Edge. Všechny tyto algoritmy jsou založeny na hledání hlavního směru budovy (každý jiným způsobem). V momentě, kdy je hlavní směr budovy nalezen, je polygon nahrazen Minimum Bounding boxem (dále jen min-max boxem), který má stejný obsah jako daný polygon a který je natočen do hlavního směru budovy.

4 Algoritmus Minimum Area Enclosing Rectangle

Po vytvoření konvexní obálky P pomocí Jarvis Scan nebo Graham Scan je nutné vytvořit min-max box. Min-max box můžeme popsat jako obdélník $\mathcal{R}$, který obklopuje celou budovu a jehož plocha je minimální. Aby min-max box splňoval tyto podmínky, je potřeba ho opakovaně rotovat okolo množiny bodů P o zápornou směrnici $-\sigma$ určité hrany P.

Nejprve je třeba vypočítat směrnici hrany konvexní obálky P tvořené body $B = [x_B, y_B]$ a $C = [x_C], y_C$:

$$tg\ \sigma = \frac{x_C - x_B}{y_C - y_B}$$

Pomocí matice rotace rotujeme konvexní obálku P o $-\sigma$:

$$\begin{bmatrix} x \\ y \end{bmatrix} \begin{bmatrix} cos(-\sigma) & -sin(-\sigma) \\ sin(-\sigma) & cos(-\sigma) \end{bmatrix} = \begin{bmatrix} x_r \\ y_r \end{bmatrix}$$

Dále nalezneme extrémní body konvexní obálky P_1, P_2, P_3, P_4 s extrémními souřadnicemi $\overline{x}, \underline{x}, \overline{y}, \underline{y}$. Tím je možné sestrojit min-max box, jehož strany budou rovnoběžné s osami x a y, a vypočítáme jeho plochu. Tento postup opakujeme pro všechny hrany konvexní obálky. Z tohoto procesu získáme obdélník $\mathcal{R}$ s nejmenší plochou a směrnici hrany konvexní obálky σ_P, z které byl min-max box vytvořen.

Dále je třeba detekovat první hlavní směr budovy. Ten představuje u tohoto algoritmu delší ze stran obdélníku $\mathcal{R}$. Tvar budovy chceme nahradit obdélníkem $\mathcal{R}'$ tak, aby měl stejnou plochu jako generalizovaná budova. Obdélník by měl být úměrně zmenšen a mít společný střed S. Nyní vypočítáme poměr obdélníků $\mathcal{R}$, $\mathcal{R}'$ (značí se k) a plochy obdélníků A (vzniklý min-max box) a A' (zmenšený obdélník):

$$A = ab$$

$$A' = kA = \sqrt{k}a'\sqrt{k}b'$$

Dále vypočítáme uhlopříčky obdélníků u_i a u_i':

$$\left(\frac{\|u_i\|}{2}\right)^2 = \left(\frac{a}{2}\right)^2 + \left(\frac{b}{2}\right)^2$$

$$\left(\frac{\|u_i'\|}{2}\right)^2 = k\left[\left(\frac{a}{2}\right)^2 + \left(\frac{b}{2}\right)^2\right] = k\left(\frac{\|u_i\|}{2}\right)^2$$

Z předchozích vztahů získáme:

$$\|u_i'\| = \sqrt{k}\|u_i\|$$

Následuje výpočet směrových vektorů pro extrémní body min-max boxu $\vec{u_{P_1}}$, $\vec{u_{P_2}}$, $\vec{u_{P_3}}$, $\vec{u_{P_4}}$:

$$\vec{u_{P_1}} = P_1 - S \qquad \vec{u_{P_1}} = P_3 - S$$

$$\vec{u_{P_1}} = P_2 - S \qquad \vec{u_{P_1}} = P_4 - S$$

A následně dopočítáme nové vrcholy obdélníku P'_1, P'_2, P'_3, P'_4:

$$P'_1 = S + \sqrt{k}u\vec{}_{P_1} \qquad P'_3 = S + \sqrt{k}u\vec{}_{P_3}$$

$$P'_2 = S + \sqrt{k}u\vec{}_{P_2} \qquad P'_4 = S + \sqrt{k}u\vec{}_{P_4}$$

5 Algoritmus Wall Average

Základním principem algoritmu Longest Edge je uvažovat orientaci každé hrany o zbytek Euklidovského dělení, tedy o $\frac{\pi}{2}$ (vychází výsledek v intervalu $\left[0, \frac{\pi}{2}\right]$) a vypočítat průměr těchto směrů, kdy váhou je délka hrany. (Duchéne a kol. 2003) Na začátku je nutné vypočítat směrnici libovolně zvolené hrany σ' a směrnice pro každou hranu budovy σ_i. Následně budou hodnoty σ_i redukovány o hodnotu σ':

$$\Delta\sigma_i = \sigma_i - \sigma'$$

Z výsledných hodnot $\Delta\sigma_i$ vypočítáme zaokrouhlený podíl k_i:

$$k_i \ = \ \lfloor(\frac{2\Delta\sigma_i}{\pi})\rfloor$$

Následuje výpočet zbytku r_i:

$$r_i \ = \ \Delta\sigma_i - k_i\frac{\pi}{2},$$

kdy

$$r_i = \begin{cases} < \frac{\pi}{4}; & \text{odchylka od vodorovného směru } (0 \pm \frac{\pi}{2}) \\ > \frac{\pi}{4}; & \text{odchylka od svislého směru } (0 \pm \frac{\pi}{2}) \end{cases} \tag{1}$$

Směr natočení budovy σ je dán váženým průměrem r_i, kde váhou je délka hran w_i:

$$\sigma = \sigma' + \frac{\sum_{i=1}^{n} r_i w_i}{\sum_{i=1}^{n} w_i}$$

6 Algoritmus Longest Edge

Algoritmus Longest Edge *„měří orientaci nejdelšího okraje budovy po odstranění zarovnaných bodů budovy“. (Duchéne a kol. 2003, str. 2)* Můžeme prohlásit, že první hlavní směr budovy je směrnice nejdelší hrany

budovy a druhý hlavní směr je na ni kolmý. Postup je podobný jako v předchozích algoritmech, kdy je vypočítaná směrnice σ, konvexní obálka P se otočí o $-\sigma$, avšak v tento moment vytvoříme min-max box a vypočítáme jeho plochu A. Dále otočíme konvexní obálku P o úhel σ, vypočítáme plochu budovy A' a upravíme min.max box tak, aby $A = A'$.

7 Data

Vstupem jsou polygonové vrstvy budov ve formátu *.shp*. Polygonové vrstvy budov byly vybrány z databáze ZABAGED. Výstupem je grafické znázornění polygonů budov a jejich generalizace.

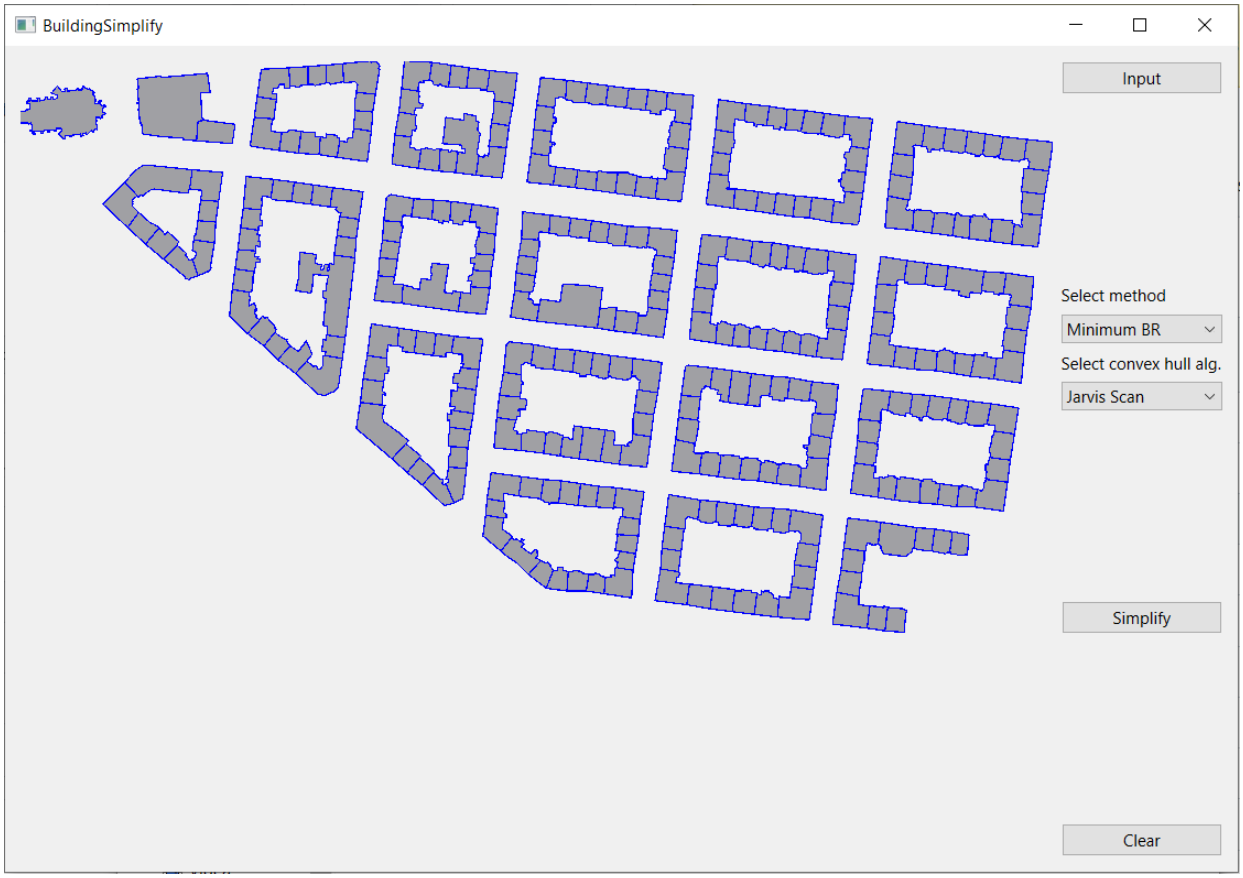
8 Dokumentace

Program je rozdělen do 4 modulů. Modul *Mainform.py* je převážně vygenerován pomocí softwaru QT Creator a slouží k vytvoření uživatelského rozhraní. Modul *algorithm.py* obsahuje algoritmy pro generalizaci budov. Modul *draw.py* slouží především k propojení předešlých dvou modulů a k zajištění vizualizace vstupu a výstupu. Modul *input.py* zajišťuje import vstupních dat v požadovaném formátu.

8.1 Modul Mainform.py

Tento modul obsahuje třídu `Ui_Mainform`, která je vytvořena automaticky pomocí QT Creatoru. (Obr. 7) Tato třída byla posléze doplněna o metody, které se spustí při interakci s uživatelským rozhraním. Jedná se o metody *input*, $simplifyClick$ a *clearClick*. *Input* vyvolá funkci $loadfile$ z modulu *input.py* a tím nahraje vstupní polygony ze souboru ve formátu *.shp* do proměnné *pol_list* ve třídě `Draw`. Metoda *simplifyClick* zajišťuje volání zvoleného algoritmu pro generalizaci budov. Toho je docíleno pomocí odečítání aktuálního itemu v příslušném comboboxu. Tento algoritmus je poté v cyklu spuštěn postupně na všechny vstupní polygony v proměnné *pol_list* ze třídy `Draw`. Jejich generalizace jsou uloženy do proměnné *er_list* ve třídě `Draw`. Poslední metoda *clear* vymaže list, který obsahuje jednotlivé vstupní polygony (*pol_list*) a list obsahující

jejich generalizace (*er_list*). Tímto odstraní veškeré zobrazované objekty.



Obr. 7 - Náhled uživatelského rozhraní

8.2 Modul input.py

Tento modul obsahuje metodu *loadfile*. V této metodě je nejprve načtena cesta ke vstupním datům. Pokud není zvolena žádná cesta, metoda se ukončí. Dále je načten vstupní soubor ve formátu *.shp*. Data z tohoto „shapefilu“ jsou převedeny do formátu *QPolygon*. Následně tato data musí být ještě zobrazena do lokálního souřadnicového systému, který je definován velikostí okna. Takto transformovaná data jsou poté jako jednotlivé polygony vloženy do listu *polygons*, který je posléze touto metodou vracen.

8.3 Modul draw.py

Tento modul obsahuje třídu `Draw`, kterou dědí od třídy `QWidget` z knihovny PyQt6. Hlavní funkce tohoto modulu je vykreslovat vstupní a generalizované polygony. Tyto polygony jsou uloženy v listech *pol_list*

(vstup) a *er_list* (generalizace). Toto vykreslení je definováno v metodě *paintEvent*. Poslední dvě metody, *gettery* a *settery*, jsou pro editaci dat z ostatních modulů.

8.4 Modul algorithm.py

Zde se nacházejí jednotlivé algoritmy pro generalizaci budov. Ve třídě `MinimumAreaEnclosingRectangle` je definován algoritmus Minimum Area Enclosing Rectangle. Tato třída se skládá z devíti metod. První metodou je *get2LinesAngle*, která má na vstupu 4 body ve formátu *QPoint*, které definují dvě úsečky. Metoda vrací velikost úhlu mezi nimi. Metoda *getPointAndLinePosition* má na vstupu 3 body *QPoint*, kde 2 definují přímku a 1 analyzovaný bod. Tato metoda zjistí, ve které polorovině se analyzovaný bod nachází. Výsledek je vracen jako 1 pro bod v levé polorovině, -1 pro bod v pravé polorovině a 0 pro kolineární bod. Další metoda se nazývá *createCHJarvis*, která vytvoří konvexní obálku polygonu pomocí algoritmu Jarvis Scan. Na vstupu je polygon *pol* ve formátu *QPolygon* a výstupem je také konvexní obálka *ch* ve formátu *QPolygon*. Metoda *createCHGraham* je stejná jako předchozí metoda s tím rozdílem, že k tvorbě konvexní obálky využívá algoritmus Graham Scan. Metoda *rotate* má na vstupu *QPolygon pol* a úhel *angel* ve formátu *float*. Metoda rotuje vrcholy polygonu o úhel *angel* a vrací jej jako proměnnou *pol_rot* ve formátu *QPolygon*. Metoda *minMaxBox* má na vstupu *QPolygon pol*, u kterého zjistí jeho min-max box, a následně vypočte obsah tohoto min-max boxu. Tyto hodnoty vrátí jako S a $minmax\_box$. Metoda *getArea* vypočte velikost obsahu vstupního QPolygonu. Metoda *resizeRectangle* změní velikost vstupního polygonu *er* tak, aby odpovídala velikosti vstupního polygonu *pol*.

Hlavní metoda této třídy se nazývá *minAreaEnclosingRectangle*, která zajišťuje generalizaci vstupního polygonu. Na vstupu je tedy *QPolygon pol* a *algorithm*, který představuje zvolený algoritmus pro tvorbu konvexní obálky. Výstupem je generalizovaná budova *er_new*, ve formátu *QPolygon*. Podrobnější průběh této metody je rozepsán v pseudokódu níže.

createCHJarvis:

Inicializuj prázdnou konvexní obálku ch

Inicializuj pivota q jako bod s minimální souřadnicí y

Inicializuj p_j *jako q a* p_{j-1} *jako bod, kde* $x = q.x \;\; 10a, \;\; y = q.y$

Přidej q do ch

Dokud $p_{j-1} \neq q$ *(ignoruj při prvním průchodu):*

Pro *všechny vstupní body p:*

Zjisti úhel mezi přímkou p_j, p_{j-1} *a přímkou* p_j *a p*

Zapamatuj si bod p definující úsečku s největším úhlem

Přidej zapamatovaný bod od ch

Aktualizuj p_{j-1} *na* p_j

Aktualizuj p_j *na zapamatovaný bod*

Vrať ch.

createCHGraham:

Inicializuj pivota q jako bod s minimální souřadnicí y

Získej prioritní frontu bodů p seřazených podle velikosti úhlů mezi přímkou, která prochází bodem q a je rovnoběžná s osou x, a přímkou definovanou body q a p

Inicializuj prázdný zásobník S

Inicializuj index J na 1

Zjisti délku prioritní fronty n

Inicializuj pivota q jako bod s minimální souřadnicí y

Přidej q do zásobníku S, přidej první prvek z prioritní fortny do S

Dokud *index j ¡ n:*

Zjisti pozici bodu P_j *z prioritní fronty vůči úsečce definovanou posledními 2 body přidanými do zásobníku S*

Když *bod* P_j *neleží v pravé polorovině:*

Přidej jej do zásobníku S

Aktualizuj j na $j = j + 1$

Jinak *odeber poslední přidaný prvek do zásobníku S*

Výsledný zásobník S představuje konvexní obálku.

minAreaEnclosingRectangle:

Vytvoř konvexní obálku ch ze vstupního polygonu

Inicializuj minimální směrnici min-max boxu na 0

Inicializuj plochu min-max boxu a jeho reprezentaci na min-max box vytvořen z nenatočeného polygonu

***Pro** všechny hrany konvexní obálky:*

Vypočítej směrnici hrany σ_i

Rotuj konvexní obálku o σ_i

Vytvoř min-max box rotované konvexní obálky

Vypočti jeho plochu

***Když** je plocha minimální:*

Zapamatuj si plochu, minimální směrnici a rotovaný min-max box

Rotuj min-max box zpět o minimální nalezenou směrnici

Změň velikost min-max boxu, aby měl stejnou plochu jako původní polygon

V druhé třídě `WallAverage` jsou definované metody pro výpočet generalizace pomocí algoritmu Wall Average. Tato třída obsahuje pět metod. První čtyři – *rotate, minMaxBox, getArea a resizeRectangle* – jsou stejné jako v předchozí třídě. Poslední metoda *computeWallAverage* bere vstupní polygon *pol QPolygon* a aplikuje na něj generalizační algoritmus Wall Average. Výstupem je poté *QPolygon*, který aproximuje vstupní polygon. Podrobnější průběh této metody je rozepsán v pseudokódu níže.

computeWallAverage:

Inicializuj směrnici sigma hrany definovanou prvními 2 body vstupního polygonu

***Pro** všechny hrany v polygonu:*

Vypočítej jejich směrnici σ_i

Redukuj σ_i podle σ'

Vypočítej k_i

Vypočítej zbytek po dělení r_i

Vypočítej velikost aktuální hrany w_i

Získej hlavní směr budovy σ

Poslední je třída `LongestEdge`, která definuje metody pro generalizaci pomocí algoritmu Longest Edge. Znovu obsahuje metody *minMaxBox, getArea, rotate* a *resizeRectangle*, které jsou opět stejné jako v předchozích

třídách. Metoda *getLongestEdgeDirection* vrací směrnici nejdelší hrany ve vstupním polygonu *pol QPolygon*. Metoda *computeLongestEdge* aplikuje generalizační metody na vstupní polygon *pol QPolygon* a vrací jeho aproximaci *mmb_new QPolygon*. Podrobnější průběh této metody je rozepsán v pseudokódu níže.

getLongestEdgeDirection:

Inicializuj velikost nejdelší hrany na -1

Pro *všechny hrany ve vstupním polygonu:*

Vypočítej velikost hrany

Když *je velikost hrany větší než nejdelší hrana:*

Aktualizuj velikost nejdelší hrany

Zapamatuj si startovní a koncový bod aktuální hrany

Získej hlavní směr budovy jako směrnici nejdelší zapamatované hrany

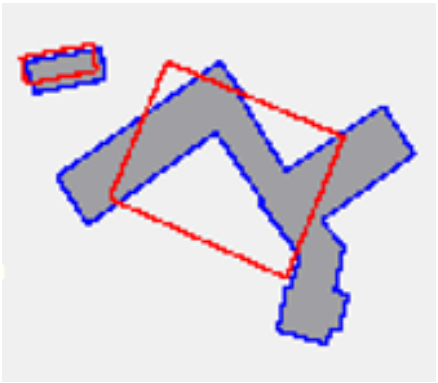
9 Porovnání algoritmů

Bylo provedeno testování algoritmů na datasetech *Branik.shp*, *Kamyk.shp* a *Zbraslav.shp* o obsahu každého 100 budov. Budovy byly hodnoceny vizuálně, jedná se tedy o subjektivní hodnocení. Výsledný poměr správně vyhodnocených budov ku celkovému počtu budov je uveden v Tabulce 1 v procentech.

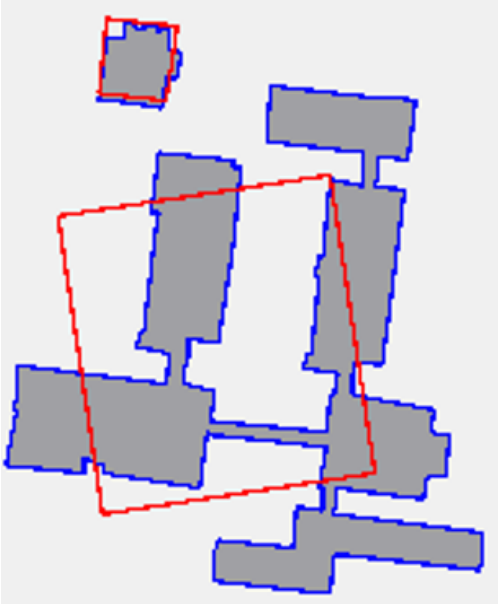
Algoritmus	oblast Braníku	oblast Kamýku	oblast Modřan
Minimum Area Enclosing Rectangle	100%	97%	99%
Wall Average	98%	100%	98%
Longest Edge	99%	100%	100%

Tabulka 1 - Úspěšnost algoritmů pro datasety Branik.shp, Kamyk.shp a Zbraslav.shp

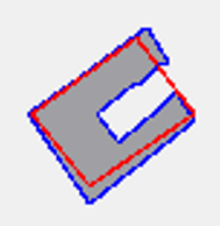
Hned na začátku je nutné říci, že si algoritmy vedly téměř výborně na třech vybraných datasetech. Pro algoritmus Minimum Area Enclosing Rectangle byly problematické budovy, jejichž tvar obsahoval písmeno Z (Obr. 9) a nebo složitější komplex (Obr. 10). Naopak si velmi dobře poradil s budovami ve tvaru U (Obr. 11)



Obr. 9 - Výsledná generalizace algoritmu Minimum Area Enclosing Rectangle

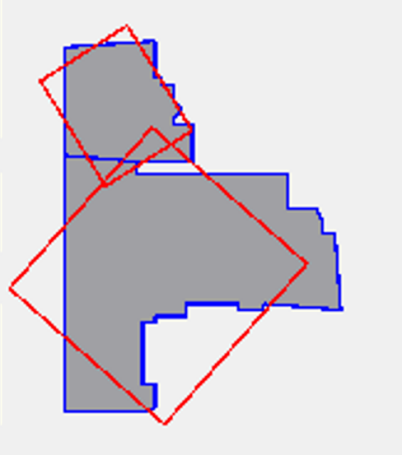


Obr. 10 - Výsledná generalizace algoritmu Minimum Area Enclosing Rectangle

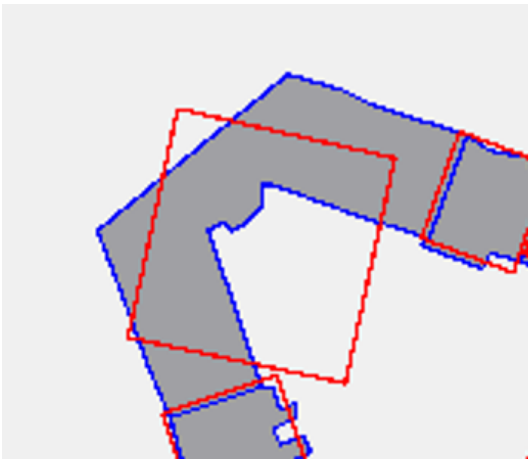


Obr. 11 - Výsledná generalizace algoritmu Minimum Area Enclosing Rectangle

Algoritmus Wall Average měl problém s budovami podobné tvaru L (Obr. 12) nebo E (Obr. 13).

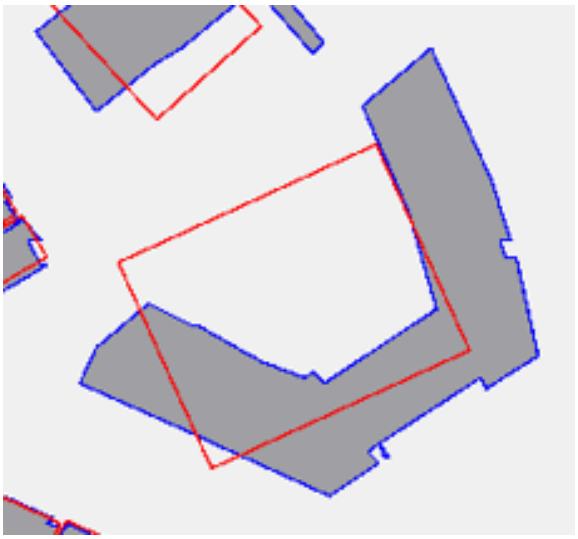


Obr. 12 - Výsledná generalizace algoritmu Wall Average



Obr. 13 - Výsledná generalizace algoritmu Wall Average

Algoritmus Longest Edge neměl téměř žádný problém s těmito datasety. Jediný problém byl u budovy ve tvaru více rozevřeného U (Obr. 14).



Obr. 14 - Výsledná generalizace algoritmu Longest Edge

Na závěr je nutné dodat, že pro jiné datasety se výsledky budou lišit a nelze říci objektivně, který je ten nejlepší. Vždy to bude záviset na orientaci vzhledem k ostatním prvkům mapy. Na závěr v Přílohách naleznete výstupy všech algoritmů pro všechny uvedené datasety.

10 Zdroje

BÆRENTZEN, J. A., GRAVESEN, J., ANTON, F., AENÆS, H. (2012): Convex Hulls. In: J. A. Bærentzen, J. Gravesen, F. Anton, H. Aenæs (ed.): Guide to Computational Geometry Processing - Foundations, Algorithms, and Methods. Springer, Londýn, 330 s.

DE BERG, M., CHEONG, O., VAN KREVELD, M., OVERMARS, M. (2008): Computational Geometry: Algorithms and Applications. Third Edition. Springer, Berlín, Heidelberg, 388 s.

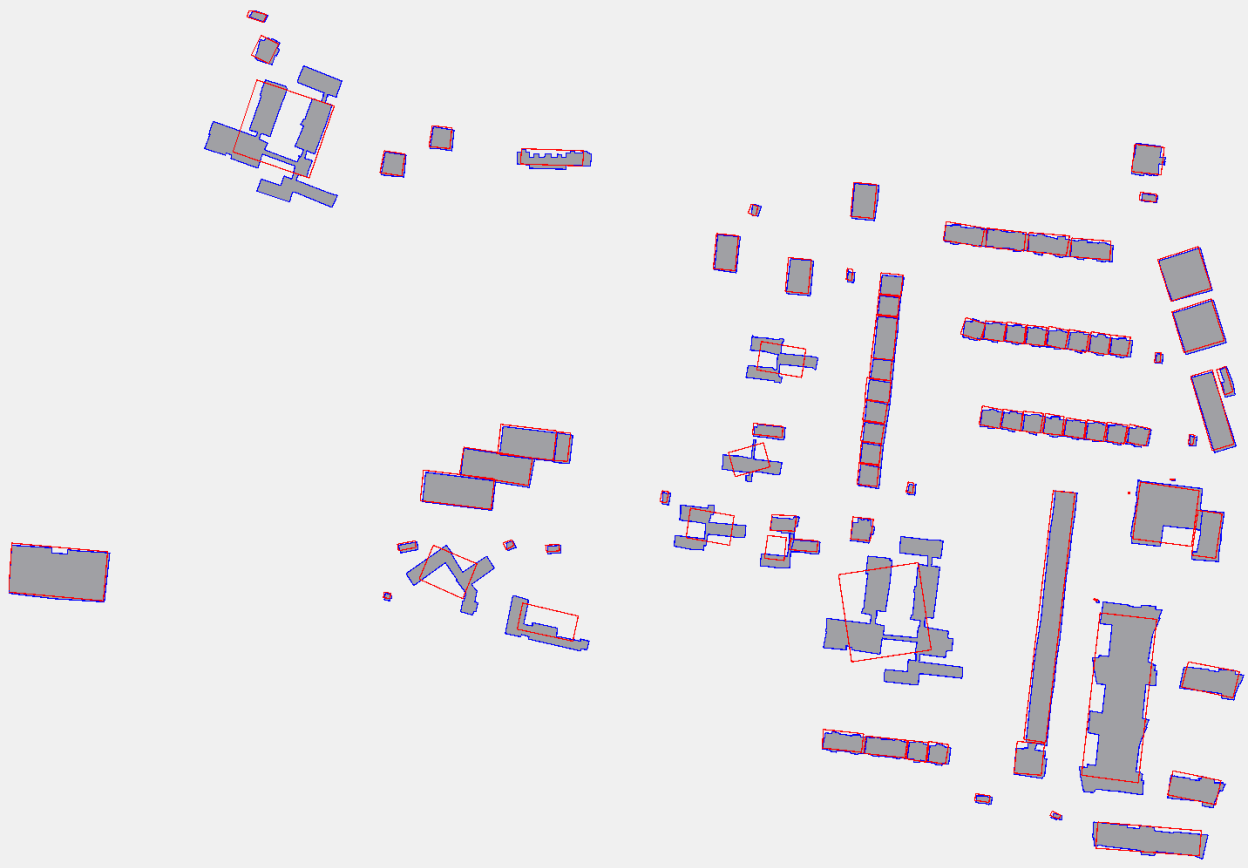
DUCHÉNE, C., BARD, S., BARILLOT, X., RUAS, A., TRÉVISAN, J., HOLZAPFEL, F. (2003): Quantitative and qualitative description of building orientation. Institut Géographique National, Paříž, 11 s.

O'ROURKE, J. (1998): Computational Geometry in C. Second Edition. Cambridge University Press, 392 s.

11 Přílohy

Příloha 1 - Výstup algoritmu Minimum Area Enclosing Rectangle pro dataset *Branik.shp*

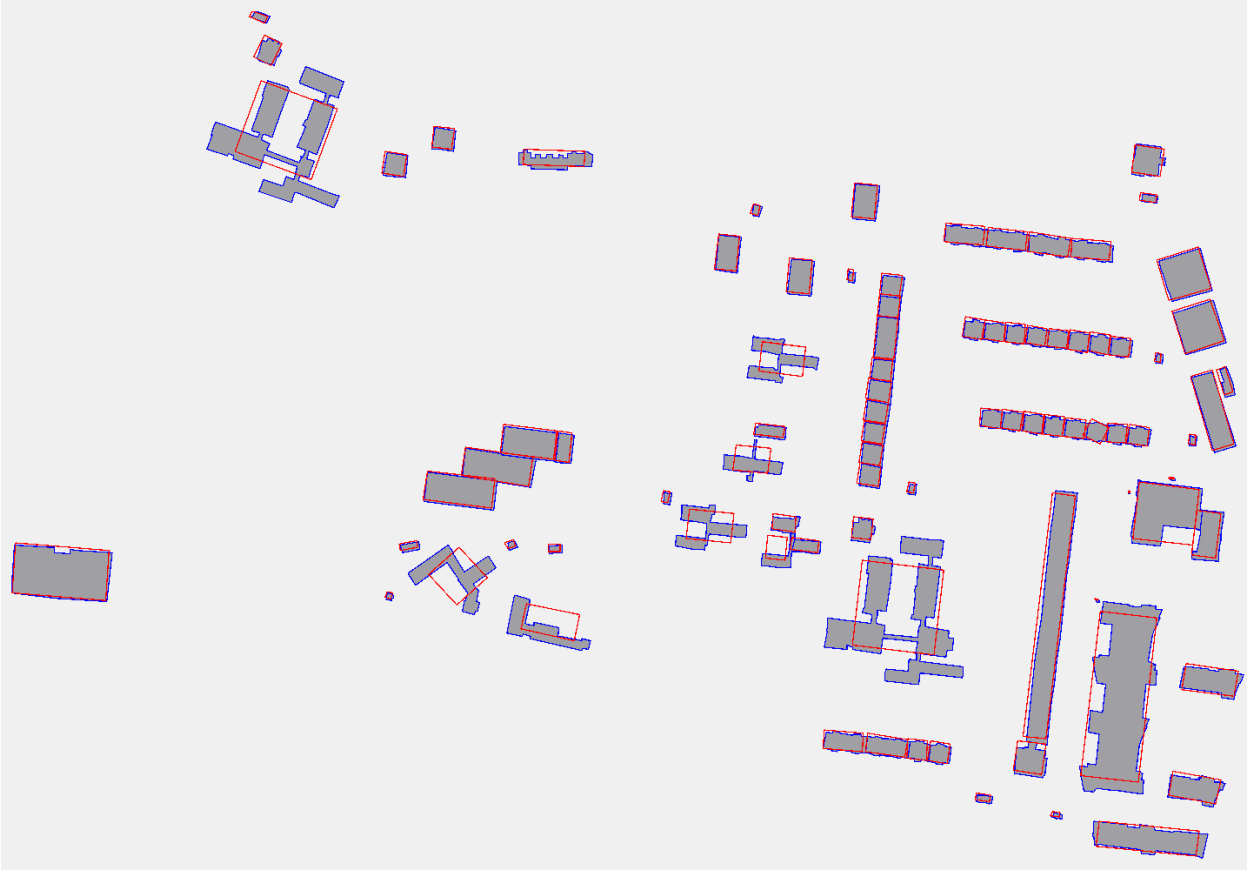
Příloha 2 - Výstup algoritmu Minimum Area Enclosing Rectangle pro dataset *Kamyk.shp*



Příloha 3 - Výstup algoritmu Minimum Area Enclosing Rectangle pro dataset *Zbraslav.shp*

Příloha 4 - Výstup algoritmu Wall Average pro dataset *Branik.shp*

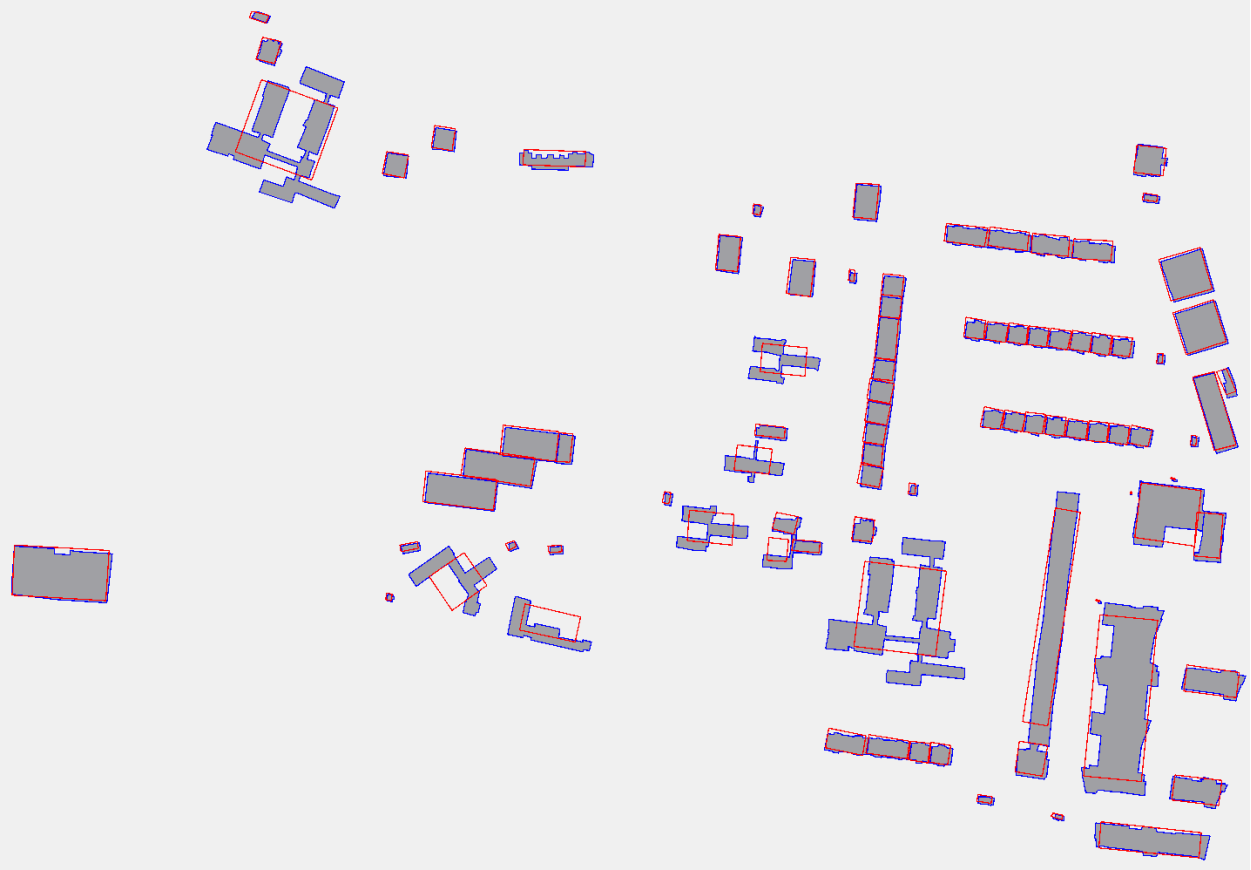
Příloha 5 - Výstup algoritmu Wall Average pro dataset *Kamyk.shp*



Příloha 6 - Výstup algoritmu Wall Average pro dataset *Zbraslav.shp*

Příloha 7 - Výstup algoritmu Longest Edge pro dataset *Branik.shp*

Příloha 8 - Výstup algoritmu Longest Edge pro dataset *Kamyk.shp*



Příloha 9 - Výstup algoritmu Longest Edge pro dataset *Zbraslav.shp*