Úloha 2 – Generalizace budov

Algoritmy počítačové kartografie

Sabina Fukalová, Klára Linková

Praha 2023

1. Zadání úlohy

1.1. Povinná část

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

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 oblouků 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 vhodnou aproximaci.

1.2. Volitelná část

V rámci tohoto úkolu nebyly řešeny žádné nepovinné části.

2. Popis a rozbor problému

Generalizace umožňuje snížit množství informací a zrychlit proces vizualizace. Snaží se odstranit z mapy prvky, které nejsou v kontextu mapy významné. Metoda je však subjektivní, jelikož závisí na pohledu kartografa a jeho zkušenostech. Proto je vhodné zavést algoritmus, který by vhodně zjednodušil původní objekty a nezávisel na kartografovi. Běžně používané generalizační algoritmy ale nemohou být v případě budov použity, protože nezachovávají vnitřní úhly polygonů (-90° a 90°) (Bayer 2008b).

2.1. Konvexní obálka

„Je dána množina bodů P_i v E^2, kde $P_i = [x_i, y_i]$. Konvexní obálka množiny P představuje hranici nejmenší konvexní množiny C obsahující celou množinu P“ (Bayer, 2008a, str. 138). Pro konvexní obálku platí následující:

- žádný z bodů P_i neleží vně konvexní množiny C
- spojnice libovolných dvou prvků P_i leží celá uvnitř konvexní množiny C
- všechny vnitřní úhly mezi sousedními segmenty konvexní obálky jsou menší než 180° (Bayer 2008a).

2.2. Metody konstrukce konvexní obálky

Pro vytvoření konvexní obálky v rovině se využívá několik algoritmů. V následujících kapitolách budou blíže popsány tři nejčastěji používané algoritmy – Grahamovo prohledávání, Jarvisovo prohledávání a algoritmus Quick Hull.

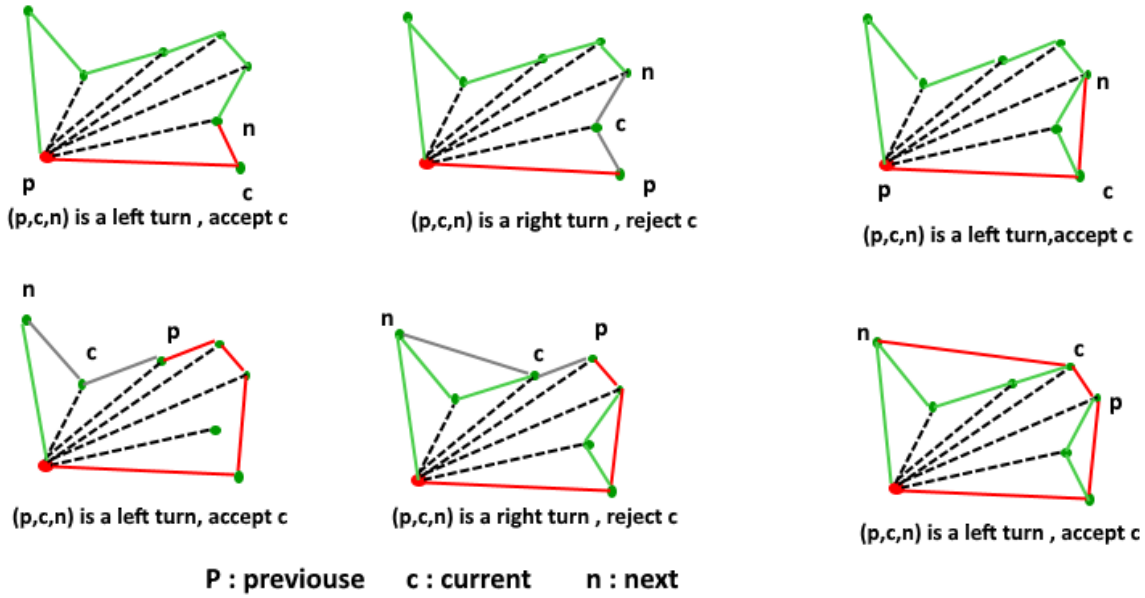
2.2.1. Grahamovo prohledávání

Při použití Grahamova skenování dochází k převodu nekonvexního mnohoúhelník na konvexní. Využívá se kritérium pravotočivosti, kdy se posuzuje úhel ω_i, který svírají úsečky procházející body P_{i-1}, P_i, P_{i+1} (obrázek 1). Pokud je úhel ω_i větší než 180°, bod P_{i+1} leží vlevo od spojnice P_{i-1}, P_i. Když je úhel ω_i menší než 180°, leží bod P_{i+1} vpravo od spojnice P_{i-1}, P_i (Bayer 2008a). Postup Grahamova prohledávání se skládá z následujících kroků:

1. Nalezení pivota Q (např. bod s maximální souřadnicí *x*)
2. Setřídění bodů P_i vzestupně podle úhlu ω mezi osou x a spojnicí Q – P_i
3. Setříděné body jsou spojeny do nekonvexního mnohoúhelníku
4. Posuzování kritéria pravotočivosti u bodů P_{i-1}, P_i a P_{i+1}
    a. Když $\omega_i > 180°$: Prostřední bod P_i leží možná na konvexní obálce – posun o jeden bod vpřed a zkouška kritéria pravotočivosti u bodů P_i, P_{i+1}, P_{i+2}
    b. Když $\omega_i < 180°$: Bod P_i určitě neleží na konvexní obálce – odstraní se ze seznamu, posun o jeden bod zpět a doplnění následujícím vrcholem – opakované prohledávání

5. Výpočet probíhá do té doby, dokud existuje alespoň jedna trojice vrcholů, u nichž by kritérium pravotočivosti ω_i bylo menší než 180° (Bayer 2008a)

Obrázek 1: Tvorba konvexní obálky pomocí Grahamova prohledávání



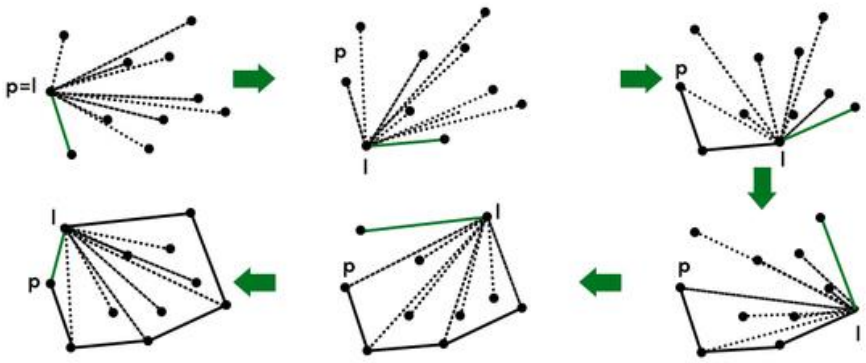
Zdroj: GeeksforGeeks (2023)

2.2.2. Jarvisovo prohledávání

Jarvisovo prohledávání je založeno na porovnávání úhlů ω mezi poslední stranou konvexní obálky tvořenou body P_{i-1} a P_i, a úsečkou spojující bod P_i a hodnotící bod P_{akt} (obrázek 2). Pro hledaný bod platí, že tento úhel je maximální. Jednotlivé kroky jsou následující:

1. Nalezení pivota Q (např. bod s nejnižší souřadnicí *y*), který leží automaticky na konvexní obálce
2. Bodem Q je vedena rovnoběžka s osou x
3. Body datasetu jsou postupně zpracovávány a měřeny úhly mezi přímkou procházející bodem Q a úsečkou spojující bod Q a P_{akt}
4. Ze zpracovávaných bodů je vybrán ten, u něhož je úhel maximální = bod konvexní obálky
5. Opakovaný výpočet úhlu ω mezi úsečkou $P_{i-1}P_i$, která spojuje poslední a předposlední bod konvexní obálky a úsečkou $P_i P_{akt}$ spojující poslední bod konvexní obálky postupně se všemi nezařazenými body – vybrán vždy bod s maximálním úhlem ω
6. Výpočet je ukončen, pokud další bod konvexní obálky bude pivot Q

Obrázek 2: Tvorba konvexní obálky pomocí Jarvisova prohledávání



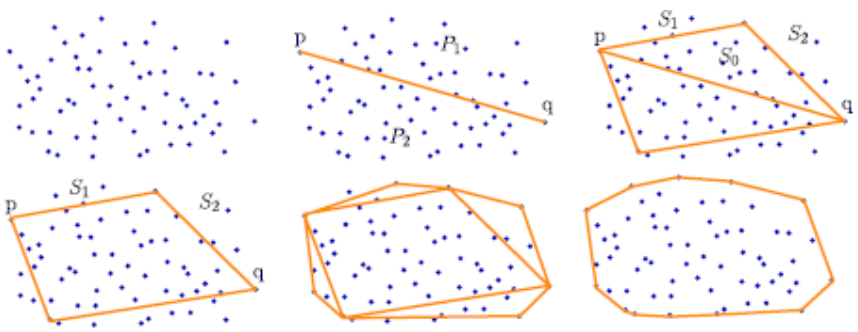
Zdroj: GeeksforGeeks (2022)

2.2.3. Quick Hull

Algoritmus Quick Hull je založen na principu, že nepracuje se všemi body vstupní množiny bodů, ale jen s body, které se nachází v blízkosti konvexní obálky. Tvar konvexní obálky je během procesu upravován, ale oproti předchozím algoritmům nejsou body přidávány postupně. Dochází tak k expanzi do všech směrů. Tento algoritmus pracuje ve dvou částech vstupní množiny (tzv. *upper hull* a *lower hull*) a výsledná konvexní obálka pak vznikne jejich spojením (Bayer 2008a). Hranice mezi horní a dolní částí je představována spojnicí dvou bodů min-max boxu s extrémními souřadnicemi *x*. Nad touto přímkou se nachází horní část, dolní část je pod touto přímkou. Jednotlivé kroky jsou následující:

1. Naleznou se body, které mají maximální a minimální souřadnici *x* a tyto body budou definovat přímku, která rozdělí vstupní body na dvě části
2. Postupně se hledají body, které jsou vůči segmentu konvexní obálky nejvzdálenější a zároveň leží v pravé polorovině (obrázek 3)
3. Výpočet probíhá do té doby, dokud existuje bod, který leží vpravo od segmentu

Obrázek 3: Tvorba konvexní obálky pomocí algoritmu Quick Hull



Zdroj: Hoang, Linh (2015)

3. Popis algoritmů

Dle Bayer (2008b) by měl algoritmus pro generalizaci budov splňovat tyto požadavky:

- Rozumná časová složitost (kvadratická nebo lepší) poskytující dostačující výsledky s minimem dodatečných manuálních oprav
- Schopnost detekce a generalizace budovy v jakékoliv poloze
- Odstranění vlastních průsečíků
- Schopnost zachovat plochu budovy
- Regulace faktoru generalizace uživatelem
- Schopnost generalizace složitých a nekonvexních polygonů

V následujících kapitolách budou blíže představeny algoritmy Minimum Area Enclosing Rectangle a Wall Average, které byly použity v aplikaci pro generalizaci budov, jež je výstupem této práce.

3.1. Minimum Area Enclosing Rectangle

Výsledkem algoritmu Minimum Area Enclosing Rectanle (MAER) je obdélník, který má minimální plochu a žádný bod neleží vně obdélníku. Zároveň alespoň jedna strana obdélníku je kolineární se stranou konvexní obálky. Algoritmus je založen na opakované rotaci – obdélník je natáčen o úhel -σ představující jednu ze směrnic strany konvexní obálky. V této poloze je vždy nalezen min-max box (MMB) a spočítá se jeho plocha. Otočením min-max boxu o úhel σ je min-max box převeden na obecný obdélník, ale velikost plochy zůstane stejná. Obdélník s nejmenší plochou bude výsledkem (Bayer 2023). Pro rotaci se využívá matice

$$\begin{bmatrix} \cos(-\sigma) & -\sin(-\sigma) \\ \sin(-\sigma) & \cos(-\sigma) \end{bmatrix}$$

Bližší popis algoritmu

1. Najdi konvexní obálku datasetu S (musí být splněna podmínka, že obsahuje alespoň 3 body)
2. Inicializuj MAER na MMB, zjisti souřadnice vrcholů a spočítej plochu: $\underline{A}$ = A(MMB(S))
3. **Projdi** hrany e konvexní obálky:
    - Vypočítej směrnici σ hrany e
    - Otoč S o -σ: $S_r = R(-\sigma)S$
    - Najdi MMB rotované množiny S_r, spočítej vrcholy a plochu A
    - **Když** je A < $\underline{A}$:
        - Zapamatuj si celý MMB (plocha, vrcholy, směrnice): $\underline{A}$ = A, $\underline{MMB}$ = MMB, $\underline{\sigma}$ = σ
4. MAER se získá tak, že se původní množina S narotuje o úhel σ_{min}, který odpovídá nejmenší ploše

3.2. Wall Average

Algoritmus Wall Average je založený na principu, že na každou stranu budovy je aplikováno modulo π/2 a z této hodnoty je poté spočten vážený průměr, kde váhy představují délky stran. Tento algoritmus dává nejlepší výsledky (Bayer 2023).

Bližší popis algoritmu

1. Vyber libovolnou stranu mnohoúhelníku a spočítej její směrnici σ'
2. **Projdi** směrnice σ_i všech hran budovy
    - Spočítej $\Delta\sigma_i = \sigma_i - \sigma'$ a natoč budovu o σ' → strany by měly být přibližně rovnoběžné s osami *x* a *y*
3. Pro každou hodnotu $\Delta\sigma_i$ spočítej k_i podle vzorce $k_i = 2\Delta\sigma_i/\pi$ a proveď zaokrouhlení
4. Pro každou hranu spočítej zbytek po dělení podle vzorce: $r_i = \Delta\sigma_i - k_i\pi/2$
5. Spočítej směr natočení budovy pomocí váženého průměru:

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

4. Data

Vstupem do vytvořené aplikace jsou polygonové vrstvy ve formátu *.shp*. Pro otestování byly použity tři soubory vstupních dat s různým typem zástavby. Prvním je dataset znázorňující část historického centrum Prahy (katastrální území Staré město), pro které jsou charakteristické úzké, nepravoúhlé ulice a nepravidelná zástavba. Druhý dataset pokrývá část katastrálního území Vinohrady, pro něhož je typická pravidelná zástavba a pravoúhlé budovy. Třetí dataset pokrývá část katastrální území Suchdol typické vilovou zástavbou. Výstup je zajištěn v podobě vizualizace v uživatelském rozhraní, kde jsou pro lepší odlišení od vstupních dat vizualizována jinou barvou.

5. Dokumentace

Vzhled výsledné aplikace je ukázán na obrázku 4. V aplikaci je možné otevřít soubor (Open File), provést generalizaci metodou *Minimum Area Enclosing Rectangle* a *Wall Average* a smazat výsledky. Vytvořený program je rozdělen do tří modulů – MainForm.py, draw.py a algorithms.py. V následujících kapitolách budou tyto moduly a obsahující funkce blíže představeny.

Obrázek 4: Náhled uživatelského rozhraní

5.1.Modul MainForm.py

Tento modul obsahuje třídu Ui_MainWindow, která byla vytvořena automaticky pomocí Qt Creatoru. Dále jsou doplněny metody, které se spustí při interakci s uživatelským rozhraním. Metoda *input* umožňuje načtení souboru. Dále modul obsahuje funkci *maerClick* a *waClick*, které volají algoritmy pro generalizace z modulu algorithms.py. Nakonec jsou funkce *clearClick* a *exit*. První zmíněná provádí smazání výsledků, druhá pak ukončení aplikace.

5.2.Modul draw.py

Modul *draw.py* obsahuje třídu *Draw*, kterou dědí od třídy *QWidget* z knihovny PyQt6. Díky tomuto modulu jsou vykreslovány vstupní data a výsledky jednotlivých funkcí.

Funkce *setPath* nastaví cestu ke vstupnímu souboru formátu *.shp* a načte ho do aplikace. Po kliknutí levým tlačítkem se spustí funkce *mousePressEvent*, která uloží souřadnice bodu p, ten přidá do polygonu a překreslí obrazovku. Metoda *paintEvent* vytvoří grafický objekt, nakreslí polygon s nastavenými atributy (barvou). Metoda clearResPol smaže všechny výsledky. Dále třída obsahuje *gettery* pro editaci dat z ostatních modulů.

5.3.Modul algorithms.py

V modulu *algorithms.py* obsahuje jednu třídu Algorithms, ve které se nachází jednotlivé algoritmy pro generalizaci budov.

Metoda *getPointPolygonPositionR* implementuje metodu pro určení polohy vzhledem k polygonu a přijímá dva argumenty: *q*, což je bodový objekt, a *pol*, což je seznam bodových objektů, které definují vrcholy polygonu. Metoda používá k určení polohy bodu algoritmus *ray casting*. Nejprve inicializuje dvě proměnné *k* a *n*, které představují počet průsečíků mezi vodorovným paprskem vycházejícím z bodu a hranami polygonu, respektive celkový počet vrcholů polygonu. Poté projdou smyčkou všechny vrcholy mnohoúhelníku a redukuje se jejich souřadnice vzhledem k zadanému bodu *q*. Pro každý vrchol také vypočítá souřadnice dalšího vrcholu polygonu, což je nutné k určení hrany, která se protíná s vodorovným paprskem. Pokud se hrana s paprskem protíná, vypočítá metoda průsečík bodu *xm* pomocí vzorce pro rovnici přímky. Pokud je tento průsečík napravo od bodu *q*, zvýší se proměnná *k*. Nakonec, je-li počet průsečíků *k* lichý, je bod *q* uvnitř mnohoúhelníku a metoda vrátí 1. V opačném případě je bod mimo polygon a metoda vrací hodnotu 0.

Metoda *get2LinesAngle* přijímá čtyři argumenty: *p1, p2, p3* a *p4*, což jsou bodové objekty reprezentující dvě přímky. Metoda nejprve vypočítá směrové vektory *u* a *v* pro obě přímky pomocí souřadnic daných bodů. Poté vypočítá skalární součin *u* a *v*, který se rovná součinu velikostí obou vektorů krát kosinus úhlu mezi nimi. Dále metoda vypočítá velikosti obou vektorů pomocí Pythagorovy věty. Poté vydělí skalární součin součinem magnitud a získá kosinus úhlu. Nakonec metoda vrátí hodnotu arccos, která udává úhel mezi oběma přímkami v radiánech. Pokud argument funkce arccos není v rozsahu od -1 do 1, vrací funkce hodnotu 0.

Metoda *createCH* implementuje metodu pro vytvoření konvexní obálky mnohoúhelníku pomocí algoritmu Jarvisova skenování. Metoda přijímá jeden argument: *Pol*, což je objekt QPolygonF reprezentující vstupní polygon. Metoda nejprve vytvoří prázdný objekt QPolygonF s názvem *ch*, do kterého uloží vrcholy konvexní obálky. Poté najde otočný bod *q* s nejmenší souřadnicí *y* ve vstupním polygonu. Inicializuje dva body *pj1* a *pj*, které představují poslední dva body přidané do konvexního trupu, přičemž *pj1* je vlevo od *q* a *pj* se rovná *q*. Metoda přidá ke konvexní obálce otočný bod *q*. Poté metoda vstoupí do smyčky, která pokračuje, dokud opět nedosáhne otočného bodu *q*. V každé iteraci metoda inicializuje maximální úhel *phi_max* na hodnotu 0 a index bodu s maximálním úhlem *i_max* na hodnotu -1. Poté projde smyčkou všechny body vstupního polygonu a vypočítá úhel mezi úsečkou z aktuálního bodu do *pj* a úsečkou z *pj1* do *pj* pomocí metody *get2LinesAngle*. Pokud je úhel větší než aktuální maximální úhel, metoda aktualizuje maximální úhel a index bodu s maximálním úhlem. Po prohledání všech bodů vstupního polygonu metoda připojí bod s maximálním úhlem do konvexní obálky a aktualizuje poslední dva body přidané do obálky. Nakonec metoda zkontroluje, zda se poslední bod

přidaný do obálky rovná otočnému bodu *q*. Pokud ano, přeruší smyčku a vrátí konvexní obálku. V opačném případě pokračuje ve smyčce, dokud nenajde otočný bod.

Metoda *rotate* nejprve vytvoří prázdný objekt QPolygonF s názvem *pol_rot*, do kterého se uloží vrcholy otočeného mnohoúhelníku. Poté projde všechny vrcholy vstupního mnohoúhelníku a pro každý vrchol vypočítá otočené souřadnice pomocí následujících vzorců:

$$x\_rot = x * \cos(sig) - y * \sin(sig)$$
$$y\_rot = x * \sin(sig) + y * \cos(sig)$$

kde *x* a *y* jsou souřadnice aktuálního vrcholu a *cos* a *sin* jsou funkce cosinus a sinus úhlu natočení *sig*. Metoda pak vytvoří objekt QPointF reprezentující otočený vrchol pomocí vypočtených souřadnic a připojí jej k otočenému polygonu *pol_rot*. Nakonec metoda vrátí otočený polygon *pol_rot*.

Funkce *minMaxBox* přijme jako vstup polygon *pol*, zjistí minimální a maximální souřadnice polygonu pro vytvoření obdélníku a vrátí obdélník jako objekt QPolygonF spolu s jeho plochou. Funkce nejprve zjistí minimální a maximální souřadnice *x* a *y* polygonu pomocí funkcí *min* a *max* a funkce *lambda*, která vrací souřadnici *x* nebo *y* bodu. Poté vytvoří čtyři objekty QPointF, které představují vrcholy obdélníku. Vrcholy jsou vytvořeny v určitém pořadí (v1, v2, v3, v4), aby byl obdélník vytvořen se správnou orientací. Nakonec funkce vrátí objekt QPolygonF reprezentující obdélník a jeho plochu, která je jednoduše součinem šířky a výšky obdélníku.

Funkce *computeArea* vypočítá plochu mnohoúhelníku pomocí vzorce, který funguje tak, že rozdělí mnohoúhelník na trojúhelníky a sečte jejich plochy. Funkce iteruje přes všechny vrcholy mnohoúhelníku a vypočítá plochu každého trojúhelníku pomocí křížového součinu dvou hran, které mají společný vrchol. Nakonec funkce sečte plochy všech trojúhelníků a vrátí polovinu absolutní hodnoty výsledku.

Funkce *resizeRectangle* vezme ohraničující obdélník konvexní obálky a změní jeho velikost tak, aby se vešel do původního mnohoúhelníku. Proměnná *k* je měřítkový faktor, který zajišťuje, že plocha zmenšeného obdélníku je rovna zlomku plochy původního mnohoúhelníku. Funkce začíná výpočtem těžiště obdélníku pomocí průměrných souřadnic jeho vrcholů. Poté vypočítá čtyři vektory, které vedou z těžiště do každého vrcholu obdélníku. Tyto vektory jsou poté škálovány koeficientem sqrt(k) a jejich nové souřadnice jsou použity k vytvoření nových vrcholů obdélníku změněné velikosti. Nakonec se pomocí nových vrcholů vytvoří nový mnohoúhelník, který se vrátí jako zmenšený obdélník.

Funkce *minAreaEnclosingRectangle* nejprve vytvoří konvexní obálku mnohoúhelníku pomocí funkce *createCH*. Poté pomocí funkce *minMaxBox* najde obdélník s minimální plochou ("minmax box") konvexního trupu. Tím se získá počáteční odhad obdélníku minimální plochy polygonu. Funkce poté

otočí konvexní obálku o malý úhel (vypočtený jako úhel mezi sousedními vrcholy konvexního obalu) a zjistí minimální plochu uzavírajícího obdélníku otočeného konvexního obalu. Pokud je plocha nového obdélníku s minimální plochou menší než předchozí odhad, funkce odhad aktualizuje a proces opakuje s otočeným konvexním trupem. Po zvážení všech rotací funkce vrátí obdélník s minimální plochou uzavírající původní polygon otočením minmax boxu o úhel, který vedl k nejmenší ploše, a změnou velikosti výsledného obdélníku tak, aby se vešel do polygonu pomocí funkce r*esizeRectangle*.

Funkce *wallAverage* přebírá objekt QPolygonF představující půdorys budovy a vrací objekt QPolygonF představující obdélník s minimální plochou, který obklopuje budovu a který byl získán pomocí metody průměrování stěn. Metoda průměrování stěn spočívá ve výpočtu průměrného směru hran budovy tak, že se vypočítá průměr zbytků úhlů mezi každou hranou a referenční hranou a poté se budova otočí o průměrný směr. Poté se najde minimální plocha ohraničujícího obdélníku výpočtem minimální plochy ohraničujícího obdélníku pootočené budovy, a nakonec se velikost obdélníku změní tak, aby odpovídal původní budově pomocí funkce *resizeRectangle*. Proměnná *n* je počet vrcholů budovy. Proměnná *sigma0* je směr první hrany budovy, který se používá jako referenční směr. Smyčka iteruje přes všechny hrany budovy a vypočítá směr každé hrany vzhledem k referenčnímu směru. Proměnná *deltasig* je rozdíl mezi směrem hrany *i* a referenčním směrem. Pokud je rozdíl záporný, opraví se přičtením 2**pi*. Proměnná *r* je zbytek úhlu po vyrovnání hrany s referenčním směrem, který se použije k výpočtu průměrného směru hrany budovy. Nakonec funkce otočí budovu o průměrný směr, vypočítá minimální plochu ohraničujícího obdélníku, otočí obdélník zpět do původní orientace a změní jeho velikost tak, aby se vešel do původní budovy.

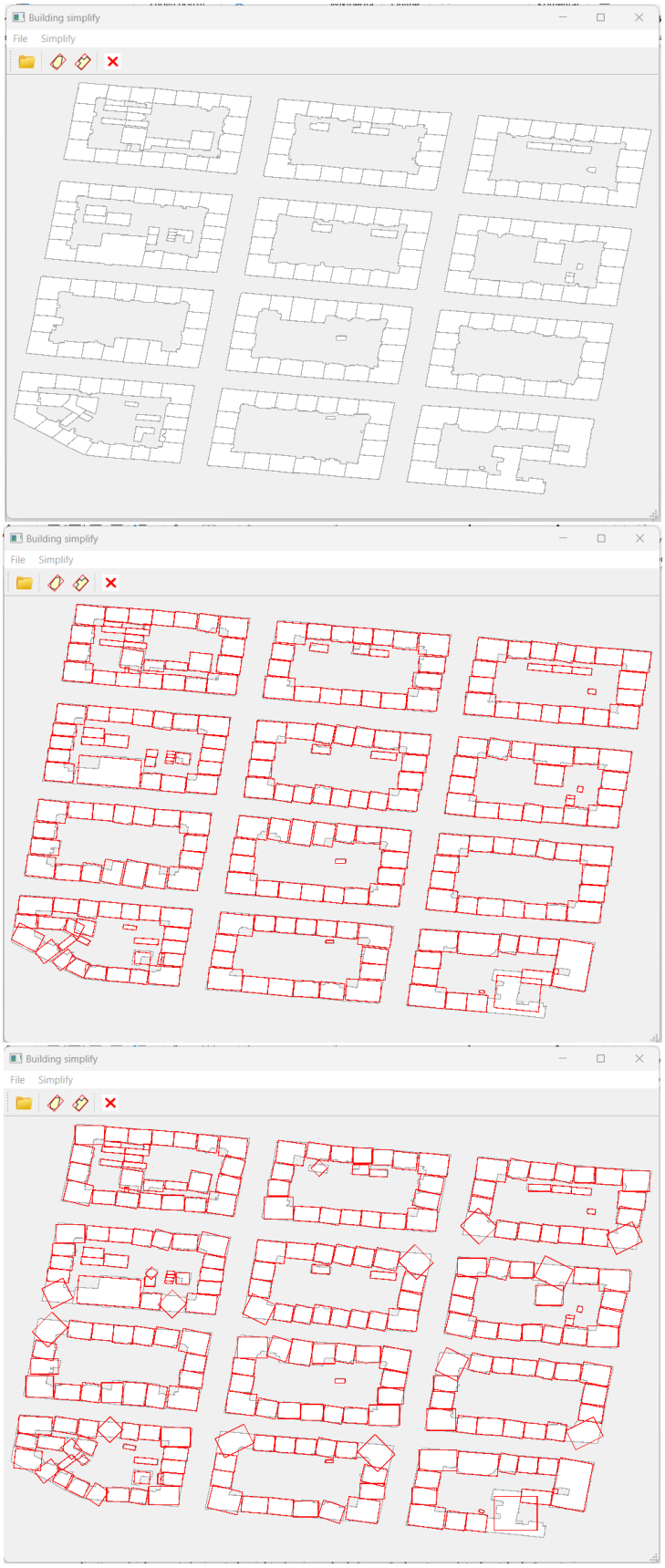
6. Výsledky

Pro otestování fungování vytvořené aplikace byly použity tři typy vstupních dat (viz kapitola 4). V případě pravidelné zástavby dávají oba algoritmy velmi dobré výsledky. Generalizace budov pomocí algoritmu Minimum Area Enclosing Rectangle dává podle vizuálního zhodnocení o něco lepší výsledky, jelikož je tvar budov ve většině případů zachován či zjednodušen tak, že nedochází k překryvům. U algoritmu Wall Average jsou některé budovy nelogicky natočeny (převážně rohové budovy jednotlivých bloků) a překryvů je více než u algoritmu Minimum Area Enclosing Rectangle. Výsledky jsou uvedeny na obrázku 5.

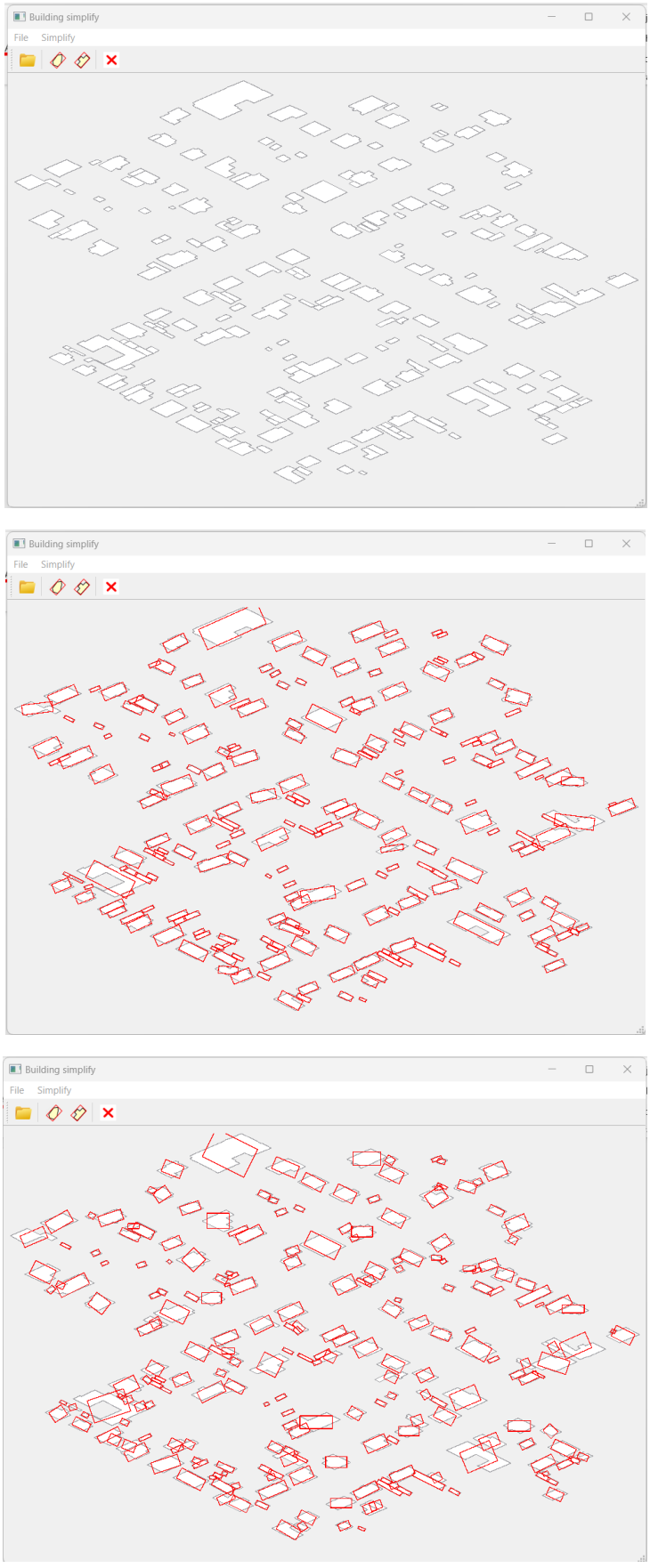
O něco horší výsledky dávají obě metody pro dataset vilové zástavby. Metoda Minimum Area Enclosing Rectangle lépe zachovává směr budov než metoda Wall Average. Jelikož jsou polygony od sebe dále, dochází k vzájemným překryvům jen v několika případech. Problém nastává u budov tvaru U, E, L či jiných složitějších tvarů, kdy jsou výchylky větší (hlavně v případě Wall Average). Výsledky jsou uvedeny na obrázku 6.

V případě datasetu historického centra dávají obě metody nejhorší výsledky v porovnání se zbylými dvěma datasety. Jelikož jsou budovy velmi blízko u sebe, dochází při generalizaci k jejich překrývání (obrázek 7). Důvodem problematické generalizace je i velké množství výběžků a jiných nepravidelností na budovách. O něco lepších výsledků u tohoto datasetu dosahuje algoritmus Minimum Area Enclosing Rectangle.

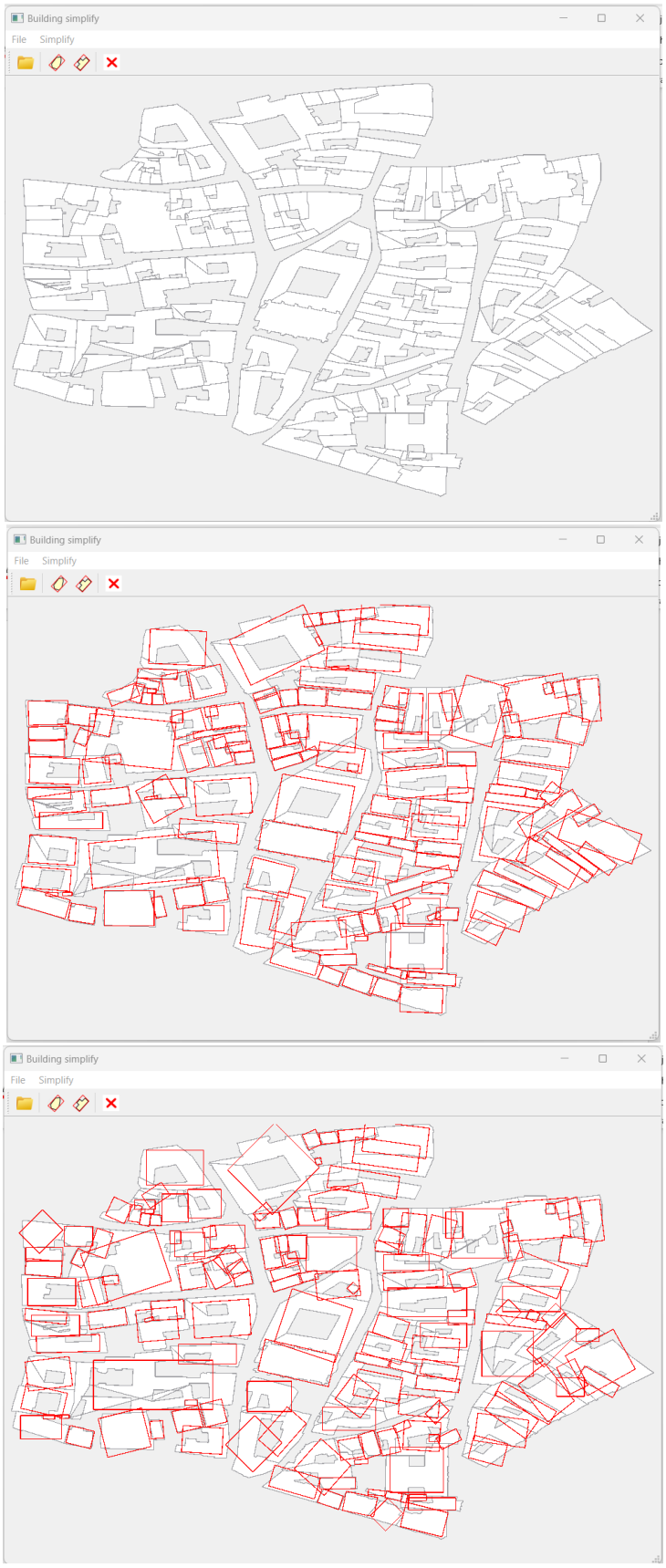
Obrázek 5: Generalizace budov pravidelné zástavby – vstupní data (nahoře), použití algoritmu Minimum Area Enclosing Rectangle (uprostřed) a Wall Average (dole)



Obrázek 6: Generalizace budov vilové zástavby – vstupní data (nahoře), použití algoritmu Minimum Area Enclosing Rectangle (uprostřed) a Wall Average (dole)



Obrázek 7: Generalizace budov historické části města – vstupní data (nahoře), použití algoritmu Minimum Area Enclosing Rectangle (uprostřed) a Wall Average (dole)



7. Závěr

Výstupem tohoto úkolu je aplikace, která by měla být schopná generalizovat mnohoúhelníky dvěma vybranými metodami – Minimum Area Enclosing Rectangle a Wall Average. Program umožňuje načíst data ve formátu *.shp*, provést samotnou generalizaci a vymazat dosavadní výsledky.

Funkčnost aplikace a obě metody byly otestovány nad třemi zvolenými datasety s různými typy zástavby – pravidelná zástavba, historické centrum a vilová zástavba. Zásadním problémem je neexistence metody pro určení přesností jednotlivých metod. Z tohoto důvodu bylo provedeno pouze vizuální porovnání, které je však velmi subjektivní. Na základě kapitoly 6 lze usoudit, že nejlepší výsledky dává aplikace pro případ pravidelné zástavby, naopak nejhorší v případě historické části města. Pro použité datasety dává o něco lepší výsledky metoda Minimum Area Enclosing Rectangle.

Vytvořená aplikace by mohla být dále doplněna o tlačítko, které by smazalo jak generalizované budovy, tak původní data, případně doladit, aby po provedení analýzy a načtení nových dat nezůstaly vykreslené výsledky předchozí analýzy. Další možností by bylo přidání dalších metod pro vytvoření konvexní obálky a samotné generalizace, čímž by bylo možné přizpůsobit metodu konkrétním datům.

8. Použité zdroje

BAYER, T. (2008a): Algoritmy v digitální kartografii. Univerzita Karlova v Praze, Nakladatelství Karolinum, Praha, 251 s.

BAYER, T. (2008b): The importance of computational geometry for digital cartography. Geoinformatics FCE CTU, 3, 15–24.

BAYER, T. (2023): Konvexní obálka množiny bodů. Výukový materiál. Dostupné z: http://web.natur.cuni.cz/~bayertom/images/courses/Adk/adk4_new.pdf (cit. 29.3.2023).

GEEKSFORGEEKS (2022): Convex Hull using Jarvis' Algorithm or Wrapping. Dostupné z: https://www.geeksforgeeks.org/convex-hull-using-jarvis-algorithm-or-wrapping/ (cit. 28.3.2023).

GEEKSFORGEESKS (2023): Convex Hull using Graham Scan. Dostupné z: https://www.geeksforgeeks.org/convex-hull-using-graham-scan/ (cit. 28.3.2023).

HOANG, N. D., LINH, N. K. (2015): Quicker than Quickhull. Vietnam Journal of Mathematics, 43, 57–70.