Úkol č. 2: Generalizácia budov

Hugo Majer, Júlia Šušková

3. apríla 2022

1 Zadanie

Úloha č. 2: Generalizace budov

Vstup: množina budov $B = \{Bi\}$, *budova* $Bi = \{P_{i,j}\}$
Výstup: $G(Bi)$.

Ze souboru načtěte vstupní data představová 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.

2 Údaje o bonusových úlohách

Boli riešené nasledujúce bonusové úlohy:

Generalizace budov metodou Longest Edge. (+5b)
Generalizace budov metodou Weighted Bisector. (+8b)
Implementace další metody konstrukce konvexní obálky (Graham Scan). (+5b)
Ošetření singulárního případu při generování konvexní obálky. (+2b)

3 Generalizácia budov

Kartografická generalizácia je procedúra redukcie reality, ktorou sa kartografovia zaoberajú od nepamäti. Pri tvorbe máp častokrát dochádza k situácií, kedy do mapy nie je možné úplne exaktne a geometricky presne vyjadriť obrazy objektov, procesov a ich vzťahy. Vtedy je nutné vytvoriť zjednodušené a zovšeobecnené obrazy týchto elementov, inými slovami potlačiť nepodstatné a ponechať len určité charakteristické črty daných elementov. Podstatné a nepodstatné prvky definuje napríklad merítko mapy a jej účel. Postup a spôsob generalizácie ale nie je unifikovaný a výsledok je vždy určitý kompromis (Droppová, 2011).

Automatizovanou generalizáciou sa kartografovia zaoberajú od 60. rokov minulého storočia. Na prelome tisícročí bolo vyvinutých mnoho geometrických algoritmov a ako výsledok vzniklo aj mnoho generalizačných algoritmov bodových, líniových a plošných prvkov mapy. Tieto algoritmy vykonávajú geometrickú transformáciu (Li et al., 2004).

Predmetom tejto práce je generalizácia budov. Tá môže zahrňovať ich výber, agregáciu, ale hlavne zjednodušenie obrysu budovy, hlavne za účelom zlepšiť vizuálnu kvalitu mapy. Počas tohto procesu musia byť zachované typické vlastnosti danej budovy (Sester, 2018). Tá je vo všeobecnosti popísaná pozíciou, tvarom, veľkosťou a orientáciou, pričom práve orientácia budovy bude skúmanou vlastnosťou v rámci tejto práce (Duchene et al., 2003).

Obrazom budovy v mape je polygón, u ktorého je pomerne obtiažné určiť orientáciu. U určitých tvarov budov je určenie orientácie celkom bezproblémové a priamočiare, ale takisto existujú tvary budov u ktorých určenie orientácie nie je možné. Práve z tohto dôvodu je ťažké definovať, čo vôbec orientácia budovy predstavuje (Duchene et al., 2003).
Orientácia budovy môže byť predstavovaná tzv. hlavnými smermi budovy, pomocou ktorých môžeme popísať jej natočenie voči ostatným prvkom mapy. Tento vzťah voči iným prvkom musí byť pri generalizácií budovy zachovaný, tj. totožný ako pred generalizáciou danej budovy.

Na určenie hlavných smerov budovy existuje viacero algoritmov, uveďme napr. *Minimum Area Enclosing Rectangle*, *Wall Average*, *Longest Edge* a *Weighted Bisector*.

4 Minimum Area Enclosing Rectangle

Uvažujme, že vstupom do procesu generalizácie budovy je polygón B tvorený lomovými bodmi $\{P_i\}$, pričom $P_i = [x_i, y_i]$. Hrany polygónu B nie sú rovnobežné s osami X a Y, ale sú voči ním orientované pod nejakým uhlom σ, ktorý musí byť detegovaný. Pre túto úlohu sa najčastejšie volí algoritmus *Minimum Area Enclosing Rectangle*, ktorý konštruuje obdĺžnik tak, aby obklopoval všetky lomové body budovy a aby jeho plocha bola minimálna. Tento obdĺžnik sa nazýva *Minimum Bounding Box*. Uhol σ je potom formovaný osou X a dlhšou stranou opisovaného obdĺžnika (Bayer, 2008). Prvý krok tohto algoritmu avšak pozostáva z tvorby konvexnej obálky, pre ktorú boli zvolené algoritmmi *Jarvis Scan* a *Graham Scan*.

4.1 Jarvis Scan

Pod pojmom konvexná obálka H rozumieme najmenší možný konvexný polygón, ktorý obklopuje vstupnú množinu bodov tak, že každý bod leží v jeho vnútri alebo na jeho obvode (tvorí hranu polygónu).

Na vstupe do *Jarvis Scan* máme množinu bodov $\{P_i\}$. Dva posledné body tvorenej konvexnej obálky H označujeme ako P_{j-1} a P_j. Bod P_{j+1} predstavuje aktuálne pridávaný bod do H. Ten musí maximalizovať uhol $\omega(P_{j-1}, P_j, P_i)$, pričom daný bod $P_i \notin H$:

$$P_{j+1} = max\, \omega(P_{j-1}, P_j, P_i)$$

Samotné uhly $\omega(P_{j-1}, P_j, P_i)$ spočítame pomocou známeho vzorca pre odchýlku dvoch vektorov $\vec{u}$ a $\vec{v}$, pričom $\vec{u}$ je smerový vektor priamky $\overleftrightarrow{P_{j-1}P_j}$ a $\vec{v}$ smerový vektor priamky $\overleftrightarrow{P_jP_i}$:

$$\cos\omega = \frac{\vec{u}\vec{v}}{\|\vec{u}\|\|\vec{v}\|}$$

Jarvis Scan je považovaný za jeden z prvých moderných algoritmov, je ľahko implementovateľný a formálne pseudokódom sa dá zapísať nasledovne:

Algorithm 1 *Jarvis Scan*

1: Nájdenie bodu P_{min} s minimálnou y súradnicou, $P_{min} = \min(y_i)$.
2: Pridaj bod P_{min} do tvorenej konvexnej obálky H: $P_{min} \in H$.
3: Inicializuj posledné dva body H: $P_{j-1} = [-\infty, \min(y_i)]$, $P_j = P_{min}$.
4: Inicializuj pridávaný bod do H: $P_{j+1} = P_{j-1}$.
5: Pokým $P_{j+1} \neq P_{min}$:
6: Pre každý vstupný bod P_i:
7: Spočítaj uhol $\omega(P_{j-1}, P_j, P_i)$.
8: Nastav $P_{j+1} = \max \omega(P_{j-1}, P_j, P_i)$.
9: Pridaj P_{j+1} do konvexnej obálky H: $P_{j+1} \in H$.
10: Aktualizuj posledné dva body H: $P_{j-1} = P_j$, $P_j = P_i$.

4.2 Graham Scan

Algoritmus *Graham Scan* je veľmi efektívny algoritmus, ktorý je založený na kritériu ľavotočivosti. Pozostáva z tvorby *star-shaped* polygónu z bodu P_{min}, ktorý predstavuje bod s minimálnou súradnicou y. Jeho vrcholy sú zoradené podľa uhlu $\omega(X, P_{min}, P_i)$. Pri uhle ω sa môže vyskytnúť situácia, kedy viacero bodov má rovnaký uhol ω. V takomto prípade sa ponechá len ten bod, ktorý je najvzdialenejší od bodu P_{min}.

Ľavotočivosť zabezpečujeme Half-Plane testom. Vždy z posledných dvoch bodov konvexnej obálky, označme ich $P_t = [x_t, y_t]$ a $P_{t-1} = [x_{t-1}, y_{t-1}]$, sa vytvorí priamka p voči ktorej určujeme pozíciu nasledujúceho bodu konvexnej obálky, označme ho $P_j = [x_j, y_j]$.
Poznámka: Posledné, resp. prvé dva body konvexnej obálky na začiatku predstavuje bod P_{min} a bod P_i, v ktorom je uhol $\omega(X, P_{min}, P_i)$ minimálny.

$$\vec{u} = (x_t - x_{t-1}, y_t - y_{t-1})$$
$$\vec{v} = (x_j - x_t, y_j - y_t)$$

Za pomoci vektorového súčinu smerových vektorov $\vec{u}$ a $\vec{v}$ zapísaného maticovo a pomocou znamienka determinantu určíme, v ktorej polrovine voči priamke $p(P_t, P_{t-1})$ sa bod P_j nachádza:

$$t = \begin{vmatrix} u_x & u_y \\ v_x & v_y \end{vmatrix} = u_x * v_y - v_x * u_y \Rightarrow \begin{cases} t > 0, & P_j(p) \in \sigma_l \\ t = 0, & P_j(p) \in p \\ t < 0, & P_j(p) \in \sigma_r \end{cases}$$

Ak bod $P_j \in \sigma_l$ alebo $P_j \in p$ (kolinearita), tak bude tvoriť konvexnú obálku H, tj. $P_j \in H$. Ak bod $P_j \in \sigma_r$, tak posledný bod konvexnej obálky P_t bude z nej vyradený, tým pádom poslednými dvoma bodmi H bude P_{t-1} a P_{t-2}.

Algorithm 2 *Graham Scan*

1: Nájdenie bodu P_{min} s minimálnou y súradnicou, $P_{min} = \min(y_i)$.
2: Pre každý bod P_i:
3: Spočítaj uhol $\omega(X, P_{min}, P_i)$.
4: Zoraď body P_i podľa uhlu ω od najmenšieho po najväčší, index j odpovedá zoradenému poradiu, n odpovedá počtu bodov.
5: Ak existujú body s rovnakým uhlom ω, ponechaj len ten najvzdialenejší od P_{min}.
6: Inicializuj prvé dva body konvexnej obálky: P_{min}, a bod $P_i = \min\ \omega(X, P_{min}, P_i)$, tj. bod s indexom $j = 0$.
7: Inicializuj $j = 1$.
8: Pokým $j < n$:
9: Ak $P_j(p) \in \sigma_l$ voči priamke $p(P_t, P_{t-1})$ (P_t, P_{t-1} sú posledné dva body H).
10: $P_j \in H$.
11: Aktualizuj posledné dva body konvexnej obálky H: $P_j = P_t$, $P_{t-1} = P_t$
12: Aktualizuj $j = j + 1$.
13: Ak je $P_j(p) \in \sigma_r$ voči priamke $p(P_t, P_{t-1})$:
14: Posledný bod konvexnej obálky P_t vyraď z konvexnej obálky H.
15: Aktualizuj posledné dva body konvexnej obálky H: $P_t = P_{t-1}$, $P_{t-1} = P_{t-2}$

4.3 Minimum Bounding Box

Po vytvorení konvexnej obálky H môžeme pristúpiť k tvorbe *Minimum Bounding Box*, tj. obdĺžnik, ktorý obklopuje všetky lomové body budovy a jeho plocha je minimálna. Dochádza ku opakovanej rotácií množiny vstupných lomových bodov $\{P_i\}$ o uhol $-\sigma$, ktorý je predstavovaný zápornou hodnotou smernice σ určitej hrany konvexnej obálky H. Dochádza vlastne k tomu, že konvexnú obálku "postavíme"na danú hranu.
Stanovme, že hrana konvexnej obálky H je tvorená bodmi $C_i = [x_i, y_i]$ a $C_{i+1} = [x_{i+1}, y_{i+1}]$. Smernicu hrany konvexnej obálky H spočítame nasledovne:

$$\Delta x = x_{i+1} - x_i$$
$$\Delta y = y_{i+1} - y_i$$
$$\tan\sigma = \frac{\Delta y}{\Delta x}$$

Vstupné lomové body, resp. konvexnú obálku rotujeme o $-\sigma$ za pomoci matice rotácie:

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

V takejto polohe je konštruovaný *Minimum Bounding Box* konvexnej obálky. V prvom rade musíme nájsť vrcholy tohto obdĺžniku, ktoré sú tvorené pomocou horných a spodných extrémov súradníc x_i a y_i. Pre vytvorený *Minimum Bounding Box* spočítame plochu S, spôsobom klasickým pre obdĺžnik. Takto vytvorený *Minimum Bounding Box* následne otočíme naspäť do pôvodnej polohy a to za pomoci uvedenej matice rotácie (uhol otočenia je ale σ) a samozrejme platí, že otočením sa plocha S nezmení. Výsledným *Minimum Bounding Box* po iterácií cez každú hranu konvexnej obálky je ten, ktorý má najmenšiu plochu S. Uhol, resp. smernicu hrany konvexnej obálky, v ktorej bol skonštruovaný označme σ_{min}.

Výsledný *Minimum Bounding Box* je nutné ešte upraviť tak, aby mal plochu S totožnú s plochou budovy S_b, ktorú aproximuje. Plochu budovy S_b, ktorá má tvar polygónu spočítame pomocou L'Huillierovho vzorca nasledovne:

$$S_b = \sum_{i=1}^{n} \frac{x_i(y_{i+1} - y_{i-1})}{2}$$

Následne spočítame pomer k, medzi plochou budovy S_b a plochou *Minimum Bounding Box* S:

$$k = \frac{S_b}{S}$$

Nech vrcholy *Minimum Bounding Box* tvorí štvorprvková množina $\{V_i\}$. Za využitia ťažiska *Minimum Bounding Box* T spočítame smerové vektory jeho uhlopriečok:

$$\vec{u_1} = V_1 - T$$
$$\vec{u_2} = V_2 - T$$
$$\vec{u_3} = V_3 - T$$
$$\vec{u_4} = V_4 - T$$

Nové vrcholy *Minimum Bounding Box* $\{V_i'\}$ vypočítame ako:

$$V_1' = T + \sqrt{k}\vec{u_1}$$
$$V_2' = T + \sqrt{k}\vec{u_2}$$
$$V_3' = T + \sqrt{k}\vec{u_3}$$
$$V_4' = T + \sqrt{k}\vec{u_4}$$

Minimum Bounding Box tvorený vrcholmi $\{V_i'\}$ ná rovnakú plochu S ako plocha budovy S_b.

Vo finále aproximáciou budovy je obdĺžnik, ktorý má rovnakú plochu S ako budova a jeho orientácia (natočenie) je totožná s uhlom σ_{min}, tj. uhlom, v ktorom bol nájdený optimálny *Minimum Bounding Box*, tj. taký, ktorý mal najmenšiu plochu S.

Pseudokód algoritmu *Minimum Area Enclosing Rectangle* zapíšeme nasledovne:

Algorithm 3 *Minimum Area Enclosing Rectangle*

1: Skonštruuj konvexnú obálku H, napr. metódou *Jarvis Scan* (detailne v kapitole 4.1) alebo *Graham Scan* (kapitola 4.2).
2: Vytvor iniciálnu aproximáciu *Minimum Bounding Box* MBB_{min} na základe konvexnej obálky H a spočítaj jeho plochu S_{min}.
3: Inicializuj $\sigma_{min} = 0$.
4: Pre každú hranu konvexnej obálky H opakuj:
5: Spočítaj smernicu σ hrany konvexnej obálky H.
6: Otoč konvexnú obálku H o uhol -σ.
7: V tejto polohe skonštruuj *Minimum Bounding Box* MBB a spočítaj jeho plochu S.
8: Ak $S < S_{min}$:
9: $S_{min} = S$, $MBB_{min} = MBB$, $\sigma_{min} = \sigma$.
10: Otoč MBB_{min} naspäť do pôvodnej polohy o uhol σ. // MBB_{min} je optimalny MBB //
11: Spočítaj plochu budovy S_b.
12: Uprav MBB_{min} tak, aby: $S(MBB_{min}) = S_b$.

5 Wall Average

Základným princípom tohto algoritmu je uvažovať orientáciu každej hrany budovy ako zvyšok po delení číslom $\frac{\pi}{2}$ (dostávame číslo od 0 do $\frac{\pi}{2}$) a z týchto hodnôt spočítať vážený priemer, pričom váhou je dĺžka hrany (Duchene et al., 2003).

V prvom kroku algoritmu sa pre ľubovoľnú hranu polygónu predstavujúceho budovu spočíta smernica σ'. Následne sa spočíta smernica σ_i pre každú hranu budovy. Hodnoty σ_i sa zredukujú o hodnotu σ':

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

Zo spočítaných hodnôt $\Delta\sigma_i$ sa vypočíta nadol zaokrúhlený podiel k_i.

$$k_i = \frac{2\Delta\sigma_i}{\pi}$$

V ďalšom kroku už dochádza ku samotnému spočítaniu zvyšku r_i, ktorého hodnota určí, či hrana je odchýlená skôr od vodorovného smeru ($r_i < \frac{\pi}{4}$) alebo skôr od vertikálneho smeru ($r_i > \frac{\pi}{4}$):

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

Ako bolo spomenuté, samotná orientácia budovy σ je určená za pomoci váženého priemeru zvyškov r_i, pričom váhou je dĺžka hrán, ktorú označme d_i, pričom d značí sumu všetkých dĺžok hrán:

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

Podobne ako pri predošlom algoritme sa budova otočí o uhol -σ, v tejto polohe sa skonštruuje *Minimum Bounding Box*, ten sa otočí naspäť o uhol σ a upraví sa tak, aby jeho plocha S bola totožná s plochou budovy S_b.

Pseudokód formálnym jazykom algoritmu *Wall Average* vypadá nasledovne:

Algorithm 4 *Wall Average*

1: Pre ľubovoľnu hranu budovy (P_i, P_{i+1}) spočítaj smernicu σ'.
2: Pre každú hranu budovy (P_i, P_{i+1}):
3: Spočítaj smernicu σ_i.
4: Spočítaj $\Delta\sigma_i$.
5: Spočítaj zaokrúhlený podiel k_i.
6: Spočítaj zvyšok r_i.
7: Spočítaj vážený priemer zvyškov r_i, váhou sú dĺžky hrán d_i: $\frac{\sum_{i=1}^{n} r_i d_i}{d}$
8: Spočítaj finálnu orientáciu budovy σ.
9: Otoč budovu o uhol -σ.
10: V tejto polohe skonštruuj *Minimum Bounding Box* MBB a spočítaj jeho plochu S.
11: Otoč *Minimum Bounding Box* o uhol σ.
12: Spočítaj plochu budovy S_b.
13: Uprav MBB_{min} tak, aby: $S(MBB_{min}) = S_b$.

6 Longest Edge

Jedná sa o veľmi jednoduchý algoritmus, v ktorom prvý hlavný smer budovy predstavuje smernica najdlhšej hrany polygónu predstavujúceho budovu. Druhý hlavný smer je kolmý na prvý smer.

Takisto ako v predošlých algoritmoch sa budova otočí o uhol -σ, v tejto polohe sa skonštruuje *Minimum Bounding Box*, ten sa otočí naspäť o uhol σ a upraví sa tak, aby jeho plocha S bola totožná s plochou budovy S_b.

Pseudokód algoritmu:

Algorithm 5 *Longest Edge*

1: Nájdi najdlhšiu hranu budovy.
2: Spočítaj jej smernicu σ.
3: Otoč budovu o uhol -σ.
4: V tejto polohe skonštruuj *Minimum Bounding Box* MBB a spočítaj jeho plochu S.
5: Otoč *Minimum Bounding Box* o uhol σ.
6: Spočítaj plochu budovy S_b.
7: Uprav MBB_{min} tak, aby: $S(MBB_{min}) = S_b$.

7 Weighted Bisector

Algoritmus spočíva v nájdení dvoch najdlhších uhlopriečok polygónu budovy a vypočítaní ich smernice. Najdlhšie uhlopriečky polygónu hľadáme spojením každého bodu s každým, tj. vznikajú úsečky. Myšlienka je, že ak táto úsečka pretína hranu budovy, nejedná sa o jej uhlopriečku. Nech $P_1 = [x_1, y_1]$ a $P_2 = [x_2, y_2]$ tvoria hranu budovy a spojnica bodov $P_i = [x_i, y_i]$ a $P_j = [x_j, y_j]$ bude úsečka, tak existenciu ich priesečníku vyšetrujeme pomocou smerových vektorov a ich súčinu zapísaného maticovo nasledovne:

$$t_1 = \begin{vmatrix} x_2 - x_1 & y_2 - y_1 \\ x_4 - x_1 & y_4 - y_1 \end{vmatrix} \quad t_2 = \begin{vmatrix} x_2 - x_1 & y_2 - y_1 \\ x_3 - x_1 & y_3 - y_1 \end{vmatrix}$$

$$t_3 = \begin{vmatrix} x_4 - x_3 & y_4 - y_3 \\ x_2 - x_3 & y_2 - y_3 \end{vmatrix} \quad t_4 = \begin{vmatrix} x_4 - x_3 & y_4 - y_3 \\ x_2 - x_3 & y_2 - y_3 \end{vmatrix}$$

Podľa znamienok vyššie uvedených determinantov určíme, či priesečník dvoch úsečiek existuje alebo neexistuje. Priesečník neexistuje, ak t_1 a t_2 majú rovnaké znamienko, alebo t_3 a t_4 majú rovnaké znamienko, čo znamená, že sa jedná o uhlopriečku.

Z nájdených uhlopriečok hľadáme dve najdlhšie a spočítame ich smernice σ_1 a σ_2. Hlavný smer budovy je daný váženým priemerom týchto smerníc, váhami sú hodnoty dĺžky uhlopriečok d_1 a d_2.

$$\sigma = \frac{d_1\sigma_1 + d_2\sigma_2}{d_1 + d_2}$$

Opäť sa budova otočí o uhol -σ, v tejto polohe sa skonštruuje *Minimum Bounding Box*, ten sa otočí naspäť o uhol σ a upraví sa tak, aby jeho plocha S bola totožná s plochou budovy S_b.

Pseudokód algoritmu:

Algorithm 6 *Weighted Bisector*

1: Nájdi dve najhlšie uhlopriečky budovy u_1 a u_2. (podrobne popísané v texte vyššie)
2: Pre u_1 a u_2 spočítaj smernicu σ_1 a σ_2.
3: Spočítaj hlavný smer budovy $\sigma = \frac{d_1\sigma_1 + d_2\sigma_2}{d_1 + d_2}$
4: Otoč budovu o uhol -σ.
5: V tejto polohe skonštruuj *Minimum Bounding Box* MBB a spočítaj jeho plochu S.
6: Otoč *Minimum Bounding Box* o uhol σ.
7: Spočítaj plochu budovy S_b.
8: Uprav MBB_{min} tak, aby: $S(MBB_{min}) = S_b$.

8 Aplikácia

Aplikácia bola vytvorená vo vývojovom prostredí QT 6. Slúži na generalizáciu tvaru budov a to za zachovania orientácie (hlavných smerov) danej budovy. Sú v nej implementované algoritmy *Minimum Area Enclosing Rectangle*, *Wall Average*, *Longest Edge* a *Weighted Bisector*. Metóda *Minimum Area Enclosing Rectangle* využíva konvexnú obálku, pre jej konštrukciu sú implementované dva algoritmy, konkrétne *Jarvis Scan* a *Graham Scan*.

Užívateľské rozhranie je veľmi jednoduché a intuitívne. Prevažnú časť okna aplikácie tvorí zobrazovacia plocha, na ktorej sa vizualizujú do aplikácie nahrané dáta. Tie užívateľ do aplikácie nahrá tlačidlom *INPUT SHP*. Po kliknutí naň vyskočí pop-up okno s možnosťou prehliadať súbory v PC. Je nutné zvoliť SHP súbor predstavujúci budovy vo forme polygónov.

Obr. 1: Výsledná generalizácia.

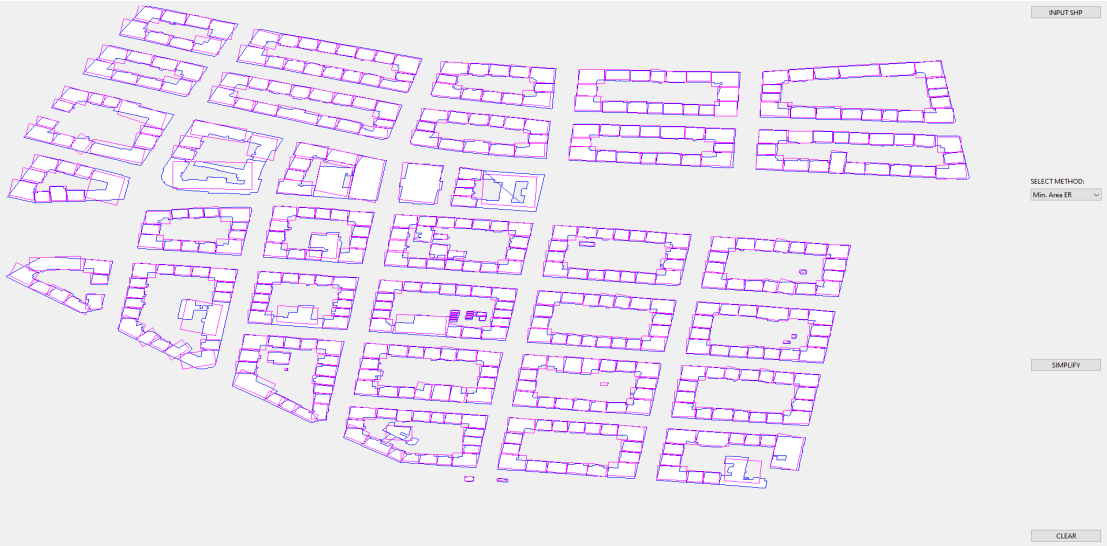
Po nahratí dát si užívateľ môže z combo-boxu vybrať, ktorý algoritmus chce použiť a ktorú metódu konštrukcie konvexnej obálky chce v rámci algoritmu *Minumum Area Enclosing Rectangle* využiť.

Samotná generalizácia sa vykoná po kliknutí na tlačidlo *SIMPLIFY* a výsledok sa zobrazí na zobrazovacej ploche (pozri Obr. 1).

Posledné tlačidlo *CLEAR* resetuje zobrazovaciu plochu.

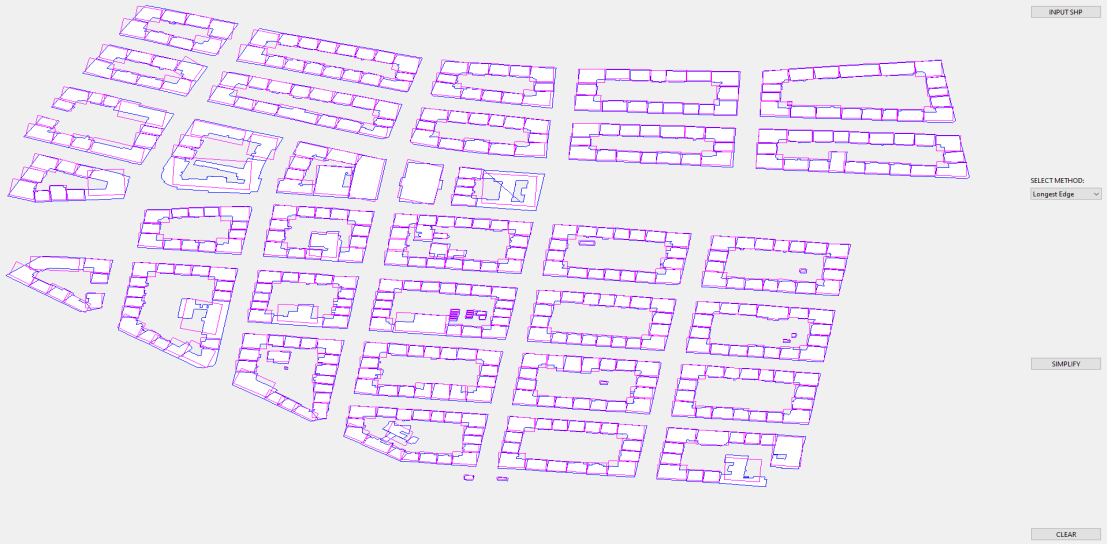
9 Zhodnotenie

V prvom rade boli všetky spomenuté algoritmy vyskúšané na objemnejších dátach, konkrétne na 505 polygónoch reprezentujúcich budovy v pražských Vinohradoch, tj. typické (pravidelné) bloky budov. Výsledky generalizácie tohto datasetu sú na Obr. 2, Obr. 3, Obr. 4, Obr. 5.



Obr. 2: Generalizácia datasetu Vinohrady algoritmom *Minimum Area Enclosing Rectangle*.

Obr. 3: Generalizácia datasetu Vinohrady algoritmom *Wall Average*.



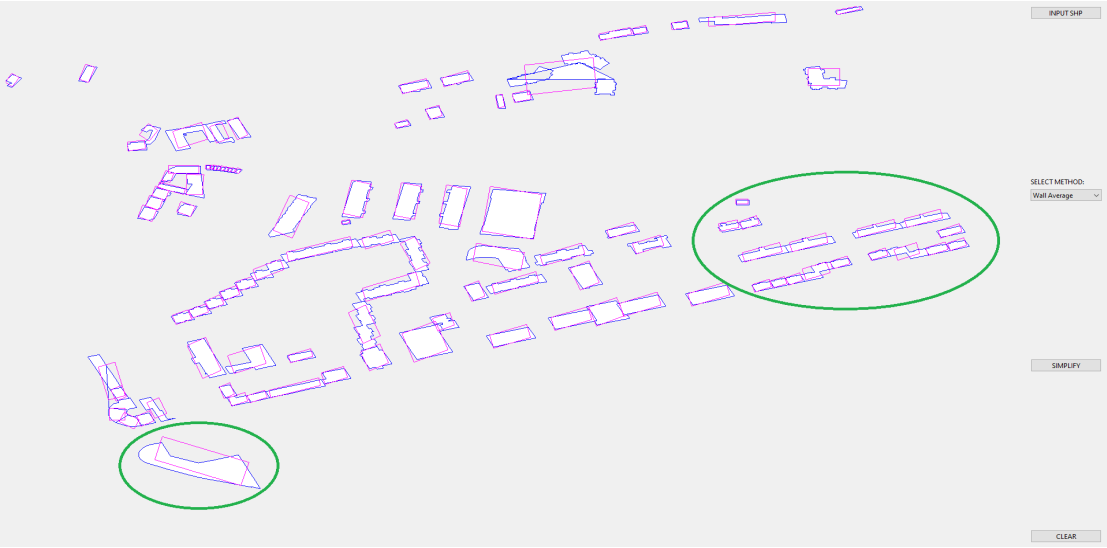
Obr. 4: Generalizácia datasetu Vinohrady algoritmom *Longest Edge*.

Obr. 5: Generalizácia datasetu Vinohrady algoritmom *Weighted Bisector*.

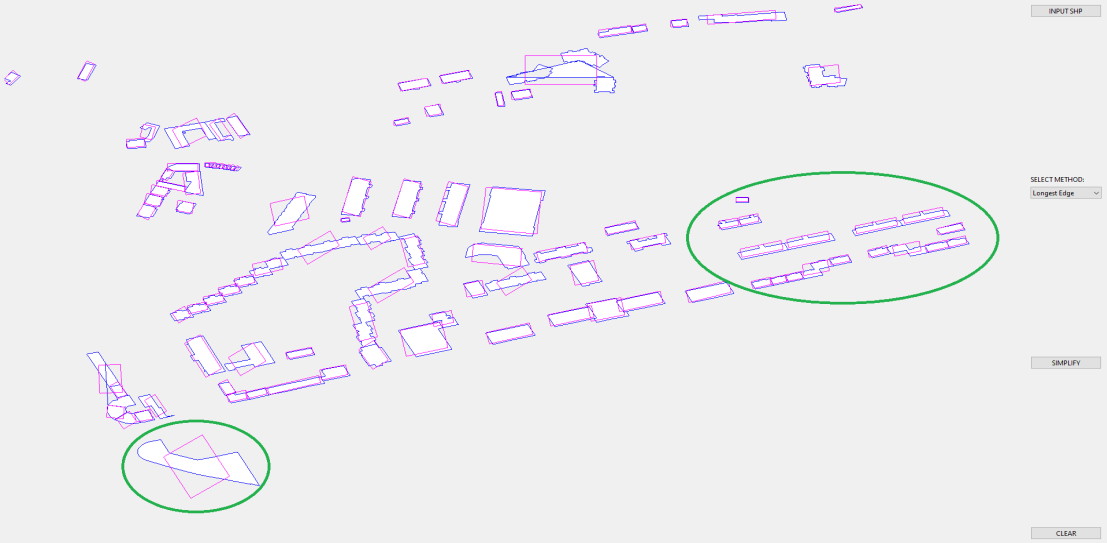
Algoritmy *Minimum Area Enclosing Rectangle*, *Wall Average* a *Longest Edge* tento dataset vyhodnotili veľmi podobne, rozdiely v orientácií generalizovaných budov sú viacmenej minimálne, a zároveň boli dosiahnuté uspokojivé výsledky. Algoritmus *Weighted Bisector* určil orientáciu pri väčšine budov nesprávne a výsledky sú neuspokojivé. Môžeme pozorovať, že častokrát budovy obdĺžnikového alebo štvorcového tvaru sú nesprávne natočené o cca 45°. Možnou príčinou chybných výsledkov tohto algoritmu je jeho nesprávna implementácia.

Druhým testovacím datasetom bol dataset obsahujúci množstvo zaujímavých tvarov budov. Výsledky jeho generalizácie sú na Obr. 6, Obr. 7, Obr. 8, Obr. 9.

Obr. 6: Generalizácia datasetu Libeň algoritmom *Minimum Area Enclosing Rectangle*.



Obr. 7: Generalizácia datasetu Libeň algoritmom *Wall Average*.

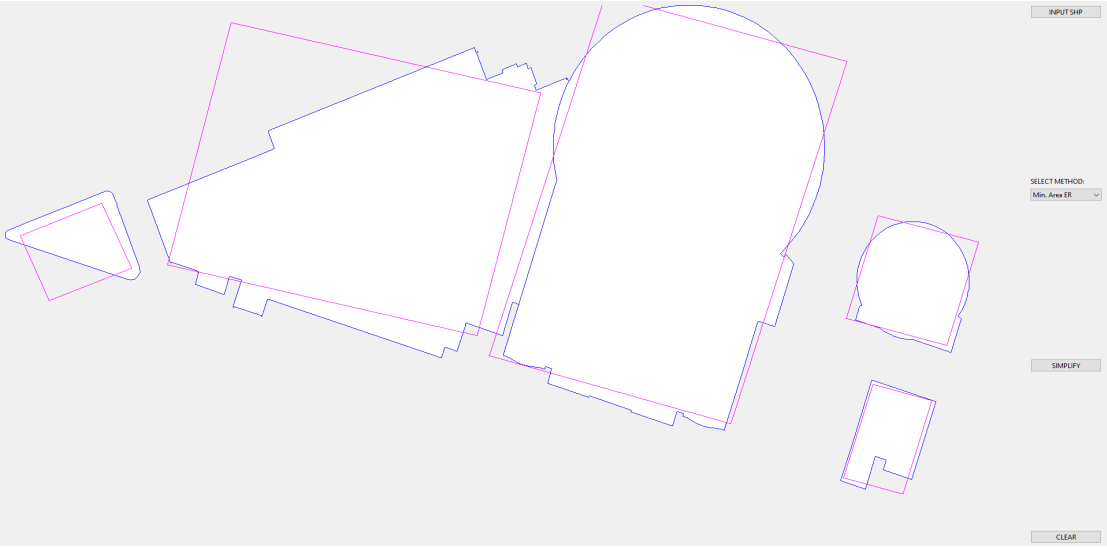


Obr. 8: Generalizácia datasetu Libeň algoritmom *Longest Edge*.

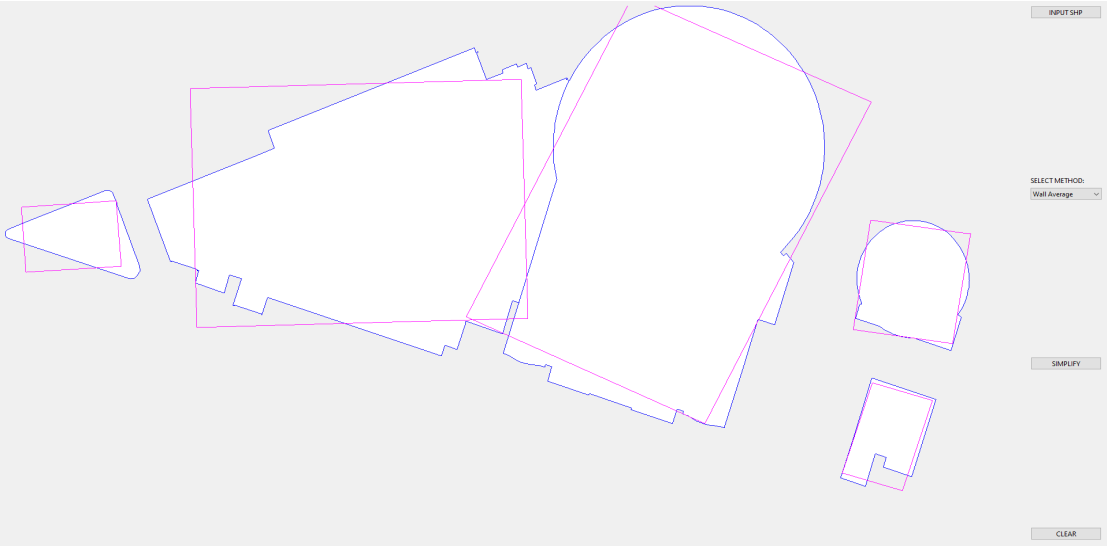
Obr. 9: Generalizácia datasetu Libeň algoritmom *Weighted Bisector*.

V pravej časti datasetu, kde sa nachádzajú budovy podlhovastého tvaru (zvýraznené) došlo ku najkvalitnejšej generalizácií algoritmom *Longest Edge*, ktorý ale pre zvýraznenú budovu iregulárneho tvaru (budova v ľavom dolnom rohu) nedosiahol uspokojivý výsledok. S tou si najlepšie poradil prekvapujúco *Weighted Bisector*, kvalitne mu sekunduje *Wall Average*. Nutno dodať, že generalizácia *Weighted Bisector* trvá dlhší čas v porovnaní s ostatnými algoritmami.

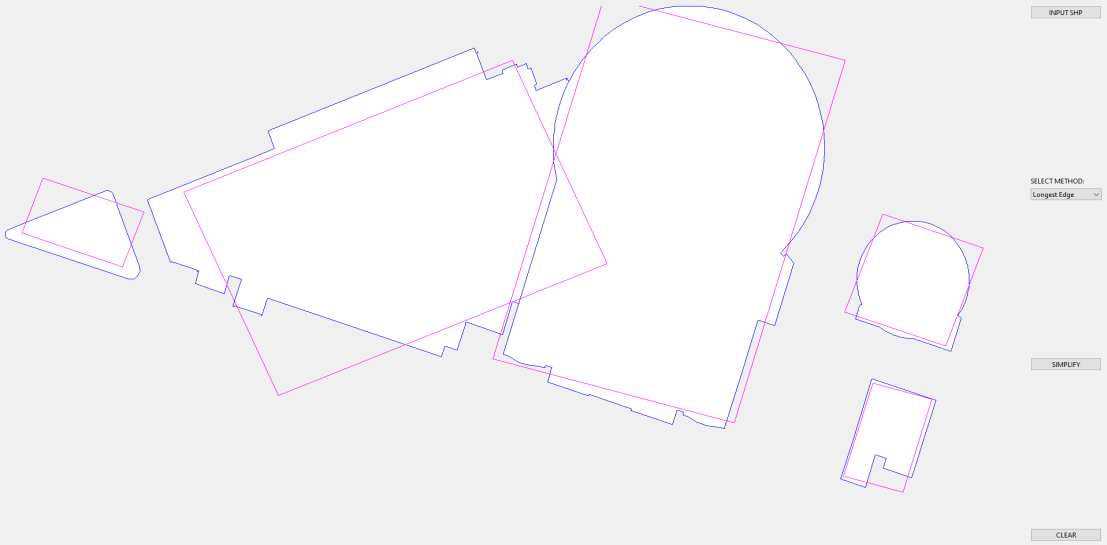
Pri treťom datasete bolo cieľom demonštrovať chovanie algoritmov pri budovách zaobleného tvaru (napr. štadióny). Výsledky pre jednotlivé algoritmy sú na Obr. 10, Obr. 11, Obr. 12, Obr. 13.



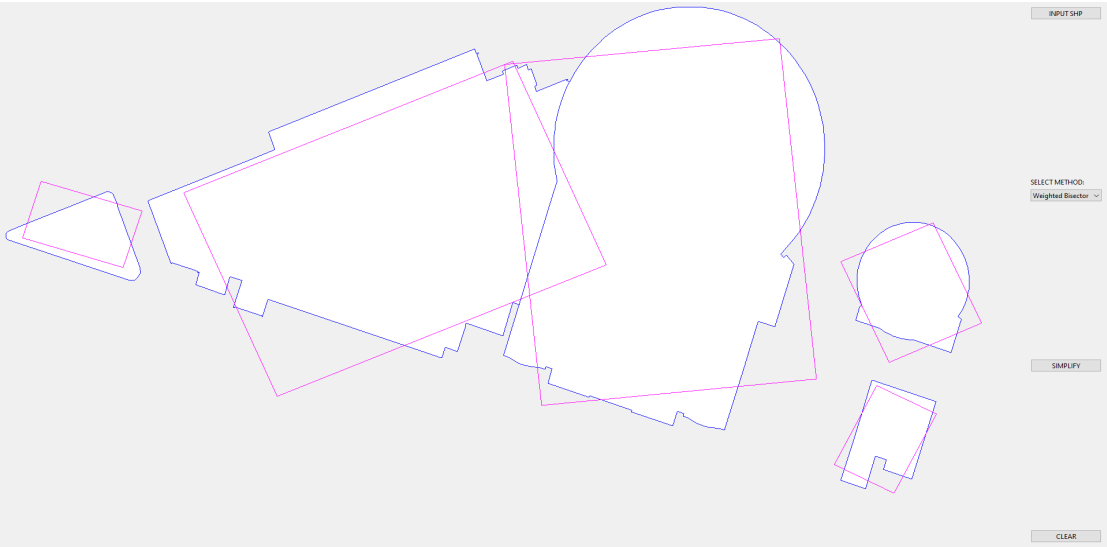
Obr. 10: Generalizácia datasetu Štadióny algoritmom *Minimum Area Enclosing Rectangle*.



Obr. 11: Generalizácia datasetu Štadióny algoritmom *Wall Average*.



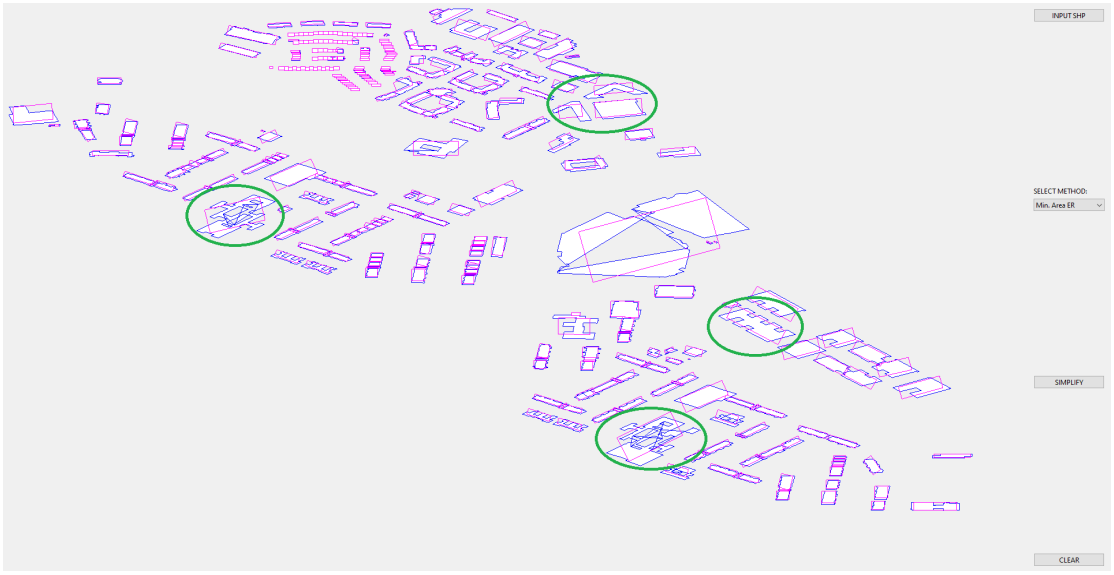
Obr. 12: Generalizácia datasetu Štadióny algoritmom *Longest Edge*.



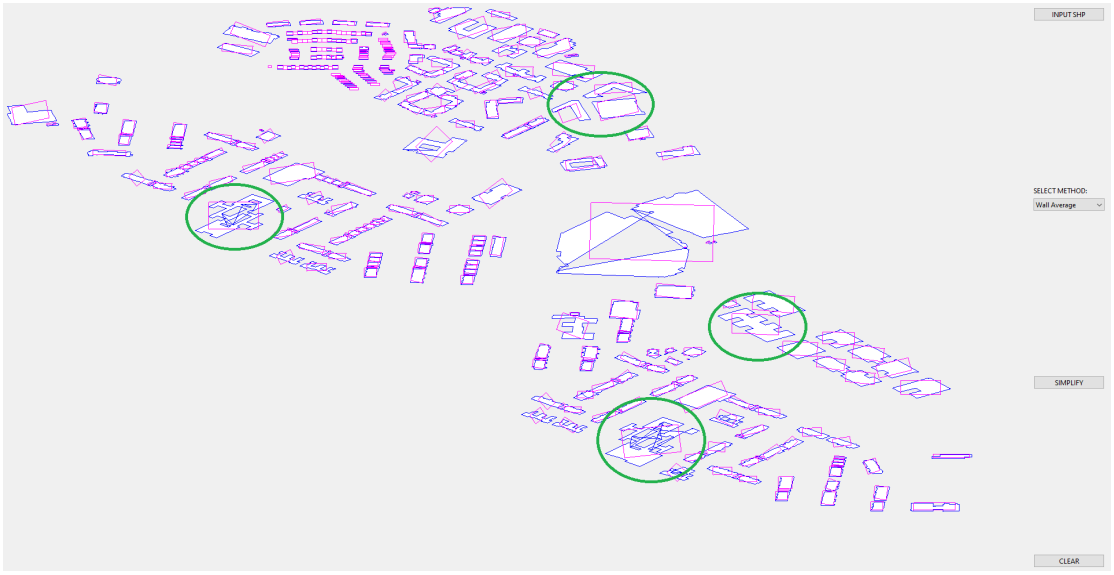
Obr. 13: Generalizácia datasetu Štadióny algoritmom *Weighted Bisector*.

Najlepšie výsledky dosiahol *Longest Edge*, čo sa dalo očakávať, pretože pri takýchto budovách skutočne najdlhšia hrana určuje jej orientáciu. Pre prvé dve budovy zľava (trojuholníkového tvaru) dopadli porovnateľne dobre *Wall Average* a *Weighted Bisector*, ktoré si ale nepodarili najlepšie so susedným štadiónom. Presne opačné závery platia pre *Minimum Area Enclosing Rectangle*.

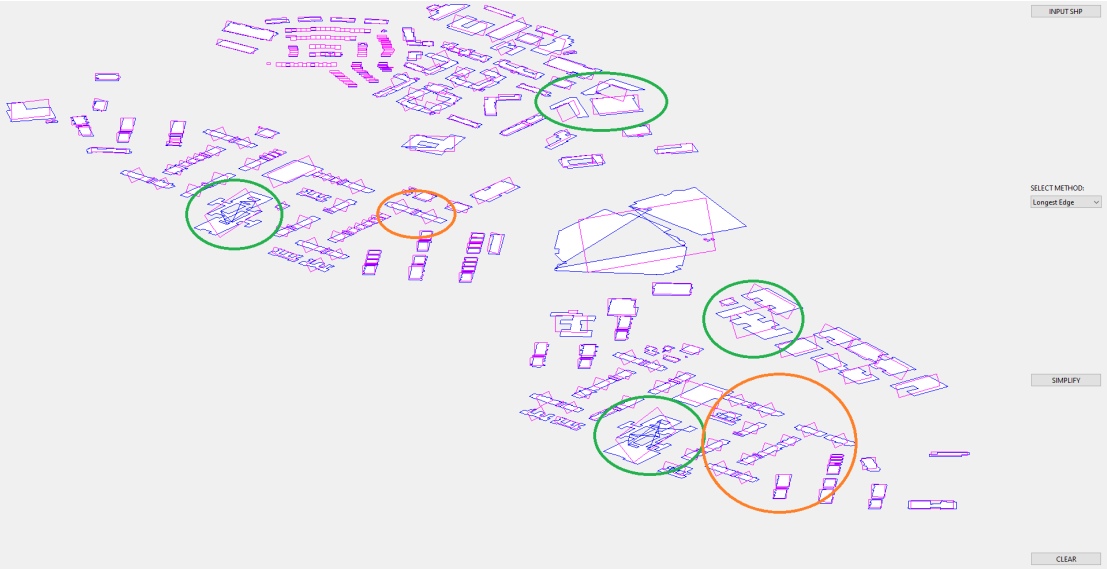
Štvrtý dataset obsahuje budovy v tvare L a dve budovy veľmi komplexného tvaru. Výsledky sú na Obr. 14, Obr. 15, Obr. 16, Obr. 17.



Obr. 14: Generalizácia datasetu Chodov algoritmom *Minimum Area Enclosing Rectangle*.



Obr. 15: Generalizácia datasetu Chodov algoritmom *Wall Average*.



Obr. 16: Generalizácia datasetu Chodov algoritmom *Longest Edge*.

Obr. 17: Generalizácia datasetu Chodov algoritmom *Weighted Bisector*.

Najlepšie výsledky pre tento dataset dosahuje *Minimum Area Enclosing Rectangle*. Druhý najkvalitnejší je *Weighted Bisector*, ktorý avšak jednu budovu negeneralizoval, pretože sa nepodarilo vypočítať jej uhlopriečky. Najhorším bol *Longest Edge*, ktorý si neporadil s budovami podlhovastého tvaru, pri ktorých najdlhšia hrana celkom jednoznačne určuje orientáciu budovy. Problémom môže byť spracovanie súradníc vstupných dát, ktoré v tomto prípade rozdelí najdlhšie hrany do krátkych segmentov, čo zásadne ovplyvní tento algoritmus.

10 Dokumentácia

Samotný program bol napísaný v jazyku Python 3.9 v SW PyCharm. Okrem externých modulov QT bol využitý aj modul `pyshp`, ktorý slúži na prečítanie SHP súboru. Program pozostáva z troch tried:

1. Trieda `MainForm`:
   Predstavuje užívateľské rozhranie pre aplikáciu, obsahuje šesť metód:

   Metódy `setupUi` a `retranslateUi` boli automaticky vygenerované prostredníctvom vývojového prostredia QT a prepisom QML do Pythonu.

   Metóda `input` inicializuje vloženie vstupných dát do programu. Zisťuje šírku a výšku zobrazovacej plochy aplikácie z dôvodu nutného preškálovania súradníc vstupných dát do súradníc zobrazovacej plochy aplikácie. Hodnoty šírky a výšky sme redukovali o hodnotu 100, aby sme dosiahli offsetu od okrajov okna aplikácie. Následne sa tieto redukované hodnoty predávajú metóde `setPath`.

   Metóda `simplifyClick` sa týka tlačidla *SIMPLIFY*, ktoré spúšťa proces generalizácie budov pomocou navoleného algoritmu v ComboBoxe. V metóde `simplifyClick` sa preto testuje, ktorý z algoritmov je aktuálne zvolený a má byť spustený (testuje sa hodnota aktuálneho indexu pomocou funkcie self.comboBox.currentIndex()). Index 0 odkazuje na algoritmus Minimum Area Enclosing Rectangle, index 1 na Wall Average, index 2 na Longest Edge a index 3 na Weighted Bisector. V metóde `simplifyClick` sú takisto inicializované pomocné premenné pre uloženie výsledkov.
   Následne sa iteruje cez každý element listu polygónov. V poslednom kroku prekresluje zobrazovaciu plochu.

   Metóda `clearClick` sa týka tlačidla *CLEAR*, ktoré zabezpečuje odstránenie akýchkoľvek útvarov na Widgete aplikácie.

2. Trieda `Draw`:
   Je zodpovedná za grafickú stránku aplikácie a zobrazovacej plochy. Trieda obsahuje inicializátor a sedem metód:

   `Inicializátor` má dva pozičné argumenty a inicializuje premenné pre túto triedu. Inicializovaný je prázdny list polygónov a takisto aj prázdny list výsledkov, do ktorého sa pri každej iterácií z metódy `analyze` ukladajú výsledky pre každý jeden polygón.
   Metóda `setPath` má dva argumenty a to šírku a výšku zobrazovacej plochy aplikácie. Do premennej ukladá cestu k SHP súboru, ktorý predstavuje vstupné dáta, ktorý užívateľ vyberie

z pop-up okna. Na prečítanie SHP súboru sa využíva externý modul `pyshp`.
V ďalšom kroku extrahuje súradnice zo SHP súboru a to tak, aby sa zachovala informácia, ktoré súradnice (vrcholy) tvoria ktorý polygón.
Z dôvodu rozličného súradnicového systému vstupných dát a zobrazovacej plochy aplikácie vstupné dáta navyše preškálováva. V prvom rade je bod najbližší počiatku súradnicového systému zobrazovacej plochy presunutý do jeho počiatku. Následne sa vykoná normalizácia do intervalu $< 0, 1 >$ a hodnoty sú vynásobené šírkou a výškou zobrazovacej plochy. Navyše Y súradnice preklápa cez osu X.
Metóda vracia list polygónov.

Metódy `getPolygon` vracia list polygónov.

Metóda `setResult` pridáva do prázdneho listu výsledných generalizácií výsledky pre jednotlivé polygony budov.

Metóda `paintEvent` má argument `QPaintEvent` z modulu `QtGui`. Dochádza tu ku začiatku vizualizácie na zobrazovacej ploche. Pomocou iterácií listu budov a listu výsledných generalizácií budov sa vykresľuje každý jeden polygon. Pre budovy ako aj pre ich generalizácie sa nastavuje rozličná farba.

3. Trieda `Algorithms`:
   Trieda obsahuje matematické metódy, pomocou ktorých sú realizované jednotlivé algoritmy analýzy, ktoré sú taktiež zapísané v tejto triede.

   Metóda `getPointAndLinePosititon` má na vstupe tri argumenty, všetky predstavujú body v dátovom type `QPoint`. Slúži na určenie polohy analyzovaného bodu a priamky.
   V prvom kroku je definovaná prijateľná odchýlka `epsilon`, nasleduje výpočet zložiek dvoch vektorov a výpočet vektorového súčinu pomoocu determinantu a testovanie podmienok, do ktorých vstupuje hodnota determinantu a podľa ktorej sa určí pozícia bodu voči priamke.
   Metóda vracia hodnotu 1 ak je bod v ľavej polrovine, hodnotu 0 ak je bod v pravej polrovine, v prípade kolinearity vráti hodnotu -1.

   Metóda `get2LinesAngle` má na vstupe štyri body, opäť všetky v dátovom type `QPoint`. Slúži na výpočet uhlu dvoch priamok. Opäť sú spočítané dva vektory, ich skalárny súčin a ich norma.
   Figuruje v nej podmienka, ktorá rieši singularitu analyzovaného bodu na vrchole polygónu, a to na základe nulovej hodnoty jednej z noriem vektorov. Metóda v tomto prípade vracia hodnotu 0.
   Druhá podmienka slúži na vyhnutie sa prípadu, kedy arccos uhlu dvoch priamok sa počíta

z hodnoty, ktorá je väčšia ako 1. V takomto prípade metóda vracia výslednú hodnotu z $|\arccos(1)|$.
Mimo spomenutých prípadov funkcia vracia hodnotu uhlu dvoch priamok v kladných hodnotách.

Metóda `jarvisScanCH` má na vstupe argument pol dátového typu `QPolygon` a argument length dátového typu `int`, ktorý predstavuje dĺžku aktuálne spracovaného polygonu. Je v nej implementovaný *Jarvis Scan algoritmus*, ktorý je detailne popísaný v kapitole 4.1. Metóda vracia konvexnú obálku vstupu (budova) v dátovom type `QPolygon`.

Metóda `grahamScanCH` má na vstupe rovnako ako predošlá metóda argument pol dátového typu `QPolygon` a rovnako ako predošlá metóda vracia konvexnú obálku vstupného polygonu v tvare `QPolygon`. Rozdiel je v implementovanom algoritme (namiesto Jarvis Scan je implementovaný Graham Scan, detailne popísaný v kapitole 4.2.

Metóda `rotate` má na vstupe dva argumenty a to pol dátového typu `QPolygon` a angle dátového typu `float`. Pomocou tejto funkcie je vstupný polygon rotovaný o uhol angle. Výstupom je takto otočený polygon dátového typu `QPolygon`.

Metóda `minMaxBox` má na vstupe pol dátového typu `QPolygon`. Funkcia vracia dva výstupy a to, obsah plochy výstupného polygonu Minimum Bounding Rectangle (typ `float`) a samotný `QPolygon`. Minimum Bounding Rectangle je popísaný v kapitole 4.3.

Metóda `getArea` má na vstupe argument pol dátového typu `QPolygon`. Vo vnútri metódy je vypočítaný obsah nekonvexného vstupného polygonu. Tento obsah je vracaný v datovom type `float`.

Metóda `resizeRectangle` má na vstupe dva argumenty a to er a pol, oba dátového typu `QPolygon`. Er je v tejto úlohe chápaný ako Minimum Bounding Rectangle. Metóda vypočíta podiel obsahov oboch polygonov a prvý vstup (polygon er) zmenšuje/zväčšuje tak, aby jeho plocha bola rovnaká ako plocha polygonu pol. Takto modifikovaný polygon er je následne funkciou vracaný.

Metóda `getLength` má na vstupe dva argumenty p1 a p2 rovnakého dátového typu `QPoint`. Metóda vracia dĺžku úsečky (v dátovom type `float`), ktorá je definovaná bodmi p1 a p2.

Metóda `minAreaEnclosingRectangle` má na vstupe `QPolygon` pol, ďalej `int` ch a takisto lenght `int`. Integer ch predáva funkcii momentálny index druhého comboboxu a teda určuje, ktorý z algoritmov pre tvorbu konvexnej obálky bude aplikovaný. Argument lenght predáva funkcii dĺžku aktuálneho polygonu. V metóde je implementovaný algoritmus Minimum Area

Enclosing Rectangle, popísaný v kapitole 4. Vracia výslednú generalizáciu vstupného polygonu v tvare `QPolygon`.

Metóda `wallAverage` má na vstupe `QPolygon` pol. V metóde je implementovaný algoritmus Wall Average, popísaný v kapitole 5. Vracia výslednú generalizáciu vstupného polygonu v tvare `QPolygon`.

Metóda `longestEdge` má na vstupe `QPolygon` pol. V metóde je implementovaný algoritmus Longest Edge, popísaný v kapitole 6. Vracia výslednú generalizáciu vstupného polygonu v tvare `QPolygon`.

Metóda `weightedBisector` má na vstupe `QPolygon` pol. V metóde je implementovaný algoritmus Weighted Bisector, popísaný v kapitole 7. Vracia výslednú generalizáciu vstupného polygonu v tvare `QPolygon`.

11 Záver

Vytvorená aplikácia a program úspešne generalizuje budovy algoritmami *Minimum Area Enclosing Rectangle*, *Wall Average*, *Longest Edge* a *Weighted Bisector*. Takisto sú v nej implementované metódy *Jarvis Scan* a *Graham Scan* pre tvorbu konvexnej obálky, ktorá sa využíva v prvom menovanom algoritme.

Problémom aplikácie je správne prečítanie polygónovej vrstvy v prípade, ak obsahuje menší polygón v polygóne (napr. prípad enkláv), prípadne multipart polygóny. V takomto prípade vzniká úsečka medzi posledným bodom prvého polygónu a prvým bodom nasledujúceho polygónu enklávy. Táto úsečka negatívne ovplyvňuje výsledky.

Čo sa týka preškálovania vstupných dát kvôli ich vizualizácii, tak dochádza k jemnej deformácii tvaru územia a to kvôli roztiahnutiu dát v oboch smeroch do maximálneho možného intervalu.

12 Použité zdroje

Prednášky z predmetu *Algoritmy počítačové kartografie*.

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

DROPPOVÁ, V. 2011: The tools of automated generalization and building generalization in an ArcGIS environment. Slovak Journal of Civil Engineering, 19(1), s. 1–7.

DUCHENE, C. et al. 2003: Quantitative and qualitative description of building orientation.

LI, Z. et al. 2004: Automated building generalization based on urban morphology and Gestalt theory. International Journal of Geographical Information Science, 18, 5, s. 513-534.