Univerzita Karlova
Přírodovědecká fakulta

ALGORITMY POČÍTAČOVÉ KARTOGRAFIE

Generalizace budov

Martina Pavlová, Martin Šíma, Ludmila Vítková
1 N-GKDPZ
Praha 2024

Zadání

Úloha č. 2: Generalizace budov

Vstup: množina budov $B = \{B_i\}_{i=0}^{n}$, *budova* $B_i = \{P_{i,j}\}_{j=1}^{m}$.
Výstup: $G(B_i)$
Ze souboru načtěte vstupní data představovaná lomovými body budov a proveďte generalizaci budov do úrovně detailu LOD0. Pro tyto účely použijte vhodnou datovou sadu, např. ZABAGED, testování proveďte nad třemi datovými sadami (historické centrum města, intravilán – sídliště, intravilán – izolovaná zástavba)
Pro každou budovu určete její hlavní směry metodami

- Minimum Area Enclosing Rectangle
- PCA

U první metody použijte některý z algoritmů pro konstrukci konvexní obálky. Budovu při generalizaci do úrovně LOD0 nahraďte obdélníkem orientovaným v obou hlavnách směrech, se středem v těžišti budovy, jeho plocha bude stejná jako plocha budovy. Výsledky generalizace vhodně vizualizujte.
Otestujte a porovnejte efektivitu obou metod s využitím hodnotících kritérií. Pokuste se rozhodnout, 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 PCA.	15b
Generalizace budov metodou Longest Edge	*+5b*
Generalizace budov metodou Wall Average	*+8b*
Generalizace budov metodou Weighted Bisector	*+10b*
Implementace další metody konstrukce konvexní obálky	*+5b*
Ošetření singulárních případů při generování konvexní obálky	*+2b*
Max celkem:	**45b**

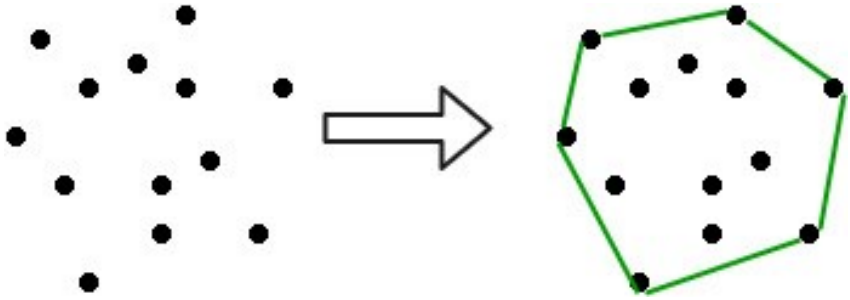
Řešeny byly všechny bonusové úlohy.

1 Popis a rozbor problému

Generalizace budov je proces, při kterém dochází k redukci detailů a zjednodušení geometrických tvarů budov tak, aby byly vhodné pro zobrazování na mapách různých měřítek. Je nezbytným procesem pro vytváření map, jelikož díky ní lze zachovat přehlednost a čitelnost map. Metody kartografické generalizace se dají dělit do několika skupin, kterými jsou například generalizace výběrem, geometrická generalizace, generalizace reklasifikací, generalizace ploch či generalizace atributů.

1.1 Konvexní obálka

Pokud definujeme množinu n bodů S v rovině, pak je konvexní obálkou nejmenší konvexní mnohoúhelník P, který obsahuje množinu bodů S. Zároveň množinu bodů S můžeme označit jako konvexní pouze v případě, kdy spojnice dvou libovolných prvků leží zcela uvnitř dané množiny bodů. Na obrázku 1 se nachází grafické znázornění konvexní obálky.



Obrázek 1: Konvexní obálka (Sharma 2024a)

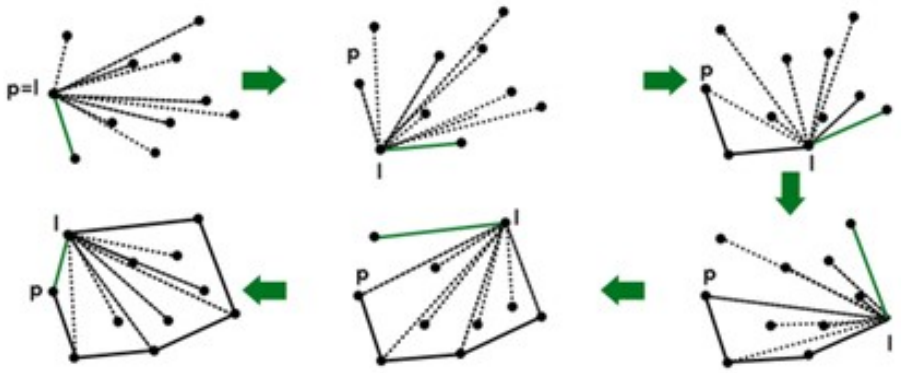
Konvexní obálky jsou používány mnoha různými algoritmy, jelikož je díky nim možné s velkou rychlostí najít tzv. bounding box nebo min-max box. Jedná se o objekt, který představuje nejzákladnější generalizaci polygonu a je to takový obdélník, který je kolmý na osy souřadnicového systému a zároveň minimálně ohraničuje daný polygon. Konvexní obálky mohou být dále použity například pro detekci kolizí, k analýze tvarů či statistické analýze.

Existuje více různých způsobů, jak konvexní obálku zkonstruovat. V této práci byly implementovány dva algoritmy, a to Jarvis Scan a Graham Scan.

1.1.1 Jarvis Scan

Jarvis Scan neboli Gift Wrapping Algorithm je jednoduchým, snadno implementovatelným a také jedním z nejvíce používaných algoritmů pro tvorbu konvexní obálky v rovině. Tento algoritmus předpokládá, že se v dané množině bodů S nenachází tři kolineární body. Tento algoritmus funguje tak, že opakovaně hledá největší úhel ω mezi poslední hranou konvexní obálky a následující hranou. Poslední hrana je tvořena body p_{j-1}, p_j a následující hrana je tvořena body p_j, p_{j+1}. Bod p_{j+1} je hledán pouze mezi body, které ještě nejsou součástí konvexní obálky. Prvním krokem při implementaci tohoto algoritmu je nalezení počátečního bodu (pivot), přičemž tento bod se automaticky stane součástí konvexní obálky. Algoritmus skončí v případě, že se aktuální bod konvexní obálky rovná bodu počátečnímu a dojde tak

k vytvoření celé konvexní obálky. Tento algoritmus se nehodí pro velké datasety, jelikož jeho časová složitost je $O(n^2)$. Na obrázku 2 je vidět grafické znázornění průběhu tohoto algoritmu.



Obrázek 2: Jarvis Scan (Greyrat 2022)

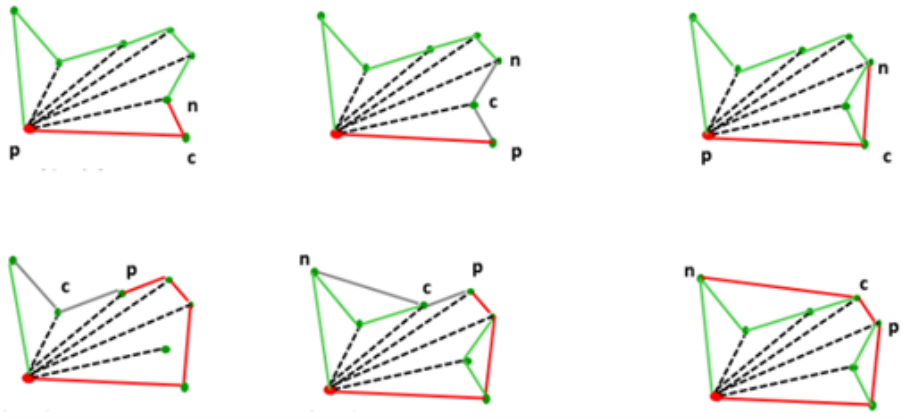
Implementace algoritmu Jarvis Scan

Algorithm 1 *Jarvis Scan*

```
Inicializuj prázdnou množinu ch
Najdi pivota q, který má nejmenší y-ovou souřadnici
Najdi pivota s, který má nejmenší x-ovou souřadnici
Inicializuj poslední dva body konvexní obálky qj a qj_1
Přidej pivota do konvexní obálky
While True
    Inicializuj proměnné omegaMax a indexMax
    Projdi všechny body polygonu
        Pokud bod polygonu není totožný s aktuálním bodem konvexní obálky
            Vypočítej úhel omega
            Pokud je omega větší než omegaMax
            Aktualizuj úhel omegaMax a indexMax
    Přidej bod do konvexní obálky ch
    Pokud je bod pivotem
        Skonči
    Aktualizuj poslední segment konvexní obálky
Vrať konvexní obálku
```

1.1.2 Graham Scan

Dalším algoritmem, který lze použít pro sestrojení konvexní obálky je Graham Scan, který byl jedním z prvních takových algoritmů. Při aplikaci tohoto algoritmu dochází k využití kritéria pravotočivosti (příp. levotočivosti), kde se posuzuje úhel ω_i. Tento úhel je mezi dvěma úsečkami, které procházejí body p_{j-1}, p_j a p_{j+1}. Prvním krokem je stejně jako u předchozího algoritmu nalezení počátečního bodu q (pivot). Tento bod se většinou hledá tak, aby jeho y-ová souřadnice byla nejmenší. Vzhledem k tomu, že může existovat více bodů se stejnou y-ovou souřadnicí, vybírá se bod, který z nich má nejmenší x-ovou souřadnici. Dalším krokem je setřídění všech bodů vzestupně na základě jejich hodnoty úhlu ω_i mezi osou x a spojnicí bodů q a p_j. Pokud nastane situace, kdy jsou úhly ω_i u více bodů stejné, dojde k seřazení bodů podle jejich euklidovské vzdálenosti od bodu q.



Obrázek 3: Graham Scan (Sharma 2024b)

Spojením všech setříděných bodů se vytvoří nekonvexní mnohoúhelní, který se následně převede na mnohoúhelník konvexní. Tento proces lze vidět na obrázku 3. Jeho časová složitost je $O(n \cdot log(n))$ a lze tedy použít i pro rozsáhlé datasety.

Implementace algoritmu Graham Scan

Algorithm 2 *Graham Scan*

Inicializuj prázdný polygon
Najdi bod q, který má nejmenší y-ovou souřadnici
Vytvoř prázdný polygon *tmp*
Projdi každý bod v původním polygonu
    Pokud aktuální bod není stejný jako pivot
        Přidej ho do *tmp*
Aktualizuj polygon
Seřaď body podle úhlu, který svírají s osou x vzhledem k pivotu
Vytvoř prázdný zásobník S
Přidej pivot q do S
Přidej první bod ze seznamu *sorted_points* do S
Urči počet bodů v polygonu *pol*
Inicializuj proměnnou j na hodnotu 2
Projdi všechny seřazené body od druhého bodu
    Přiřaď aktuální bod pj ze seznamu *sorted_points*
    Pokud bod leží vlevo
        Přidej ho do pj
        Inkrementuj proměnnou j o 1
    Jinak odstraň poslední bod
Pro každý bod v S
    Přidej bod do konvexní obálky
Vrať *ch*

1.1.3 Singulární případy

U obou výše zmíněných algoritmů pro tvorbu konvexní obálky se vychází z předpokladu, že počet vrcholů polygonu je větší než 2. Pokud by byl počet bodů menší než 2 nejednalo by se o polygon a nemohlo by tím pádem dojít k vytvoření konvexní obálky. Ve vlastním kódu je tedy podmínka, že polygon nesmí mít méně než 3 body. Dále je také nutné, aby dvě úsečky, mezi kterými se měří úhel nebyly nulové. Při

výpočtu úhlu mezi třemi body (qj, $qj1$ a p) se, mimo jiné, počítá také velikost daných vektorů (v, u), které poté figurují ve jmenovateli:

$$\cos\varphi = \frac{v \cdot u}{\|v\|\|u\|}$$

Dále každý výpočet úhlu probíhá pouze pro body, které se nerovnají p_j. Touto podmínkou je zaručeno, že úhel bude počítán mezi třemi unikátními body, a tudíž vektory mají nenulovou velikost.

1.2 Detekce hlavních směrů budov

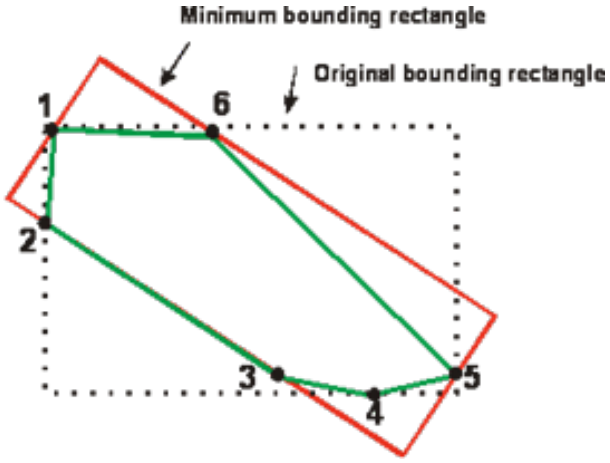
Abychom mohli jakoukoliv budovu generalizovat, je potřeba najít její hlavní směry. Tímto dojde k zachování orientace původního objektu a také návaznosti na ostatní prvky v mapě. Předejde se tím tak například tomu, aby budova přesahovala okolní liniové prvky (silnice, potok). Pro nalezení hlavních směrů budov existuje několik různých algoritmů. V této práci došlo k použití algoritmů Minimum Area Enclosing Rectangle, Principal Component Analysis, Longest Edge, Wall Average a Weighted Bisector. Všechny tyto algoritmy jsou popsány v následujících odstavcích.

1.2.1 Minimum Area Enclosing Rectangle

Minimum Area Enclosing Rectangle (MAER) neboli také Minimum Area Bounding Rectangle funguje na principu nalezení takového obdélníku, jehož plocha je minimální. Grafické znázornění tohoto algoritmu lze vidět na obrázku 4. Na počátku je definována množina n bodů S v rovině. Dále proběhne zpracování této množiny do konvexní obálky a následně je vytvořen obdélník, který opíše množinu S. Výsledný obdélník má alespoň jednu stranu kolineární s hranou konvexní obálky.

Tento algoritmus prochází postupně všechny hrany konvexní obálky, přičemž pro každou z nich hledá min-max box, který má hranu rovnoběžnou s aktuální hranou konvexní obálky. Ze všech vzniklých obdélníků se následně vybere ten, jehož plocha je minimální.

Algoritmus řeší problém v čase $O(n \cdot log(n))$ a jeho alternativou je algoritmus *Rotating Calipers*, který má časovou složitost $O(n)$.



Obrázek 4: Minimum Area Enclosing Rectangle (Scientificlib.com 2024)

Implementace algoritmu Minimum Area Enclosing Rectangle

Algorithm 3 *Minimum Area Enclosing Rectangle*

```
Pokud je k dispozici méně než 3 body
    Vrať prázdný polygon
Vytvoř konvexní obálku
Inicializuj proměnnou mmb_min
Inicializuj proměnnou area_min
Inicializuj proměnnou sigma_min
Učti počet bodů v konvexní obálce
Projdi všechno body konvexní obálky
    Vypočítej rozdíl souřadnic mezi aktuálním bodem a následujícím bodem v konvexní obálce
    Vypočítej úhel mezi těmito dvěma body
    Otoč konvexní obálku o úhel −sigma
    Vypočítej minimální obdélník
    Vypočítej plochu minimálního obdélníku
    Pokud je plocha aktuálního obdélníku menší než minimální plocha
        Aktualizuj minimální obdélník
        Aktualizuj minimální plochu
        Aktualizuj úhel sigma
Otoč minimální obdélník o úhel sigma_min
Uprav velikost obdélníku
Vrať mmb_res
```

1.2.2 Principal Component Analysis (PCA)

Algoritmus Principal Component Analysis funguje na principu, nalezení hlavních směrů pomocí kovarianční matice C a následným singulárním rozkladem. Kovarianční matici vypadá následovně:

$$C = \begin{bmatrix} C(A,A) & C(A,B) \\ C(B,A) & C(B,B) \end{bmatrix}$$

Pro singulární rozklad (SVD) platí $C = U\Sigma V^T$.

$$\begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix} = \begin{bmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \end{bmatrix} \begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{bmatrix} \begin{bmatrix} (v_{11} & v_{12} \\ v_2 1 & v_{22}) \end{bmatrix}^T$$

Úhel, o který budeme polygon následně otáčet se vypočítá jako:

$$\sigma = atan2(v_{12}, v_{11})$$

Tato metoda generalizace budov je snadno zobecnitelná i do vyšších dimenzí. Pro pravidelné objekty, jako jsou například čtverce či kruhy, má tento algoritmus nízkou efektivitu, jelikož vrcholy tím pádem leží na kružnici a neexistuje tak dominantní směr. Jedná se o poměrně citlivou metodu.

Implementace algoritmu PCA

Algorithm 4 *Principal Component Analysis*

```
Pokud je k dispozici méně než 3 body
    Vrať prázdný polygon
Vytvoř list souřadnic
Projdi všechny body polygonu
    Přidej souřadnice aktuálního bodu do seznamu
Vytvoř numpy pole
Vypočítej kovarianční matici souřadnic bodů polygonu
Proveď singulární rozklad kovarianční matice
Vypočítej úhel pro převrácen polygonu
Otoč polygon o úhel −sigma
Vytvoř minimální obdélník pro daný polygon
Otoč minimální obdélník o úhel sigma
Uprav velikost obdélníku
Vrať er_r
```

1.2.3 Longest Edge

Algoritmus Longest Edge je jednoduchou metodou generalizace budov, která funguje na principu, kdy je hlavní směr budovy přestavován nejdelší stranou dané budovy. Druhý hlavní je směr je na něj pak kolmý. Výsledný polygon je tedy natočený podél nejdelší strany budovy a zároveň je jeho delší strana rovnoběžná s touto nejdelší stranou budovy. Tento algoritmus nedosahuje příliš dobrých výsledků, jelikož nejdelší strana nemusí nutně reprezentovat hlavní směr budovy.

Implementace algoritmu Longest Edge

Algorithm 5 *Longest Edge*

```
Urči počet bodů v polygonu
Inicializuj proměnnou longest_edge
Inicializuj proměnnou angle
Projdi všechny hrany polygonu
    Vypočítej délku aktuální hrany
    Pokud je edge_length větší než longest_edge
        Aktualizuj proměnnou longest_edge
        Vypočítej sklon hrany
Otoč polygon o úhel −sigma
Vytvoř minimální obdélník pro daný polygon
Otoč minimální obdélník o úhel sigma
Uprav velikost obdélníku
Vrať er_r
```

1.2.4 Wall Average

Algoritmus Wall Average patří k těm komplexnějším. Je založen na principu, kdy je pro každou hranu v daném polygonu spočítán zaokrouhlený podíl k_i:

$$k_i \ = \ \left\lfloor \frac{\Delta\sigma_i}{\frac{\pi}{2}} \right\rceil$$

Proměnná $\Delta\sigma_i$ značí rozdíl úhlů σ_i a σ', kde σ_i je směrnice aktuálně zpracované hrany a σ' je náhodně zvolená směrnice jedné z hran daného polygonu. Tento rozdíl vypadá tedy následovně:

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

Po vypočtení k_i dojde k vypočítání zbytků r_i a odchylky od $0 \pm \ k\pi$, respektive $\frac{\pi}{2} \pm \ k\pi$:

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

Hlavní směr natočení budovy se následně vypočítá pomocí vztahu:

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

Proměnná s_i značí délku hrany i.

Implementace algoritmu Wall Average

Algorithm 6 *Wall Average*

```
Urči počet bodů v polygonu
Vypočítej rozdíl souřadnic mezi prvním a druhým bodem polygonu
Vypočítej směr hrany
Inicializuj proměnnou av_r
Projdi všechny hrany polygonu
    Vypočítej rozdíl souřadnic mezi aktuálním a následujícím bodem polygonu
    Vypočítej směr aktuální hrany polygonu
    Vypočítej rozdíl směru mezi aktuální hranou a první hranou
    Pokud je dir_dif větší než 0
        Přičti 2π
    Vypočítej počet čtvrtin otočení
    Vypočítej zbytek po odečtení počtu čtvrtin
    Aktualizuj průměrný zbytek
Vypočítej průměrný zbytek
Vypočítej průměrný úhel rotace
Otoč polygon o úhel −sigma_av
Vytvoř minimální obdélník pro daný polygon
Otoč minimální obdélník o úhel sigma_av
Uprav velikost obdélníku
Vrať er_r
```

1.2.5 Weighted Bisector

Algoritmus Weighted Bisector používá pro detekci hlavních směrů dvě nejdelší úhlopříčky, které se v daném polygonu nachází. Jedná se o úhlopříčky, které neprotínají ani jednu z hran polygonu. Z bodů $P_1 = [x_1,\ y_1]$ a $P_2 = [x_2,\ y_2]$ se vytvoří úsečka P_1P_2 a z bodů $P_3 = [x_3,\ y_3]$ a $P_4 = [x_4,\ y_4]$ vznikne úsečka P_3P_4. Pro tyto body se následně provede čtyřikrát analýza koncového bodu úsečky vzhledem ke druhé úsečce, aby se zjistilo, zda úsečka prochází či neprochází hranou polygonu. Dochází k výpočtu determinantu:

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

$$t_3 = \begin{vmatrix} x_4 - x_3 & y_4 - y_3 \\ x_1 - x_3 & y_1 - y_3 \end{vmatrix} \quad t_4 = \begin{vmatrix} x_4 - x_3 & y_4 - y_3 \\ x_2 - x_3 & y_2 - y_3 \end{vmatrix}$$

Průsečík neexistuje buď pokud mají t_1 a t_2 stejné znaménko anebo pokud t_3 a t_4 mají stejné znaménko. Znamená to totiž, že se oba body nacházejí ve stejné polorovině vůči druhé úsečce. Úsečka je tím pádem úhlopříčkou, přičemž následně dojde k výběru dvou nejdelších úhlopříček a je pro ně spočítána jejich směrnice. Hlavní směr budovy σ je poté určen jako vážený průměr směrnic nejdelších úhlopříček, kde jsou váhami délky uhlopříček. Vzorec pro tento výpočet vypadá tedy takto:

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

Implementace algoritmu Weighted Bisector

Algorithm 7 *Weighted Bisector*

```
1: Pokud je k dispozici méně než 3 body
2:     Vrať prázdný polygon
3: Najdi dvě nejdelší diagonály polygonu
4: Vypočítej sklon první a druhé diagonály
5: Vypočítej vážený průměr úhlů diagonál sigma
6: Otoč polygon o úhel −sigma
7: Vytvoř minimální obdélník pro daný polygon
8: Otoč minimální obdélník o úhel sigma
9: Uprav velikost obdélníku
10: Vrať er_res
```

1.3 Hodnocení efektivity detekce hlavních směrů

Pro porovnání jednotlivých algoritmů na generalizaci budov došlo k definování hodnocení efektivity detekce hlavních směrů. Toto hodnocení se zpravidla provádí na jednotlivých objektech a jeho předpokladem je, že budovy jsou pravoúhlé. Základem je určení nejdelší strany generalizované budovy. Byla vypočítána střední hodnota úhlových odchylek jednotlivých segmentů, pro kterou platí vztah:

$$\Delta\sigma_1 \;=\; \frac{\pi}{2n}\sum_{i=1}^{n}[r_i - r]$$

kde

$$r_i \;=\; (k_i - \lfloor k_i \rfloor)\frac{\pi}{2}, \qquad k_i = \frac{2\sigma_i}{\pi}$$

Druhým výpočtem je střední hodnota čtverců úhlových odchylek jednotlivých segmentů:

$$\Delta\sigma_2 \;=\; \frac{\pi}{2n}\sqrt{\sum_{i=1}^{n}(r_i - r)^2}$$

2 Struktura programu

Program se skládá ze sedmi souborů, kterými jsou *Data*, *Images*, *Algorithms*, *Building*, *Draw*, *Load* a *MainForm*.

Ve složce *Data* se nacházejí vstupná data, která se skládají z pěti různých souborů. Prvním z nich je soubor budov, které se nacházejí v centru města a jsou tedy velmi blízko sebe. Pro tento dataset bylo vybráno Staré Město v Praze. Druhým datasetem jsou budovy, které se nacházejí na sídlišti a mají tedy většinou pravidelný tvar. V tomto případě se jedná o sídliště Opatov. Třetí dataset obsahuje vilovou oblast Mydlářka, kde stojí budovy samostatně dá od sebe. Další dataset zobrazuje oblast Vinohrad a poslední dataset s názvem *Test* obsahuje pouze jeden polygon. Všechny tyto datasety jsou ve formátu JSON. Ve vstupních datech je klíč POLYGONS, který obsahuje seznam budov, kde je každá budova reprezentována seznamem bodů ve formátu [x, y]. Každá souřadnice je zadána jako desetinné číslo s devíti desetinnými místy. Jelikož se jedná o uzavřené polygony, je první a poslední bod u každé budovy totožný.

Ve složce *Images* se nachází celkem 15 ikon, které slouží k vytvoření aplikace. Těmito ikonami jsou *Jarvis_scan.png*, *Graham_scan.png*, *Validate.png*, *about.png*, *ch.png*, *clear.png*, *clear_ch.png*, *clear_er.png*, *exit.png*, *longestedge.png*, *maer.png*, *open_file.png*, *pca.png*, *wa.png*, *weightedbisector.png*.

V *Algorithms.py* je definována třída *Algorithms*, která obsahuje skoro dvacet metod. Metoda *errorWindow* slouží k zobrazení informačních, varovných a chybových zpráv pomocí dialogového okna. Pomocí *analyzePointLinePosition* dochází k určení polohy bodu p vzhledem k přímce určené body p_1 a p_2. Pokud se bod nachází vlevo, je vrácena hodnota 1. V jiném případě je vrácena hodnota 0. Další metodou je *getTwoLineAngle*, která vypočítá pomocí skalárního součinu úhel mezi dvěma liniemi definovanými čtyřmi body. Pro vytvoření konvexní obálky lze použít metodu *jarvisScan* nebo *grahamScan*. Při použití

metody implementující Jarvis Scan dojde k nalezení počátečního bodu a následně k vytvoření konvexní obálky na základě maximálního úhlu. Při použití metody Graham Scan dojde také k nalezení počátečního bodu, následně k seřazení bodů a z nich se vytvoří konvexní obálka. Pomocí metody *mmb* dochází k vytvoření min-max boxu, který obsahuje všechny body vstupního seznamu. Metoda *rotate* slouží k rotaci polygonu o úhel *sig*, metoda *getArea* slouží k výpočtu plochy polygonu a metoda *l2* vypočítá Eukleidovskou vzdálenost mezi dvěma body. V rámci *searchDiagonals* dochází k hledání nejdelších úhlopříček daného polygonu, pomocí *weightedMean* dochází k výpočtu váženého průměru dvou hodnot a *slope* slouží k výpočtu sklonu úhlopříčky, která je určena dvěma body. Metoda *resizeRectangle* slouží k přizpůsobení velikosti polygonu tak, aby se vešel do oblasti původní budovy. Dalších pět metod slouží k samotné generalizaci budov v závislosti na zvoleném algoritmu. Těmito metodami jsou *createMBR*, *createERPCA*, *longestEdge*, *wallAverage* a *weightedBisector*. Poslední metodou je *validation*, ve které dochází k hodnocení efektivity detekce hlavních směrů.

V *Building.py* je definována třída *Building*, která slouží k reprezentaci budovy. Dochází k určení počtu vrcholů v polygonu či k přidání vrcholu do polygonu. Slouží k nastavení zjednodušené verze budovy.

V *draw.py* se nachází třída *Draw*, ve které je definovaných celkem šest různých metod. Metoda *getBuildigns* vrací seznam budov uložených v proměnné *buildings*. Metoda *getJarvis* slouží k přepínání použití algoritmu Jarvis Scan pro vytvoření konvexní obálky. *clearData* slouží ke smazání dat v proměnné *buildings* a *clearmbr* slouží ke smazání polygonů generalizovaných budov. Metoda *mousePressEvent* je zavolána při stisknutí tlačítka myši, přičemž získává souřadnice a přidává nový bod budovy. Poslední metodou je *paintEvent*, která je volána v případě, kdy je potřeba překreslit plátno. Pro každou budovu se vykreslí původní polygon (černý obrys, žlutá výplň) a její zjednodušená verze (červený obrys).

V *Load.py* dochází k definování *load_buildings* a *transformBuildings*. Funkce *load_buildings* slouží k načtení dat budov ze souboru formátu JSON. Pro každý polygon se projdou všechny body a spočítají se minimální a maximální hodnoty x a y, aby bylo možné určit velikost oblasti, ve které se budovy nacházejí. Pokud při načítání dat dojde k nějaké chybě, objeví se odpovídající chybová hláška. Po načtení dat a určení rozměrů oblasti, ve které se budovy nacházejí, se spočítají posuny *dx* a *dy*, které umožňují umístit budovy do středu zobrazovací oblasti, a měřítko *scale*, které určuje, jak moc se budovy musí zmenšit/zvětšit, aby se vešly do zobrazovací oblasti. Nakonec se provede transformace souřadnic všech polygonů budov podle vypočtených hodnot měřítka a posunů pomocí funkce *transformBuildings*.

V *MainForm.py* je definována třída *Ui_MainWindow*, ve které je následně definováno několik metod. Metody *setupUI* a *retranslateUI* byly vygenerovány v prostředí Qt. Nově implementovanými byly metody *openClick*, *mbrClick*, *pcaClick*, *clearClick*, *clearAllClick*, *jarvisScan*, *grahamScan*, *longestEdge*, *wallAverage*, *weightedBisector* a *validation*. Zavolání metody *openClick* proběhne při kliknutí na možnost *Open* v menu. Dochází k otevření dialogového okna pro výběr souboru typu JSON a po výběru souboru dojde pomocí funkce *load_buildings* k načtení polygonů budov, které jsou následně vykresleny. Metoda *mbrClick* přijímá seznam budov a pro každou z nich vytvoří její ohraničující obdélník pomocí algoritmu Minimum Area Enclosing Rectangle. Metody *pcaClick*, *longestEdge*, *wallAverage* a *weightedBisector* fungují stejně jako předchozí metoda, avšak pro algoritmy Principal Component Analysis, Longest Edge, Wall Average a Weighted Bisector. Dalšími dvěma metodami jsou *clearClick* a *clearAllClick*. Pomocí

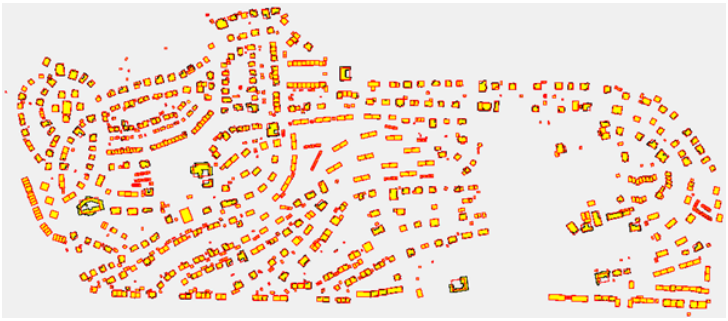
clearClick dojde k vymazání všech minimálních ohraničujících obdélníků a po kliknutí na *clearAllClick* dojde ke smazání všech dat. Obě tyto metody nakonec překreslí plátno. Metody *jarvisScan* a *grahamScan* slouží k vytvoření konvexní obálky na základě zvoleného algoritmu. Poslední metodou je *validation*, která provedení hodnocení efektivity detekce hlavních směrů. Výsledek se zobrazí v dialogovém okně.

3 Výsledky

V rámci této úlohy došlo k vytvoření uživatelského rozhraní s využitím frameworku QT, ve kterém je demonstrována funkčnost algoritmů Jarvis Scan a Graham Scan pro tvorbu konvexní obálky a algoritmů Minimum Area Enclosing Rectangle, Principal Component Analysis, Longest Edge, Wall Average a Weighted Bisector pro generalizaci polygonů budov. Toto grafické rozhraní aplikace vytvořené v prostředí Qt Designer lze vidět na obrázku 5 a dále bylo upravováno v prostředí programovacího jazyka Python. Po spuštění aplikace může uživatel otevřít soubor obsahující polygony budov.

Obrázek 5: Grafické rozhraní aplikace po načtení souboru s polygony budov

V rámci tohoto rozhraní lze zvolit, kterou metodou proběhne tvorba konvexní obálky i samotná generalizace budov. Po zvolení těchto parametrů proběhne analýza dat a dojde k vykreslení generalizovaných polygonů budov červeným obrysem, zatímco původní data mají žlutou výplň a černý obrys. Jak takový výsledek může vypadat lze vidět na obrázku 6.



Obrázek 6: Ukázka možného výsledku

Prvním analyzovaným datasetem bylo Staré Město v Praze. Jedná se o budovy v centru města, které jsou blízko u sebe a mají nepravidelné tvary. Výřez tohoto datasetu lze vidět na obrázku 7. Na obrázcích 8 až 12 jsou vidět výsledky jednotlivých algoritmů pro generalizaci polygonů budov. Při pohledu na tyto obrázky lze vidět, že nejhorší výsledek měl algoritmus Weighted Bisector, zatímco dobrý výsledek měly algoritmy Minimum Area Enclosing Rectangle a Wall Average.

Obrázek 7: Staré Město – původní budovy

Obrázek 8: Staré Město – výsledek PCA

Obrázek 9: Staré Město – výsledek LE

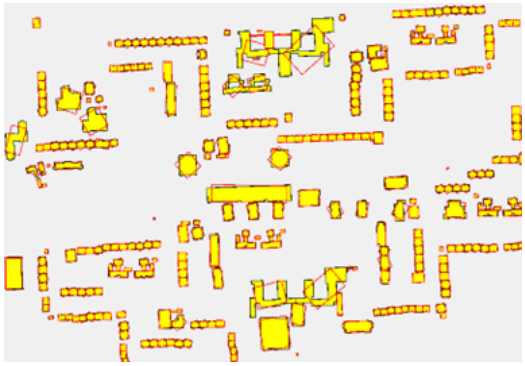
Obrázek 10: Staré Město – výsledek WA

Obrázek 11: Staré Město – výsledek MAER

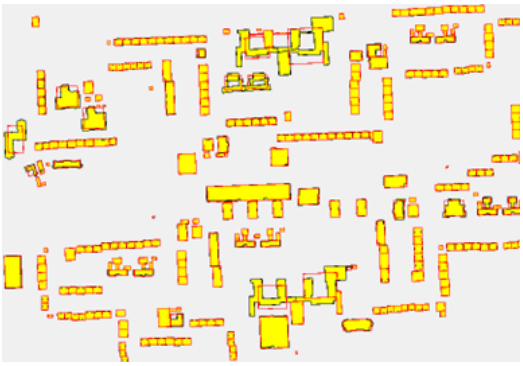
Obrázek 12: Staré Město – výsledek WB

Druhým analyzovaným datasetem bylo sídliště Opatov v Praze. Jedná se o budovy, které mají pravidelný tvar a jsou uspořádané do bloků. Výřez tohoto datasetu lze vidět na obrázku 13. Na obrázcích 14 až 18 jsou vidět výsledky jednotlivých algoritmů pro generalizaci polygonů budov. Při pohledu na tyto obrázky lze vidět, že nejhorší výsledek měl opět algoritmus Weighted Bisector, zatímco podobně dobrý výsledky měly algoritmy Longest Edge, Minimum Area Enclosing Rectangle a Wall Average.

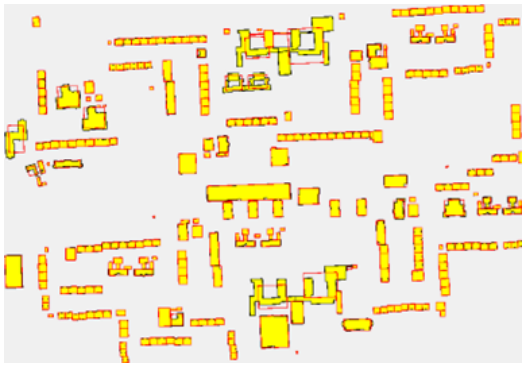
Obrázek 13: Opatov – původní budovy



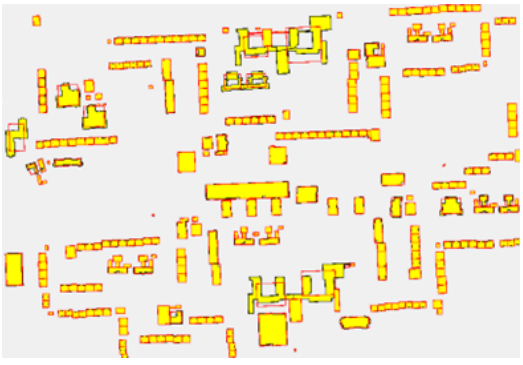
Obrázek 14: Opatov – výsledek PCA



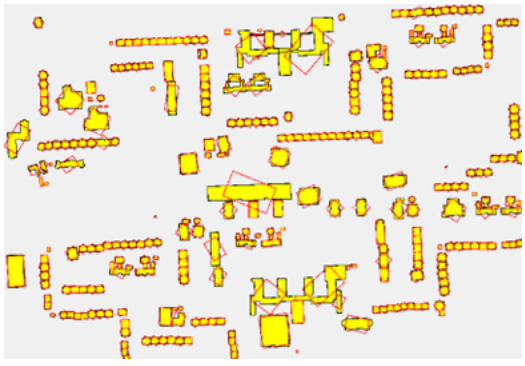
Obrázek 15: Opatov – výsledek LE



Obrázek 16: Opatov – výsledek WA



Obrázek 17: Opatov – výsledek MAER



Obrázek 18: Opatov – výsledek WB

Posledním analyzovaným datasetem byla Mydlářka v Praze. Jedná se o vilovou oblast, kde jsou samostatně stojící budovy umístěny daleko od sebe. Výřez tohoto datasetu lze vidět na obrázku 19. Na obrázcích 20 až 24 jsou vidět výsledky jednotlivých algoritmů pro generalizaci polygonů budov. Na první pohled z těchto obrázků nelze říct, který z algoritmů byl nejlepší či nejhorší.

Obrázek 19: Mydlářka – původní budovy

Obrázek 20: Mydlářka – výsledek PCA

Obrázek 21: Mydlářka – výsledek LE

Obrázek 22: Mydlářka – výsledek WA

Obrázek 23: Mydlářka – výsledek MAER

Obrázek 24: Mydlářka – výsledek WB

3.1 Hodnocení efektivity

Prvním výpočtem byla střední hodnota úhlových odchylek jednotlivých segmentů. Výsledky jednotlivých algoritmů lze vidět v tabulce 1, kde je uvedeno kolik procent budov mělo hodnotu menší než 10 °. Na posledním řádku této tabulky je vypočítána úspěšnost jednotlivých datasetů. Nejlepší výsledky měl dataset Mydlářka, kde jsou samostatně stojící domy pravidelného tvaru, které se nacházejí daleko od sebe. Největší úspěšnost měl algoritmus Minimum Area Enclosing Rectangle, který měl více než 73 % úspěšnost. Algoritmy Longest Edge a Wall Average měli přibližně jen o jedno procento horší úspěšnost. Nejhůře dopadl algoritmus Weighted Bisector, kde nebyla správně generalizována ani čtvrti všech budov.

Metoda	**Staré Město [%]**	**Opatov [%]**	**Mydlářka [%]**	**Celkem[%]**
MAER	52,01	83,00	84,45	**73,15**
PCA	26,95	30,31	35,00	**30,75**
Longest Edge	48,94	84,14	83,94	**72,34**
Wall Average	51,30	82,44	82,50	**72,08**
Weighted Bisector	22,46	15,86	29,57	**22,63**
Celková úspěšnost	**40,33**	**59,15**	**63,09**	

Tabulka 1: Střední hodnota úhlových odchylek jednotlivých segmentů

Druhým výpočtem je střední hodnota čtverců úhlových odchylek jednotlivých segmentů. Výsledky jednotlivých algoritmů lze vidět v tabulce 2, kde je uvedeno kolik procent budov mělo hodnotu menší než 10 °. Na posledním řádku této tabulky je vypočítána úspěšnost jednotlivých datasetů. Nejlepší výsledky měl opět dataset Mydlářka, přičemž jeho úspěšnost byla u všech algoritmů největší. Největší úspěšnost měl algoritmus Minimum Area Enclosing Rectangle, který měl více než 83 % úspěšnost. Algoritmy Longest Edge a Wall Average měli přibližně jen o jedno procento horší úspěšnost. Nejhůře dopadl algoritmus Weighted Bisector, kde mělo přes 40 % budov větší odchylku.

Metoda	**Staré Město [%]**	**Opatov [%]**	**Mydlářka [%]**	**Celkem[%]**
MAER	72,58	87,82	89,46	**83,29**
PCA	55,56	51,27	58,37	**50,07**
Longest Edge	69,98	88,95	89,29	**82,74**
Wall Average	70,21	88,10	88,87	**82,39**
Weighted Bisector	42,55	30,59	49,36	**40,83**
Celková úspěšnost	**62,18**	**69,35**	**75,07**	

Tabulka 2: Střední hodnota čtverců úhlových odchylek jednotlivých segmentů

4 Závěr

V rámci této úlohy došlo k přestavení a implementování algoritmů Jarvis Scan a Graham Scan pro tvorbu konvexní obálky. Dále byly pro generalizaci polygonů budov použity algoritmy Minimum Area Enclosing Rectangle, Principal Component Analysis, Longest Edge, Wall Average a Weighted Bisector. Jako vstupní data byly použity tři datasety ve formátu JSON obsahující budovy z centra města (Staré Město), ze sídliště (Opatov) a z vilové oblasti (Mydlářka). Budovy lze generalizovat ve vzniklém grafickém rozhraní s využitím frameworku QT, kde si uživatel mže vybrat jak algoritmus pro tvorbu konvexní obálky, tak i algoritmus pro samotnou generalizaci budov.

Výstupem jsou nově vzniklé obdélníky, které generalizují vstupní data, přičemž vstupní i výstupní data se zobrazí současně. Součástí aplikace je také validace výsledků, která se zobrazí v samostatném dialogovém okně. Celkově měl nejhorší výsledky algoritmus Weighted Bisector, zatímco nejlepší výsledky měl algoritmus Minimum Area Enclosing Rectangle. Algoritmy Longest Edge a Wall Average měly přibližně o jedno procento horší úspěšnost než MAER a měli tedy také dobré výsledky. Tento výsledek se dal předpokládat již z vizuálního porovnání výsledků algoritmů.

5 Zdroje

GREYRAT, R. (2022): Casco convexo – Conjunto 1 (Algoritmo de Jarvis o Wrapping) [online]. [cit. 9. 4. 2024]. Dostupné z: https://barcelonageeks.com/casco-convexo-conjunto-1-algoritmo-o-envoltura-de-jarvis/

SCIENTIFICLIB.COM (2024): Minimum Bounding box [online]. [cit. 9. 4. 2024]. Dostupné z: https://www.scientificlib.com/en/ComputationalGeometry/MinimumBoundingBox.html

SHARMA, P. (2024a): Gift Wrap Algorithm (Jarvis March Algorithm) to find Convex Hull [online]. [cit. 9. 4. 2024]. Dostupné z: https://iq.opengenus.org/gift-wrap-jarvis-march-algorithm-convex-hull/

SHARMA, P. (2024b): Graham Scan Algorithm to find Convex Hull [online]. [cit. 9. 4. 2024]. Dostupné z: https://iq.opengenus.org/graham-scan-convex-hull/