UNIVERZITA KARLOVA
PŘÍRODOVĚDECKÁ FAKULTA

ALGORITMY POČÍTAČOVÉ KARTOGRAFIE

Generalizace budov

Daniela DANČEJOVÁ,
Anna KOŽÍŠKOVÁ,
Praha 2023

Zadání

Vstup: množina budov $B = \{B_i\}_{i=0}^{n}$, *budova* $B_i = \{P_{i,j}\}_{j=1}^{m}$.

Výstup: $G(B_i)$.

Ze souboru nečtěte vstupní data představovaná lomovými body budov. Pro tyto účely použijte vhodnou datovou sadu, např. ZABAGED.

Pro každou budovu určtete její hlavní směry metodami.

- Minimum Area Enclosing Rectangle,
- Wall Average.

U první metody použijte některý z algoritmů pro konstrukci konvexní obálky. Budovu nahraďte obdélníkem se středem v jejím těžišti orientovaným v obou hlavních směrech, jeho plocha bude stejná jako plocha budovy. Výsledky generalizace vhodně vizualizujte.

Odhadněte efektivitu obou metod, vzájemně je porovnejte a zhodnoťte. Pokuste se identifikovat, pro které tvary budov dávají metody nevhodné výsledky, a pro které naopak poskytují vhodnou aproximaci.

Hodnocení:

Krok	**Hodnocení**
Generalizace budov metodami Minimum Area Enclosing Rectangle a Wall Average.	15b
Generalizace budov metodou Longest Edge.	*+5b*
Generalizace budov metodou Weighted Bisector.	*+8b*
Implementace další metody konstrukce konvexní obálky.	*+5b*
Ošetření singulárního případu u / při generování konvexní obálky.	*+2b*
Max celkem:	**35b**

Popis a rozbor problému

Geoinformační systémy a jejich softwary přinášejí celou řadu funkcí pro vizualizaci, zpracování a analýzu geoprostorových dat. Jejich součástí je také spousta funkcí používaných v kartografii. Ovšem mnoho potřebných funkcí, jako je například generalizace prvků, není součástí těchto softwarů. V dnešní digitální době je ovšem i taková funkce pro kartografické práce potřebná a jelikož se v současnosti téměř veškerá činnost přesouvá do digitálního světa, objevují se pro kartografy nové výzvy v podobě řešení kartografických problémů pomocí počítačových algoritmů.

Každý prvek na mapě musí být zobrazen vhodnými kartografickými vyjadřovacími prostředky (bod, linie, polygon). Součástí každé mapy je měřítko. Podle určitého měřítka se zobrazují na mapě i prvky obsažené v mapě. S tím souvisí i vhodné zobrazení prvků v závislosti na měřítku. Bude –li mapa zobrazovat vesnici s přilehlými lesy a ornou půdou, nemá smysl, aby byly všechny prvky na mapě vykresleny do detailů. Čtenáře mapy bude hlavně zajímat, jaký prvek se v daném místě vyskytuje. Je tedy žádoucí jednotlivé objekty v mapě zjednodušit, ale zároveň musí platit, aby stále nesly stejnou informaci o objektu a aby generalizovaný tvar stále odpovídal danému prvku. Jinak řečeno, podle Monmoniera (2018) dobrá mapa čtenáři do jisté míry lže (potlačuje pravdu) proto, aby mohl vidět, co potřebuje.

Generalizovat se tedy mohou liniové a polygonové prvky. Tato úloha se zabývá generalizací polygonů, konkrétně generalizací budov pomocí tzv. *konvexní obálky* a *min-max boxu* (dále jen *MMB*), též označovaného jako *bounding box*.

Konvexní obálka

Je dána konečná množina n bodů $S = [p_1, p_2, ..., p_n]$ v $\mathbb{R}^2$, kde $p_i = [x_i, y_i]$. Podle Bayera (2023a) je konvexní obálka (v angličtině *convex hull*) $\mathcal{H}$ nejmenší konvexní mnohoúhelník P obsahující S (a tedy neexistuje vlastní podmnožina $P' \subset P$, která splňuje tuto definici). Další definice konvexní obálky jsou uvedeny následovně:

1. Konvexní obálka $\mathcal{H}$ konečné množiny S představuje konvexní mnohoúhelník P s nejmenší plochou.

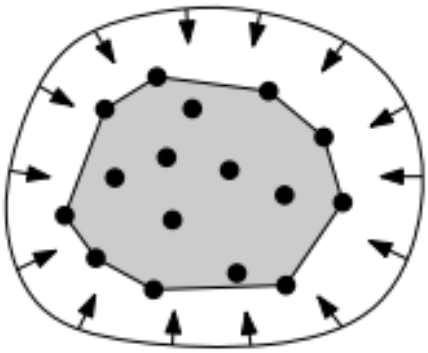
2. Konvexní obálka $\mathcal{H}$ konečné množiny S představuje průsečnici všech polorovin obsahujících S.

3. Konvexní obálka $\mathcal{H}$ konečné množiny S představuje sjednocení všech trojúhelníků, jejichž vrcholy tvoří body v S.

Podle Bayera (2023, s. 4) *„množinu S označíme jako konvexní, pokud spojnice libovolných dvou prvků leží zcela uvnitř této množiny“*.

Konvexní obálka (*Obrázek 1*) se používá pro řadu algoritmů. Pomocí její konstrukce je možné rychle najít tzv. *bounding box*, označován i jako *min-max box* (podle Wooda (2008) takový obdélník, který je

kolmý na osy souřadnicového systému x a y a minimálně ohraničuje složitější polygon), který představuje nejzákladnější generalizaci polygonu. Dále lze pomocí konstrukce několika konvexních obálek nad množinou bodů najít střed datasetu (Bayer 2023a), v jiných oborech nachází využití především v detekci kolizí (plánování pohybu robotů), různých statistických analýzách, analýzách tvarů objektů a pod.

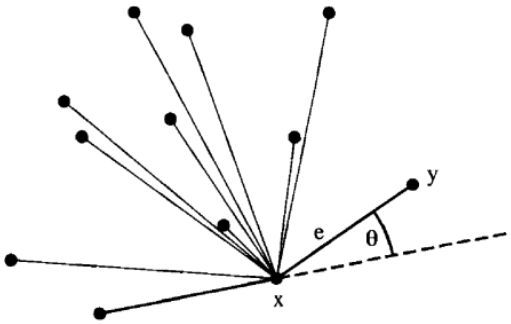


Obrázek 1: Ukázka konvexní obálky (převzato z de Berg a kol. (2008, s. 3)).

Způsobů konstrukce konvexní obálky je mnoho, v této úloze jsou konkrétně implementovány dva algoritmy: ***Jarvis Scan*** a ***Graham Scan***. Některé algoritmy pro konstrukci konvexní obálky jsou také převoditelné do vyšších dimenzí než je $\mathbb{R}^2$. V této úloze bude ovšem věnovaná pozornost čistě problémům v $\mathbb{R}^2$ prostoru.

Jarvis Scan

Tento algoritmus (označován také jako *Gift Wrapping Algorithm*) je jednoduchý a snadno implementovatelný. Představuje vůbec jeden z nejpoužívanějších postupů pro tvorbu konvexní obálky v $\mathbb{R}^2$ i v $\mathbb{R}^3$. Vychází z předpokladu, že v množině bodů S nejsou tři kolineární body (Bayer 2008). Je založen na principu opakovaného hledání maximálního úhlu ω_{max} mezi poslední hranou konvexní obálky $\mathcal{H}$ tvořenou body p_{j-1}, p_j a následující hranou tvořenou body p_j, p_{j+1}. Bod p_{j+1} hledáme ze všech $p_i \in S$, které dosud nejsou součástí $\mathcal{H}$. Složitost algoritmu je $O(n^2)$, nehodí se pro velké datasety (Bayer 2023a). Pro nalezení bodu p_{j+1} lze nalézt i minimální úhel θ s poslední hranou (*Obrázek 2*).



Obrázek 2: Jiný přístup ke konstrukci Jarvis Scan algoritmu. Následující hrana e svírá nejmenší úhel θ s předchozí hranou (převzato z Rourke (2005, s. 69)).

V prvním kroku algoritmu je nutné nalézt počáteční bod – pivota $q = [x_q, y_q]$ (časová složitost $O(n)$), který se automaticky stane součástí konvexní obálky $\mathcal{H}$:

$$y_q = \min_{\forall p_i \in S}(y_i)$$

Poté se inicializuje pomocný bod $p_{j-1} = [x_q - 1, y_q]$, pak $p_j = q$ a provede se vyhledání následujícího bodu p_{j+1} nalezením úhlu ω_{max} způsobem popsaným výše.

Algorithm 1 Jarvis Scan

Require: *pol* : *polygon*
Ensure: $\mathcal{H}$: *konvexní obálka*

$q \leftarrow \min_{\forall p_i \in S}(y_i)$
$p_{j-1} \leftarrow$ inicializuj pomocný bod se souřadnicemi $[x_q - 1, y_q]$
$p_j \leftarrow q$
přidej $q \rightarrow \mathcal{H}$

while true **do**
  $\omega_{max} \leftarrow 0$ ▷ inicializuj maximální úhel
  $i_{max} \leftarrow -1$ ▷ inicializuj index patřící bodu p_{j+1}

  for každý bod p_i z polygonu *pol* **do**
    if $p_j \neq p_i$ **then**
      $\omega \leftarrow \measuredangle p_{j-1}, p_j, p_i$
      if $\omega > \omega_{max}$ **then**
        $\omega_{max} \leftarrow \omega$ ▷ aktualizuj maximální úhel
        $i_{max} \leftarrow i$ ▷ aktualizuj index patřící bodu p_{j+1}
      end if
    end if
  end for

  přidej $p_{j+1} \rightarrow \mathcal{H}$
  $p_{j-1} \leftarrow p_j$
  $p_j \leftarrow p_{j+1}$

  if $p_j = q$ **then**
    break ▷ zlom cyklus pokud je následující bod pivotem
  end if
end while

vrať konvexní obálku $\mathcal{H}$

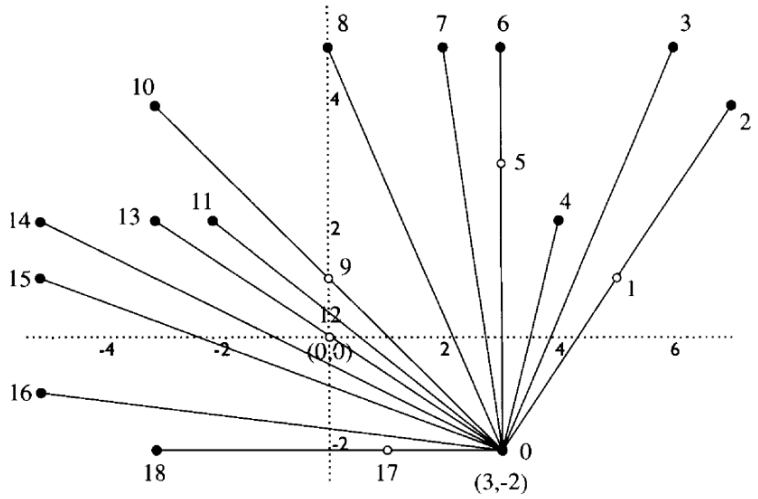
Graham Scan

Za jeden z vůbec prvních algoritmů pro nalezení konvexní obálky se považuje algoritmus popsaný Ronaldem Grahamem v roku 1972 (Rourke 2005). Tento algoritmus využívá pro tvorbu konvexní obálky kritérium pravotočivosti (resp. levotočivosti (Bayer 2023)), při kterém se posuzuje úhel ω_i mezi dvěma úsečkami procházejícími trojici bodů: p_{j-1}, p_j, p_{j+1}. Uvažujme počáteční bod (pivot) $q = [x_q, y_q]$ se souřadnicemi:

$$x_q, y_q = \min_{\forall p_i \in S}(x_i, y_i)$$

Protože může existovat více bodů se stejnou hodnotou y, je zvykem také uvažovat s jednou z extrémních hodnot x pro jednoznačné nalezení pivota.

Dále jsou všechny body p_i setříděné vzestupně podle jejich úhlu ω_i mezi osou x a spojnicí $\overline{qp_j}$. Protože může existovat více bodů p_i se stejným ω_i, setříděné body se pak znovu setřídí podle jejich euklidovské vzdálenosti od q (*Obrázek 3*).



Obrázek 3: Setřídění bodů p_i nejprv podle ω_i, pak podle jejich vzdálenosti od zvoleného q (převzato z Rourke (2005, s. 76)).

Spojením setříděných bodů p_i je pak vytvořen nekonvexní mnohoúhelník tvořen body p_j, který se pomocí kritéria pravotočivosti (resp. levotočivosti) provádí na konvexní mnohoúhelník (viz *Obrázek 4*). Do konvexní obálky $\mathcal{H}$ se okamžitě přidají body q a první setříděný bod p_j.

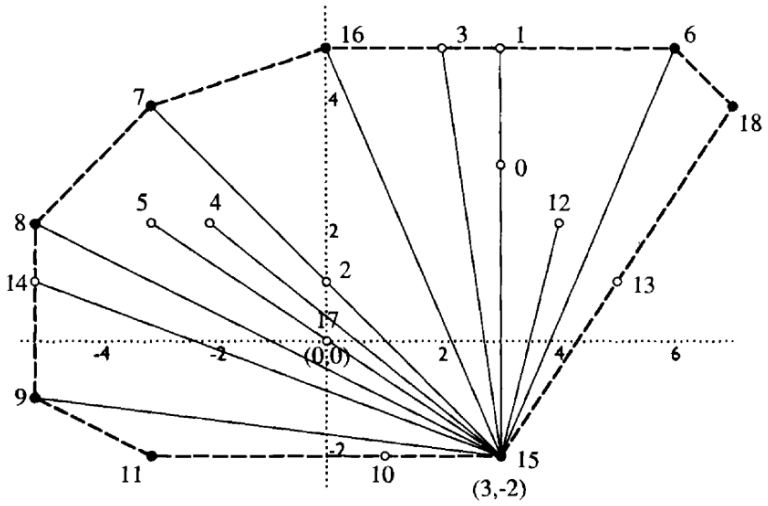
Kritérium levotočivosti pak determinuje příslušnost dalších bodů p_j ke konvexní obálce $\mathcal{H}$ následovně:

$$p_j \begin{cases} \notin \mathcal{H}, & p_{j+1} \in \sigma_r(p_j, p_{j+1}), \\ ? \in \mathcal{H}, & p_{j+1} \in \sigma_l(p_j, p_{j+1}), \end{cases}$$

kde σ_r a σ_l označují pravou, resp. levou polorovinu vzhledem k vektoru $\overrightarrow{p_{j-1}p_j}$.

V prvním případě bod p_j nebude součástí konvexní obálky $\mathcal{H}$ a dále s ním nebudeme uvažovat, vrátíme se o jeden vrchol zpět ($p_j = p_{j-1}$), doplníme dvojici bodů předcházejícím a následujícím doposud nevyloučeným vrcholem a provádíme sken znova. V druhém případě bude p_j možná součástí konvexní obálky. Posuneme se o jeden vrchol vpřed a zkoumáme toto kritérium u bodů p_j, p_{j+1}, p_{j+2}; tento postup opakujeme i pro další trojice bodů.

Algoritmus skončí, když $p_j = p_{n-1}$ (kde n je počet vrcholů polygonu). Jeho časová složitost je $O(n \log n)$. Ve vlastní implementaci se posuzuje pravotočivost místo levotočivosti vzhledem k orientaci souřadnicového systému.



Obrázek 4: Vytvořena konvexní obálka pomocí algoritmu Graham Scan (převzato z Rourke (2005, s. 85)).

Singulární případy

U obou zmíněných algoritmů vycházíme z předpokladu, že počet vrcholů polygonu $n \geq 3$. Ve vlastní implementaci je na případ, když $n < 3$ upozorněno, a konvexní obálka nad takovým prvkem nebude vytvořena.

Algorithm 2 Graham Scan

Require: *pol* : *polygon*
Ensure: $\mathcal{H}$: *konvexní obálka*

```
q ← min_{∀p_i∈S} (x_i, y_i)
seřaďBodyVPolygonu(pol, q)                          ▷ pomocná funkce
S ← inicializuj zásobník
n ← počet vrcholů pol

for každý bod p_j ze setříděného polygonu pol do
    while délka zásobníku S ≥ 2 do
        if p_{j+1} ∈ σ_r(p_j, p_{j+1}) then          ▷ pravotočivost
            break
        else                                         ▷ levotočivost nebo kolinearita bodů
            S.pop()                                  ▷ vyjmi p_j ze zásobníku
        end if
    end while
    přidej p_{j+1} → H
end for
vrať konvexní obálku H
```

Detekce hlavních směrů budov

V rámci generalizace budovy je nutné najít její hlavní směry pro zachování orientace původního nezgeneralizovaného objektu a návaznosti na ostatní prvky v mapě. Není například žádoucí, aby generalizovaná budova vbíhala za uliční, či říční linie. Je důležité, aby výsledný zjednodušený tvar budovy byl vhodně natočen vzhledem k respektování polohy ostatních prvků v mapě. Existuje několik algoritmů pro nalezení hlavních směrů budovy:

- *Minimum Area Enclosing Rectangle*,
- *Wall Average*,
- *Longest Edge*,
- *Weighted Bisector*.

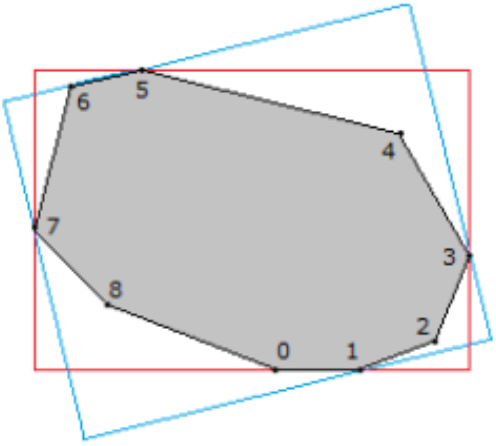
Minimum Area Enclosing Rectangle Algorithm

Principem algoritmu *Minimum Area Enclosing Rectangle* (dále jen *MAER*) je nalézt takový obdélník, jehož plocha bude minimální. Jinde je tento algoritmus znám pod názvem *Minimum-Area Bounding Rectangle*. Na podobném principu pracuje i algoritmus *Rotating Calipers*.

Je dána množina n bodů S v $\mathbb{R}^2$. Výstupem je takový obdélník $\mathcal{R}$, který opíše množinu S. Obdélník $\mathcal{R}$ nelze zkonstruovat napřímo. Proto je potřebné množinu S předzpracovat do konvexní obálky $\mathcal{H}$. Výsledný $\mathcal{R}$ má pak alespoň jednu hranu kolineární s hranou konvexní obálky $\mathcal{H}$ (Bayer 2023a).

Hlavní myšlenkou tohoto algoritmu je (v momentě, kdy je zkonstruována konvexní obálka tvořena n body) procházet všechny hrany konvexní obálky a pro každou její hranu nalézt MMB, jehož hrana je rovnoběžná s právě procházenou hranou konvexní obálky (viz *Obrázek 5*). Ze všech n obdélníků se vybere ten s minimální plochou. Během chodu algoritmu je tedy nutné uchovávat informaci o nejmenší možné ploše a zároveň si k této ploše pamatovat hranu, jež je kolineární s hranou minimálního opsaného obdélníku (Eberley 2020).

Algoritmus pak bude probíhat následovně: Během procházení každé hrany v konvexní obálce se vypočte úhel směrnice σ příslušné hrany. Pak se konvexní obálka $\mathcal{H}$ otočí o příslušný úhel σ a zkonstruuje se pro ni MMB. Pokud plocha nového MMB bude menší než dosud nejmenší spočtená plocha, pak se tento MMB a jeho příslušný úhel zapamatují (Bayer 2023a).



Obrázek 5: Ukázka zkonstruování min-max boxu, který je rovnoběžný s hranou konvexní obálky tvořenou body 5 a 6. (převzato z Eberley (2020, s. 4)).

Algorithm 3 Minimum Area Enclosing Rectangle

Require: pol : $polygon$
Ensure: $\mathcal{R}$: $polygon$

$\mathcal{H} \leftarrow$ vytvoř konvexní obálku
$R \leftarrow$ první opsaný obdélník
$A \leftarrow$ spočti iniciální plochu $\mathcal{R}$
$sigma\_min \leftarrow 0$ ▷ inicializuj úhel

for každou hranu $edge$ z konvexní obálky $\mathcal{H}$ **do**
  pro $edge$ spočti směrnici $sigma$
  natoč $\mathcal{H}$ o úhel $-sigma$
  zkonstruuj nový obdélník $\mathcal{R}'$ a spočti novou plochu A'

  if $A' < A$ **then** ▷ nová minimální plocha
    A' se stává nejmenší nalezenou plochou A
    zapamatuj si hodnoty $\mathcal{R} = \mathcal{R}'$ a $sigma\_min = sigma$
  end if
end for

vrať nejmenší $\mathcal{R}$ natočený o příslušnou hodnotu $sigma\_min$

Wall Average Algorithm

Tento algoritmus patří k více komplexnějším. Základním principem je pro každou hranu v polygonu spočítat poměr

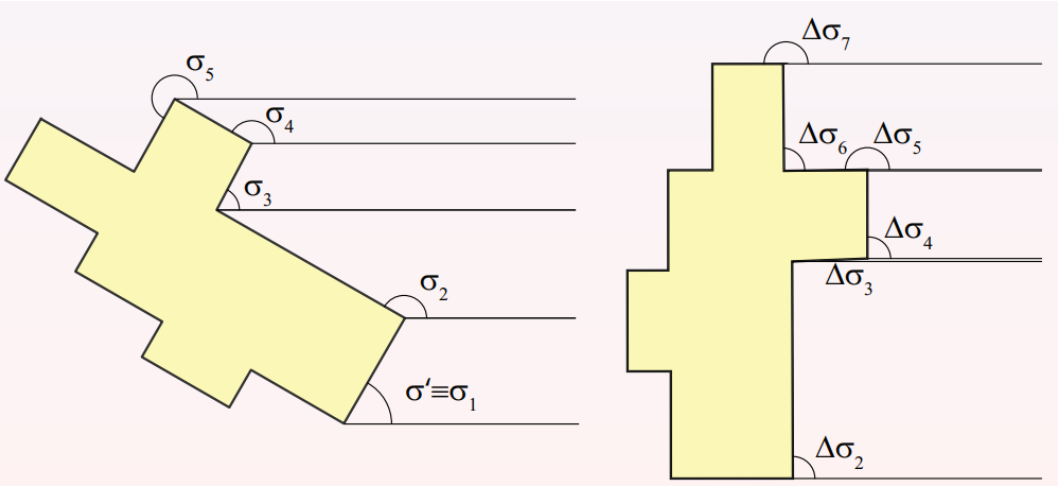
$$\left\lfloor \frac{\Delta\sigma_i}{\frac{\pi}{2}} \right\rfloor, \tag{1}$$

kde $\Delta\sigma_i$ je rozdíl úhlů σ_i a σ', σ_i je směrnice právě zpracované hrany polygonu a σ' představuje jednu libovolně zvolenou směrnici jedné z hran polygonu.

Pro každou hranu polygonu tvořeného n body se vypočte směrnice σ_i. Dále se vybere libovolná hrana s úhlem σ', o níž se otočí polygon. Poté se pro každou hranu spočte rozdíl

$$\Delta\sigma_i = \sigma_i - \sigma'. \tag{2}$$

Pro jasnější představu je přiložen *Obrázek 6*, který vlevo ukazuje vypočtené směrnice σ_i a vpravo rozdíl úhlů $\Delta\sigma_i$ (Bayer 2023a).



Obrázek 6: Metoda Wall average: vpravo: směrnice σ_i spočtená pro každou hranu, vlevo: rozdíl $\Delta\sigma_i$ (převzato z Bayer (2023a s. 40).

Následně pro každou hodnotu $\Delta\sigma_i$ spočteme (1), čímž se získá zaokrouhlený podíl k_i:

$$k_i = \left\lfloor \frac{\Delta\sigma_i}{\frac{\pi}{2}} \right\rfloor. \tag{3}$$

Po vypočtení k_i je pak potřebné vypočítat zbytky r_i a odchylky od $0 \pm k\pi$, respektive $\frac{\pi}{2} \pm k\pi$:

$$r_i = \Delta\sigma_i - k_i \frac{\pi}{2}. \tag{4}$$

Hlavní směr budovy natočení je pak dán vztahem:

$$\sigma = \sigma' + \sum_{i=1}^{n} \frac{r_i \cdot s_i}{s_i}, \tag{5}$$

kde s_i je délka hrany i. V rámci vlastní implementace je používán aritmetický průměr místo váženého průměru.

Algorithm 4 Wall Average Algorithm

Require: pol : $polygon$
Ensure: $\mathcal{R}$: $polygon$

$\sigma' \leftarrow$ spočítej směrnici první hrany polygonu pol
$r\_aver \leftarrow 0$ ▷ inicializuj průměrný zbytek

for každou hranu $edge$ v polygonu pol **do**
  pro $edge$ spočti směrnici σ_i
  $\Delta\sigma_i \leftarrow$spočti rozdíl $\sigma_i - \sigma'$

  if $\Delta\sigma_i < 0$ **then** ▷ korekce úhlu $\Delta\sigma_i$
    k úhlu $\Delta\sigma_i$ přičti periodu 2π
  end if

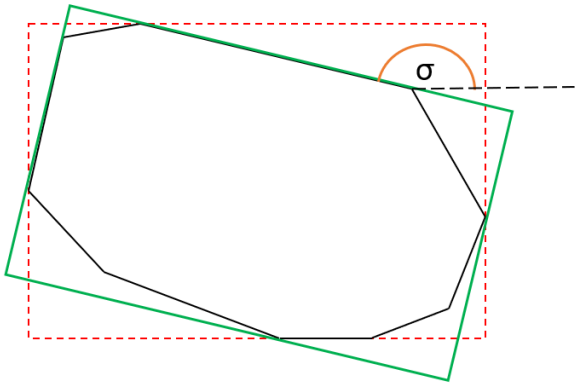
  vypočti koeficient k_i a výsledek zaokrouhli
  vypočti zbytek r_i pro aktuální iteraci
  aktualizuj průměrný zbytek r_aver o hodnotu r_i
end for

vypočti průměrný zbytek pro polygon r_aver
nastav průměrný úhel $\sigma\_aver = \sigma' + r\_aver$
$\mathcal{R} \leftarrow$ vytvoř obdélník kolem r_aver

vrať $\mathcal{R}$

Longest Edge Algorithm

Tento jednoduchý algoritmus předpokládá, že hlavní směr polygonu (budovy) je jeho nejdelší hrana. Druhý hlavní směr je pak kolmý na nejdelší hranu. Podél nejdelší strany budovy pak bude natočen výsledný obdélník, jehož delší strana bude rovnoběžná s nejdelší hranou. *Obrázek 7* ilustruje princip algoritmu. Zelený obdélník je rovnoběžný s nejdelší hranou polygonu (Bayer 2023a).



Obrázek 7: Opsaný obdélník podél nejdelší hrany polygonu natočený o úhel σ (vlastní zpracování).

Pro danou množinu S, jejíž hranice tvoří n bodů v $\mathbb{R}^2$ je hledána nejdelší hrana polygonu, podél které se zkonstruuje MMB. Výstupem algoritmu je opsaný obdélník $\mathcal{R}$ kolem množiny S rovnoběžný s nejdelší hranou polygonu.

Algorithm 5 Longest Edge Algorithm

Require: pol : $polygon$
Ensure: $\mathcal{R}$: $polygon$

$longest\_edge \leftarrow -1$ ▷ inicializuj nejdelší hranu

for každou hranu $edge$ v polygonu **do**
  pro $edge$ spočti její délku $dist$

  if $dist > longest\_edge$ **then** ▷ nová nejdelší hrana
    $edge$ s délkou $dist$ se stává nejdelší hranou v polygonu
    pro tuto $edge$ vypočti její směrnici σ
  end if
end for

$rotate\_pol \leftarrow$ rotuj polygon o $-\sigma$
$\mathcal{R} \leftarrow$ vytvoř obdélník kolem $rotate\_pol$

vrať $\mathcal{R}$

Weighted Bisector Algorithm

Tento algoritmus pro detekci hlavních směrů použije dvě nejdelší vnitřní úhlopříčky nalezené v polygonu. Jde tedy o nalezení takových úhlopříček, které neprotínají žádnou z hran polygonu. Pro test, zda nalezená úhlopříčka protíná hranu polygonu, se použije opakovaná aplikace Halfplane testu (Bayer 2023a).

Mějme úsečku $\overline{\mathrm{P_1P_2}}$ tvořenou body $P_1 = [x_1, y_1]$ a $P2 = [x_2, y_2]$ a úsečku $\overline{\mathrm{P_3P_4}}$ tvořenou body $P3 = [x_3, y_3]$ a $P4 = [x_4, y_4]$.

Pro tyto body se provede 4x analýza koncového bodu úsečky vzhledem ke druhé úsečce a naopak. Jedná se o výpočet determinantu (Bayer 2023b).

$$\begin{aligned} t_1 &= \begin{vmatrix} x_2 - x_1 & y_2 - y_1 \\ x_4 - x_1 & y_4 - y_1 \end{vmatrix} \quad t_2 = \begin{vmatrix} x_2 - x_1 & y_2 - y_1 \\ x_3 - x_1 & y_3 - y_1 \end{vmatrix} \\ t_3 &= \begin{vmatrix} x_4 - x_3 & y_4 - y_3 \\ x_1 - x_3 & y_1 - y_3 \end{vmatrix} \quad t_4 = \begin{vmatrix} x_4 - x_3 & y_4 - y_3 \\ x_2 - x_3 & y_2 - y_3 \end{vmatrix} \end{aligned} \tag{6}$$

Pokud t_1 a t_2 mají stejné znaménko, nebo t_3 a t_4 mají stejné znaménko, pak mezi úsečkami $\overline{\mathrm{P_1P_2}}$ a $\overline{\mathrm{P_3P_4}}$ neexistuje průsečík. Bude-li mít t_1 a t_2 stejné znaménko, pak se koncové body úsečky nacházejí ve stejné polorovině vůči druhé úsečce (2023b).

Po nalezení dvou nejdelších úhlopříček se spočte pro každou úhlopříčku jejich délka (s_1 a s_2) a směrnice (σ_1 a σ_2). Hlavní směr budovy σ se pak vypočte pomocí vztahu:

$$\sigma = \frac{s_1\sigma_1 + s_2\sigma_2}{s_1 + s_2} \tag{7}$$

Algorithm 6 Weighted Bisector Algorithm

Require: *pol* : *polygon*
Ensure: $\mathcal{R}$: *polygon*

zkontroluj, zda má polygon více jak 3 body ▷ pro polygon tvořený 3 body neexistuje úhlopříčka
diagonals ←najdi všechny možné úhlopříčky
seřaď úhlopříčky podle délky
inicializuj hodnoty $\sigma 1$, *dist*1, $\sigma 2$, *dist*2 ← *None*

for každou *diagonal* z listu *diagonals* **do**
  for každou hranu *edge* v polygonu *pol* **do**
    if *diagonal* má průsečík s *edge* **then**
      pokračuj
    else ▷ úhlopříčka nemá průsečík – jedná se o vnitřní úhlopříčku
      test = *True*
    end if
  end for

  if *test* = *True* **then**
    pokračuj ▷ nejde o vnitřní uhlopříčku
  else
    if *dist*1 = *None* **then** ▷ vypočti směrnici a délku první úhlopříčky
      $\sigma 1$ ←spočti směrnici úhlopříčky *diagonal*
      *dist*1 ← vypočti délku úhlopříčky *diagonal*
    else ▷ vypočti směrnici a délku druhé úhlopříčky
      $\sigma 2$ ←spočti směrnici úhlopříčky *diagonal*
      *dist*2 ← vypočti dílku úhlopříčky *diagonal*
      break
    end if
  end if
end for

$\sigma \leftarrow (dist1 * \sigma 1 + dist2 * \sigma 2)/(dist1 + dist2)$ ▷ hlavní směr polygonu
rotate_pol ← rotuj polygon o $-\sigma$
$\mathcal{R}$ ← vytvoř obdélník kolem *rotate_pol*

vrať $\mathcal{R}$

V rámci této úlohy byl pro každý algoritmus výsledný obdélník $\mathcal{R}$ zmenšen tak, aby jeho plocha odpovídala ploše původního polygonu.

Implementace

V rámci této úlohy bylo ve frameworku QT vytvořené uživatelské prostředí pro demonstraci generalizace polygonů budov s využitím výše zmíněných algoritmů. V následující kapitole jsou pak algoritmy pro detekci hlavních směrů generalizovaných polygonů vyhodnoceny a porovnány.

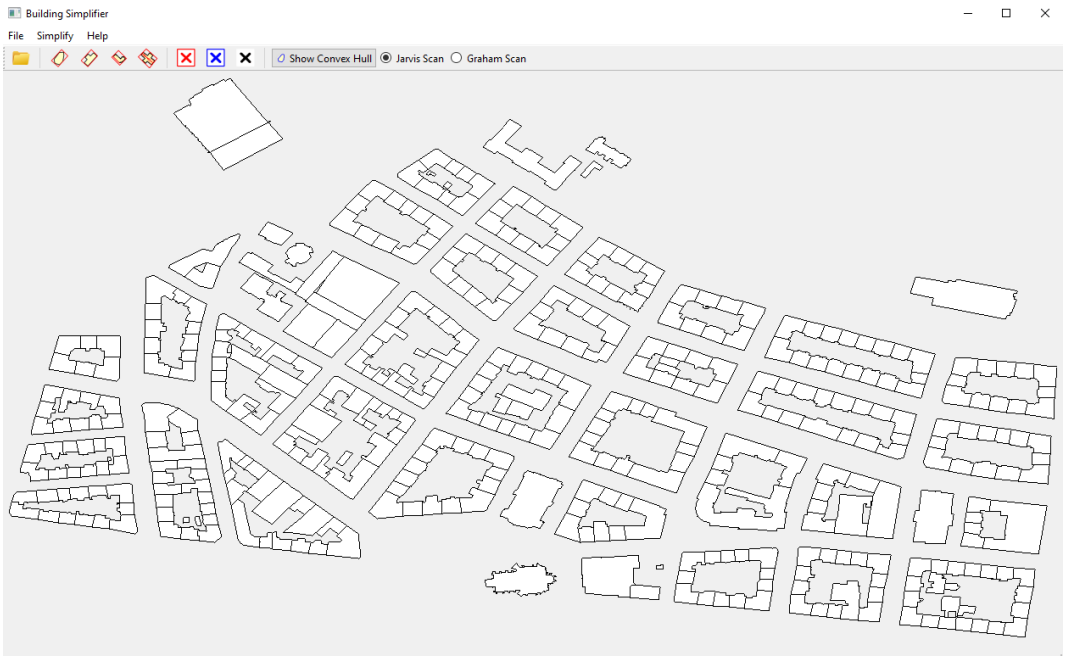
Vstupní data

Aplikace je přizpůsobená na načítání geografických vstupních dat v Křovákově souřadnicovém referenčním systému (`EPSG:5514`) ve formátu `JSON` a `GeoJSON`. Předpokládá se načítání polygonové mapy (*Obrázek 8*), pokud však vstupní soubor obsahuje i jiné prvky (např. linie), uživatel je na tuto skutečnost upozorněn a generalizovat se budou jen polygonové prvky.

Pro možnost porovnání generalizace na různých tvarech polygonů jsou k aplikaci přiloženy testovací data ve formátu `GeoJSON` ve složce *input_files*:

- `chotebor.geojson`,
- `Prosek.geojson`,
- `vinohrady.geojson`.

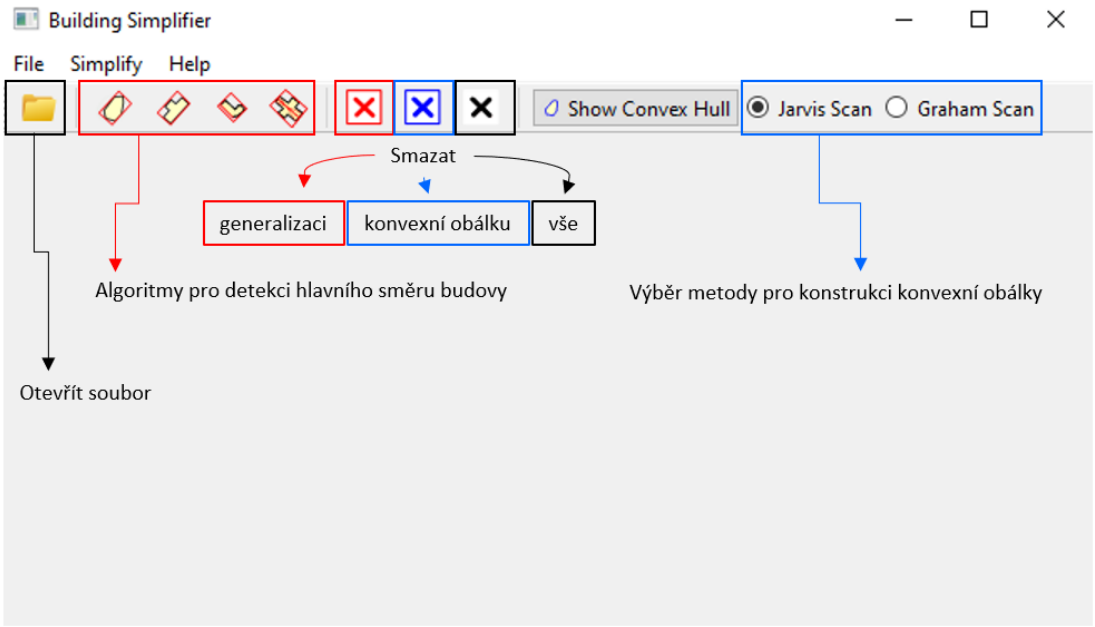
Jedná se o různé typy zástavby, a tedy různé kompozice a tvary polygonů reprezentujících budovy.



Obrázek 8: Ukázka načtených dat Vinohrad (vlastní zpracování).

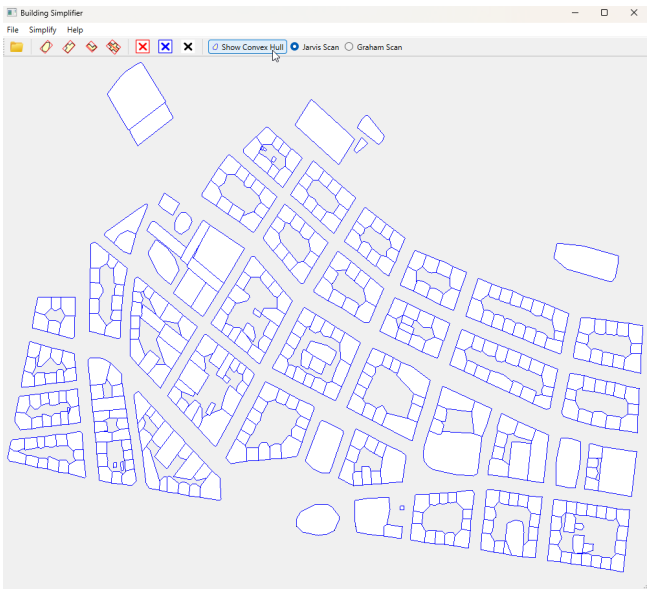
Aplikace

Grafické rozhraní aplikace bylo vytvořeno v prostředí `Qt Creator 9.0.1` a dále upravováno v prostředí programovacího jazyka `Python 3.11`. Ovladatelnost a možnosti aplikace popisuje detailněji *Obrázek 9*.

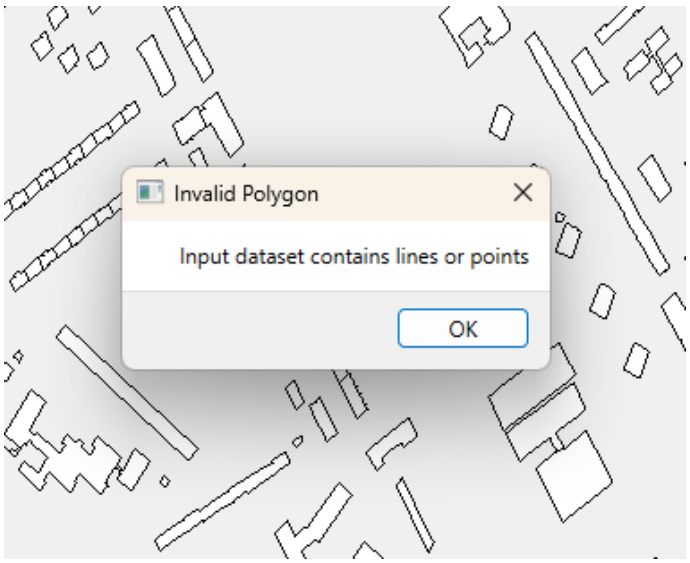


Obrázek 9: Ukázka grafického okna aplikace s popisem funkcí (vlastní zpracování).

Uživatel má možnost na vstupních datech také zobrazit konvexní obálky jednotlivých polygonů (*Obrázek 10*). Pokud je v datasetu nevalidní polygon (tj. např. polygon o dvou vrcholech nebo prázdný polygon), uživatel je upozorněn vyskakovacím oknem (*Obrázek 11*) a tento prvek je vyloučen z dalšího zpracování.



Obrázek 10: Ukázka konvexních obálek vinohrad (vlastní zpracování).



Obrázek 11: Ukázka vyskakovacího okna upozorňujícího na nevalidní polygon (vlastní zpracování).

Třídy a metody

Funkční chod aplikace si vyžaduje tři povinné skripty, v kterých jsou implementovány potřebné třídy a metody: `mainform.py`, `algorithms.py` a `draw.py`.

Třída MainForm

Třída MainForm ze souboru `mainform.py` zabezpečuje inicializaci okna aplikace, vrchní lišty, panelu nástrojů, ikon a tlačítek. Zároveň přepojuje jednotlivé interaktivní položky okna s metodami, které vykonají specifické akce. Týkají se především otevření souboru, přepínání algoritmů apod. Část této třídy byla vygenerována v prostředí `Qt Creator 9.0.1` (metody `setupUi()` a `translateUi()`). Níže jsou vyjmenovány nově implementované metody:

- `switchToJarvis()`

  Nastaví algoritmus pro konstrukci konvexní obálky na *Jarvis Scan*.

- `switchToGraham()`

  Nastaví algoritmus pro konstrukci konvexní obálky na *Graham Scan*.

- `constructCH()`

  Provede konstrukci konvexní obálky. Metodě je předán seznam vstupních polygonů, na kterých se s využitím zvoleného algoritmu vytvoří jejich konvexní obálky. Pokud je nějaký z předaných polygonů nevalidní, konvexní obálka nebude vytvořena.

- `simplifyMAERClick()`

  Provede generalizaci polygonu s využitím algoritmu *Minimum Area Enclosing Rectangle*.

- `simplifyWallAverageClick()`

  Provede generalizaci polygonu s využitím algoritmu *Wall Average*.

- `simplifyLongestEdgeClick()`

  Provede generalizaci polygonu s využitím algoritmu *Longest Edge*.

- `simplifyWeightedBisectorClick()`

  Provede generalizaci polygonu s využitím algoritmu *Weighted Bisector*.

- `clearButton()`

  Zavolá metodu `clearCanvas()` z třídy `Draw`, která vyprázdní okno.

- `clearERButton()`

  Zavolá metodu `clearERs()` z třídy `Draw`, která smaže generalizované polygony.

- `clearCHButton()`

  Zavolá metodu `clearCHs()` z třídy `Draw`, která smaže konvexní obálky.

- `exitClick()`

  Ukončí aplikaci.

- `aboutClick()`

  Otevře repozitář s aplikací v portálu `GitHub`.

- `processFile()`

  Zabezpečuje otevření souboru. Samotný soubor načte do proměnné pomocí metody `openFile()` a následně zavolá metodu `clearCanvas()` pro vyprázdnění okna. Pokud je vstupní `JSON` nebo `GeoJSON` nečitelný (např. nestandardní hierarchie položek v slovnících), uživatele na to upozorní vyskakovacím oknem.

- `openFile()`

  Otevře `JSON` nebo `GeoJSON` soubor a načte ho do proměnné.

Třída Algorithms

V této třídě jsou obsaženy algoritmy, které byly popsány v kapitole *Popis problému* věnované algoritmům pro konstrukci konvexní obálky a generalizace. Obsahuje tři skupiny metod:

1. Metody pro konstrukci konvexní obálky: Jarvis Scan, Graham Scan;
2. Metody pro určení hlavního směru budovy: MAER, Wall Average, Longest Edge, Weighted Bisector;
3. Pomocní metody pro zmíněné algoritmy (výpočet euklidovské vzdálenosti, nalezení min-max boxu, přizpůsobení velikosti min-max boxu, výpočet úhlů apod).

První dvě skupiny již byly detailně popsány výše, třetí skupina obsahuje celkem 14 algoritmů uvedených níže. Jejich podrobnější popis se nachází uvnitř této třídy ve skriptu `algorithms.py`.

- `get2LinesAngle(p1, p2, p3, p4)`

  Vypočte úhel mezi dvěma vstupními liniemi.

- `getPolarAngle(p1, p2)`

  Vypočte směrnici zadané linie.

- `vectorOrientation(p1, p2, p3)`

  Udává pravotočivost, levotočivost nebo kolinearitu vektorů.

- `findPivot(pol)`

  Nalezne pivota jako bod s minimálními souřadnicemi x a y.

- `sortPoints(pol, q)`

  Seřadí body vstupního polygonu vzestupně nejprv podle velikosti jejich polárního úhlu s pivotem q, pak podle jejich vzdálenosti od pivota q.

- `rotate(pol, sigma)`

  Otočí polygon o úhel σ.

- `minMaxBox(pol)`

  Vytvoří min-max box a vypočte jeho plochu.

- `computeArea(pol)`

  Vypočte plochu libovolného konvexního útvaru.

- `resizeRectangle(er, pol)`

  Změní velikost generalizované budovy vzhledem k její původní ploše.

- `checkValidity(pol)`

  Zkontroluje, jestli je vstupní polygon validní.

- `euclidDistance(p1, p2)`

  Vypočte euklidovskú vzdálenost mezi dvěma body.

- `findDiagonals(ch)`

  Nalezne všechny uhlopříčky konvexní obálky a jejich velikosti.

- `intersectionTest(p1, p2, pol)`

  Zjišťuje, jestli se vstupní úhlopříčka protíná s jednou z hran budovy.

- `setDiagonals(diagonals, pol)`

  Vrátí dvě nejdelší uhlopříčky a jejich směrnice.

Třída Draw

Třída Draw ze souboru `Draw.py` slouží pro inicializaci proměnných nesoucí prostorovou informaci, načítání a vykreslování geoprostorové informace. Při spuštění aplikace se inicializují proměnné pro seznam polygonů, seznam ohraničujících obdélníků (generalizovaných polygonů) a seznam konvexních obálek:

- `self.__polyg_list`,
- `self.__er_list`,
- `self.__ch_list`.

Třída Draw obsahuje následující metody:

- `paintEvent(e:QPaintEvent)`

  Vykresluje objekty (bod a polygony) na plátno (Canvas).

- `clearCanvas()`

  Smaže všechny objekty na plátně (Canvas).

- `clearERs()`

  Vyprázdní seznam ohraničujících obdélníků.

- `clearCHs()`

  Vyprázdní seznam konvexních obálek.

- `getPolygonList()`

  Vrátí seznam vstupních polygonů.

- `getEnclosingRectangles(pols)`

  Vrátí seznam ohraničujících obdélníků.

- `getConvexHulls(pols)`

  Vrátí seznam konvexních obálek.

- `resizePolygons(xmin, ymin, xmax, ymax)`

  Roztáhne vstupní data na plátno podle velikosti okna aplikace.

- `findBoundingPoints(p:QPointF, xmin, ymin, xmax, ymax)`

  Nalezne minimální a maximální souřadnice pro ohraničení polygonu (tzv. bounding box).

- `loadData(data)`

  Prochází slovník ze vstupního souboru `JSON/GeoJSON` a načte geoprostorovou informaci.

Výsledky

Na všechny datasety zmíněné v kapitole *Implementace* byly aplikovány algoritmy pro detekci hlavních směrů budovy. Níže jsou detailněji popsány výsledky jednotlivých algoritmů a slovně zhodnoceny výsledné generalizace.

Dataset Vinohrady

Hustě zastavěná oblast Vinohrad obsahuje mnoho bloků budov, které na sebe plynule navazují. Půdorysy budov jsou především obdélníkového tvaru. Půdorysy některých budov, jako je např. bazilika sv. Ludmily (dole vlevo na jednotlivých oknech *Obrázku 12*), jsou však složitější a nepravidelné.

Obrázek 12: Porovnání generalizačních algoritmů pro Vinohrady. Algoritmy: (a) *MAER*, (b) *Wall Average*, (c) *Longest Edge* a (d) *Weighted Bisector*. (vlastní zpracování).

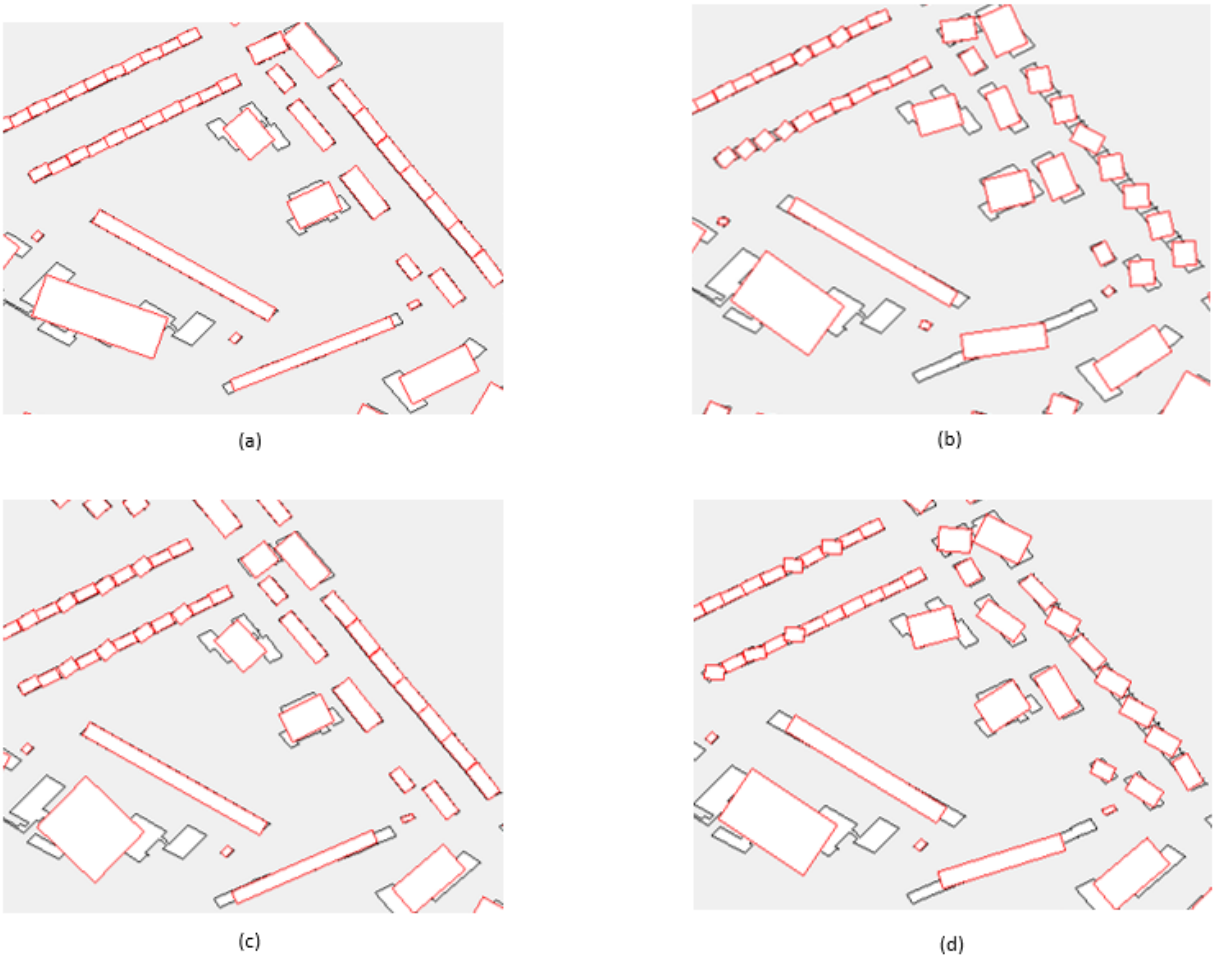
Vzhledem na tvar většiny půdorysů dokázala většina algoritmů poměrně přesně určit hlavní směr budovy. Pro tento dataset vystihl hlavní směr budov nejpřesněji ***MAER***. Vizuálně velmi podobných výsledků dosahl i algoritmus ***Longest Edge***.

V případě algoritmu ***Wall Average*** již docházelo v některých případech k detekci jiných hlavních směrů budov jako u algoritmů MAER a Longest Edge. Došlo tedy ke konstrukci zjednodušených tvarů s jinou konfigurací, která už méně charakterizovala skutečné tvary půdorysů budov. To je však

nejvíc pozorovatelné u algoritmu ***Weighted Bisector***, který si nedokázal poradit s některými méně pravidelnými tvary budov a natočil zjednodušující obdélníky podle nalezených uhlopříček, které zřejmě nedokážou pro nepravidelné tvary polygonů přesně vystihnout jejich hlavní směr. V případě baziliky sv. Ludmily byl hlavní směr budovy určen správně, ovšem u budovy Vinohradského divadla a Národního domu výsledný generalizovaný tvar neodpovídal jejich hlavním směrům.

Dataset Prosek

Sídliště Prosek obsahuje především půdorysy na sebe navazujících panelových budov a domů s jednoduchými obdélníkovými tvary (*Obrázek 13*). V oblasti se nachází i několik nekonvexních budov, jako je např. základní škola, nebo budovy ve tvaru písmena C.



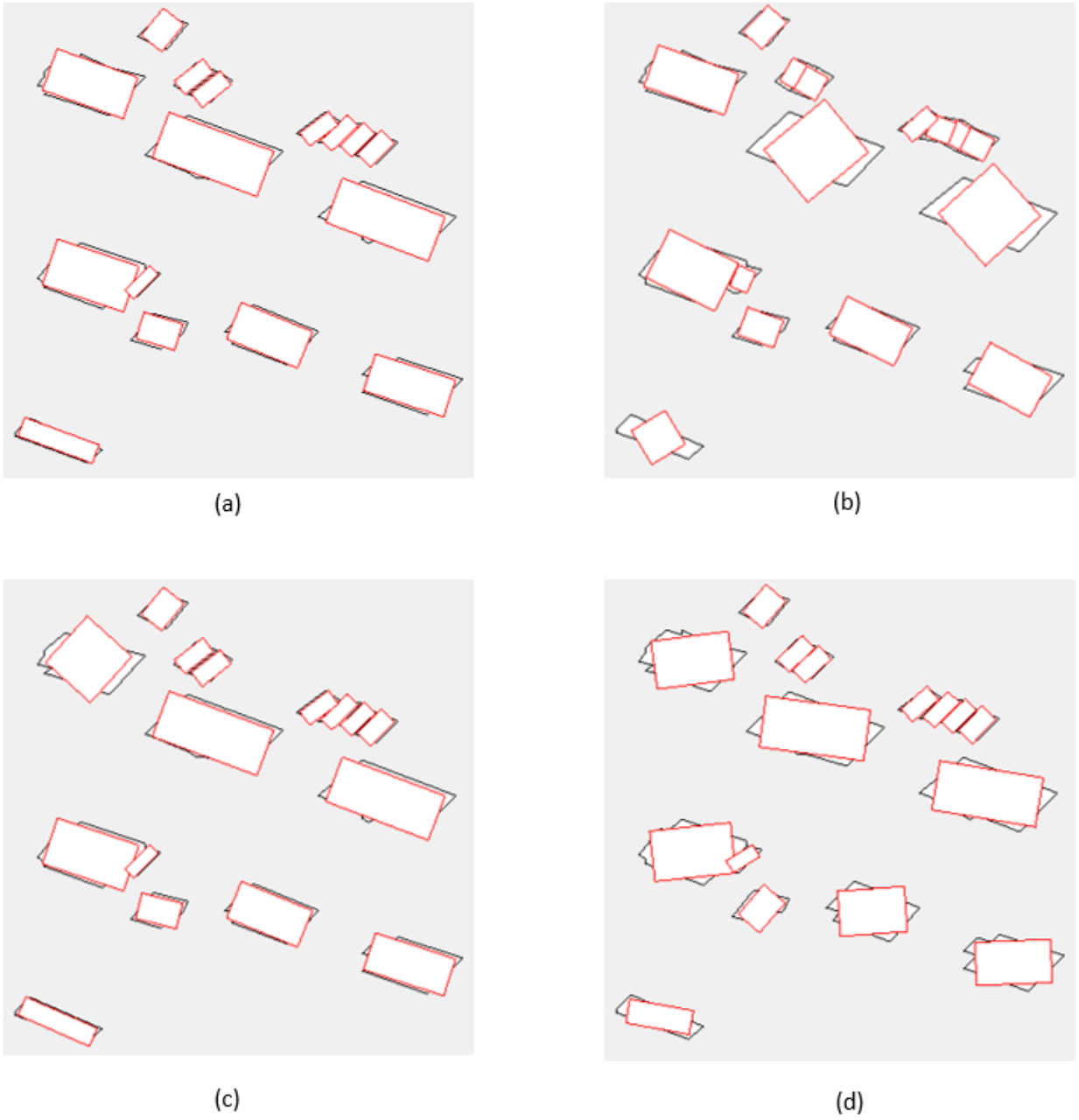
Obrázek 13: Porovnání generalizačních algoritmů pro Prosek. Algoritmy: (a) *MAER*, (b) *Wall Average*, (c) *Longest Edge* a (d) *Weighted Bisector*. (vlastní zpracování).

Protože je většina budov datasetu obdélníkového tvaru, není na nich co generalizovat. Pro tyto tvary se jako nejvhodnější algoritmus pro určení hlavního směru budov opět jeví ***MAER***, který i nejpřesněji zachoval natočení jednotlivých budov. O něco méně přesný, avšak podobně úspěšný byl opět ***Longest Edge***, který měl problémy s navazujícími budovami v horní části výřezů.

Algoritmy ***Wall Average*** a ***Weighted Bisector*** si s přesností vedly podobně. Oba tyto algoritmy nebyly schopny zachytit skutečný směr budov v navazující zástavbě, a to i přesto, že jsou jenom obdélníkového tvaru. V případě nekonvexního půdorysu základní školy (ve výřezech dole vlevo) však nejpřesněji vystihly jeho orientaci a vedly si lépe než MAER a Longest Edge.

Dataset Chotěboř

Vybraná část města Chotěboř obsahuje především domovou zástavbu, která na sebe příliš nenavazuje. Jednotlivé půdorysy jsou primárně obdélníkového nebo čtvercového tvaru (*Obrázek 14*). V horní části datasetu (není ve výřezech) je domov důchodců ve tvaru písmena L. Tento tvar mají i některé další půdorysy domů.



Obrázek 14: Porovnání generalizačních algoritmů pro Chotěboř. Algoritmy: (a) *MAER*, (b) *Wall Average*, (c) *Longest Edge* a (d) *Weighted Bisector*. (vlastní zpracování).

Protože jsou tvary půdorysů opět především pravidelné, konfiguraci obdélníkových budov nejpřesněji detekovaly ***MAER*** a ***Longest Edge*** s téměř stejnými výsledky.

U algoritmu ***Wall Average*** již došlo k nesprávnému odhadu hlavního směru některých větších budov, avšak přibližně zachovává jejich původní tvar. Algoritmus ***Weighted Bisector*** fungoval uspokojivě při obdélníkových budovách. O něco komplikovanější tvary však již nenatočil správně a z tohoto

hlediska si vedl hůře než ostatní algoritmy.

Budovu domovu důchodců ani jeden algoritmus nedokázal adekvátně generalizovat pro její tvar. Půdorys ve tvaru písmena L byl ve všech případech nahrazen téměř čtvercovým obdélníkem.

V tabulce 1 je vyhodnocena relativní přesnost jednotlivých algoritmů.

Tabulka 1: Úspěšnost jednotlivých algoritmů v %

Algoritmus	**Vinohrady**	**Prosek**	**Chotěboř**	**průměr**
MAER	92	90	95	92,3
Wall Average	87	78	89	84,6
Longest Edge	95	83	91	89,6
Weighted B.	55	70	80	68,3

Shrnutí

Pro vybrané datasety se jako nejvhodnější algoritmus pro určení hlavního směru budovy jeví ***Minimum Area Enclosing Rectangle*** (toto tvrzení potvrzuje i Tabulka 1), pak ***Longest Edge***. Nejméně uspokojivé výsledky dosahoval algoritmus ***Weighted Bisector***, který si však v několika případech dokázal lépe poradit s nekonvexními půdorysy. Algoritmus ***Wall Average*** dosahoval různé úspěšnosti. Ve vlastní implementaci tohoto algoritmu se pracuje s aritmetickým průměrem zbytků úhlů a bylo by vhodné porovnat tyto výsledky s verzí, která pracuje s váženým průměrem zbytků.

Závěr

V teto úloze byly představeny a implementovány vybrané algoritmy pro generalizaci tvarů polygonů. Nejprv byly představeny a implementovány dva algoritmy pro konstrukci konvexní obálky *Jarvis Scan* a *Graham Scan*. S jejich využitím byly poté implementovány čtyři algoritmy pro určení hlavního směru budov: *Minimum Area Enclosing Rectangle*, *Wall Average*, *Longest Edge* a *Weighted Bisector*. Tyto algoritmy byly aplikovány na tři vybrané datasety a následně mezi sebou porovnány a slovně zhodnoceny.

Vybrané datasety obsahují různé typy zástavby: historickou část města, sídliště a nesouvislou domovou zástavbu. Ve všech případech se jako nejpřesnější algoritmus pro celkovou generalizaci jeví *Minimum Area Enclosing Rectangle*, poté *Longest Edge*, *Wall Average* a nakonec *Weighted Bisector*. Protože půdorysy většiny budov byly v datasetech především obdélníkového tvaru, pro úspěšné určení hlavního směru budov postačí algoritmy, které pracují pouze s délkami hran budov. V několika případech si však vedli úspěšněji algoritmy pracující primárně s úhly natočení, a to u nekonvexních půdorysů.

Aplikaci je vhodné vylepšit o implementování možnosti přiblížení/oddálení polygonů v okně, při načtení většího datasetu se totiž stane obsah okna nečitelný.

Vytvořená aplikace je volně dostupná přes `GitHub` na adrese `https://github.com/koziskoa/APK_2023/tree/master/building_simplify`.

Literatura

[1] BAYER, T. (2023a): Konvexní obálka množiny bodů. Přednáška pro předmět Algoritmy počítačové kartografie, Katedra aplikované geoinformatiky a kartografie. Přírodovědecká fakulta UK, dostupné *zde* (cit. 30. 3. 2023).

[2] BAYER, T. (2023): Úvod do výpočetní geometrie. Základní vztahy. Přednáška pro předmět Algoritmy počítačové kartografie, Katedra aplikované geoinformatiky a kartografie. Přírodovědecká fakulta UK, dostupné *zde* (cit. 30. 3. 2023).

[3] DE BERG, M., CHEONG, O., VAN KREVELD, M., OVERMARS, B. (2008): Computational Geometry. Algorithms and Applications, third edition. Springer, Berlin.

[4] EBERLEY, D. (2020): Minimum-Area Rectangle Containing a Set of Points. Geometric Tools, Redmond WA 98052, dostupné *zde* (cit. 30. 3. 2023).

[5] MONMONIER, M. (2018): How to lie with maps, second edition. The University of Chicago Press, Chicago.

[6] ROURKE, O. J. (2005): Computational Geometry in C. Cambridge University Press, Cambridge.

[7] WOOD, J. (2008): Minimum Bounding Rectangle. In: Shekhar, S., Xiong, H. (eds.) Encyclopedia of GIS. Springer, Boston.