ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

FAKULTA STAVEBNÍ, OBOR GEODÉZIE A KARTOGRAFIE

KATEDRA GEOMATIKY

Název předmětu:					
Algoritmy digitální kartografie a GIS					
Úloha: U2	**Název úlohy:** Generalizace budov				
Akademický rok:	**Semestr:**	**Studijní skupina:**	**Vypracoval:**	**Datum:**	**Klasifikace:**
2024/2025	letní	101C	Michal Kovář Filip Roučka	7. 4. 2025	

1 Zadání

Zadáním úlohy bylo implementovat generalizaci budov do úrovně detailu LOD0.

- Vstup: množina budov $B = \{B_i\}_{i=1}^{n}$, kde budova B_i je reprezentována množinou lomových bodů $\{P_{i,j}\}_{j=1}^{m}$.
- 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, například ZABAGED. Testování proveďte nad třemi datovými sadami (historické centrum města, sídliště, 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 hlavních směrech, se středem v těžišti budovy, jehož 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.

2 Bonus

- *Generalizace budov metodou Longest Edge.*
- *Generalizace budov metodou Wall Average.*
- *Generalizace budov metodou Weighted Bisector.*
- *Implementace další metody konstrukce konvexní obálky.*
- *Ošetření singulárních případů při generování konvexní obálky.*
- *Načtení vstupních dat ze *.shp.*

3 Popis problému

Problém generalizace budov do úrovně detailu LOD0 spočívá v zjednodušení tvaru polygonů na obdélníky za účelem zmenšení množství dat a zlepšení čitelnosti v mapě.

3.1 Formulace problému

Dáno:

- Množina n budov $B = \{B_i\}$, kde každá budova B_i je reprezentována množinou lomových bodů $\{P_{i,j}\}$.

Určováno:

- Generalizovaný tvar budovy $G(B_i)$ do úrovně detailu LOD0.

3.2 Techniky řešení problému

- **Minimum Area Enclosing Rectangle** – Tato metoda určuje obdélník s nejmenší plochou, který obsahuje všechny body budovy. Pro určení tohoto obdélníku se využívá konstrukce konvexní obálky. Nalezený obdélník je následně zmenšen, aby měl stejný obsah jako původní polygon budovy a těžiště zůstalo na stejném míste.
- **PCA (Principal Component Analysis)** – Metoda, která určuje hlavní směry budovy pomocí analýzy hlavních komponent. Výsledný obdélník je natočený podle hlavních směrů budovy a má obsah shodný s obsahem budovy. Těžiště obdélníku je budovy je shodné s těžištěm polygonu.
- **Longest Edge** – Metoda, která určuje natočení obdélníku na základě nejdelší hrany polygonu. Obsah obdélníku je shodný s obsahem polygonu. Těžiště obdélníku je budovy je shodné s těžištěm polygonu.
- **Wall Average** – Metoda, která určuje natočení obdélníku na základě průměrné orientace všech hran polygonu. Velikost obdélníku je volena tak, aby měl stejný obsah jako původní polygon. Těžiště obdélníku je budovy je shodné s těžištěm polygonu.
- **Weighted Bisector** – Metoda, která určuje natočení obdélníku na základě váženého průměru orientací hran polygonu. Váhy jsou přiřazeny jednotlivým hranám podle jejich délky. Velikost obdélníku je volena tak, aby měl obdélník obsah shodný s polygonem. Těžiště obdélníku je budovy je shodné s těžištěm polygonu.

4 Popis algoritmů

4.1 Algoritmy pro tvorbu konvexní obálky

4.1.1 Jarvis scan

Jarvisův sken, známý také jako Gift Wrapping Algorithm, je algoritmus sloužící ke konstrukci konvexní obálky z množiny bodů v rovině. Princip spočívá v postupném „obalování“ bodů – algoritmus vždy přidává takový bod, který tvoří s aktuálním směrem největší úhel.

Nejprve se začne vybráním pivotu q, bodu s nejmenší y souřadnicí:

$$q = \min(y_i) \tag{1}$$

Následně se vybere bod r s nejmenší x souřadnicí:

$$r = \min(x_i) \tag{2}$$

Jou inicializovány body p_j a p_{j1}, p_{j1} tvoří posunutá x souřadnice bodu r a y souřadnice bodu y:

$$p_j = q \tag{3}$$

$$p_{j1}(r.x - 1, q.y) \tag{4}$$

Pivot q je přidán do do konvexní obálky ch:

$$ch.\text{push\_back}(q) \tag{5}$$

Následně se projdou všechny body a najde se takový, který tvoří svírá s pj a p_{j-1} maximální úhel. Výběr je provázen pouze u bodů, které již nejsou v ch:

$$p_{j+1} = \max\angle(p_{j-1}, p_j, p_i) \tag{6}$$

Bod je následně do konvexní obálky:

$$ch.\text{push\_back}(p_{j+1}) \tag{7}$$

Metoda `Jarvis Scan`

```
Vstup: polygon P
Výstup: konvexní obal polygonu P
CH ← ∅
if size(P) = 3 then
    return P
end if
q ← min(P) podle y-ové souřadnice
r ← min(P) podle x-ové souřadnice
p_j ← q
p_j1 ← (r_x − 1, q_y)
přidej  p_j do CH
repeat
    ω_max ← 0
    i_max ← −1
    for i ← 0 to size(P) − 1 do
        ω ← get2LinesAngle(p_j, p_j1, p_j, P[i])
        if ω > ω_max then
            ω_max ← ω
            i_max ← i
        end if
    end for
    přidej P[i_max] do CH
    p_j1 ← p_j
    p_j ← P[i_max]
until p_j = q
return CH
```

4.1.2 Graham scan

Grahamův scan je algoritmus, který vytváří konvexní obálku množiny bodů. Algoritmus začíná dvěma počátečními body p_1 a p_2. Pro každý kandidátský bod k se zjišťuje, zda leží vlevo od přímky tvořené body p_1 a p_2. Pokud ano, bod k se přidá do konvexní obálky. Pokud však leží vpravo od této přímky, bod p_2 je z obálky odebrán a nahrazen kandidátem k.

Nejprve je nalezen bod q který má minimální y souřadnici:

$$q = \min(y_i) \tag{8}$$

Pokud je více bodů, které mají stejnou y souřadnici, vybere se bod, který má menší x souřadnici:

$$\text{if pocet } q > 1 \text{ then } q = min(x_q) \tag{9}$$

Následně je vypočítán od přímky která tvoří je rovnoběžná s osou x a prochází bodem q úhel ke všem ostatním bodům v množině:

$$\omega_i = \angle(q.x + 1, q, p_i) \tag{10}$$

Všechny body p_i jsou seřazeny sestupně podle úhlů ω_i vzhledem k pivotu q, kde platí:

$$p = \text{sort}(\omega, \downarrow) \tag{11}$$

Pokud mají dva body stejný úhel $\omega_i = \omega_j$, je ponechán pouze ten, který má větší vzdálenost od pivota q, tedy ten pro nějž platí.

Následně jsou do konvexní obálky přidány body q,p_1,p_2:

$$ch.\text{push\_back}(q, p_1, p_2) \tag{12}$$

Projdou se všechny zbylé body a pokud p_i je vlevo od přímky tvořenou p_{i-1} a p_{i-2}. Je přidán do ch:

$$ch.\text{push\_back}(p_i) \tag{13}$$

Pokud je pokud bod p_i vpravo od přímky tvořenou p_{i-1} a p_{i-2}. Je odebrán bod p_{i-1} a přidán p_i:

$$ch.\text{pop\_back}(p_{i-1}) \tag{14}$$

$$ch.\text{push\_back}(p_i) \tag{15}$$

Metoda `createCHGS`

Vstup: polygon P
Výstup: konvexní obal polygonu P
CH $\leftarrow \emptyset$
if size(P) $= 3$ **then**
  return P
end if
$q \leftarrow$ findPivotGS(P) {nejnižší bod podle y}
$P' \leftarrow P \setminus \{q\}$ {odstraň pivot z množiny bodů}
$\alpha \leftarrow$ anglesWithPoints(P', q) {úhly vůči pivotu}
sortAnglesPoints(q, α, P') {seřazení bodů podle úhlu}
přidej q, $P'[0]$, $P'[1]$ do CH
for $i \leftarrow 2$ **to** size(P') $- 1$ **do**
  $c \leftarrow P'[i]$
  while size(CH) ≥ 2 **and** findSide(CH_{-2}, CH_{-1}, c) $= -1$ **do**
    odstraň poslední bod z CH
  end while
  přidej c do CH
end for
return CH

Metoda `findPivotGS` – hledání pivotu pro Graham Scan

Vstup: polygon P
Výstup: pivotní bod q s nejmenší y souřadnicí (případně nejmenší x)
$q \leftarrow P[0]$
$q_y \leftarrow$ y-ová souřadnice bodu q
for $i \leftarrow 1$ **to** size(P) $- 1$ **do**
  $c \leftarrow P[i]$
  $c_y \leftarrow$ y-ová souřadnice bodu c
  if $c_y \leq q_y$ **then**
    if $c_y = q_y$ **then**
      $c_x \leftarrow$ x-ová souřadnice bodu c
      $q_x \leftarrow$ x-ová souřadnice bodu q
      if $c_x < q_x$ **then**
        $q \leftarrow c$
        $q_y \leftarrow c_y$
      end if
    else
      $q \leftarrow c$
      $q_y \leftarrow c_y$
    end if
  end if
end for
return q

Metoda `sortAnglesPoints` – třídění bodů podle úhlů od pivotu

Vstup: pivotní bod q, vektor úhlů α, polygon P'
Výstup: seřazené a vyfiltrované α a P'
vytvoř vektor indexů $I \leftarrow [0, 1, \ldots, n-1]$
vytvoř vektor indexů α_{sorted}
$P'_{\text{sorted}} \leftarrow []$
if I není prázdný **then**
  přidej $\alpha[I_0]$ do α_{sorted}
  přidej $P'[I_0]$ do P'_{sorted}
  for $i \leftarrow 1$ **to** size$(I) - 1$ **do**
    $i_{\text{cur}} \leftarrow I[i]$
    $i_{\text{prev}} \leftarrow I[i-1]$
    if $\alpha[i_{\text{cur}}] \neq \alpha[i_{\text{prev}}]$ **then**
      přidej $\alpha[i_{\text{cur}}]$ do α_{sorted}
      přidej $P'[i_{\text{cur}}]$ do P'_{sorted}
    else
      $d_{\text{cur}} \leftarrow \text{vzdálenost}(P'[i_{\text{cur}}], q)$
      $d_{\text{prev}} \leftarrow \text{vzdálenost}(P'_{\text{sorted}}.\text{last}, q)$
      if $d_{\text{cur}} > d_{\text{prev}}$ **then**
        nahraď poslední bod v P'_{sorted} bodem $P'[i_{\text{cur}}]$
      end if
    end if
  end for
end if
$\alpha \leftarrow \alpha_{\text{sorted}}$
$P' \leftarrow P'_{\text{sorted}}$

4.2 Algoritmy pro generalizaci budov

4.2.1 Minimum Area Enclosing Rectangle

Tento algoritmus slouží k nalezení obdélníku s nejmenší plochou, který obsahuje všechny body daného polygonu. Využívá se přitom konstrukce konvexní obálky, která se postupně otáčí podle orientace jednotlivých jejích hran. Pro každou rotaci se vypočítá min-max box a vybírá se ten s nejmenší plochou. Po nalezení min-max boxu s nejmenší plochou se tento min-max box rotuje zpět do původní orientace. Výsledný obdélník je dále upraven tak, aby měl stejný obsah jako původní polygon a zachoval jeho těžiště[1].

Nejprve se vytvoří konvexní obálka CH z množiny bodů budovy P:

$$CH = \text{createCH}(P) \tag{16}$$

Pro každou hranu konvexní obálky mezi body p_i a p_{i+1} se vypočítá úhel rotace σ:

$$\sigma = \arctan 2(y_{i+1} - y_i, x_{i+1} - x_i) \tag{17}$$

Konvexní obálka se otočí o úhel $-\sigma$:

$$CH_{\text{rot}} = \mathbf{R}(-\sigma) CH \tag{18}$$

Kde $\mathbf{R}$(-σ) je rotační matice:

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

Po rotaci se vypočítá min-max box a jeho plocha:

$$\text{MMBox} = \left\{ \begin{bmatrix} x_{\min} \\ y_{\min} \end{bmatrix}, \begin{bmatrix} x_{\max} \\ y_{\min} \end{bmatrix}, \begin{bmatrix} x_{\max} \\ y_{\max} \end{bmatrix}, \begin{bmatrix} x_{\min} \\ y_{\max} \end{bmatrix} \right\} \tag{19}$$

$$\text{area} = (x_{\max} - x_{\min})(y_{\max} - y_{\min}) \tag{20}$$

Pokud je plocha menší než aktuálně nejmenší nalezená:

$$\text{if area} < \text{area}_{\min} \text{ then area}_{\min} = \text{area}, \quad \sigma_{\min} = \sigma, \quad \text{MMBox}_{\min} = \text{MMBox} \tag{21}$$

Po projití všech hran se spočítá plocha původního polygonu pomocí L'Huilierových vzorců:

$$A_P = \frac{1}{2} \left| \sum_{i=1}^{n} x_i (y_{i+1} - y_{i-1}) \right| \tag{22}$$

Min-max box se zmenší tak, aby měl stejný obsah jako původní polygon budovy A_P. Nejprve se spočítá těžiště:

$$\mathbf{c} = \frac{1}{4} \sum_{j=1}^{4} \mathbf{v}_j, \quad \mathbf{v}_j \in \text{MMBox}_{\min}$$

Pak se každý bod redukuje k těžišti:

$$\mathbf{q}_j = \mathbf{c} + \sqrt{\frac{A_P}{\text{area}_{\min}}} \cdot (\mathbf{v}_j - \mathbf{c}) \tag{23}$$

Výsledný zmenšený box:

$$\text{MMBox}_{\text{res}} = \{\mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_3, \mathbf{q}_4\}$$

Nakonec se box otočí zpět o úhel $\sigma_{\min}$:

$$\text{MAER} = \{\mathbf{R}(\sigma_{\min}) \cdot \mathbf{q}_j \mid j = 1, \ldots, 4\} \tag{24}$$

Metoda `Minimum Area Enclosing Rectangle`

Vstup: polygon P
Výstup: minimální obdélník obsahující všechny body polygonu P
$\sigma_{\min} \leftarrow 2\pi$
$\text{mmbox\_min}, \text{area\_min} \leftarrow \text{minMaxBox}(P)$
$CH \leftarrow \text{createCH}(P)$
$n \leftarrow \text{size}(CH)$
for $i \leftarrow 0$ **to** $n - 1$ **do**
  $dx \leftarrow x_{i+1} - x_i$
  $dy \leftarrow y_{i+1} - y_i$
  $\sigma \leftarrow \arctan 2(dy, dx)$
  $CH_{\text{rot}} \leftarrow \text{rotate}(CH, -\sigma)$
  $\text{mmbox}, \text{area} \leftarrow \text{minMaxBox}(CH_{\text{rot}})$
  if $\text{area} < \text{area\_min}$ **then**
    $\text{area\_min} \leftarrow \text{area}$
    $\sigma_{\min} \leftarrow \sigma$
    $\text{mmbox\_min} \leftarrow \text{mmbox}$
  end if
end for
$\text{mmbox\_min\_res} \leftarrow \text{resize}(P, \text{mmbox\_min})$
return $\text{rotate}(\text{mmbox\_min\_res}, \sigma_{\min})$

4.2.2 Principal Component Analysis (PCA)

Metoda, která určuje hlavní směry budovy pomocí analýzy hlavních komponent. Výsledný obdélník je natočený podle hlavních směrů budovy a má obsah shodný s obsahem budovy. Těžiště obdélníku je shodné s těžištěm polygonu. Pro výpočet hlavních komponent je využita knihovna eigen[2].

Nejprve se vytvoří matice $\mathbf{A}$ z množiny bodů budovy P:

$$\mathbf{A} = \begin{pmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \vdots & \vdots \\ x_n & y_n \end{pmatrix} \tag{25}$$

Vypočítají se průměry souřadnic přes sloupce:

$$\mathbf{M} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{A}_i \tag{26}$$

Od matice $\mathbf{A}$ se odečtou průměry $\mathbf{M}$, čímž vznikne matice $\mathbf{B}$:

$$\mathbf{B} = \mathbf{A} - \mathbf{M} \tag{27}$$

Vypočítá se kovarianční matice $\mathbf{C}$:

$$\mathbf{C} = \frac{\mathbf{B}^T \mathbf{B}}{n-1} \tag{28}$$

Provede se singulární rozklad (SVD) kovarianční matice $\mathbf{C}$:

$$\mathbf{C} = \mathbf{U} \mathbf{\Sigma} \mathbf{V}^T \tag{29}$$

kde $\mathbf{U}$ a $\mathbf{V}$ jsou matice vlastních vektorů a $\mathbf{\Sigma}$ je diagonální matice singulárních hodnot.
Úhel rotace σ se vypočítá jako:

$$\sigma = \arctan 2(\mathbf{V}_{1,0}, \mathbf{V}_{0,0}) \tag{30}$$

Polygon P se otočí o úhel $-\sigma$:

$$P_{\text{rot}} = \mathbf{R}(-\sigma) P \tag{31}$$

Po rotaci se vypočítá min-max box a jeho plocha:

$$\text{MMBox} = \left\{ \begin{bmatrix} x_{\min} \\ y_{\min} \end{bmatrix}, \begin{bmatrix} x_{\max} \\ y_{\min} \end{bmatrix}, \begin{bmatrix} x_{\max} \\ y_{\max} \end{bmatrix}, \begin{bmatrix} x_{\min} \\ y_{\max} \end{bmatrix} \right\} \tag{32}$$

$$\text{area} = (x_{\max} - x_{\min})(y_{\max} - y_{\min}) \tag{33}$$

Min-max box se zmenší tak, aby měl stejný obsah jako původní polygon budovyA_P. Nejprve se spočítá těžiště:

$$\mathbf{c} = \frac{1}{4} \sum_{j=1}^{4} \mathbf{v}_j, \quad \mathbf{v}_j \in \text{MMBox}_{\min}$$

Pak se každý bod redukuje k těžišti:

$$\mathbf{q}_j = \mathbf{c} + \sqrt{\frac{A_P}{\text{area}_{\min}}} \cdot (\mathbf{v}_j - \mathbf{c}) \tag{34}$$

Výsledný zmenšený box:

$$\text{MMBox}_{\text{res}} = \{\mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_3, \mathbf{q}_4\}$$

Nakonec se box otočí zpět o úhel σ:

$$\text{PCA} = \{\mathbf{R}(\sigma) \cdot \mathbf{q}_j \mid j = 1, \ldots, 4\} \tag{35}$$

Metoda `PCA (Principal Component Analysis)`

Vstup: polygon P
Výstup: obdélník obsahující všechny body polygonu P pomocí PCA
$n \leftarrow \text{size}(P)$
$\mathbf{A} \leftarrow \text{matrix}(n, 2)$
for $i \leftarrow 0$ **to** $n-1$ **do**
  $\mathbf{A}(i,0) \leftarrow P[i].x$
  $\mathbf{A}(i,1) \leftarrow P[i].y$
end for
$\mathbf{M} \leftarrow \text{mean}(\mathbf{A})$
$\mathbf{B} \leftarrow \mathbf{A} - \mathbf{M}$
$\mathbf{C} \leftarrow \frac{\mathbf{B}^T\mathbf{B}}{n-1}$
$\mathbf{U}, \mathbf{\Sigma}, \mathbf{V} \leftarrow \text{SVD}(\mathbf{C})$
$\sigma \leftarrow \arctan 2(\mathbf{V}_{1,0}, \mathbf{V}_{0,0})$
$P_{\text{rot}} \leftarrow \text{rotate}(P, -\sigma)$
$\text{mmbox}, \text{area} \leftarrow \text{minMaxBox}(P_{\text{rot}})$
$\text{mmbox\_min\_res} \leftarrow \text{resize}(P, \text{mmbox})$
return $\text{rotate}(\text{mmbox\_min\_res}, \sigma)$

4.2.3 Longest Edge

Metoda, která určuje natočení obdélníku na základě nejdelší hrany polygonu. Obsah obdélníku je shodný s obsahem polygonu. Těžiště obdélníku je shodné s těžištěm polygonu.
Nejprve se najde nejdelší hrana polygonu P:

$$\text{maxLength} = 0 \tag{36}$$

Pro každou hranu mezi body p_i a p_{i+1} se vypočítají rozdíly souřadnic:

$$dx_i = x_{i+1} - x_i \tag{37}$$

$$dy_i = y_{i+1} - y_i \tag{38}$$

Délka hrany se vypočítá jako:

$$\text{length} = \sqrt{dx_i^2 + dy_i^2} \tag{39}$$

Pokud je délka hrany větší než maxLength, aktualizuje se maxLength a směrové rozdíly dx a dy:

$$\text{if length} > \text{maxLength then} \tag{40}$$

$$\text{maxLength} = \text{length} \tag{41}$$

$$dx = dx_i \tag{42}$$

$$dy = dy_i \tag{43}$$

Úhel rotace σ se vypočítá jako:

$$\sigma = \arctan 2(dy, dx) \tag{44}$$

Polygon P se otočí o úhel $-\sigma$:

$$P_{\text{rot}} = \mathbf{R}(-\sigma)P \tag{45}$$

Po rotaci se vypočítá min-max box a jeho plocha:

$$\text{MMBox} = \left\{ \begin{bmatrix} x_{\min} \\ y_{\min} \end{bmatrix}, \begin{bmatrix} x_{\max} \\ y_{\min} \end{bmatrix}, \begin{bmatrix} x_{\max} \\ y_{\max} \end{bmatrix}, \begin{bmatrix} x_{\min} \\ y_{\max} \end{bmatrix} \right\} \tag{46}$$

$$\text{area} = (x_{\max} - x_{\min})(y_{\max} - y_{\min}) \tag{47}$$

Min-max box se zmenší tak, aby měl stejný obsah jako původní polygon budovyA_P. Nejprve se spočítá těžiště:

$$\mathbf{c} = \frac{1}{4} \sum_{j=1}^{4} \mathbf{v}_j, \quad \mathbf{v}_j \in \text{MMBox}_{\min} \tag{48}$$

Pak se každý bod redukuje k těžišti:

$$\mathbf{q}_j = \mathbf{c} + \sqrt{\frac{A_P}{\text{area}_{\min}}} \cdot (\mathbf{v}_j - \mathbf{c}) \tag{49}$$

Výsledný zmenšený box:

$$\text{MMBox}_{\text{res}} = \{\mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_3, \mathbf{q}_4\} \tag{50}$$

Nakonec se box otočí zpět o úhel σ:

$$\text{LE} = \{\mathbf{R}(\sigma) \cdot \mathbf{q}_j \mid j = 1, \ldots, 4\} \tag{51}$$

Metoda `Longest Edge`

Vstup: polygon P
Výstup: obdélník obsahující všechny body polygonu P pomocí nejdelší hrany
maxLength $\leftarrow 0$
$dx \leftarrow 0$, $dy \leftarrow 0$
$n \leftarrow \text{size}(P)$
for $i \leftarrow 0$ **to** $n - 1$ **do**
    $dx_i \leftarrow x_{i+1} - x_i$
    $dy_i \leftarrow y_{i+1} - y_i$
    length $\leftarrow \sqrt{dx_i^2 + dy_i^2}$
    if length $>$ maxLength **then**
        maxLength $\leftarrow$ length
        $dx \leftarrow dx_i$
        $dy \leftarrow dy_i$
    end if
end for
$\sigma \leftarrow \arctan 2(dy, dx)$
$P_{\text{rot}} \leftarrow \text{rotate}(P, -\sigma)$
mmbox, area $\leftarrow \text{minMaxBox}(P_{\text{rot}})$
mmbox_min_res $\leftarrow$ resize(P, mmbox)
return rotate(mmbox_min_res, σ)

4.2.4 Wall Average

Na každou stranu budovy je aplikována operace mod$(\pi/2)$. Ze zbytků hodnot je spočten vážený průměr, kde váhou je délka strany. Tato metoda je robustní, avšak citlivá na nepravé úhly. Nejprve jsou určeny směrnice σ_i všech hran, poté jsou spočteny vnitřní úhly ω_i.
Nejprve se vypočítá počáteční směr $\sigma_{1,2}$:

$$\sigma_{1,2} = \arctan 2(y_2 - y_1, x_2 - x_1) \tag{52}$$

Pro každý vrchol p_i se vypočítají směrnice σ_i a σ_{i+1}:

$$\sigma_i = \arctan 2(y_i - y_{i-1}, x_i - x_{i-1}) \tag{53}$$

$$\sigma_{i+1} = \arctan 2(y_{i+1} - y_i, x_{i+1} - x_i) \tag{54}$$

Délka strany s_i je:

$$s_i = \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2} \tag{55}$$

Vnitřní úhel ω_i je:

$$\omega_i = |\sigma_i - \sigma_{i+1}| \tag{56}$$

Násobek $\pi/2$ je:

$$k_i = \frac{2\omega_i}{\pi} \tag{57}$$

Orientovaný zbytek po dělení je:

$$r_i = (k_i - \lfloor k_i \rfloor)\frac{\pi}{2} \tag{58}$$

Pokud $\omega_i \mod \pi/2 > \pi/4$, pak:

$$r_i = \frac{\pi}{2} - r_i \tag{59}$$

Hlavní směr budovy σ je:

$$\sigma = \sigma_{1,2} + \frac{\sum_{i=1}^{n} r_i s_i}{\sum_{i=1}^{n} s_i} \tag{60}$$

Dále je postup obdobný jako u předchozích metod.

Metoda `Wall Average`

Vstup: polygon P
Výstup: obdélník představující generalizaci polygonu
$n \leftarrow \text{size}(P)$
$\sigma \leftarrow 0$
$S_i\_sum \leftarrow 0$
$dx \leftarrow x_1 - x_0$
$dy \leftarrow y_1 - y_0$
$\sigma_{12} \leftarrow \arctan 2(dy, dx)$
for $i \leftarrow 0$ **to** $n - 1$ **do**
    $dx1 \leftarrow x_{i-1} - x_i$
    $dy1 \leftarrow y_{i-1} - y_i$
    $\sigma_i \leftarrow \arctan 2(dy1, dx1)$
    $dx2 \leftarrow x_{i+1} - x_i$
    $dy2 \leftarrow y_{i+1} - y_i$
    $\sigma_{i+1} \leftarrow \arctan 2(dy2, dx2)$
    $S_i \leftarrow \sqrt{dx2^2 + dy2^2}$
    $\omega_i \leftarrow |\sigma_i - \sigma_{i+1}|$
    $k_i \leftarrow \frac{2\omega_i}{\pi}$
    $r_i \leftarrow (k_i - \lfloor k_i \rfloor)\frac{\pi}{2}$
    if $\omega_i \mod \pi/2 > \pi/4$ **then**
        $r_i \leftarrow \frac{\pi}{2} - r_i$
    end if
    $\sigma \leftarrow \sigma + r_i S_i$
    $S_i\_sum \leftarrow S_i\_sum + S_i$
end for
$\sigma \leftarrow \sigma_{12} + \frac{\sigma}{S_i\_sum}$
$P_{\text{rot}} \leftarrow \text{rotate}(P, -\sigma)$
$\text{mmbox}, \text{area} \leftarrow \text{minMaxBox}(P_{\text{rot}})$
$\text{mmbox\_min\_res} \leftarrow \text{resize}(P, \text{mmbox})$
return $\text{rotate}(\text{mmbox\_min\_res}, \sigma)$

4.2.5 Weighted Bisector

Metoda, která určuje natočení obdélníku na základě váženého průměru orientací hran polygonu. Váhy jsou přiřazeny jednotlivým hranám podle jejich délky. Velikost obdélníku je volena tak, aby měl obdélník obsah shodný s polygonem. Těžiště obdélníku je shodné s těžištěm polygonu.
Nejprve se najdou dvě nejdelší úhlopříčky polygonu P:

$$\text{max\_diag\_1} = 0, \quad \text{max\_diag\_2} = 0 \tag{61}$$

Pro každou dvojici vrcholů p_i a p_j se vypočítají rozdíly souřadnic:

$$dx_i = x_j - x_i \tag{62}$$

$$dy_i = y_j - y_i \tag{63}$$

Délka úhlopříčky se vypočítá jako:

$$\text{len}_i = \sqrt{dx_i^2 + dy_i^2} \tag{64}$$

Pokud je délka úhlopříčky větší než max_diag_1, aktualizuje se max_diag_2 a max_diag_1 a směrové rozdíly dx a dy:

$$\text{if } \text{len}_i > \text{max\_diag\_1} \text{ then} \tag{65}$$

$$\text{max\_diag\_2} = \text{max\_diag\_1} \tag{66}$$

$$\text{max\_diag\_1} = \text{len}_i \tag{67}$$

$$dx_2 = dx_1, \quad dy_2 = dy_1 \tag{68}$$

$$dx_1 = dx_i, \quad dy_1 = dy_i \tag{69}$$

Pokud je délka úhlopříčky větší než max_diag_2 a menší než max_diag_1, aktualizuje se max_diag_2 a směrové rozdíly dx a dy:

$$\text{else if } \text{len}_i > \text{max\_diag\_2} \text{ and } \text{len}_i < \text{max\_diag\_1} \text{ then} \tag{70}$$

$$\text{max\_diag\_2} = \text{len}_i \tag{71}$$

$$dx_2 = dx_i, \quad dy_2 = dy_i \tag{72}$$

Směr nejdelších úhlopříček se vypočítá jako:

$$\sigma_1 = \arctan 2(dy_1, dx_1) \tag{73}$$

$$\sigma_2 = \arctan 2(dy_2, dx_2) \tag{74}$$

Vážený průměr směru úhlopříček se vypočítá jako:

$$\sigma = \frac{\sigma_1 \cdot \text{max\_diag\_1} + \sigma_2 \cdot \text{max\_diag\_2}}{\text{max\_diag\_1} + \text{max\_diag\_2}} \tag{75}$$

Dále je postup obdobný jako u předchozích metod.

Metoda `Weighted Bisector`

Vstup: polygon P
Výstup: obdélník představující generalizaci polygonu
$\text{max\_diag\_1} \leftarrow 0$, $\text{max\_diag\_2} \leftarrow 0$
$dx_1 \leftarrow 0$, $dy_1 \leftarrow 0$, $dx_2 \leftarrow 0$, $dy_2 \leftarrow 0$
$n \leftarrow \text{size}(P)$
for $i \leftarrow 0$ **to** $n-1$ **do**
  for $j \leftarrow 0$ **to** $n-1$ **do**
    $dx_i \leftarrow x_{i+j+2} - x_i$
    $dy_i \leftarrow y_{i+j+2} - y_i$
    $\text{len}_i \leftarrow \sqrt{dx_i^2 + dy_i^2}$
    if $\text{len}_i > \text{max\_diag\_1}$ **then**
      $\text{max\_diag\_2} \leftarrow \text{max\_diag\_1}$
      $\text{max\_diag\_1} \leftarrow \text{len}_i$
      $dx_2 \leftarrow dx_1$, $dy_2 \leftarrow dy_1$
      $dx_1 \leftarrow dx_i$, $dy_1 \leftarrow dy_i$
    else if $\text{len}_i > \text{max\_diag\_2}$ and $\text{len}_i < \text{max\_diag\_1}$ **then**
      $\text{max\_diag\_2} \leftarrow \text{len}_i$
      $dx_2 \leftarrow dx_i$, $dy_2 \leftarrow dy_i$
    end if
  end for
end for
$\sigma_1 \leftarrow \arctan 2(dy_1, dx_1)$
$\sigma_2 \leftarrow \arctan 2(dy_2, dx_2)$
$\sigma \leftarrow \frac{\sigma_1 \cdot \text{max\_diag\_1} + \sigma_2 \cdot \text{max\_diag\_2}}{\text{max\_diag\_1} + \text{max\_diag\_2}}$
$P_{\text{rot}} \leftarrow \text{rotate}(P, -\sigma)$
$\text{mmbox}, \text{area} \leftarrow \text{minMaxBox}(P_{\text{rot}})$
$\text{mmbox\_min\_res} \leftarrow \text{resize}(P, \text{mmbox})$
return $\text{rotate}(\text{mmbox\_min\_res}, \sigma)$

5 Problematické situace

5.1 Souřadnice shapefilu

Při nahrávání a vykreslování polygonů ze shapefile je nutné provést transformaci souřadnic z geografického souřadnicového systému do souřadnicového systému okna aplikace.

- **Načtení hranic shapefile**
  Nejprve se pomocí funkce `SHPGetInfo` zjistí minimální a maximální souřadnice polygonů obsažených ve shapefile (bounding box).

- **Zjištění rozměrů okna aplikace**
  Rozměry vykreslovací plochy se zjistí funkcemi `width()` a `height()`. Tyto hodnoty definují maximální prostor, který může být využit pro zobrazení polygonů.

- **Výpočet měřítka transformace**
  Aby se polygon přizpůsobil velikosti vykreslovací oblasti vypočte se měřítkový koeficient:

$$\text{scale} = \min\left(\frac{\text{widgetWidth}}{\text{maxX} - \text{minX}}, \frac{\text{widgetHeight}}{\text{maxY} - \text{minY}}\right) \tag{76}$$

- **Výpočet posunu polygonu**
  Po aplikaci měřítka je třeba polygon zarovnat do středu vykreslovací plochy. Posun v ose X a Y se vypočte následovně:

$$\text{offsetX} = \frac{\text{widgetWidth} - (\text{maxX} - \text{minX}) \cdot \text{scale}}{2} - \text{minX} \cdot \text{scale} \tag{77}$$

$$\text{offsetY} = \frac{\text{widgetHeight} - (\text{maxY} - \text{minY}) \cdot \text{scale}}{2} + \text{maxY} \cdot \text{scale} \tag{78}$$

- **Transformace jednotlivých bodů**
  Každý bod X a Y v polygonu se transformuje následujícím způsobem:

$$\text{transformedX} = \text{x} \cdot \text{scale} + \text{offsetX} \tag{79}$$

$$\text{transformedY} = -\text{y} \cdot \text{scale} + \text{offsetY} \tag{80}$$

- **Uložení transformovaných bodů**
  Transformované body jsou uloženy do datové struktury pro polygon.

6 Vstupní data

Data pro analýzu lze do aplikace získat manuálním zadáním, popřípadě načtením z textového souboru, nebo ze souboru shapefile.

6.1 Vstupní data od uživatele

Data jsou získány čtením souřadnic cursoru myši nad widgetem aplikace.

- Levé tlačítko vloží bod do polygonu.
- Dvojklik levého tlačítka ukončí polygon.

6.2 Načítání dat z textového souboru

Při načítání dat z textového souboru je třeba, aby soubor měl specifický formát. Každý bod v polygonu je reprezentován souřadnicemi x a y, oddělenými čárkou. Každý polygon je oddělen prázdným řádkem. Příklad formátu souboru:

```
100, 100
100, 200
200, 200
200, 100

100, 200
200, 200
200, 300
100, 300
```

V tomto formátu:

- Každá skupina souřadnic (například $100, 100$; $100, 200$; $200, 200$; $200, 100$) tvoří jeden polygon.
- Každý prázdný řádek mezi skupinami souřadnic označuje konec jednoho polygonu a začátek dalšího.

Před načtením jsou vymazány veškeré údaje, které jsou načtené, nebo vytvořené v aplikaci. Polygony z textového souboru jsou načítány do proměnné pro analýzu polygonů, tak do proměnné pro vykreslování polygonů.

6.3 Načítání dat z shapefile souboru

Načítání dat z formátu shapefile probíhá pomocí knihovny `shapelib`, přičemž je použita její upravená verze pro C++. Ještě před načtením dat jsou vymazány veškeré údaje, které jsou načtené nebo vytvořené v aplikaci. Funkce `loadPolygonFromShapefile` načítá soubor shapefile a každý objekt (tedy polygon) v souboru je přetvořen na seznam bodů. Následně jsou tyto body transformovány tak, aby se správně zobrazily v uživatelském rozhraní aplikace.

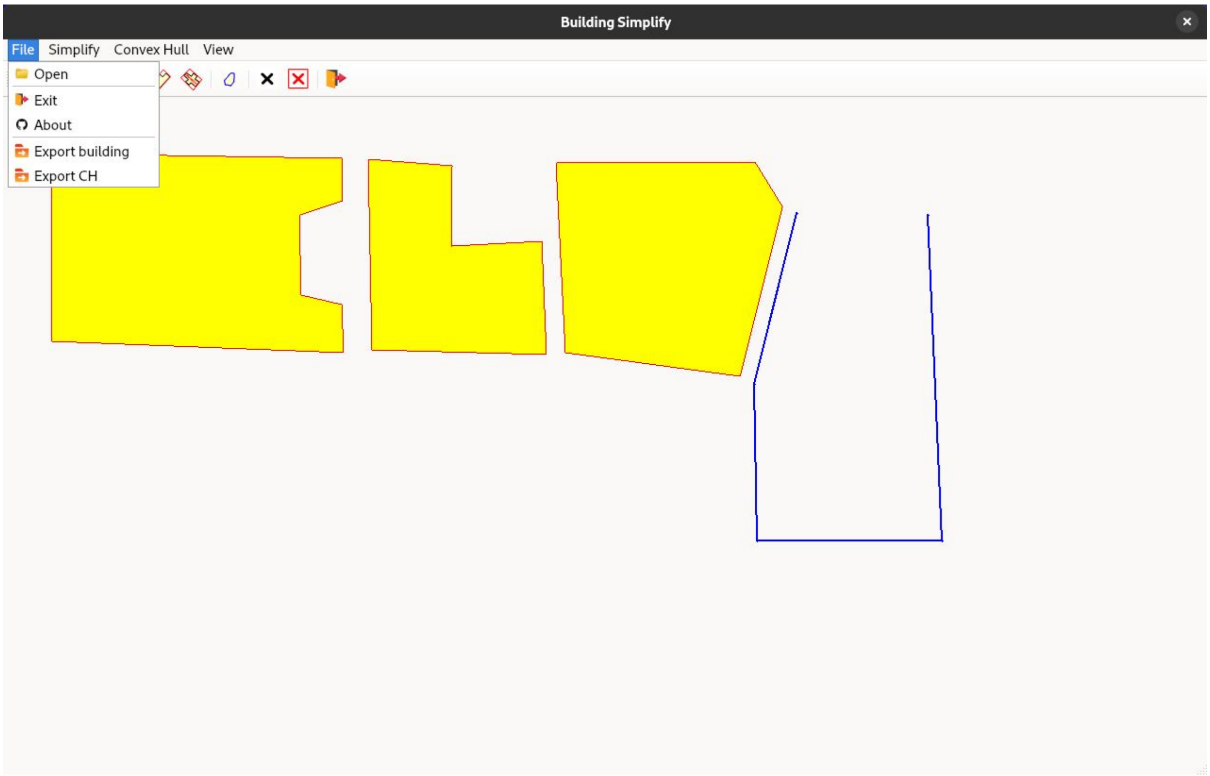
- Soubor je otevřen pro čtení pomocí funkce `SHPOpen`, která otevře shapefile soubor v režimu čtení binárního souboru `"rb"`. Pokud není soubor dostupný, aplikace zobrazuje chybovou hlášku.
- Po otevření shapefile souboru se načítají základní informace o počtu entit (polygonů), typu tvaru a souřadnicích hranic (bounding box) pomocí funkce `SHPGetInfo`. Pokud soubor neobsahuje žádné entity, aplikace zobrazí varovnou zprávu.
- Pro každý polygon se získají souřadnice jeho vrcholů pomocí funkce `SHPReadObject`, která načte polygon z shapefile souboru. Tyto souřadnice jsou následně transformovány, aby se správně zobrazily na widgetu aplikace. Změna měřítka škáluje polygony a posun je použit pro umístění polygonů na správné místo na obrazovce.
- Funkce `SHPDestroyObject` je volána po zpracování každého polygonu, aby se uvolnila paměť alokovaná pro daný objekt shapefile.
- Polygony jsou následně ukládány do struktury, která se skládá z vnějšího obvodu a případných děr.
- Po zpracování všech polygonů je soubor zavřen pomocí funkce `SHPClose`.

Pro ukázku byla použita data budov v centru města, panelovém sídlišti a ve vilové čtvrti získaná z RÚIAN (Registr územní identifikace, adres a nemovitostí)[3].

7 Výstupní data

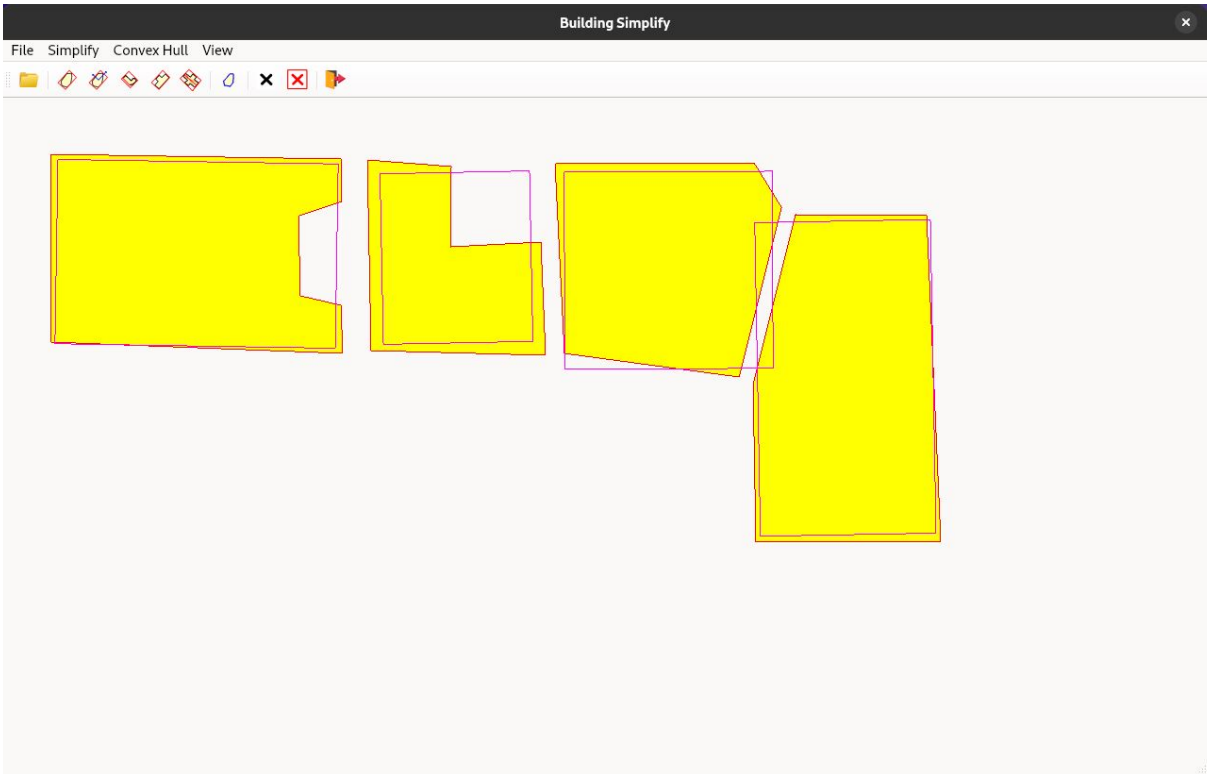
Výsledky jsou vizualizovány přímo v aplikaci. Původní budova je zobrazena červeným obrysem se žlutou výplní, zatímco generalizované budovy jsou zobrazeny růžovou barvou. Konvexní obálka je vykreslena tmavě zelenou čárkovanou linií a doplněna zelenou výplní. Zobrazení konvexní obálky lze dle potřeby vypnout či zapnout. Popřípadě nastavit viditelnost výplně Pro účely uložení výsledků je možné exportovat jak generalizované budovy, tak i konvexní obálku do textového souboru ve stejném formátu jako vstupní data.

8 Výsledná aplikace



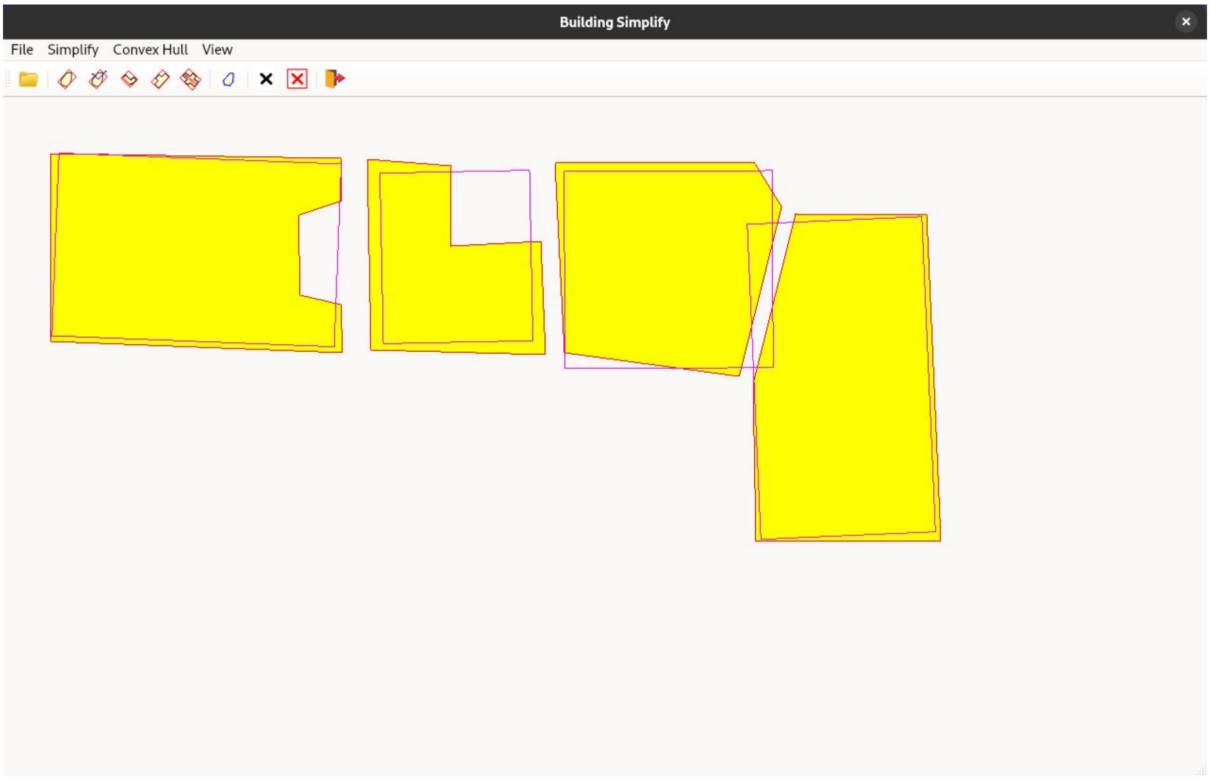
Obrázek 1: Ukázka vzhledu výsledné aplikace

Obrázek 2: Tvorba konvexní obálky

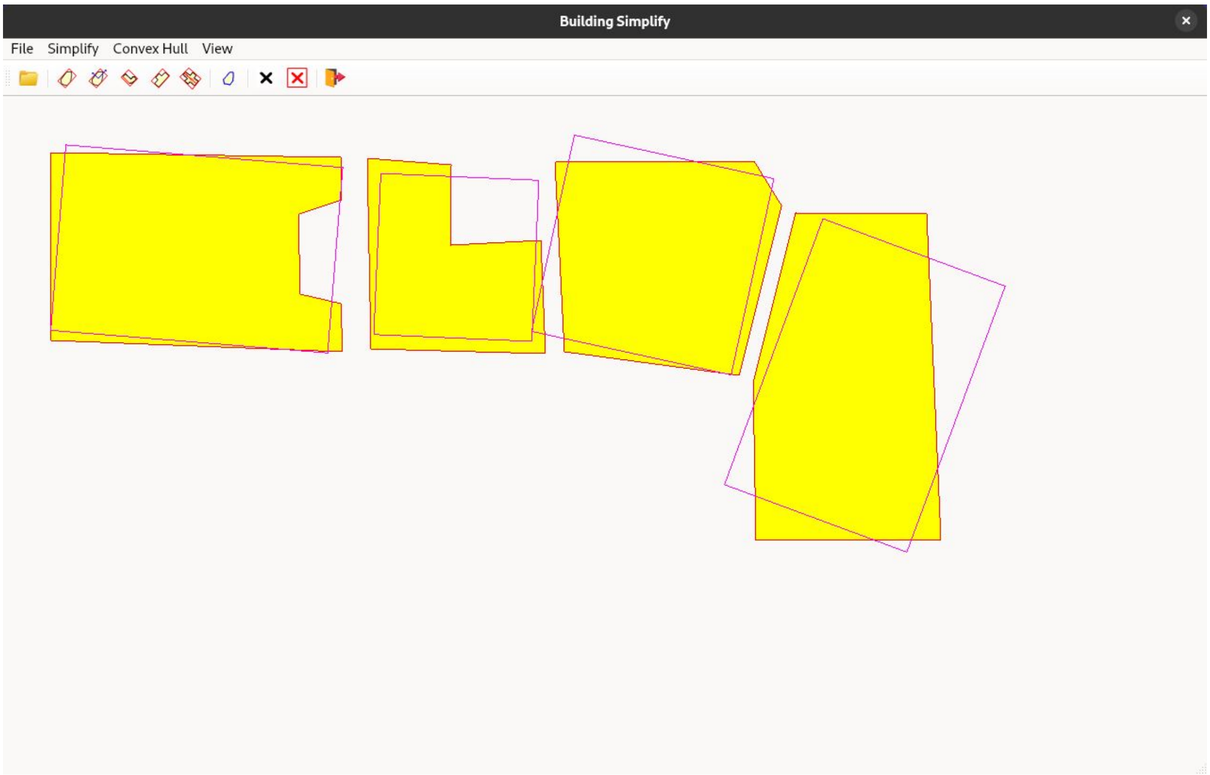


Obrázek 3: Generalizace metodou Minimum Area Enclosing Rectangle

Obrázek 4: Generalizace metodou PCA

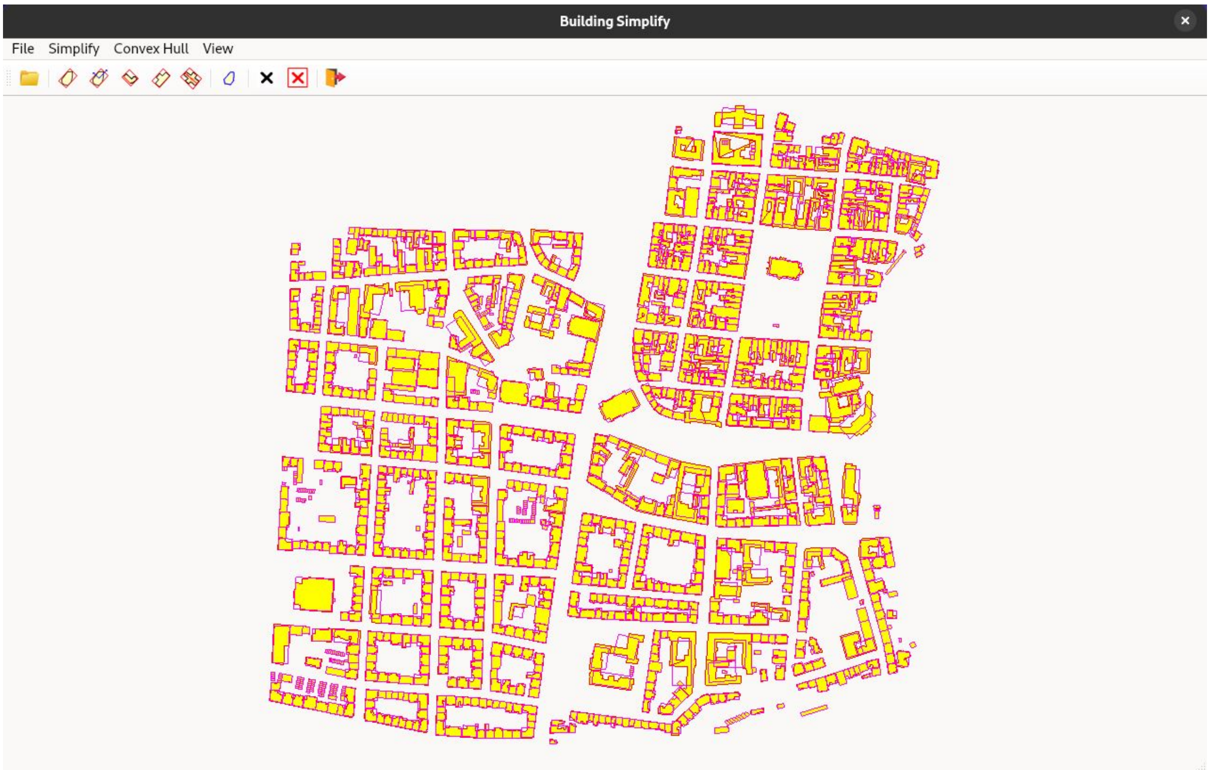


Obrázek 5: Generalizace metodou Longest Edge



Obrázek 6: Generalizace metodou Wall Average

Obrázek 7: Generalizace metodou Weighted Bisector



Obrázek 8: Generalizace budov v centru města metodou MAER

Obrázek 9: Generalizace budov panelového sídliště metodou MAER

Obrázek 10: Generalizace budov ve vilové čtvrti metodou MAER

9 Dokumentace

9.1 Použité knihovny

Kód využívá následující knihovny:

- `Qt` – Knihovna pro správu grafických objektů, včetně tříd jako `QPointF`, `QPolygonF`.
- `shapelib` – Knihovna pro práci se soubory formátu Shapefile [4]. Upravená verze této knihovny do C++ byla použita z projektu shapefile-viewer-qt od Zhihao Liu [5].
- `Eigen` – Knihovna pro lineární algebru v C++, včetně operací s maticemi a vektory[2].

9.2 Třída Algorithms

Třída poskytuje metody pro generalizaci bodů

Veřejné statické metody:

- `QPolygonF createMAER(const QPolygonF &pol)` – Generalizuje budovu pomocí metody Minimum Area Enclosing Rectangle.
- `QPolygonF createERPCA(const QPolygonF &pol)` – Generalizuje budovu pomocí metody Enclosing Rectangle using PCA.
- `QPolygonF createERLE(const QPolygonF &pol)` – Generalizuje budovu pomocí metody Enclosing Rectangle using Longest Edge.
- `QPolygonF createERWA(const QPolygonF &pol)` – Generalizuje budovu pomocí metody Enclosing Rectangle using Wall Average.
- `QPolygonF createERWB(const QPolygonF &pol)` – Generalizuje budovu pomocí metody Enclosing Rectangle using Weighted Bisector.
- `QPolygonF createCHJS(const QPolygonF &pol)` – Vytváří konvexní obálku pomocí motdy Jarvis Scan.
- `QPolygonF createCHGS(const QPolygonF &pol)` – Vytváří konvexní obálku pomocí motdy Graham Scan.
- `void exportFile(const std::vector<QPolygonF> &results,const QString &fileName)` – Vytvoří textový soubor výsledků.

Soukromé statické metody:

- `double get2LinesAngle(const QPointF &p1, const QPointF &p2, const QPointF &p3, const QPointF &p4)` – Vypočítá úhel mezi přímkou tvořenou body (`p1` a `p2`) a přímkou tvořenou body (`p3` a `p4`).
- `double getArea(const QPolygonF &pol)` – Vypočítá plochu polygonu.
- `QPolygonF resize(const QPolygonF &pol, const QPolygonF &mmbox)` – Změní velikost polygonu.
- `QPolygonF rotate(const QPolygonF &pol, double sigma)` – Otočí polygon o úhel `sigma`.
- `std::tuple<QPolygonF, double> minMaxBox(const QPolygonF &pol)` – Vytvoří Minimum Bounding Box daného polygonu.
- `double getDistance(const QPointF &p1, const QPointF &p2)` – Vypočítá euklidovskou vzdálenost mezi dvěma body (`p1` a `p2`).
- `QPointF findPivotGS(const QPolygonF &pol)` – Nalezne pivot pro Graham Scan.
- `std::vector<double> anglesWithPoints(const QPolygonF &pol, const QPointF &q)` – Vypočte úhel mezi body a pivotem.

- `void sortAnglesPoints(const QPointF &q, std::vector<double> &angles, QPolygonF &pol_)` – Seřadí úhly sestupně a korespondující body sestupně podle úhlu.

- `double pointLineDistance(const QPointF& A, const QPointF& B, const QPointF& P)` – Vypočte vzdálenost bodu od přímky.

- `short findSide(const QPointF& a, const QPointF& b, const QPointF& p)` – Vrátí informaci o pozici bodu `p1` vůči přímce tvořenou body `a` a `b`

9.3 Třída sortPointsByX

Třída pro seřazení bodů podle souřadnice `x`.

- `bool operator()(const QPointF &p1, const QPointF &p2)` – Vratí `True`, pokud `x.p1` je menší než `x.p2`.

9.4 Třída sortPointsByY

Třída pro seřazení bodů podle souřadnice `y`.

- `bool operator()(const QPointF &p1, const QPointF &p2)` – Vratí `True`, pokud `y.p1` je menší než `y.p2`.

9.5 Třída Draw

Třída pro vykreslování polygonů a interakci s uživatelem.
Veřejné metody:

- `void mousePressEvent(QMouseEvent *e)` – Zpracuje událost kliknutí myší.
- `void paintEvent(QPaintEvent *event)` – Vykreslí polygony a body.
- `void switch_source()` – Přepíná typ vstupních dat.
- `QPolygonF getPol()` – Vrátí polygon.
- `void loadPolygonFromFile(const QString &fileName)` – Načte polygon z `*.TXT`.
- `void loadPolygonFromShapefile(const QString &fileName)` – Načte polygon z `*.SHP`.
- `void clearPolygons()` – Vymaže všechny polygony.
- `void clearResults()` – Vymaže výsledky.
- `void clearCHs()` – Vymaže konvexní obálky.
- `const std::vector<QPolygonF> getPolygons()` – Vrátí všechny polygony.
- `void setResults(const std::vector<QPolygonF>& newResults)` – Uloží výsledky do vektoru polygonů.
- `void setConvexHulls(const std::vector<QPolygonF>& newCHs)` – Uloží konvexní obálky do veltoru polygonů.
- `void changeColourCHOutline(const bool &status)` – Nastaví barvu obrysu.
- `changeColourCHFilling(const bool &status)` – Nastaví barvu výplně.

9.6 Třída MainForm

Hlavní uživatelské rozhraní aplikace.

- `void on_actionOpen_triggered()` – Otevře nabídku pro výběr souboru.
- `void on_actionMBR_triggered()` – Spustí výpočet minimálního ohraničujícího obdélníku (MBR).
- `void on_actionPCA_triggered()` – Spustí výpočet pomocí hlavních komponent (PCA).
- `void on_actionClear_All_triggered()` – Vymaže všechny objekty a výsledky.
- `void on_actionExit_triggered()` – Ukončí aplikaci.
- `void on_actionClear_results_triggered()` – Vymaže výsledky analýz.
- `void on_actionLongest_edge_triggered()` – Spustí výpočet pomocí nejdelší hrany (LE).
- `void on_actionWall_average_triggered()` – Spustí výpočet pomocí wall avarage (WA).
- `void on_actionWeighted_bisector_triggered()` – Spustí výpočet pomocí weighted bisector (WB).
- `void on_actionCovvex_Hull_triggered()` – Spustí výpočet konvexního obalu.
- `void on_actionAbout_triggered()` – Zobrazí informace o aplikaci.
- `void on_actionGraham_Scan_triggered()` – Spustí Graham Scan algoritmus.
- `void on_actionJarvis_Scan_triggered()` – Spustí Jarvis Scan algoritmus.
- `void on_actionExport_building_triggered()` – Exportuje generalizované budovy do souboru.
- `void on_actionExport_CH_triggered()` – Exportuje konvexní obálky do souboru.
- `void on_actionFill_triggered()` – Změní viditelnost výplně konvexní obálky.
- `void on_actionOutline_triggered()` – Změní viditelnost obrysu konvexní obálky.

10 Závěr

V rámci této úlohy byla vytvořena aplikace, která umožňuje generalizaci budov do úrovně detailu LOD0 pomocí několika metod: `Minimum Area Enclosing Rectangle`, `Principal Component Analysis (PCA)`, `Longest Edge`, `Wall Average` a `Weighted Bisector`. Pro výpočet metodou MAER je využita tvorba konvexní obálky metodou Jarvis scan a Graham scan, mezi nimiž může uživatel přepínat a které může také vizualizovat. Aplikace umožňuje uživateli ruční tvorbu polygonů budov nebo nahrání polygonů ze souboru ve formátech `*.TXT` a `*.SHP`. Výsledky generalizovaných budov nebo konvexních obálek lze exportovat do formátu `*.TXT` ve stejném formátu, který slouží pro nahrávání dat. Aplikace byla testována na třech datových sadách (historické centrum města, panelové sídliště a vilová čtvrť).

10.1 Další Možné Neřešené Problémy a Náměty na Vylepšení

- **Podpora dalších formátů pro vstup:** V současnosti aplikace podporuje načítání polygonů ze souborů ve formátech TXT a SHP. Bylo by vhodné rozšířit podporu také o formáty jako GeoJSON nebo GeoPackage, které jsou běžně využívány v GIS aplikacích.
- **Podpora dalších formátů pro výstup** V současnosti aplikace podporuje ukládání výsledků pouze pro TXT. Bylo by vhodné rozšířit podporu také o formáty jako CSV, SHP GeoJSON nebo GeoPackage, které jsou běžně využívány v GIS aplikacích.
- **Generování náhodných polygonů:** Aplikace by mohla být rozšířena o možnost generování náhodných polygonů.

- **Dávkové zpracování:** Dalším vylepšením by mohlo být aplikaci kromě zpracování v GUI umožnit i dávkové zpracování polygonů v příkazové řádce, které by mohlo vypadat například takto:
`BuildingSimplification.exe -CHGS Puvodni.shp > ConvexHull.txt`
`BuildingSimplification.exe -MAER Puvodni.shp > Generalizovane.txt`.
- **Editace budov:** Aktuální implementace umožňuje pouze základní práci s polygony. Bylo by možné rozšířit funkcionalitu o editaci polygonů, například odebírání jednotlivých vrcholů nebo přidávání nových.
- **Zoomování a posouvání mapy:** Aplikace v současnosti nezahrnuje podporu pro přibližování a posun mapy, což by mohlo usnadnit práci s polygony.
- **Odstranění jednotlivých prvků:** Dalším možným vylepšením by byla možnost selektivního mazání nakreslených prvků, například odstranění konkrétního polygonu nebo jeho editace, aniž by bylo nutné smazat veškeré polygony.
- **Rušení aktuálně kresleného prvku:** Možnost zrušit aktuální kresbu pomocí tlačítka.
- **Nastavení:** Přidat tlačítko umožňující nastavení parametrů, jako jsou barva, tloušťka a velikost.
- **Formát textového souboru:** Vytvořit načítání souboru z TXT, které bude umět přijímat různou strukturu dat, například více možností oddělovačů.
- **Vizuál aplikace:** Zlepšit vizuální stránku aplikace.
- **Konvexní obálky:** Vytvářet konvexní obálky pomocí dalších metod například Quick Hull, Sweep Line a nebo Devide and Conquer.
- **Výběr metody generalizace:** Umožnit nastavení metody generalizace pro jednotlivé budovy.
- **Vizualizace výsledků:** Zajistit možnost překryvného zobrazení výsledků jednotlivých metod pro snadnější porovnání.

Odkazy

[1] Tomáš Bayer. *Algoritmy v digitální kartografii.* 2008. vyd. Praha: Karolinum, 2008. ISBN: 978-80-246-1499-1.

[2] *Eigen Library.* `https://gitlab.com/libeigen/eigen`. Přístup k 2025-04-07.

[3] *ArcGIS REST Services.* `https://ags.cuzk.gov.cz/arcgis/rest/services/RUIAN/MapServer`. Přístup k 2025-04-07.

[4] Frank Warmerdam. *Shapelib.* `http://shapelib.maptools.org/`. Přístup k 2025-03-16. 1999.

[5] Zhihao Liu. *Shapelib pro C++.* `https://github.com/zhihao-liu/shapefile-viewer-qt/tree/master/shapelib`. Přístup k 2025-03-16. 2025.