<table>
<tr><td colspan="6">ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA STAVEBNÍ
KATEDRA GEOMATIKY</td></tr>
<tr><td colspan="6">Název předmětu
Algoritmy digitální kartografie a GIS</td></tr>
<tr><td>Úloha
U2</td><td colspan="5">Název úlohy:
Generalizace budov</td></tr>
<tr><td>akademický rok
2024/2025</td><td>semestr
letní</td><td>studijní skupina
C102</td><td>vypracoval
Matyáš Pokorný
Tereza Černohousová</td><td>datum
7.04.2025</td><td>klasifikace</td></tr>
</table>

Technická zpráva

1 Zadání

Úloha č. 2: Generalizace budov

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 $\{P_{i,j}\}_{j=1}^{m_i}$.
- **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 Bonusové úlohy

Z bonusových úloh námi byly zpracovány:

1. Generalizace budov metodami Wall Average.
2. Generalizace budov metodou Longest Edge.
3. Generalizace budov metodou Weighted Bisector.
4. Implementace další metody konstrukce konvexní obálky.
5. Ošetření singulárního případu při generování konvexní obálky.
6. Načtení z *.shp

3 Pracovní postup a použité algoritmy

3.1 Konstrukce konvexní obálky

3.1.1 Jarvis Scan

Tato metoda iterativně konstruuje konvexní obálku polygonu postupným výběrem vrcholu, který spolu s předchozím vrcholem a dočasným bodem tvoří největší kladný úhel. Algoritmus je známý také jako *gift wrapping*.

Nejprve zvolíme výchozí bod q s nejmenší souřadnicí y. Jako počáteční směr se použije horizontální vektor směřující doprava. V každém kroku hledáme bod $p \in P$, který tvoří s aktuálním bodem p_j a předchozím směrem $\overrightarrow{p_j p_{j-1}}$ největší orientovaný úhel ω. Tedy hledáme bod maximalizující:

$$\omega_i = \angle(\overrightarrow{p_j p_{j-1}}, \overrightarrow{p_j p_i})$$

Tento bod přidáme do výsledné množiny obálky. Iteraci opakujeme, dokud se algoritmus nevrátí zpět do výchozího bodu q. Pokud výsledná obálka obsahuje méně než tři body, vstupní polygon nebyl validní.

Metoda je robustní a jednoduchá na implementaci, ale v nejhorším případě má kvadratickou časovou složitost $\mathcal{O}(nh)$, kde h je počet bodů obálky.

3.1.2 Graham scan

Grahamův algoritmus nejprve seřadí body podle orientace (úhlu) vzhledem k pivotnímu bodu q s nejmenší souřadnicí y, a poté postupně buduje konvexní obálku s využitím zásobníku.

Po nalezení bodu q odstraníme jej ze seznamu a ke každému dalšímu bodu vypočteme úhel vzhledem k q:

$$\theta_i = \arctan 2(y_i - y_q,\ x_i - x_q)$$

Body se seřadí podle rostoucí hodnoty θ_i a vznikne sekvence P'. Následně přidáme první tři body do zásobníku a zbytek zpracováváme iterativně: pro každý další bod kontrolujeme orientaci trojice posledních dvou bodů zásobníku a nového kandidáta. Pokud je orientace pravotočivá nebo kolineární, bod na vrcholu zásobníku se odstraní. Postup se opakuje, dokud poslední tři body netvoří levotočivý trojúhelník. Nakonec se bod přidá do zásobníku.

Grahamův scan má časovou složitost $\mathcal{O}(n \log n)$ díky nutnosti třídění podle úhlu. Výsledná obálka je tvořena body v zásobníku.

3.2 Generalizace budov

3.2.1 Minimum Area Enclosing Rectangle

Pro výpočet minimálního obalového obdélníku (Minimum Area Enclosing Rectangle, MAER) vstupního polygonu využíváme algoritmus založený na rotaci konvexního obalu. Nejprve je třeba vytvořit konvexní obal daného polygonu $P \subset \mathbb{R}^2$, který představuje nejmenší konvexní množinu obsahující všechny jeho vrcholy. V našem případě je konvexní obal určen metodou založenou na iterativním výběru bodu s maximálním úhlem mezi dvěma orientovanými úsečkami. Tento přístup nezávisí na třídění bodů, což zajišťuje robustnost i při zpracování degenerovaných polygonů.

Po vytvoření konvexního obalu s vrcholy $\{p_0, p_1, \ldots, p_{n-1}\}$, kde $n \geq 3$, iterujeme přes všechny jeho hrany. Každá hrana $e_i = \overrightarrow{p_i p_{i+1}}$ (s indexací modulo n) je kandidátem na základnu minimálního obdélníku. Pro každou tuto hranu vypočítáme její orientaci takto:

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

Celý konvexní obal následně otočíme o úhel $-\sigma_i$, aby se aktuální hrana zarovnala s osou x. Rotace bodu (x, y) se provádí podle transformační rovnice:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos(-\sigma_i) & -\sin(-\sigma_i) \\ \sin(-\sigma_i) & \cos(-\sigma_i) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Tím získáme natočený konvexní obal, pro který vypočítáme minimální osu zarovnaný ohraničující obdélník (tzv. min-max box). Ten je určen extrémními souřadnicemi:

$$x_{\min} = \min_{(x,y)\in\mathcal{P}} x, \quad x_{\max} = \max_{(x,y)\in\mathcal{P}} x$$
$$y_{\min} = \min_{(x,y)\in\mathcal{P}} y, \quad y_{\max} = \max_{(x,y)\in\mathcal{P}} y$$

Rozměry obdélníku pak získáme jako rozdíly mezi těmito extrémními hodnotami:

$$\text{šířka} = x_{\max} - x_{\min}, \quad \text{výška} = y_{\max} - y_{\min}$$

Z těchto hodnot sestavíme obdélník se stranami rovnoběžnými s osami x, y. Spočítáme jeho obsah a z porovnání všech těchto obdelníků vybereme ten s nejměnším obsahem:

$$A_{\min} = \min_i A_i$$

Výsledný minimální obalový obdélník následně upravíme tak, aby jeho plocha odpovídala ploše původního polygonu P. Toho docílíme přepočtem měřítka pomocí faktoru:

$$k = \frac{S_P}{S_R}$$

kde S_P je plocha polygonu a S_R plocha obalového obdélníku. Následně provádíme změnu měřítka obdélníku vůči jeho těžišti a výsledek zpětně rotujeme o úhel $\sigma_{\min}$, abychom získali finální orientaci minimálního obdélníku:

$$\text{MAER}(P) = R_{\sigma_{\min}}\left(\sqrt{k} \cdot \text{BBox}\left(R_{-\sigma_{\min}}(\text{CH}(P))\right)\right)$$

kde:

- $\mathrm{CH}(P)$ je konvexní obal polygonu P,
- $R_{\pm\sigma_{\min}}$ označuje rotaci o úhel $\pm\sigma_{\min}$ kolem těžiště obdélníku,
- $\mathrm{BBox}(\cdot)$ je osa zarovnaný obdélník (axis-aligned bounding box),
- $\sigma_{\min}$ je úhel natočení, pro který má obalový obdélník minimální plochu,
- $k = \frac{S_P}{S_R}$ je měřítkový faktor, kde S_P je plocha polygonu a S_R plocha původního obalového obdélníku.

Tato konstrukce zajišťuje, že výsledný obdélník má stejnou orientaci jako původní nejlepší nalezený minimální obdélník, ale jeho plocha odpovídá ploše původního polygonu P, čímž se eliminuje vliv měřítka a umožňuje lepší srovnání např. mezi různými tvary.

3.2.2 Principal Component Analysis (PCA)

Pro výpočet minimálního obalového obdélníku pomocí analýzy hlavních komponent (PCA) používáme přístup založený na statistickém zpracování dat. Tento algoritmus najde optimální orientaci obdélníku, který obklopuje daný polygon $P \subset \mathbb{R}^2$, a to prostřednictvím zjištění hlavních komponent datového souboru. Následující kroky detailně popisují, jak tento algoritmus funguje.

Nejprve vytvoříme matici A, jejíž řádky obsahují souřadnice vrcholů polygonu P. Předpokládejme, že polygon má n vrcholů, což znamená, že A bude mít rozměry $n \times 2$. Pro každý vrchol $p_i = (x_i, y_i)$ polygonu P, přidáme jeho souřadnice do matice A:

$$A = \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \vdots & \vdots \\ x_n & y_n \end{bmatrix}$$

Dále vypočítáme střední hodnoty souřadnic x a y pro všechny vrcholy:

$$M = \frac{1}{n} \sum_{i=1}^{n} \begin{bmatrix} x_i \\ y_i \end{bmatrix} = \begin{bmatrix} \bar{x} \\ \bar{y} \end{bmatrix}$$

Poté od všech souřadnic odečteme střední hodnoty, čímž získáme matici B, která představuje „centrální“ hodnoty bodů polygonu:

$$B = A - M = \begin{bmatrix} x_1 - \bar{x} & y_1 - \bar{y} \\ x_2 - \bar{x} & y_2 - \bar{y} \\ \vdots & \vdots \\ x_n - \bar{x} & y_n - \bar{y} \end{bmatrix}$$

Následně vypočítáme kovarianční matici C podle vzorce:

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

Kde B^T je transponovaná matice B. Kovarianční matice C vyjadřuje rozptyl dat a jejich vzájemnou korelaci.

Dále provedeme dekompozici matic pomocí singulární hodnotové dekompozice (SVD):

$$C = U \Sigma V^T$$

Kde U a V jsou ortogonální matice a Σ je diagonální matice obsahující singularity. Z matice V získáme hlavní směry dat, které odpovídají hlavním komponentám.

Zajímá nás druhá komponenta (orientace dat), která je reprezentována úhlem σ mezi osou x a hlavní komponentou:

$$\sigma = \arctan 2(V_{1,0}, V_{0,0})$$

Tento úhel nám říká, jakým způsobem je třeba data rotovat, aby orientace polygonu odpovídala hlavnímu směru.

Po získání úhlu rotace σ provedeme rotaci polygonu P o tento úhel:

$$P_{\text{rot}} = R_{-\sigma}(P)$$

Následně vypočítáme minimální ohraničující obdélník pro rotovaný polygon, který je určen souřadnicemi minimálního a maximálního bodu na ose x a y:

$$\text{šířka} = x_{\text{max}} - x_{\text{min}}, \quad \text{výška} = y_{\text{max}} - y_{\text{min}}$$

Tento obdélník je definován čtyřmi body, které tvoří jeho rohy. Získaný obdélník představuje minimální ohraničující obdélník pro rotovaný polygon.

Nakonec přepočteme velikost tohoto obdélníku podle původního polygonu a zpětně provedeme rotaci o úhel σ, abychom vrátili obdélník do původní orientace:

$$\text{PCA}(P) = R_\sigma \left(\text{BBox} \left(R_{-\sigma}(P) \right) \right)$$

kde:

- P je původní polygon,
- $R_{\pm\sigma}$ značí rotaci o úhel $\pm\sigma$,
- $\text{BBox}(\cdot)$ je osa zarovnaný obdélník (axis-aligned bounding box),
- σ je úhel rotace, který odpovídá hlavní komponentě,
- $\text{PCA}(P)$ je minimální obalový obdélník pro polygon P.

3.2.3 Longest Edge

Pro výpočet minimálního obalového obdélníku s využitím nejdelší hrany polygonu (Longest Edge) postupujeme následovně. Tento algoritmus je založen na nalezení nejdelší hrany polygonu a jejím použití pro určení orientace obalového obdélníku. Nejprve iterujeme přes všechny hrany polygonu a pro každou vypočítáme její délku. Po nalezení nejdelší hrany určujeme její orientaci a podle ní otáčíme polygon tak, aby tato hrana byla zarovnána s osou x.

Nejprve inicializujeme proměnné pro maximální vzdálenost, index nejdelší hrany a její složky v x a y-ové ose. Poté procházíme všechny hrany polygonu P a pro každou hranu e_i, která je určena dvojicí bodů $p_i = (x_i, y_i)$ a $p_{i+1} = (x_{i+1}, y_{i+1})$, spočítáme její délku podle vzdálenostní formule:

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

Pokud je délka hrany d_i větší než dosud nalezená maximální vzdálenost, aktualizujeme maximální hodnoty a uložíme index hrany, její složky Δx a Δy.

Následně vypočítáme úhel rotace σ, který je určen směrem nejdelší hrany. Tento úhel získáme pomocí funkce atan2, která vrací úhel mezi vektorem a osou x:

$$\sigma = \arctan 2(\Delta y, \Delta x)$$

Poté rotujeme celý polygon o úhel $-\sigma$, aby se nejdelší hrana zarovnala s osou x. Následně vypočítáme minimální ohraničující obdélník (min-max box) pro otočený polygon. Z těchto hodnot určíme rozměry obdélníku a vypočítáme jeho obsah. Tento minimální obdélník následně přizpůsobíme velikosti původního polygonu pomocí změny měřítka, aby byl zachován jeho poměr stran.

Nakonec obdélník rotujeme zpět o úhel σ, čímž vrátíme jeho původní orientaci. Výsledkem je minimální obalový obdélník, jehož orientace je určena nejdelší hranou původního polygonu.

Výsledný obdélník je dán vzorcem:

$$\mathrm{LE}(P) = R_\sigma\left(\mathrm{BBox}\left(R_{-\sigma}(P)\right)\right)$$

kde:

- $R_{\pm\sigma}$ označuje rotaci polygonu o úhel $\pm\sigma$,
- $\mathrm{BBox}(\cdot)$ je osa zarovnaný obdélník (axis-aligned bounding box),
- σ je úhel natočení, který zarovnává nejdelší hranu s osou x,
- P je původní polygon.

Tato metoda je efektivní pro geometrické analýzy, kde je důležité najít obdélník odpovídající orientaci nejdelší hrany, čímž dochází k optimalizaci prostorového zobrazení polygonu.

3.2.4 Wall Average (WA)

Algoritmus Wall Average (WA) slouží k výpočtu průměrné orientace polygonu na základě směru jeho hran. Nejprve se spočítají směry a délky všech hran polygonu. Pro každou hranu e_i vypočítáme její směr σ_i, což je úhel mezi vektorem hrany a osou x:

$$\sigma_i = \arctan 2(dy_i, dx_i)$$

kde dx_i a dy_i jsou změny souřadnic mezi dvěma sousedními vrcholy polygonu. Délka hrany l_i je pak dána vzorcem:

$$l_i = \sqrt{(dx_i)^2 + (dy_i)^2}$$

Dále si zvolíme referenční směr σ_{ref}, což může být směr první hrany. Poté vypočítáme vážený průměr směrů hran podle jejich délek. To se provádí pomocí následujícího vzorce:

$$\sigma = \sigma_{\text{ref}} + \frac{\sum r_i \cdot l_i}{\sum l_i}$$

kde r_i je upravený směr hrany e_i, tedy rozdíl mezi směrem hrany σ_i a referenčním směrem σ_{ref}, normalizovaný na interval $[-\pi, \pi]$:

$$r_i = \sigma_i - \sigma_{\text{ref}}$$

Po získání směru σ provedeme rotaci polygonu tak, aby hrany byly orientovány v souladu s tímto průměrným směrem. Následně spočítáme minimální obalový obdélník pro otočený polygon, upravíme jeho velikost a opět jej otočíme zpět do původní orientace.

- σ_i je směr i-té hrany polygonu.
- l_i je délka i-té hrany polygonu.
- σ_{ref} je referenční směr (směr první hrany).
- r_i je upravený směr hrany, normalizovaný na interval $[-\pi, \pi]$.
- Výsledný směr σ je vážený průměr směrů hran s ohledem na jejich délky.

3.2.5 Weighted Bisector (WE)

Algoritmus Weighted Bisector (WE) se zaměřuje na výpočet váženého bisektoru polygonu na základě dvou nejdelších diagonál. Nejprve spočítáme vzdálenosti mezi všemi dvojicemi vrcholů polygonu a zjistíme dvě nejdelší diagonály d_1 a d_2. Každá diagonála má svůj směr σ_1 a σ_2, které jsou dány vzorcem:

$$\sigma_1 = \arctan 2(dy_1, dx_1), \quad \sigma_2 = \arctan 2(dy_2, dx_2)$$

kde (dx_1, dy_1) a (dx_2, dy_2) jsou rozdíly souřadnic mezi vrcholy diagonál. Výsledný směr váženého bisektoru je dán váženým průměrem směrů těchto dvou diagonál:

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

Po výpočtu směru σ provedeme rotaci polygonu tak, aby diagonály byly orientovány v souladu s hlavním směrem. Poté vypočítáme minimální obalový obdélník pro otočený polygon, upravíme jeho velikost a opět jej otočíme zpět do původní orientace.

- d_1 a d_2 jsou délky dvou nejdelších diagonál polygonu.
- σ_1 a σ_2 jsou směry těchto diagonál.
- Vážený směr σ je vážený průměr směrů diagonál, kde váhy jsou jejich délky.

4 Zhodnocení přesnosti algoritmů

Pro zhodnocení přesnosti algoritmů byl použit poměr mezi průnikem a sjednocením (Intersection over Union = IoU). Ten je dán vztahem:

$$IoU = \frac{A \cap B}{A \cup B}, \tag{1}$$

kde A je obdélník reprezentující generalizovanou budovu a B je generalizovaná budova.

Tato metrika vhodně zhodnocuje překryt dvou ploch nehledě na jejich velikost a tvar. Nabývá hodnot 1 (pro kompletní shodu) až 0 (pro žádnou shodu, plochy jsou mimo). Jako dobrá shoda je považována hodnota $IoU \geq 0.5$.

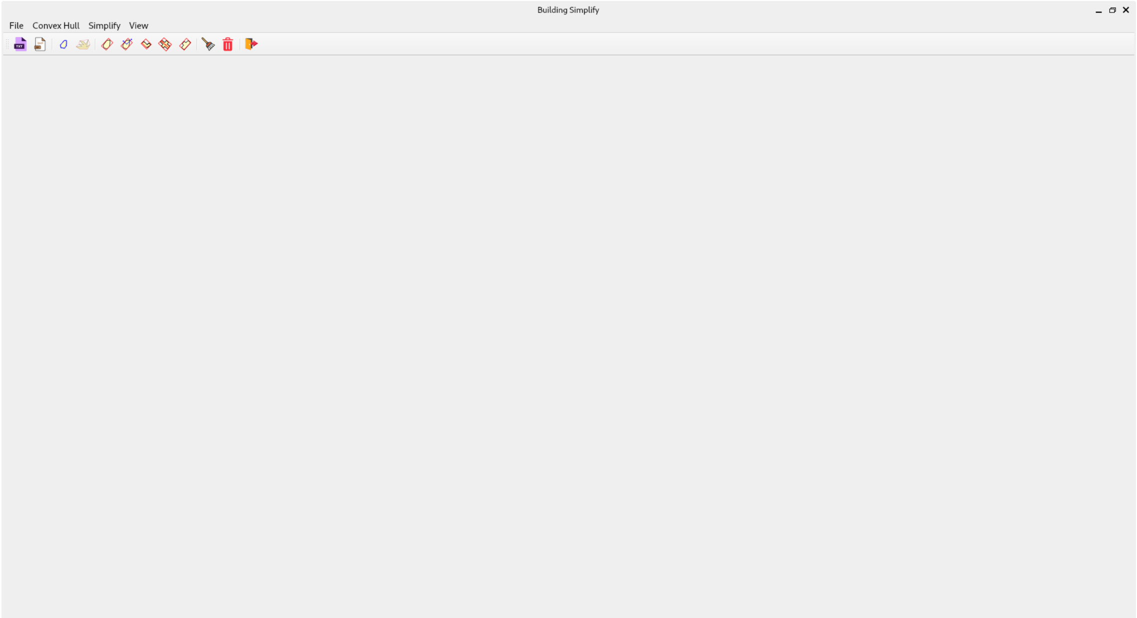
Po přenásobení hodnoty IoU stem, dostáváme hodnotu v procentech: $IoU * 100 = IoU\%$.

Algoritmus	Sídliště [%]	Vilovka [%]	Centrum [%]
MAER	94	89	88
PCA	78	69	69
LE	94	89	88
WE	72	71	70
WA	94	89	88

Tabulka 1: Zhodnocení přesnosti algoritmů pomocí IoU v %

5 Ukázka aplikace

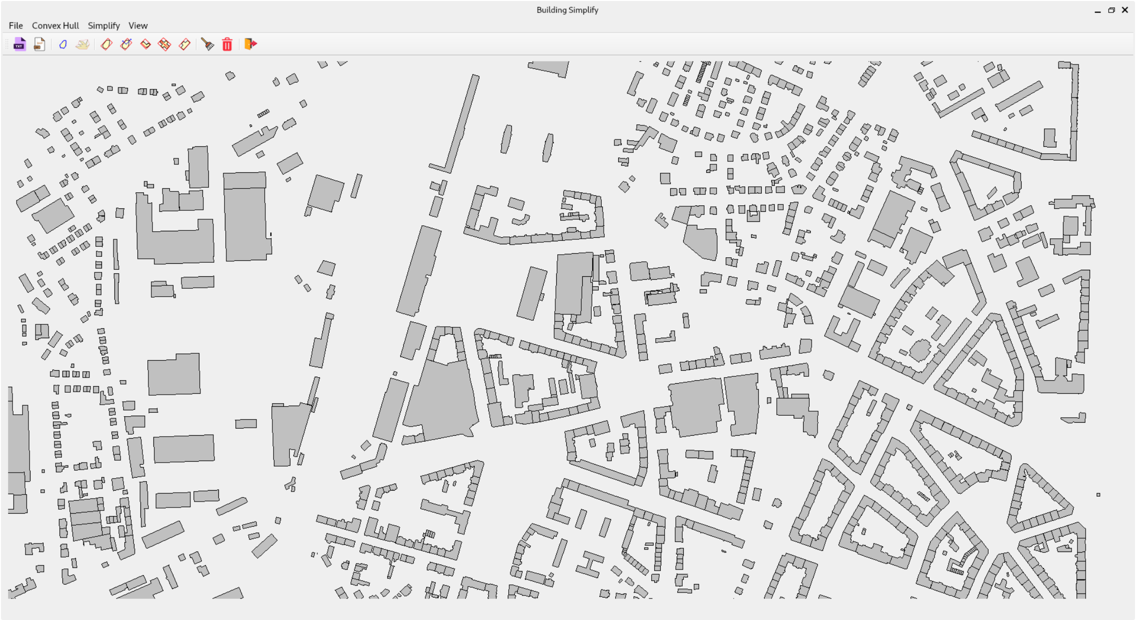
Po spuštění kódu je otevřeno grafické rozhraní, ve kterém je možné naši aplikaci ovládat. Při spuštění se zobrazí prázdná aplikace s několika ikonami.



Obrázek 1: Po otevření aplikace

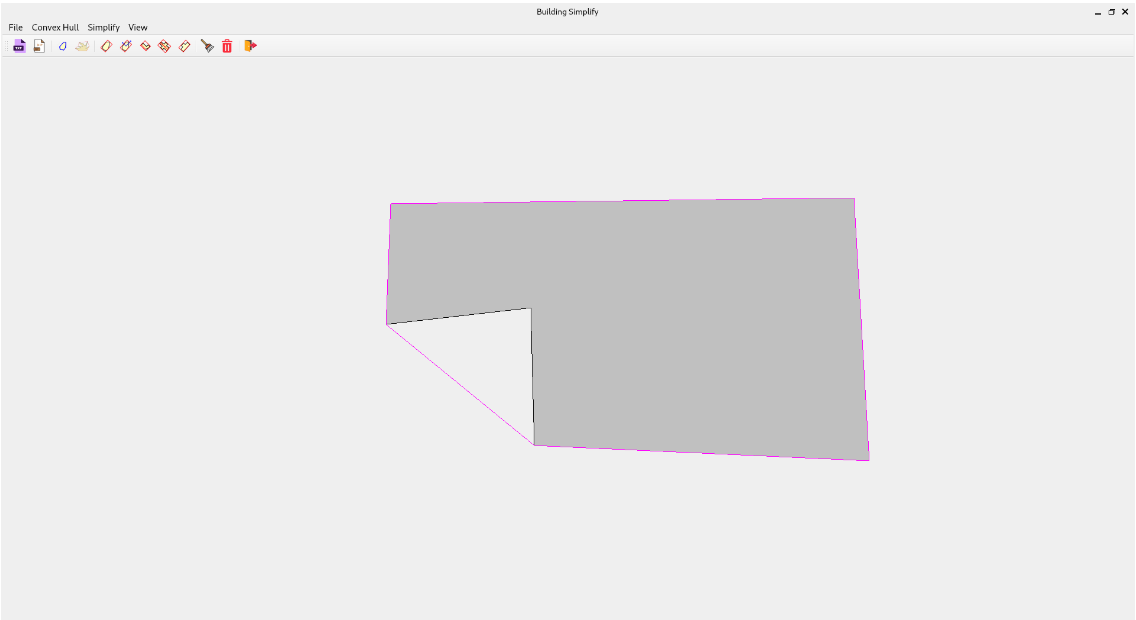
Polygon, který bude vyhodnocován může nakreslit uživatel sám nebo může být načten z SHP. Kreslení polygonu je možné ihned po spuštění aplikace. Pro SHP formátu je vytvořena speciální ikona.

Obrázek 2: Nakreslený polygon

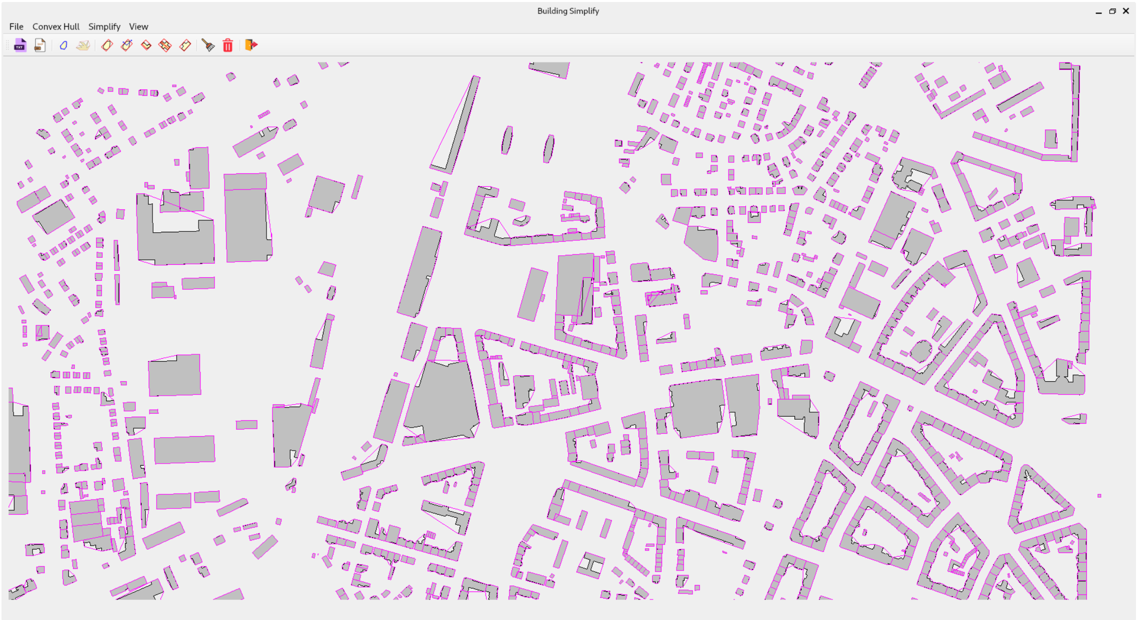


Obrázek 3: Otevření SHP souboru z počítače

Aplikace obsahuje dvě různé metody pro konstrukci konvexní obálky. Obě jsou k nalezení v hlavním panelu ikon nebo v záložce *Convex Hull*.



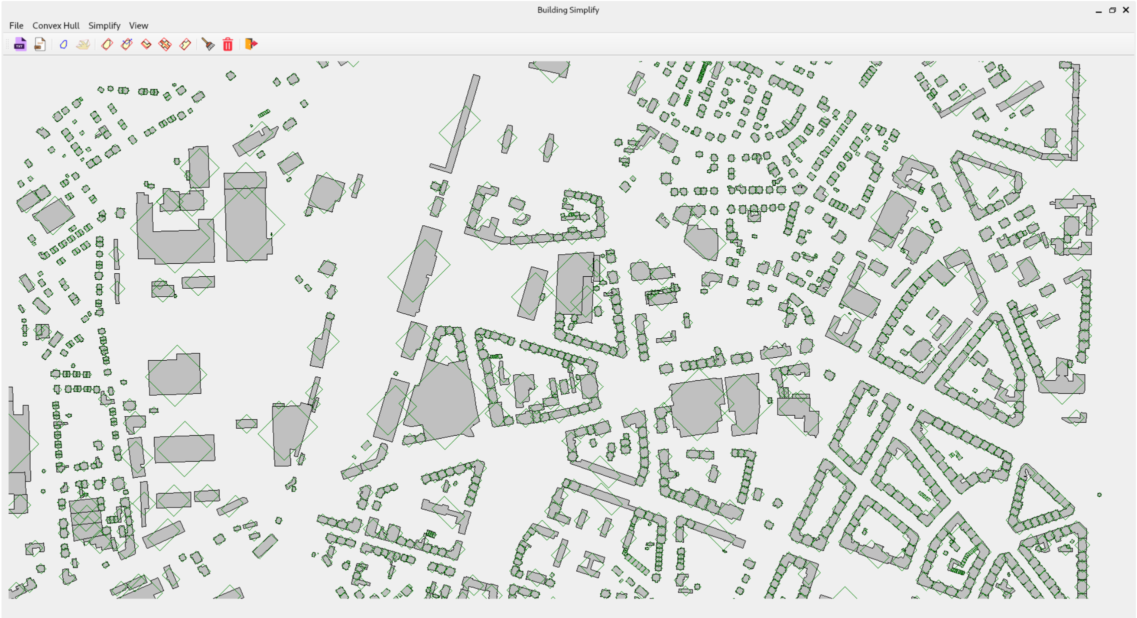
Obrázek 4: Konvexní obálka pro samostatný polygon



Obrázek 5: Konvexní obálka pro polygony z SHP

Hlavním cílem této úlohy bylo vyzkoušet si několik algoritmů pro generalizaci budov. Bylo vytvořeno celkem 5 algoritmů, které jsou k dispozici v kartě *Simplify* nebo v horním pásu ikon.

Obrázek 6: Generalizace budovy, uživatelský polygon



Obrázek 7: Generalizace budov, SHP

6 Závěr

Pomocí QT Creatoru byla vytvořena jednoduchá aplikace pro generalizaci budov (polygonů obecně). Aplikace umožňuje vytvořit ručně jeden polygon nebo polygony načíst buď z textového souboru ve formátu *x,y* nebo ze *ShapeFilu*.

Pro generalizaci lze volat různé algoritmy, které přes načtenou vrstvu polygonů vykreslí budovy generalizované. Mezi použitými algoritmy jsou *Minimum Area Enclosing Rectangle*, *Principal Component Analysis*, *Longest Edge*, *Longest Diagonals* a *Wall average*. Některé z těchto algoritmů používají pro výpočet konvexní obálku. Ta je tvořena algoritmy *Graham Scan* a *Jarvis Scan*.

V tabulce 1 jsou uvedeny procentuální hodnoty vyjadřující přesnost použitých algoritmů v oblastech s různou zástavbou. Je z ní vidět, že algoritmy si nejlépe vedou v oblasti sídlišť, kde má zástavba už přirozeně jednoduché tvary (panelové domy mívají obdélníkový půdorys) a nejhůře zase v centrech starých měst, kde půdorysy budov nejsou pravidelné ani pravoúhlé a bývají často "přilepené"na sebe. Ve vilových čtvrtích si algoritmy vedly o něco lépe než v centrech, což je způsobeno nejspíš různými verandami či balkóny, které tyto budovy mají. Co se týče algoritmů, tak nejlepších výsledků dosáhl algoritmus MAER, WA a LE, zato nejhorších PCA a WE. Žádný z algoritmů však neklesl pod hodnotu 50%, což je hranice pro hodnocení algoritmu jako úspěšného.

Algoritmy jsou ošetřené i pro singulární případy, kdy například je vstupní polygon tvořen pouze dvěma body.

Možným návrhem na zlepšení je například tlačítko pro odstranění výchozích polygonů pro zobrazení pouze výsledků nebo odstraňování výsledků z různých algoritmů samostatně. Dalším zlepšením by pak bylo zapínání či vypínání vrstev. Další úpravou by mohla být funkce *zoom*.

V Praze dne: 07.04. 2025

T. Černohousová
M. Pokorný

Pseudokód pro algoritmy

algorithms.cpp

Algorithm 1 Create Convex Hull from Polygon

Vstup: Polygon *pol*
Výstup: Konvexní obálka *ch* polygonu *pol*
$ch \leftarrow$ prázdný polygon
if velikost $pol < 3$ **then**
    vytiskni "Input polygon has too few points!"
    vrátit *ch*
end if
$q \leftarrow$ bod s nejmenší y-souřadnicí (při shodě nejmenší x)
$r \leftarrow$ bod s nejmenší x-souřadnicí (při shodě nejmenší y)
$pj \leftarrow q$
$pj1 \leftarrow$ bod (r_x, q_y)
Přidej pj do ch
repeat
    $\omega_{max} \leftarrow 0$
    $i_{max} \leftarrow -1$
    for každý bod p_i v *pol* **do**
        $\omega \leftarrow$ úhel mezi $(pj1, pj)$ a (pj, p_i)
        if $\omega > \omega_{max}$ **then**
            $\omega_{max} \leftarrow \omega$
            $i_{max} \leftarrow i$
        end if
    end for
    if $i_{max} = -1$ **then**
        vytiskni "Error: Could not compute convex hull."
        přerušit
    end if
    Přidej bod $pol[i_{max}]$ do ch
    $pj1 \leftarrow pj$
    $pj \leftarrow pol[i_{max}]$
until $pj = q$
if velikost $ch < 3$ **then**
    vytiskni "Convex hull has less than 3 points!"
    vrátit prázdný polygon
end if
vrátit *ch*

Algorithm 2 Get Angle Between Two Vectors

Vstup: Body $p1, p2, p3, p4$
Výstup: Úhel mezi vektory $\overrightarrow{p1p2}$ a $\overrightarrow{p3p4}$
Vypočítat složky vektoru: $u_x \leftarrow p2.x - p1.x$, $u_y \leftarrow p2.y - p1.y$
Vypočítat složky vektoru: $v_x \leftarrow p4.x - p3.x$, $v_y \leftarrow p4.y - p3.y$
Vypočítat skalární součin: $dot \leftarrow u_x \cdot v_x + u_y \cdot v_y$
Vypočítat velikosti vektorů: $n_u \leftarrow \sqrt{u_x^2 + u_y^2}$, $n_v \leftarrow \sqrt{v_x^2 + v_y^2}$
vrátit $\arccos\left(\frac{dot}{n_u \cdot n_v}\right)$

Algorithm 3 Create Convex Hull using Graham Scan [1]

Vstup: Polygon *pol*
Výstup: Konvexní obálka *ch_g* polygonu *pol*
if velikost $pol < 3$ **then**
    vrátit prázdný polygon
end if
$ch\_g \leftarrow$ prázdný polygon
$q \leftarrow$ bod s nejmenší y-souřadnicí (při shodě nejmenší x)
$new\_pol \leftarrow pol$ bez bodu q
if *new_pol* je prázdný **then**
    vrátit prázdný polygon
end if
$points\_with\_angles \leftarrow$ prázdný seznam
for každý bod p_i v *new_pol* **do**
    $dx \leftarrow p_i.x - q.x$
    $dy \leftarrow p_i.y - q.y$
    $angle \leftarrow \arctan 2(dy, dx)$
    Přidej $(p_i, angle)$ do *points_with_angles*
end for
Seřaď *points_with_angles* vzestupně podle *angle*
$stack \leftarrow$ prázdný zásobník
Vlož q, první a druhý bod ze seznamu do *stack*
for každý další bod p_i v *points_with_angles* **do**
    while poslední dvě hodnoty ve *stack* a p_i tvoří pravotočivý ohyb **do**
        Odeber poslední bod ze *stack*
    end while
    Vlož p_i do *stack*
end for
$ch\_g \leftarrow$ body ve *stack*
vrátit *ch_g*

Algorithm 4 Normalize Polygons

Vstup: Seznam polygonů *polygons*, Šířka okna *width*, Výška okna *height*
Výstup: Normalizované polygonu centrované v okně
Vypočítat centroid polygonů
Vypočítat offset pro centrování polygonů
for každý polygon v seznamu *polygons* **do**
    for každý bod v polygonu **do**
        Přelož bod podle offsetu
        Uprav Y-souřadnici pro výšku okna
    end for
end for

Algorithm 5 Rotate Polygon by Angle Sigma

Vstup: Polygon *pol*, úhel σ
Výstup: Otáčený polygon *rotated*
$rotated \leftarrow$ prázdný polygon
for každý bod $p_i = (x_p, y_p)$ v *pol* **do**
    $x_{pr} \leftarrow x_p \cdot \cos(\sigma) - y_p \cdot \sin(\sigma)$
    $y_{pr} \leftarrow x_p \cdot \sin(\sigma) + y_p \cdot \cos(\sigma)$
    Vytvoř nový bod $p_{rotated} = (x_{pr}, y_{pr})$
    Přidej $p_{rotated}$ do *rotated*
end for
vrátit *rotated*

Algorithm 6 Compute Area of Polygon using Gauss's Formula (LH Formula)

Vstup: Polygon *pol*
Výstup: Plocha *area* polygonu *pol*
$n \leftarrow$ velikost polygonu *pol*
$area \leftarrow 0$
for $i \leftarrow 0$ to $n - 1$ **do**
    $i_1 \leftarrow (i + 1)\%n$ ▷ Index dalšího bodu
    $i_{-1} \leftarrow (i - 1 + n)\%n$ ▷ Index předchozího bodu
    $area \leftarrow area + pol[i].x() \cdot (pol[i_1].y() - pol[i_{-1}].y())$
end for
$area \leftarrow \text{fabs}(area/2)$ ▷ Vezmeme absolutní hodnotu a vydělíme 2
vrátit *area*

Algorithm 7 Resize Polygon to Minimum Area Bounding Box

Vstup: Polygon pol, minimální ohraničující obdélník $mmbox$
Výstup: Změněný polygon pol_res
$Ab \leftarrow$ plocha polygonu pol (volání funkce get_area)
$A \leftarrow$ plocha ohraničujícího obdélníku $mmbox$ (volání funkce get_area)
if $A = 0$ **then**
    vytiskni "Area of bounding box is zero!"
    vrátit prázdný polygon
end if
$k \leftarrow \frac{Ab}{A}$ ▷ Koeficient změny velikosti
$xt \leftarrow$ střed x-souřadnic $mmbox$ (průměr ze čtyř bodů)
$yt \leftarrow$ střed y-souřadnic $mmbox$ (průměr ze čtyř bodů)
Vytvoř vektory:
$u_1x \leftarrow mmbox[0].x - xt$, $u_1y \leftarrow mmbox[0].y - yt$
$u_2x \leftarrow mmbox[1].x - xt$, $u_2y \leftarrow mmbox[1].y - yt$
$u_3x \leftarrow mmbox[2].x - xt$, $u_3y \leftarrow mmbox[2].y - yt$
$u_4x \leftarrow mmbox[3].x - xt$, $u_4y \leftarrow mmbox[3].y - yt$
Vypočítej nové vrcholy ohraničujícího obdélníku:
$x_1r \leftarrow xt - \sqrt{k} \cdot u_1x$, $y_1r \leftarrow yt - \sqrt{k} \cdot u_1y$
$x_2r \leftarrow xt - \sqrt{k} \cdot u_2x$, $y_2r \leftarrow yt - \sqrt{k} \cdot u_2y$
$x_3r \leftarrow xt - \sqrt{k} \cdot u_3x$, $y_3r \leftarrow yt - \sqrt{k} \cdot u_3y$
$x_4r \leftarrow xt - \sqrt{k} \cdot u_4x$, $y_4r \leftarrow yt - \sqrt{k} \cdot u_4y$
Vytvoř nové body $p_1 = (x_1r, y_1r)$, $p_2 = (x_2r, y_2r)$, $p_3 = (x_3r, y_3r)$, $p_4 = (x_4r, y_4r)$
Vytvoř polygon $pol\_res = (p_1, p_2, p_3, p_4)$
vrátit pol_res

Algorithm 8 Create Minimum Area Enclosing Rectangle (MAER)

Vstup: Polygon *pol*
Výstup: Minimum Area Enclosing Rectangle $mmbox\_min$
$\sigma_{min} \leftarrow 2 \cdot \pi$
$[mmbox_{min}, area_{min}] \leftarrow$ volání funkce `minmaxBox(pol)` ▷ Získej minimální ohraničující obdélník a plochu
$ch \leftarrow$ volání funkce `createCH(pol)` ▷ Vytvoř konvexní obálku polygonu
$n \leftarrow$ velikost polygonu ch
if ch má méně než 2 body **then**
    vrátit prázdný polygon ▷ Není dostatek bodů pro konvexní obálku
end if
for $i \leftarrow 0$ to $n - 1$ **do**
    $dx \leftarrow ch[(i+1)\%n].x - ch[i].x$
    $dy \leftarrow ch[(i+1)\%n].y - ch[i].y$
    $\sigma \leftarrow \arctan 2(dy, dx)$ ▷ Výpočet úhlu otáčení
    $ch\_rotate \leftarrow$ volání funkce rotate(ch, -σ) ▷ Otoč konvexní obálku o -σ
    $[mmbox, area] \leftarrow$ volání funkce minmaxBox(ch_rotate) ▷ Vypočítej minimální ohraničující obdélník pro otočený polygon
    if $area < area_{min}$ then
        $area_{min} \leftarrow area$
        $\sigma_{min} \leftarrow \sigma$
        $mmbox_{min} \leftarrow mmbox$
    end if
end for
$mmbox\_min\_res \leftarrow$ volání funkce resize(pol, mmbox$_{min}$) ▷ Změň velikost minimálního ohraničujícího obdélníku
vrátit volání funkce rotate(mmbox_min_res, σ_{min}) ▷ Otoč zpět minimální ohraničující obdélník do původní orientace

Algorithm 9 Create Minimum Enclosing Rectangle using Principal Component Analysis (PCA)

Vstup: Polygon *pol*
Výstup: Minimum enclosing rectangle *mmbox*
$n \leftarrow$ velikost polygonu *pol*
Vytvoř matici A o rozměrech $n \times 2$
for $i \leftarrow 0$ to $n-1$ **do**
    $A(i,0) \leftarrow pol[i].x$ ▷ X souřadnice bodu i
    $A(i,1) \leftarrow pol[i].y$ ▷ Y souřadnice bodu i
end for
$M \leftarrow$ průměr souřadnic: $A.colwise().mean()$ ▷ Spočítáme průměr souřadnic
$B \leftarrow A - M$ ▷ Odečteme průměr od každého bodu
$C \leftarrow B^T \cdot B/(n-1)$ ▷ Kovarianční matice
SVD: $[U, S, V] \leftarrow$ `svd`(C, `ComputeFullV | ComputeFullU`) ▷ Provedeme SVD na kovarianční matici
$V \leftarrow$ matice V z SVD ▷ Uložíme matice z decompozice
$sigma \leftarrow \arctan 2(V(1,0), V(0,1))$ ▷ Spočítáme úhel rotace
pol_rot $\leftarrow$ volání funkce `rotate(pol, -sigma)` ▷ Otočíme polygon o $-\sigma$
$[mmbox, area] \leftarrow$ volání funkce `minmaxBox(pol_rot)` ▷ Vypočítáme minimální ohraničující obdélník pro otočený polygon
mmbox_res $\leftarrow$ volání funkce `resize(pol, mmbox)` ▷ Změníme velikost minimálního ohraničujícího obdélníku
vrátit volání funkce `rotate(mmbox_res,` σ`)` ▷ `Otočíme zpět obdélník podle vypočítaného úhlu`

Algorithm 10 Create Minimum Enclosing Rectangle based on the Longest Edge (LE) [1]

Vstup: Polygon pol
Výstup: Minimum Enclosing Rectangle $mmbox$
$max\_d \leftarrow 0$ ▷ Maximální délka hrany
$max\_index \leftarrow -1$ ▷ Index hrany s maximální délkou
$max\_dx \leftarrow 0$, $max\_dy \leftarrow 0$ ▷ Rozdíly mezi souřadnicemi pro maximální hranu
for $i \leftarrow 0$ to $pol.size() - 1$ **do**
    $p1 \leftarrow pol[i]$ ▷ První bod hrany
    $p2 \leftarrow pol[(i+1)\%pol.size()]$ ▷ Druhý bod hrany (cyklický index)
    $dx \leftarrow p1.x - p2.x$ ▷ Rozdíl X souřadnic
    $dy \leftarrow p1.y - p2.y$ ▷ Rozdíl Y souřadnic
    $d \leftarrow \sqrt{dx^2 + dy^2}$ ▷ Vzdálenost mezi dvěma body
    if $d > max\_d$ **then**
        $max\_d \leftarrow d$ ▷ Aktualizace maximální délky
        $max\_index \leftarrow i$ ▷ Uložení indexu hrany
        $max\_dx \leftarrow dx$
        $max\_dy \leftarrow dy$
    end if
end for
$sigma \leftarrow \arctan 2(max\_dy, max\_dx)$ ▷ Výpočet úhlu hlavní orientace hrany
$pol\_rot \leftarrow$ volání funkce `rotate(pol, -sigma)` ▷ Otočení polygonu o $-\sigma$
$[mmbox, area] \leftarrow$ volání funkce `minmaxBox(pol_rot)` ▷ Výpočet minimálního ohraničujícího obdélníku pro otočený polygon
$mmbox\_res \leftarrow$ volání funkce `resize(pol, mmbox)` ▷ Změna velikosti minimálního ohraničujícího obdélníku
vrátit volání funkce `rotate(mmbox_res, sigma)` ▷ Otočení zpět minimálního ohraničujícího obdélníku do původní orientace

Algorithm 11 Create Minimum Enclosing Rectangle based on the Longest Diagonals (WE) [1]

Vstup: Polygon pol
Výstup: Minimum Enclosing Rectangle $mmbox$
$n \leftarrow pol.size()$ ▷ Počet bodů polygonu
$d1max \leftarrow 0$, $d2max \leftarrow 0$ ▷ Maximální délky diagonál
$dx1 \leftarrow 0$, $dy1 \leftarrow 0$, $dx2 \leftarrow 0$, $dy2 \leftarrow 0$ ▷ Koordináty rozdílů pro diagonály
for $i \leftarrow 0$ to $n-1$ **do**
  for $j \leftarrow 0$ to $n-1$ **do**
    if $i = j$ **then**
      Pokračuj ▷ Přeskoč, když i a j jsou stejné
    end if
    $p1 \leftarrow pol[(i+j+2)\%n]$ ▷ Další bod diagonály
    $p2 \leftarrow pol[i\%n]$ ▷ První bod diagonály
    $dx \leftarrow p1.x - p2.x$ ▷ Rozdíl X souřadnic
    $dy \leftarrow p1.y - p2.y$ ▷ Rozdíl Y souřadnic
    $d \leftarrow \sqrt{dx^2 + dy^2}$ ▷ Vzdálenost mezi body diagonály
    if $d > d1max$ **then**
      $d2max \leftarrow d1max$ ▷ Aktualizace druhé nejdelší diagonály
      $d1max \leftarrow d$ ▷ Aktualizace první nejdelší diagonály
      $dx1 \leftarrow dx$, $dy1 \leftarrow dy$
      $dx2 \leftarrow dx1$, $dy2 \leftarrow dy1$
    else if $d > d2max$ **a** $d < d1max$ **then**
      $d2max \leftarrow d$ ▷ Aktualizace druhé nejdelší diagonály
      $dx2 \leftarrow dx$, $dy2 \leftarrow dy$
    end if
  end for
end for
$sigma1 \leftarrow \arctan 2(dy1, dx1)$ ▷ Výpočet úhlu první diagonály
$sigma2 \leftarrow \arctan 2(dy2, dx2)$ ▷ Výpočet úhlu druhé diagonály
$sigma \leftarrow \frac{d1max \cdot sigma1 + d2max \cdot sigma2}{d1max + d2max}$ ▷ Průměrný úhel diagonál
$pol\_rot \leftarrow$ volání funkce `rotate(pol, -sigma)` ▷ Otočení polygonu o $-\sigma$
$[mmbox, area] \leftarrow$ volání funkce `minmaxBox(pol_rot)` ▷ Výpočet minimálního ohraničujícího obdélníku pro otočený polygon
$mmbox\_res \leftarrow$ volání funkce `resize(pol, mmbox)` ▷ Změna velikosti minimálního ohraničujícího obdélníku
vrátit volání funkce `rotate(mmbox_res, sigma)` ▷ Otočení zpět minimálního ohraničujícího obdélníku do původní orientace

Algorithm 12 Wall Average algorithm [1]

Vstup: Polygon *pol*
Výstup: Generalizovaný polygon zarovnaný na hlavní směr
$n \leftarrow$ počet vrcholů polygonu *pol*
inicializuj prázdné seznamy *sigmas*, *lengths*
for $i \leftarrow 0$ to $n-1$ **do**
    $p_1 \leftarrow pol[i]$
    $p_2 \leftarrow pol[(i+1) \bmod n]$
    $dx \leftarrow p_2.x - p_1.x$
    $dy \leftarrow p_2.y - p_1.y$
    $\sigma \leftarrow \mathrm{atan2}(dy, dx)$ ▷ směr hrany
    $length \leftarrow \sqrt{dx^2 + dy^2}$ ▷ délka hrany
    přidej σ do *sigmas*
    přidej *length* do *lengths*
end for
$\sigma_{ref} \leftarrow sigmas[0]$ ▷ referenční směr
$sum\_r\_s \leftarrow 0$, $sum\_s \leftarrow 0$
for $i \leftarrow 0$ to $n-1$ **do**
    $\Delta\sigma \leftarrow sigmas[i] - \sigma_{ref}$
    while $\Delta\sigma \leq -\pi$ **do**
        $\Delta\sigma \leftarrow \Delta\sigma + 2\pi$
    end while
    while $\Delta\sigma > \pi$ **do**
        $\Delta\sigma \leftarrow \Delta\sigma - 2\pi$
    end while
    $k_i \leftarrow \mathrm{round}\left(\frac{2\Delta\sigma}{\pi}\right)$
    $r_i \leftarrow \Delta\sigma - k_i \cdot \left(\frac{\pi}{2}\right)$
    $sum\_r\_s \leftarrow sum\_r\_s + r_i \cdot lengths[i]$
    $sum\_s \leftarrow sum\_s + lengths[i]$
end for
$\sigma \leftarrow \sigma_{ref} + \frac{sum\_r\_s}{sum\_s}$
$pol_{rot} \leftarrow \mathrm{rotate}(pol, -\sigma)$
$[box, area] \leftarrow \mathrm{minmaxBox}(pol_{rot})$
$box_{resized} \leftarrow \mathrm{resize}(pol, box)$
vrátit $\mathrm{rotate}(box_{resized}, \sigma)$

Algorithm 13 Výpočet IoU (Intersection over Union) dvou polygonů

Vstup: Polygon pol_0, generovaný polygon pol_{gen}
Výstup: Hodnota IoU mezi pol_0 a pol_{gen}
$path_1 \leftarrow$ nová kreslící cesta
Přidej pol_0 do $path_1$
$path_2 \leftarrow$ nová kreslící cesta
Přidej pol_{gen} do $path_2$
$intersection \leftarrow$ průnik $path_1$ a $path_2$
$unionPath \leftarrow$ sjednocení $path_1$ a $path_2$
$area_{inter} \leftarrow$ plocha polygonu z $intersection$
$area_{union} \leftarrow$ plocha polygonu z $unionPath$
if $area_{union} = 0.0$ **then**
    vrátit 0.0
end if
vrátit $area_{inter}/area_{union}$

draw.cpp

Algorithm 14 Konstruktor třídy Draw

Vstup: Rodičovský widget *parent*
Výstup: Objekt třídy `Draw`
`isShapefileLoaded` ← `false` ▷ Inicializace flagu pro načtení shapefile

Algorithm 15 Zpracování události stisknutí myši (mousePressEvent)

Vstup: Objekt události e typu QMouseEvent
Výstup: Aktualizovaný seznam polygonů
$x \leftarrow e.pos().x()$ ▷ Získání X souřadnice kliknutého bodu
$y \leftarrow e.pos().y()$ ▷ Získání Y souřadnice kliknutého bodu
$p \leftarrow$ QPointF(x, y) ▷ Vytvoření bodu z kliknutých souřadnic
if polygons.isEmpty() **then** ▷ Pokud není žádný polygon, vytvoř nový
    polygons.append(QPolygonF()) ▷ Vytvoření nového prázdného polygonu
end if
polygons.last().append(p) ▷ Přidání bodu do posledního polygonu
repaint() ▷ Překreslení obrazovky

Algorithm 16 Vymazání výsledků geometrických operací

Vstup: Žádný vstup
Výstup: Vykreslený widget bez geometrických operací
maer.clear() ▷ Vymazání polygonů z kolekce maer
erpca.clear() ▷ Vymazání polygonů z kolekce erpca
le.clear() ▷ Vymazání polygonů z kolekce le
we.clear() ▷ Vymazání polygonů z kolekce we
wa.clear() ▷ Vymazání polygonů z kolekce wa
ch.clear() ▷ Vymazání polygonů z kolekce ch
chGraham.clear() ▷ Vymazání polygonů z kolekce chGraham
repaint() ▷ Rekreslení widgetu, což znamená vymazání zobrazených výsledků

Algorithm 17 Vymazání všech polygonů a geometrických objektů

```
Vstup: Žádný vstup
Výstup: Vykreslený widget bez polygonů a geometrických objektů
polygons.clear()        ▷ Vymazání všech polygonů z kolekce polygons
maer.clear()            ▷ Vymazání všech polygonů z kolekce maer
erpca.clear()           ▷ Vymazání všech polygonů z kolekce erpca
le.clear()              ▷ Vymazání všech polygonů z kolekce le
we.clear()              ▷ Vymazání všech polygonů z kolekce we
wa.clear()              ▷ Vymazání všech polygonů z kolekce wa
ch.clear()              ▷ Vymazání všech polygonů z kolekce ch
chGraham.clear()        ▷ Vymazání všech polygonů z kolekce chGraham
repaint()               ▷ Rekreslení widgetu, což znamená vymazání zobrazených objektů
```

Algorithm 18 Zpracování události vykreslování (paintEvent)

```
Vstup: Objekt události event typu QPaintEvent
Výstup: Vykreslený widget s polygonálními objekty
painter ← QPainter(this)                                  ▷ Vytvoření objektu pro kreslení
painter.begin(this)                                       ▷ Inicializace kreslení
painter.setPen(Qt::GlobalColor::black)                    ▷ Nastavení barvy pera na černou
painter.setBrush(Qt::GlobalColor::lightGray)              ▷ Nastavení barvy výplně na světle šedou
for i = 0 to polygons.size() - 1 do                       ▷ Pro každý polygon v seznamu polygons
    painter.drawPolygon(polygons[i])                      ▷ Vykresli polygon
end for
pen_maer.setColor(Qt::GlobalColor::cyan)                  ▷ Nastavení barvy pera na cyan pro MAER
painter.setPen(pen_maer)                                  ▷ Použití nastaveného pinu
painter.setBrush(Qt::GlobalColor::transparent)            ▷ Nastavení transparentní výplně
for i = 0 to maer.size() - 1 do                           ▷ Pro každý polygon v seznamu maer
    painter.drawPolygon(maer[i])                          ▷ Vykresli polygon MAER
end for
painter.setPen(Qt::GlobalColor::darkGreen)                ▷ Nastavení barvy pera na tmavě zelenou pro PCA
painter.setBrush(Qt::GlobalColor::transparent)            ▷ Transparentní výplň
for i = 0 to erpca.size() - 1 do                          ▷ Pro každý polygon v seznamu erpca
    painter.drawPolygon(erpca[i])                         ▷ Vykresli polygon PCA
end for
painter.setPen(Qt::GlobalColor::blue)                     ▷ Nastavení barvy pera na modrou pro LE
painter.setBrush(Qt::GlobalColor::transparent)            ▷ Transparentní výplň
for i = 0 to le.size() - 1 do                             ▷ Pro každý polygon v seznamu le
    painter.drawPolygon(le[i])                            ▷ Vykresli polygon LE
end for
penWE.setColor(QColor(255, 165, 0))                       ▷ Nastavení barvy pera na oranžovou pro WE
painter.setPen(penWE)                                     ▷ Použití nastaveného pinu
for i = 0 to we.size() - 1 do                             ▷ Pro každý polygon v seznamu we
    painter.drawPolygon(we[i])                            ▷ Vykresli polygon WE
end for
painter.setPen(Qt::GlobalColor::yellow)                   ▷ Nastavení barvy pera na žlutou pro WA
for i = 0 to wa.size() - 1 do                             ▷ Pro každý polygon v seznamu wa
    painter.drawPolygon(wa[i])                            ▷ Vykresli polygon WA
end for
painter.setPen(Qt::GlobalColor::magenta)                  ▷ Nastavení barvy pera na magentu pro CH Jarvis
for i = 0 to ch.size() - 1 do                             ▷ Pro každý polygon v seznamu ch
    painter.drawPolygon(ch[i])                            ▷ Vykresli polygon CH Jarvis
end for
painter.setPen(Qt::GlobalColor::cyan)                     ▷ Nastavení barvy pera na cyan pro CH Graham
for i = 0 to chGraham.size() - 1 do                       ▷ Pro každý polygon v seznamu chGraham
    painter.drawPolygon(chGraham[i])                      ▷ Vykresli polygon CH Graham
```

Algorithm 19 Funkce openFile

```
Otevři dialog pro výběr souboru
if soubor je vybrán then
    Otevři soubor
    if soubor je otevřen then
        Vyčisti seznam polygonů
        Vytvoř dočasný seznam pro polygon
        while nebyl dočten celý soubor do
            Přečti řádek a ořež bílé znaky
            if řádek je prázdný then
                if dočasný seznam polygonu není prázdný then
                    Přidej polygon do seznamu
                    Vyčisti dočasný seznam
                end if
            end if
            Rozděl řádek na souřadnice
            if má řádek 2 souřadnice then
                Získej hodnoty x a y
                if hodnoty jsou platné then
                    Přidej bod do dočasného polygonu
                end if
            end if
        end while
        if dočasný seznam není prázdný then
            Přidej poslední polygon do seznamu
        end if
        Zavři soubor
        Překresli obrazovku
    end if
end if
```

Algorithm 20 Funkce openSHP

```
Otevři dialog pro výběr souboru typu SHP
if soubor je vybrán then
    Registrovat všechny formáty GDAL
    Otevři SHP soubor
    if soubor je otevřen then
        Načti první vrstvu shapefile
        Vyčisti seznam polygonů
        while nejsou přečteny všechny prvky do
            Načti geometrický prvek
            if geometrie je polygon then
                Získej exteriérový prstenec polygonu
                for každý bod v prstenci do
                    Získej souřadnice x a y
                    Přidej bod do polygonu
                end for
                Přidej polygon do seznamu
            end if
        end while
        Zavři soubor
        Normalizuj polygony
        Překresli obrazovku
    end if
end if
```

mainforma.cpp

Algorithm 21 Konstruktor MainForm

Inicializuj `ui` pomocí `setupUi(this)`

Algorithm 22 Destruktor MainForm

Uvolni paměť pro `ui`

Algorithm 23 Vytvoření minimálního obvodového obdélníku (MBR)

Vstup: Žádný vstup (používá data z Canvas)
Výstup: Vykreslený minimální obvodový obdélník na Canvasu
polygons ← ui->$Canvas$->$getPolygons()$ ▷ Načtení polygonů z Canvasu
maer ← prázdný seznam polygonů ▷ Inicializace seznamu pro výsledky MBR
for každý polygon p_i v seznamu polygons **do**
  if maer je prázdný **then**
    maer.append(QPolygonF()) ▷ Přidej první prázdný polygon do seznamu maer
  end if
  maer.append(Algorithms::createMAER(p$_i$)) ▷ Vytvoř MBR pro aktuální polygon a přidej ho do seznamu maer
end for
ui->Canvas->setMAER(maer) ▷ Nastavení výsledků MBR na Canvas
repaint() ▷ Rekreslení widgetu pro zobrazení nových výsledků

Algorithm 24 Vytvoření "Area Enclosing Rectangle"(ERPCA)

Vstup: Žádný vstup (používá data z Canvasu)
Výstup: Vykreslený ERPCA na Canvasu
polygons ← ui->$Canvas$->$getPolygons()$ ▷ Načtení polygonů z Canvasu
erpca ← prázdný seznam polygonů ▷ Inicializace seznamu pro výsledky ERPCA
for každý polygon p_i v seznamu polygons **do**
  if erpca je prázdný **then**
    erpca.append(QPolygonF()) ▷ Přidej první prázdný polygon do seznamu erpca
  end if
  erpca.append(Algorithms::createERPCA(p$_i$))▷ Vytvoř ERPCA pro aktuální polygon a přidej ho do seznamu erpca
end for
ui->Canvas->setERPCA(erpca) ▷ Nastavení výsledků ERPCA na Canvas
repaint() ▷ Rekreslení widgetu pro zobrazení nových výsledků

Algorithm 25 Vytvoření "Longest Edge"(LE)

Vstup: Žádný vstup (používá data z Canvasu)
Výstup: Vykreslený polygon s nejdelšími hranami (LE) na Canvasu
polygons ← *ui->Canvas->getPolygons*() ▷ Načtení polygonů z Canvasu
le ← prázdný seznam polygonů ▷ Inicializace seznamu pro výsledky LE
for každý polygon p_i v seznamu polygons **do**
    if le je prázdný **then**
        le.append(QPolygonF()) ▷ Přidej první prázdný polygon do seznamu le
    end if
    le.append(Algorithms::createLongesEdge(p$_i$)) ▷ Vytvoř LE pro aktuální polygon a přidej ho do seznamu le
end for
ui->Canvas->setLE(le) ▷ Nastavení výsledků LE na Canvas
repaint() ▷ Rekreslení widgetu pro zobrazení nových výsledků

Algorithm 26 Vytvoření "Wide Enclosing"(WE)

Vstup: Žádný vstup (používá data z Canvasu)
Výstup: Vykreslený polygon s nejširšími obalovými diagonálami (WE) na Canvasu
polygons ← *ui->Canvas->getPolygons*() ▷ Načtení polygonů z Canvasu
we ← prázdný seznam polygonů ▷ Inicializace seznamu pro výsledky WE
for každý polygon p_i v seznamu polygons **do**
    if we je prázdný **then**
        we.append(QPolygonF()) ▷ Přidej první prázdný polygon do seznamu we
    end if
    we.append(Algorithms::createWE(p$_i$)) ▷ Vytvoř WE pro aktuální polygon a přidej ho do seznamu we
end for
ui->Canvas->setWE(we) ▷ Nastavení výsledků WE na Canvas
repaint() ▷ Rekreslení widgetu pro zobrazení nových výsledků

Algorithm 27 Vytvoření "Wall Average"(WA)

Vstup: Žádný vstup (používá data z Canvasu)
Výstup: Vykreslený polygon pro "Wall Average"(WA) na Canvasu
polygons ← *ui->Canvas->getPolygons*() ▷ Načtení polygonů z Canvasu
wa ← prázdný seznam polygonů ▷ Inicializace seznamu pro výsledky WA
for každý polygon p_i v seznamu polygons **do**
    if wa je prázdný **then**
        wa.append(QPolygonF()) ▷ Přidej první prázdný polygon do seznamu wa
    end if
    wa.append(Algorithms::createWA(p$_i$)) ▷ Vytvoř WA pro aktuální polygon a přidej ho do seznamu wa
end for
ui->Canvas->setWA(wa) ▷ Nastavení výsledků WA na Canvas
repaint() ▷ Rekreslení widgetu pro zobrazení nových výsledků

Algorithm 28 Vytvoření Convex Hull pomocí Jarvisova skenu (Jarvis Scan)

Vstup: Žádný vstup (používá data z Canvasu)
Výstup: Vykreslený polygon Convex Hull na Canvasu
polygons ← *ui->Canvas->getPolygons*() ▷ Načtení polygonů z Canvasu
ch ← prázdný seznam polygonů ▷ Inicializace seznamu pro výsledky Convex Hull
for každý polygon p_i v seznamu polygons **do**
    if ch je prázdný **then**
        ch.append(QPolygonF()) ▷ Přidej první prázdný polygon do seznamu ch
    end if
    ch.append(Algorithms::createCH(p$_i$)) ▷ Vytvoř Convex Hull pro aktuální polygon pomocí Jarvisova skenu
end for
ui->Canvas->setCH(ch) ▷ Nastavení výsledků Convex Hull na Canvas
repaint() ▷ Rekreslení widgetu pro zobrazení nových výsledků

Algorithm 29 Vytvoření Convex Hull pomocí Grahamova skenu (Graham Scan)

Vstup: Žádný vstup (používá data z Canvasu)
Výstup: Vykreslený polygon Convex Hull na Canvasu
polygons ← *ui->Canvas->getPolygons*() ▷ Načtení polygonů z Canvasu
chGraham ← prázdný seznam polygonů ▷ Inicializace seznamu pro výsledky Convex Hull
for každý polygon p_i v seznamu polygons **do**
    if chGraham je prázdný **then**
        chGraham.append(QPolygonF()) ▷ Přidej první prázdný polygon do seznamu chGraham
    end if
    chGraham.append(Algorithms::createCH$_Graham(p_i)$) ▷ Vytvoř Convex Hull pro aktuální polygon pomocí Grahamova skenu
end for
ui->Canvas->setCH(chGraham) ▷ Nastavení výsledků Convex Hull na Canvas
repaint() ▷ Rekreslení widgetu pro zobrazení nových výsledků

Algorithm 30 Funkce on_actionOpen_triggered

Zavolej funkci `openFile` na `Canvas` pro otevření souboru s polygonem

Algorithm 31 Funkce on_actionOpen_SHP_triggered

Zavolej funkci `openSHP` na `Canvas` pro načtení shapefile souboru

Algorithm 32 Funkce on_actionExit_triggered

Ukonči aplikaci pomocí `QApplication::quit()`

Algorithm 33 Vymazání výsledků na plátně (Clear Results)

Vstup: žádný
Výstup: prázdné plátno
ui->Canvas->clearResults() ▷ Vymazání všech výsledků z plátna (odstranění výsledků analýz)
repaint() ▷ Rekreslení plátna pro zobrazení prázdného stavu

Algorithm 34 Vymazání všech dat na plátně (Clear All)

Vstup: žádný
Výstup: prázdné plátno
ui->Canvas->clear() ▷ Vymazání všech dat: uživatelských polygonů i výsledků analýz
repaint() ▷ Rekreslení plátna pro zobrazení prázdného stavu

Odkazy

1. CHATGPT. *Asistenční pomoc s kódem a dokumenty* [`https://chat.openai.com`]. 2025. Pomocí AI asistenta ChatGPT od OpenAI.