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

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

Úloha byla vypracována v prostředí QT Creator, jazyk C++. Při tvorbě úlohy bylo vycházeno z probrané látky na cvičení a přednáškách. Pro generalizaci budov byly použity algoritmy *Minimum Area Enclosing Rectangle*, *Wall Average*, *Longest Edge* a *Weighted Bisector*. Jako součást nektěrých algoritmů bylo třeba sestrojit konvexní obálku, k tomu byly použity algoritmy *Jarvis Scan* a *Graham Scan*. Následující odstavce popisují jednotlivé soubory projektu a použité funkce, které byly sestaveny pro správné fungování aplikace.

3.1 algorithms.cpp

3.1.1 `normalizePolygons`

Tato funkce je určena pro úpravu souboru polygonů tak, aby všechny polygony byly umístěny do středu obrazovky nebo daného okna. Nejprve je vypočítáno *centroid*, což je průměrná pozice všech bodů napříč všemi polygonálními objekty. Centroid je považován za střed těžiště všech polygonů. Po výpočtu centroidu se každý polygon posune (nebo normalizuje) tak, aby jeho těžiště bylo umístěno ve středu okna nebo obrazovky. Posunutí se provádí pomocí ofsetů, které se získají z rozdílu mezi polohou centroidu a středem okna. Tento krok je užitečný zejména při zobrazení polygonů na obrazovce, protože zajistí, že všechny polygony budou dobře viditelné a vycentrované ve vymezeném prostoru.

3.1.2 `calculateCentroid`

Tato funkce vypočítá *centroid* (střed těžiště) souboru polygonů, což je průměrná pozice všech bodů, které tvoří všechny polygony. Funkce prochází všechny body každého polygonu a sčítá jejich souřadnice x a y, čímž získává celkový součet pro všechny souřadnice. Poté vyděluje tento součet počtem bodů, čímž získá průměrnou hodnotu pro souřadnice x a y. Výsledkem je bod, který představuje střední hodnotu pozice všech polygonů. Tento střed je užitečný pro různé operace, jako je centrální normalizace polygonů nebo pro výpočty geometrických vlastností. Tato funkce je užitečná například při pokusech o nalezení vyvážené pozice pro zobrazení polygonů nebo pro výpočty těžiště v dalších geometrických algoritmech.

3.1.3 `get2LineAngle`

Tato funkce vypočítá úhel mezi dvěma vektory, které jsou definovány čtyřmi body, tedy dvěma dvojicemi bodů. Pro výpočet úhlu mezi těmito vektory se používá *skalární součin* (dot product), který je definován jako:

$$\vec{a} \cdot \vec{b} = |a||b| \cos(\theta)$$

kde $\vec{a}$ a $\vec{b}$ jsou vektory, $|a|$ a $|b|$ jsou jejich velikosti (délky), a θ je úhel mezi nimi. Funkce tedy nejprve spočítá velikosti obou vektorů pomocí vzorců pro velikost vektoru a jejich skalární součin. Následně pomocí inverzního kosinu (arccos) vypočítá hodnotu úhlu θ mezi těmito dvěma

vektory. Tato funkce je užitečná při analýze geometrických vztahů mezi liniemi nebo vektory, například pro určení orientace dvou čar nebo při výpočtu sklonu dvou hran polygonu.

3.1.4 `createCH`

Funkce createCH generuje konvexní obálku pro zadaný polygon pomocí algoritmu Jarvis Scan, který iterativně vybírá body s největším úhlem vůči předchozím dvěma bodům. Začíná z bodem s nejnižší Y souřadnicí (pivotem) a postupně přidává body vytvářející maximální úhel s aktuálním směrem, dokud se nevrátí zpět do výchozího bodu. Výsledkem je nejmenší konvexní polygon, který obklopuje všechny body vstupního tvaru. Pokud vstupní data nejsou dostatečná nebo dojde k chybě během konstrukce, funkce vrací prázdný polygon.[1]

3.1.5 `createCH_Graham`

Funkce createCH_Graham implementuje algoritmus Graham Scan pro výpočet konvexní obálky zadaného polygonu. Nejprve najde pivot – bod s nejmenší Y souřadnicí – a spočítá úhly všech ostatních bodů vzhledem k tomuto pivotu. Body se poté seřadí podle těchto úhlů a přidávají se jeden po druhém do výsledné obálky, přičemž se průběžně kontroluje orientace posledních tří bodů. Pokud tvoří pravotočivý ohyb nebo jsou kolineární, poslední bod se odstraní, aby se zachovala konvexita. Výsledkem je polygon tvořený vrcholy v konvexním uspořádání.[1]

3.1.6 `minmaxBox`

Funkce minmaxBox vytvoří obdélík ohraničující polygon, který má hrany rovnoběžné se souřadnicovými osami. Nejprve nalezne extrémní souřadnice všech bodů polygonu:

- $x_{\min} = \min(x_i)$
- $x_{\max} = \max(x_i)$
- $y_{\min} = \min(y_i)$
- $y_{\max} = \max(y_i)$

Z těchto souřadnic sestaví vrcholy obdélníku:

$$v_1 = (x_{\min}, y_{\min})$$
$$v_2 = (x_{\max}, y_{\min})$$
$$v_3 = (x_{\max}, y_{\max})$$
$$v_4 = (x_{\min}, y_{\max})$$

Tento obdélník vrací ve formě objektu `QPolygonF` spolu s jeho obsahem, který je vypočten jako:

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

Výsledkem je dvojice `(polygon, area)`.

3.1.7 `rotate`

Funkce rotate provádí otočení (rotaci) všech bodů polygonu `pol` o zadaný úhel σ (v radiánech) kolem počátku souřadného systému.

Každý bod (x, y) je transformován pomocí rotační matice:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\sigma & -\sin\sigma \\ \sin\sigma & \cos\sigma \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} \quad \Rightarrow \quad \begin{cases} x' = x\cos\sigma - y\sin\sigma \\ y' = x\sin\sigma + y\cos\sigma \end{cases}$$

Nově vypočítané body jsou vloženy do výsledného polygonu, který je vrácen jako výstup. Otočení se provádí vzhledem k počátku $(0, 0)$.

3.1.8 `get_area`

Funkce get_area vypočítává obsah polygonu `pol` pomocí *L'Huilierových vzorců*):

$$\text{area} = \frac{1}{2}\left|\sum_{i=0}^{n-1} x_i \cdot (y_{i+1} - y_{i-1})\right|$$

kde indexy jsou počítány cyklicky pomocí modulo operace (tedy $x_{-1} = x_{n-1}$ a $x_n = x_0$). Výsledná hodnota je absolutní hodnota součtu dělená dvěma, čímž se získá kladná plocha i pro polygon s hodinovým směrem průchodu.

3.1.9 `resize`

Funkce resize upravuje velikost minimálního ohraničujícího obdélníku (`mmbox`) tak, aby měl stejný obsah jako vstupní polygon `pol`, přičemž zachovává jeho tvar a orientaci.

Nejprve se spočítá obsah polygonu A_b a obsah obdélníku A, ze kterého se určí škálovací faktor:

3.1.10 `createMAER`

Funkce createMAER slouží k vytvoření minimálního obalujícího obdélníku (MAER - Minimum Area Enclosing Rectangle) pro zadaný polygon pol. Využívá konvexní obálku (Convex Hull) a rotaci obdélníku k nalezení orientace s nejmenší plochou.[1]

3.1.11 `createERPCA`

Funkce createERPCA používá metodu *Principal Component Analysis* (PCA) k vytvoření minimálního ohraničujícího obdélníku pro zadaný polygon `pol`, který je orientován podle hlavní osy polygonu. vodní polygon, přičemž je orientován podle hlavní osy polygonu.[1]

3.1.12 `getDistance`

Tato funkce vrací vzdálenost dvou bodů ze souřadnicových rozdílů.

3.1.13 `createLongestEdge`

Funkce createLongestEdge vytváří minimální ohraničující obdélník pro polygon `pol`, který je orientován podle nejdelší hrany polygonu.

Po nalezení nejdelší hrany se spočítá úhel rotace σ, který orientuje tuto hranu podél osy x. Polygon je následně otočen o úhel $-\sigma$, čímž je nejdelší hrana umístěna podél osy x.

Dále je pro otočený polygon spočítán minimální ohraničující obdélník pomocí funkce `minmaxBox`. Tento obdélník je následně upraven, aby měl stejnou plochu jako původní polygon, pomocí funkce `resize`. Nakonec je tento obdélník otočen zpět o úhel σ a vrácen jako výstup.

3.1.14 `createWE`

Funkce createWE vytváří minimální ohraničující obdélník pro polygon `pol`, který je orientován podle dvou nejdelších diagonál polygonu.

Funkce nejprve prochází všechny možné dvojice bodů v polygonu a pro každou dvojici bodů i a j (kdy $i \neq j$) vypočítává vzdálenost mezi těmito dvěma body.

Po nalezení dvou nejdelších diagonál se spočítá úhel rotace σ_1 pro první diagonálu a σ_2 pro druhou diagonálu. Kombinovaný úhel rotace σ se pak vypočítá jako vážený průměr těchto dvou úhlů podle délek diagonál.

Polygon je následně otočen o úhel $-\sigma$, čímž jsou dvě nejdelší diagonály umístěny v optimálním směru. Poté je pro otočený polygon spočítán minimální ohraničující obdélník pomocí funkce `minmaxBox`. Tento obdélník je následně upraven pomocí funkce `resize`, aby měl stejnou plochu jako původní polygon. Nakonec je tento obdélník otočen zpět o úhel σ a vrácen jako výstup.[1]

3.1.15 `createWA`

Funkce createWA vytváří minimální ohraničující obdélník pro polygon `pol` s orientací založenou na směrech jeho hran.

Nejprve se pro každou hranu polygonu spočítají její směr a délka. Směr hrany se určuje pomocí arktangentu z rozdílu souřadnic bodů hrany, a délka hrany je vypočítána jako Eukleidovská vzdálenost mezi těmito dvěma body. Směry a délky hran se ukládají do vektorů `sigmas` a `lengths`.

Jako referenční směr σ' je vybrán směr první hrany. Poté se pro každý směr hrany spočítá rozdíl od referenčního směru, který je normalizován do rozsahu $(-\pi, \pi)$. Na základě tohoto rozdílu se vypočítají hodnoty k_i a r_i, které jsou použity k výpočtu váženého průměru směrů.

Konečný směr σ je určen jako vážený průměr směru hrany, přičemž váhami jsou délky hran. Tento směr je následně použit k rotaci polygonu, jehož výsledek je poté uložen do nového polygonu `pol_rot`.

Dalšími kroky jsou výpočet minimálního ohraničujícího obdélníku pro otočený polygon, jeho úprava pomocí funkce `resize`, a nakonec zpětná rotace o směr σ, čímž se získá požadovaný výsledek.[1]

3.2 draw.cpp

Tento kód implementuje třídu `Draw`, která umožňuje vykreslování polygonů. Umožňuje interakci s uživatelem pomocí myši pro zadávání polygonů a poskytuje funkce pro načítání polygonů z textového a SHP souboru a vykreslování polygonů.

3.2.1 Zachycení stisku myši

Funkce `mousePressEvent(QMouseEvent *e)` reaguje na stisk myši. Přidá bod do posledního polygonu a následně zavolá `repaint()` pro překreslení.

3.2.2 Vymazání polygonů

Funkce `clearPolygons()` odstraní všechny polygony a vykreslí plátno. Funkce `clear()` odstraní všechny polygony a také provede překreslení.

3.2.3 Vykreslování objektů

Funkce `paintEvent(QPaintEvent *event)` vykresluje polygony červeně s žlutým vyplněním.

3.2.4 Načtení polygonů z textového souboru

Funkce `openFile()` otevře textový soubor s polygonovými daty. Každý řádek obsahuje dvě hodnoty souřadnic bodu polygonu. Prázdný řádek znamená konec polygonu. Funkce načte souřadnice a vytvoří `QPolygonF` objekty, které uloží do vektoru `polygons`. Po načtení polygonů se plátno překreslí a režim přepne na přidávání bodu.

3.2.5 Načtení polygonů z SHP souboru

Funkce `openSHP()` načítá data z SHP souboru pomocí knihovny GDAL. Po načtení polygonů je normalizuje pomocí `algorithms::normalizePolygons()` a plátno překreslí. Režim přepne na přidávání bodu.

3.3 mainform.cpp

Tento kód představuje implementaci hlavní třídy `MainForm`, která slouží k interakci s uživatelem v prostředí Qt. Obsahuje různé akce pro analýzu polohy bodu vůči polygonu, načítání dat a vykreslování výsledků. Kód používá grafickou knihovnu Qt a knihovnu GDAL pro načítání shapefile souborů.

Inicializace okna aplikace

Funkce `MainForm::MainForm(QWidget *parent)` je konstruktor, který nastavuje uživatelské rozhraní aplikace pomocí `ui->setupUi(this)`. V destruktoru `MainForm:: MainForm()` je pak uvolněna paměť pro uživatelské rozhraní.

Volání algortimů

Funkce on$_a ction MBR_t riggered(), on_a ction PCA_t riggered(), on_a ction LE_t riggered(), on_a ction WE_t riggere$

Získání polygonů: Funkce začínají získáním seznamu polygonů z uživatelského rozhraní (pomocí metody ui-¿Canvas-¿getPolygons()).

Vytvoření prázdného seznamu pro výsledky: Pro každý algoritmus je vytvořen prázdný seznam, do kterého budou ukládány výsledky zpracování jednotlivých polygonů.

Iterace přes polygony: Funkce iterují přes každý polygon ve vstupním seznamu a aplikují na něj konkrétní algoritmus.

Přidání výsledků do seznamu: Výstupy z jednotlivých algoritmů jsou přidávány do příslušného výsledného seznamu.

Zobrazení výsledků: Po zpracování všech polygonů jsou výsledky zobrazeny na uživatelském rozhraní pomocí příslušných metod.

Obnovení zobrazení: Nakonec se obrazovka obnoví, aby se zobrazily nové výsledky, a to pomocí metody repaint().

Všechny tyto funkce vykonávají podobné kroky, liší se pouze tím, jaký konkrétní algoritmus nebo geometrickou transformaci aplikují na vstupní polygony.

Načtení polygonů z textového souboru

Funkce `on_actionOpen_triggered()` umožňuje načíst polygonální data z textového souboru. Po načtení jsou polygony vykresleny na plátno.

Načtení souboru SHP pomocí GDAL

Funkce `on_actionOpen_SHP_triggered()` načítá shapefile soubor (.shp) pomocí knihovny GDAL. Po načtení souboru jsou polygony vykresleny na plátno.

Ukončení aplikace

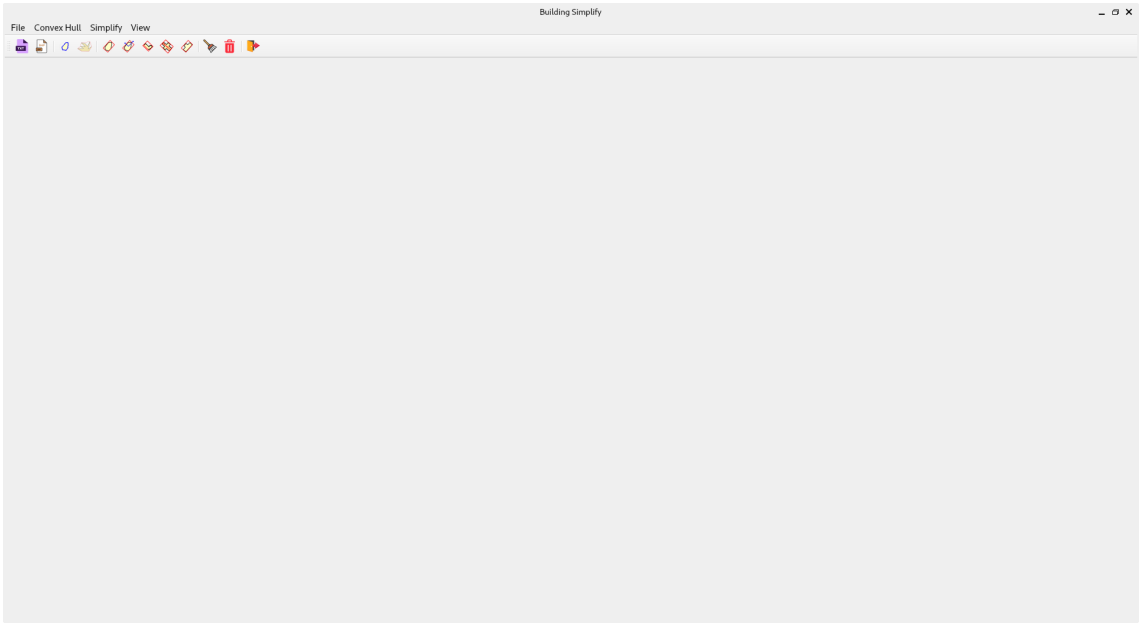
Funkce `on_actionExit_triggered()` ukončuje běh aplikace tím, že volá metodu `QApplication::quit()`.

Vymazání dat

Funkce `on_actionClear_all_triggered()` odstraní všechny vykreslené polygony a výsledky z plátna. Funkce `on_actionClear_results_triggered()` vymaže výsledky, a provede překreslení plátna.

4 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

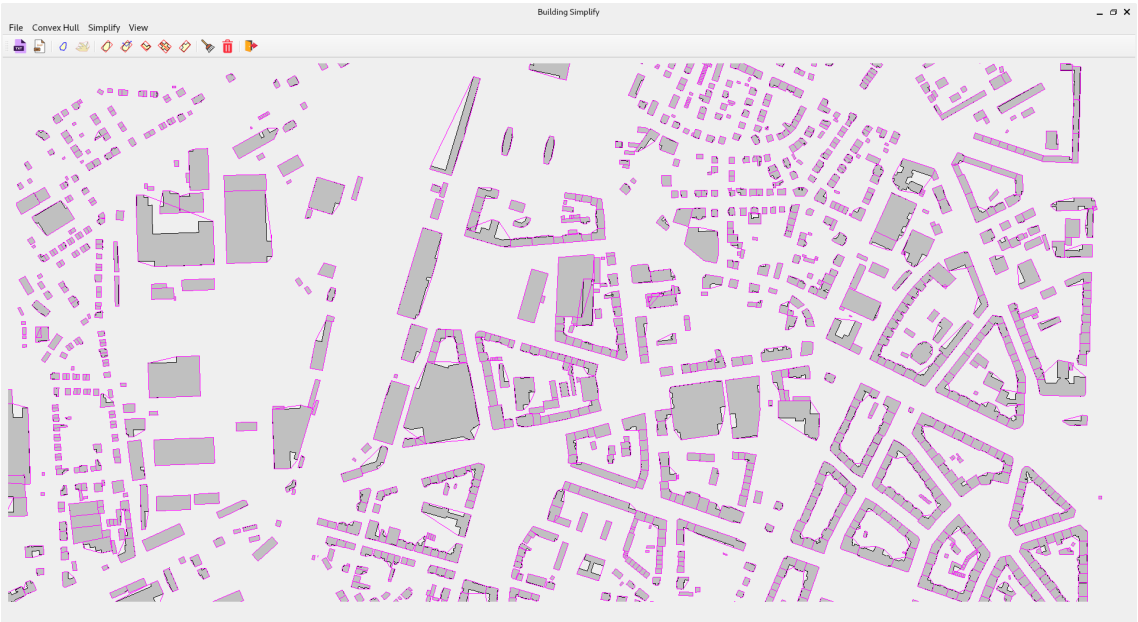
Polygon, který bude vyhodnocován může nakreslit uživatel sám nebo může být načten z SHP. Kreslení polygonu je možné ihned po spuštění aplikace. Pro SHP formátu je vytvořena speciální ikona.

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*.

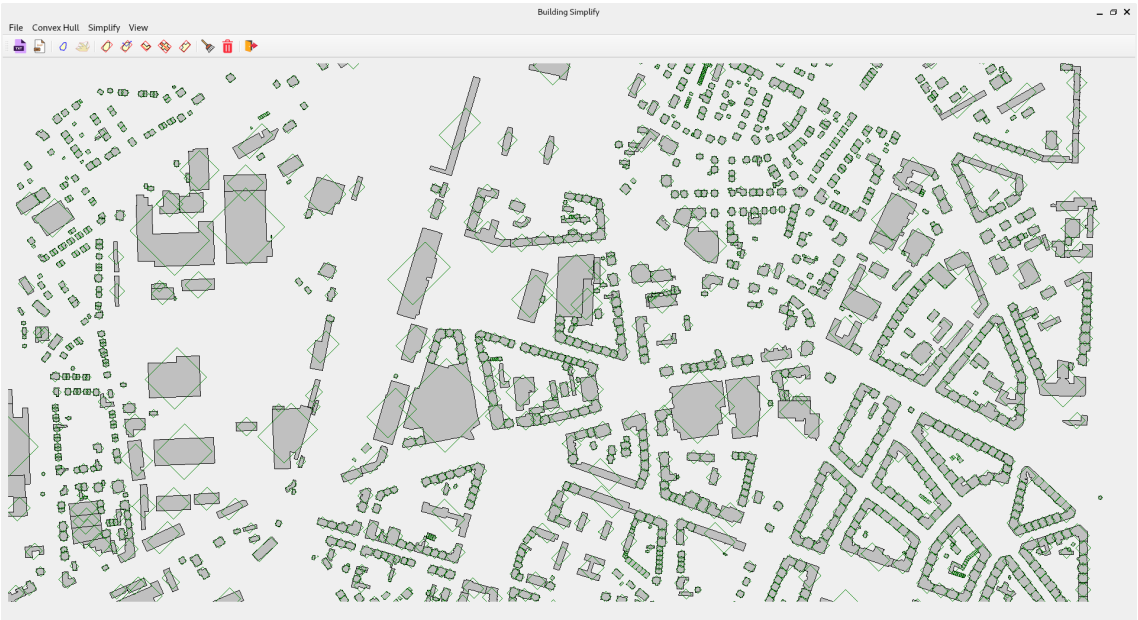
Obrázek 4: Konvexní obálka pro samostatný polygon



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

5 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*.

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 ← 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 [2]

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) [2]

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) [2]

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 [2]

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**
    $p_1 \leftarrow pol[i]$
    $p_2 \leftarrow pol[(i+1) \bmod n]$
    $dx \leftarrow p_2.x - p_1.x$
    $dy \leftarrow p_2.y - p_1.y$
    $\sigma \leftarrow \text{atan2}(dy, dx)$ ▷ 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
    $k_i \leftarrow \text{round}\left(\dfrac{2\Delta\sigma}{\pi}\right)$
    $r_i \leftarrow \Delta\sigma - k_i \cdot \left(\dfrac{\pi}{2}\right)$
    $sum\_r\_s \leftarrow sum\_r\_s + r_i \cdot lengths[i]$
    $sum\_s \leftarrow sum\_s + lengths[i]$
end for
$\sigma \leftarrow \sigma_{ref} + \dfrac{sum\_r\_s}{sum\_s}$
$pol_{rot} \leftarrow \text{rotate}(pol, -\sigma)$
$[box, area] \leftarrow \text{minmaxBox}(pol_{rot})$
$box_{resized} \leftarrow \text{resize}(pol, box)$
vrátit $\text{rotate}(box_{resized}, \sigma)$

draw.cpp

Algorithm 13 Konstruktor třídy Draw

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

Algorithm 14 Zpracování události stisknutí myši (mousePressEvent)

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

Algorithm 15 Vymazání výsledků geometrických operací

1: **Vstup:** Žádný vstup
2: **Výstup:** Vykreslený widget bez geometrických operací
3: maer.clear() ▷ Vymazání polygonů z kolekce maer
4: erpca.clear() ▷ Vymazání polygonů z kolekce erpca
5: le.clear() ▷ Vymazání polygonů z kolekce le
6: we.clear() ▷ Vymazání polygonů z kolekce we
7: wa.clear() ▷ Vymazání polygonů z kolekce wa
8: ch.clear() ▷ Vymazání polygonů z kolekce ch
9: chGraham.clear() ▷ Vymazání polygonů z kolekce chGraham
10: repaint() ▷ Rekreslení widgetu, což znamená vymazání zobrazených výsledků

Algorithm 16 Vymazání všech polygonů a geometrických objektů

1: **Vstup:** Žádný vstup
2: **Výstup:** Vykreslený widget bez polygonů a geometrických objektů
3: polygons.clear() ▷ Vymazání všech polygonů z kolekce polygons
4: maer.clear() ▷ Vymazání všech polygonů z kolekce maer
5: erpca.clear() ▷ Vymazání všech polygonů z kolekce erpca
6: le.clear() ▷ Vymazání všech polygonů z kolekce le
7: we.clear() ▷ Vymazání všech polygonů z kolekce we
8: wa.clear() ▷ Vymazání všech polygonů z kolekce wa
9: ch.clear() ▷ Vymazání všech polygonů z kolekce ch
10: chGraham.clear() ▷ Vymazání všech polygonů z kolekce chGraham
11: repaint() ▷ Rekreslení widgetu, což znamená vymazání zobrazených objektů

Algorithm 17 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 18 Funkce openFile

```
1: Otevři dialog pro výběr souboru
2: if soubor je vybrán then
3:     Otevři soubor
4:     if soubor je otevřen then
5:         Vyčisti seznam polygonů
6:         Vytvoř dočasný seznam pro polygon
7:         while nebyl dočten celý soubor do
8:             Přečti řádek a ořež bílé znaky
9:             if řádek je prázdný then
10:                if dočasný seznam polygonu není prázdný then
11:                    Přidej polygon do seznamu
12:                    Vyčisti dočasný seznam
13:                end if
14:            end if
15:            Rozděl řádek na souřadnice
16:            if má řádek 2 souřadnice then
17:                Získej hodnoty x a y
18:                if hodnoty jsou platné then
19:                    Přidej bod do dočasného polygonu
20:                end if
21:            end if
22:        end while
23:        if dočasný seznam není prázdný then
24:            Přidej poslední polygon do seznamu
25:        end if
26:        Zavři soubor
27:        Překresli obrazovku
28:    end if
29: end if
```

Algorithm 19 Funkce openSHP

```
1: Otevři dialog pro výběr souboru typu SHP
2: if soubor je vybrán then
3:     Registrovat všechny formáty GDAL
4:     Otevři SHP soubor
5:     if soubor je otevřen then
6:         Načti první vrstvu shapefile
7:         Vyčisti seznam polygonů
8:         while nejsou přečteny všechny prvky do
9:             Načti geometrický prvek
10:            if geometrie je polygon then
11:                Získej exteriérový prstenec polygonu
12:                for každý bod v prstenci do
13:                    Získej souřadnice x a y
14:                    Přidej bod do polygonu
15:                end for
16:                Přidej polygon do seznamu
17:            end if
18:        end while
19:        Zavři soubor
20:        Normalizuj polygony
21:        Překresli obrazovku
22:    end if
23: end if
```

mainforma.cpp

Algorithm 20 Konstruktor MainForm

1: Inicializuj `ui` pomocí `setupUi(this)`

Algorithm 21 Destruktor MainForm

1: Uvolni paměť pro `ui`

Algorithm 22 Vytvoření minimálního obvodového obdélníku (MBR)

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

Algorithm 23 Vytvoření "Area Enclosing Rectangle"(ERPCA)

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

Algorithm 24 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 25 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 26 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 27 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 28 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$_{G}raham(p_i)$) ▷ 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 29 Funkce on_actionOpen_triggered

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

Algorithm 30 Funkce on_actionOpen_SHP_triggered

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

Algorithm 31 Funkce on_actionExit_triggered

Ukonči aplikaci pomocí `QApplication::quit()`

Algorithm 32 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 33 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. BAYER, Tomáš. *ADK 4: Digitální kartografie a geoinformatika*. 2022. Dostupné také z: `https://web.natur.cuni.cz/~bayertom/images/courses/Adk/adk4.pdf`. Přednáškové materiály, PřF UK.
2. CHATGPT. *Asistenční pomoc s kódem a dokumenty* [`https://chat.openai.com`]. 2025. Pomocí AI asistenta ChatGPT od OpenAI.