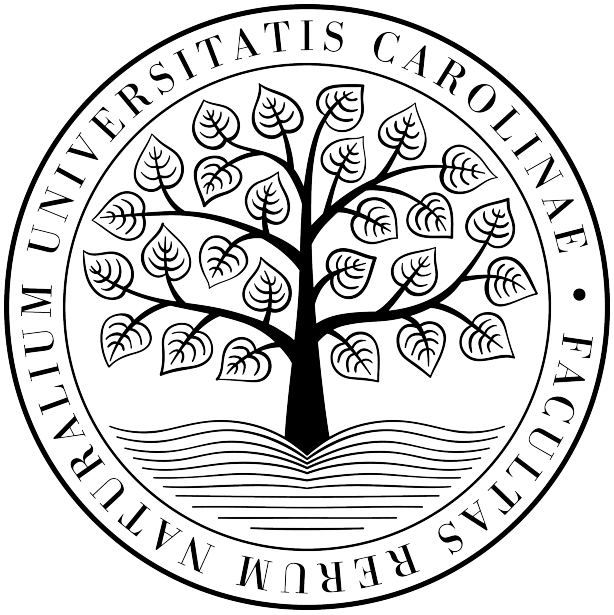
Přírodovědecká fakulta
Univerzita Karlova



Algoritmy počítačové kartografie

Úkol č. 2: Generalizace budov

Vanda Hlaváčková a Petra Krsková

1.N-GKDPZ

Praha 2023

1 Zadání

Úloha č. 2: Geometrické vyhledávaní bodu

Vstup: množina budov $B = \{Bi\}$*, budova* $Bi = \{P_{i,j}\}$
Výstup: $G(Bi)$*.*

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

Pro každou budovu určete její hlavní směry metodami *Minimum Area Enclosing Rectangle*, *Wall Average*.

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

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

Hodnocení:

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

2 Popis a rozbor problému

2.1 Generalizace budov

S automatizovanou generalizací map se můžeme setkat už od 60. letech minulého století. K výraznému vývoji však dochází až v posledních dvou desetiletích (LI a kol. 2004). Snadný přístup do online prostředí zvýšil poptávku po mapování ve více měřítkách, tedy jak například oblast zahrnující nejbližší sousedství domu až po celou zeměkouli. Webové mapy jako jsou Bing Maps, Google Maps nebo ArcGIS Online World Streetmap zobrazují silnice a ulice. Je tedy nutné mít automatizovaný přístup pro generalizaci podrobných dat (Punt, Watkins 2010).

Generalizace budov je vzhledem k často komplikovanému prostorovému uspořádání či kvůli prostorovému rozpoznávání objektů poměrně složitou operací. Výstupem efektivní kartografické generalizace je tedy zachování charakteru a vztahů mezi sousedními prvky a zjednodušené zobrazení, které ale zachová vizuální přesnost. Generalizační algoritmy vychází z manuální praxe, kdy reklasifikace nebo výběr/vyloučení prvků je v digitálním prostředí snadněji implementovatelné oproti rozpoznávání prostorového uspořádání a vztahů mezi objekty. Příkladem obtížně proveditelnou operací je nahrazení mnoho prvků jedním – agregace. Pokud je cílem seskupit například budovy znázorněné pomocí bodů, je nutné uvažovat jejich vztah vůči poloze ulic, řek nebo vůči dalším relevantním prvků (Punt, Watkins 2010).

Celý proces automatické generalizace budov bývá často rozdělen do třech kroků. V prvním části je popsán a provedena analýza prostorového kontextu geografických prvků. Následně je s ohledem na zjištěné parametry provedeno algoritmické zpracování. Třetím krokem je pak vyhodnocení výsledků generalizace (LI a kol. 2004).

Pro automatickou generalizaci budov je důležité znát jejich orientaci, což na rozdíl od úsečky nebo bodu je pro mnohoúhelník poněkud obtížné. Orientace budov odpovídá buď orientaci jejích stěn (slouží pro porovnání orientace dvou rovnoběžných stěn) nebo obecné orientaci (slouží k prodloužení budovy) (Duchene 2003). Důležitým kritériem kartografické generalizace je to, aby budova měla před i po generalizaci stejnou orientaci vzhledem k ostatním prvkům na mapě. Je tedy nutné definovat hlavní směry budovy, které právě tuto orientaci popisují. Algoritmy, pomocí kterých lze určit hlavní směry budov, jsou například *Longest Edge*, *Weighted Bisektor*, *Minimum Area Enclosing Rectangle*, *Wall Average* nebo *Metoda hlavních komponent*.

2.2 Konvexní obálka

Konstrukci konvexních obálek řadíme mezi problémy optimalizace. Jedná se o sestrojení nejkratší možné křivky, která by ohraničovala vstupní sadu bodů v rovině. Výsledná křivka pak odpovídá konvexnímu mnohoúhelníku v jehož středu leží některé zadané body. Zbylé se pak nachází buď vně nebo na hraně mnohoúhelníku.

V konvexním mnohoúhelníku platí, že všechny jeho vnitřní úhly jsou menší nebo rovny 180° (tedy jsou konvexní). Dále pak platí, že jakékoliv dva body a úsečka je propojující leží v konvexní množině (Felkel 2020). Konvexní obálky lze pak mimo detekci tvaru a orientaci budov v kartografii využít například k analýze shluků, plánování pohybu robotů nebo pro statické analýzy. Pro sestrojení konvexních obálek existuje několik algoritmů, například: *Jarvis Scan*, *Graham Scan*, *Quick Hull*, *Inkrementální konstrukce* nebo *Divide and Conquer*. Jednotlivé algoritmy se pak od sebe liší především časovou a paměťovou náročností.

2.2.1 Jarvis Scan

Jarvis Scan nebo také *Gift Wrapping Algorithm* patří mezi nejpoužívanější algoritmy pro konstrukci konvexní obálky a to především díky jednoduchosti implementace. Je vhodné ho použít především pro vstupní data, jejichž množina bodů n není příliš velká.

Na začátku algoritmus vybere bod P ze vstupní množiny tak, že jeho souřadnice jsou minimální k ose x, jelikož takový bod leží uvnitř konvexní obálky (označován jako *pivot*). Následně je bod přidán do seznamu vrcholů tvořících konvexní obálku. Z bodu P je pak hledán další bod q, který leží na přímce procházející bodem P tak, aby maximalizoval úhel σ (Bayer).

Pseudokód metody Jarvis Scan

Algorithm 1 *Jarvis Scan*

```
Najdi bod P s min y souřadnicí
Přidej bod P do konvexní obálky ch
Inicializuj bod p_{j-1} a p_min = P
Dokud p_{j+1} ≠ p_min
    Pro každý bod P:
        Vypočítej ω
    p_{j+1} = max ω
    Přidej p_{j+1} do ch
    Aktualizuj poslední dva body ch
```

2.2.2 Graham Scan

U algoritmu *Graham Scan* vstupuje kritérium levotočivosti, které musí splňovat každá uspořádaná trojice bodů. Nejprve je nalezen bod P s nejnižší $y-ovou$ souřadnicí. V případě, že je takových bodů více, je vybrán bod s nejnižší $x-ovou$ souřadnicí. Následně je množina vstupních bodů seřazena vzestupně v závislosti na úhlu, který svírají s bodem P a s osou x. Pokud je pro více bodů stejná hodnota úhlu, jsou vymazány všechny body kromě toho nejvzdálenějšího. Algoritmus dále postupuje tak, že pro každý bod určí, zda cesta z předcházejících dvou bodů představuje zatočení doprava nebo doleva. V případě, že se jedná o pravotočivou orientaci, je předposlední bod vyloučen, jelikož se nenachází na, ale vně konvexní obálky. Při splnění levotočivého kritéria algoritmus přejde k dalšímu bodu v setříděné množině bodů (odečteny jsou ty body, pro které bylo zjištěno, že se nachází uvnitř konvexní obálky).

Pseudokód metody Graham Scan

Algorithm 2 *Graham Scan*

```
1: Inicializuj konvexní obálku ch
2: Najdi bod P s min y souřadnicí
3: Seřaď body podle úhlu od nejmenšího po největší
4:     Vypočítej úhel Ω
5: Inicializuj j = 1
6: Dokud j < počet bodů n:
7:     Pokud bod v levé polorovině:
8:         Přidej bod do ch
9:         Aktualizuj j+ = 1
10:    Pokud bod v pravé polorovině:
11:        Odeber ho z ch
```

2.3 Minimum Area Enclosing Rectangle

Jak již bylo výše uvedeno, *Minimum Area Enclosing Rectangle* je algoritmus, který slouží k určování hlavních směrů budovy, čímž lze následně určit její orientaci. Nejprve je sestrojena konvexní obálka, která je následně opakovaně otáčena na základě matice rotace o úhel $-\sigma$ (směrnice konvexní obálky).

$$S_r = R(-\sigma)S \Rightarrow \begin{bmatrix} x_r \\ y_r \end{bmatrix} = \begin{bmatrix} \cos(-\sigma) & -\sin(-\sigma) \\ \sin(-\sigma) & \cos(-\sigma) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

V této poloze je pak nalezen minimální *Minimum Bounding Box* a vypočítána jeho plocha. Pomocí následujícího vzorečku zjistíme poměr mezi plochou budovy a plochou minimálního MMB:

$$k = \frac{S_b}{S_{er}}$$

Na závěr jsou vypočítány polohy nově vzniklých bodů pomocí těžiště, kdy do výpočtu vstupuje vektor pro

všechny vrcholy obdélníku MMB (Arnon, Gieselmann 1983).

Pseudokód metody Minimum Area Enclosing Rectangle

Algorithm 3 *Minimum Area Enclosing Rectangle*

```
Vytvoř konvexní obálku ch pro množiny bodů S
Najdi MMB a vypočítej jeho plochu S
Pro každou hranu e v ch:
    Vypočítej směrnici σ_i pro danou hranu e
    Otoč ch o −σ
    Zkonstuuj MMB a jeho plochu S_i
    Pokud S_i < S:
        S = S_i
        σ = σ_i
        MMB = MMB_i
Otoč MMB_i o σ_i a vypočítej jeho plochu
Uprav plochu tak, aby S_i = S
```

2.4 Wall Average

Dalším algoritmem pro hledání orientace budov je *Wall Average*. Pro orientaci budov je uvažována hodnota modulo $\frac{\pi}{2}$ v rozmezí 0 a $\frac{\pi}{2}$, kdy je vypočítán průměr těchto směrů. Nejprve je určena směrnice σ' a následně směrnice σ_i pro všechny strany, které jsou redukovány právě o hodnotu σ'.

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

Následně lze vypočítat zaokrouhlený podíl k_i jako:

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

V případě, že je hodnota zbytku $r_i < \frac{\pi}{4}$, je daná hrana odchýlena od vodorovného směru $(0 \pm k\pi)$. Pokud $r_i > \frac{\pi}{4}$, je hrana odchýlena od svislého směru $(\frac{\pi}{2} \pm k\pi)$. Výsledný směr natočení budovy je určen pomocí výpočtu průmerného úhlu.

Pseudokód metody Wall Average

Algorithm 4 *Wall Average*

1: Vypočítej směrnici σ' pro libovolnou hranu
2: Pro všechny hrany budovy:
3: Vypočítej směrnici σ_i
4: Vypočítej $\Delta\sigma'$
5: Vypočítej k_i a r_i
6: Vypočítej výslednou směrnici σ a otoč všechny budovy o její hodnotu
7: Vytvoř MMB a otoč ho o σ
8: Změň obsah MMB tak, aby $S_{MMB} = S_b$

2.5 Weighted Bisector

Při zjišťování orientace pomocí algoritmu Weighted Bisector jsou nalezeny dvě nejdelší úhlopříčky v polygonu a to tak, že dojde ke spojení postupně každého bodu s každým. V případě, že nově vzniklá úsečka prochází skrz hranu budovy, je vyloučena jako možná úhlopříčka. Jestli úsečka prochází nebo neprochází hranou je zjištěno pomocí průsečíků, kdy jsou porovnány polohy koncových bodů jedné úsečky oproti koncovým bodům úsečky druhé.

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

Podle znaménka výsledného determinantu jednotlivých směrových vektorů lze určit, zda daný průsečík existuje. V případě, že by $t1$ a $t2$ nebo $t3$ a $t4$ měly totožné znaménko, průsečík neexistuje, úsečka neprochází hranou, jedná se tedy o úhlopříčku. Následně jsou vybrány dvě nejdelší úhlopříčky, pro které je spočítána jejich směrnice. Výsledná orientace budovy je dána váženým průměrem vypočítaných směrnic. Jako váhy do vzorečku vstupují délky daných úhlopříček (s_1 a s_2) (Regnauld 1998, Duchene a kol. 2003).

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

Pseudokód metody Weighted Bisector

Algorithm 5 *Weighted Bisector*

1: Najdi dvě nejdelší úhlopříčky budovy
2: Vypočítej směrnice σ_1 a σ_2 pro dané úhlopříčky
3: Urči hlavní směr budovy
4: Otoč budovu o $-\sigma$
5: Zkonstruuj MMB a vypočítej jeho plochu S
6: Otoč MMB o σ
7: Vypočítej plochu budovy S_b a uprav MMB tak, aby $S = S_b$

2.6 Longest Edge

Hlavní směr je u algoritmu *Longest Edge* určen směrnicí nejdelší strany budovy. Druhý směr je pak na první směr kolmý. Budova je následně opět jako u výše zmíněných algoritmů otočena o $-\sigma$. V této poloze je vytvořen *Minimum Bounding Box*, který je pak otočen o kladnou hodnotu téhož úhlu zpět a následně upraven tak, aby jeho plocha odpovídala ploše budovy (Regnauld 1998).

Pseudokód metody Longest Edge

Algorithm 6 *Longest Edge*

1: Najdi nejdelší stranu budovy a vypočítej její směrnici
2: Vypočítej směrnici σ a otoč budovu o její zápornou hodnotu
3: Zkonstruuj MMB a vypočítej jeho plochu S
4: Otoč MMB o úhel σ
5: Vypočítej plochu budovy S_b a uprav MMB tak, aby $S = S_b$

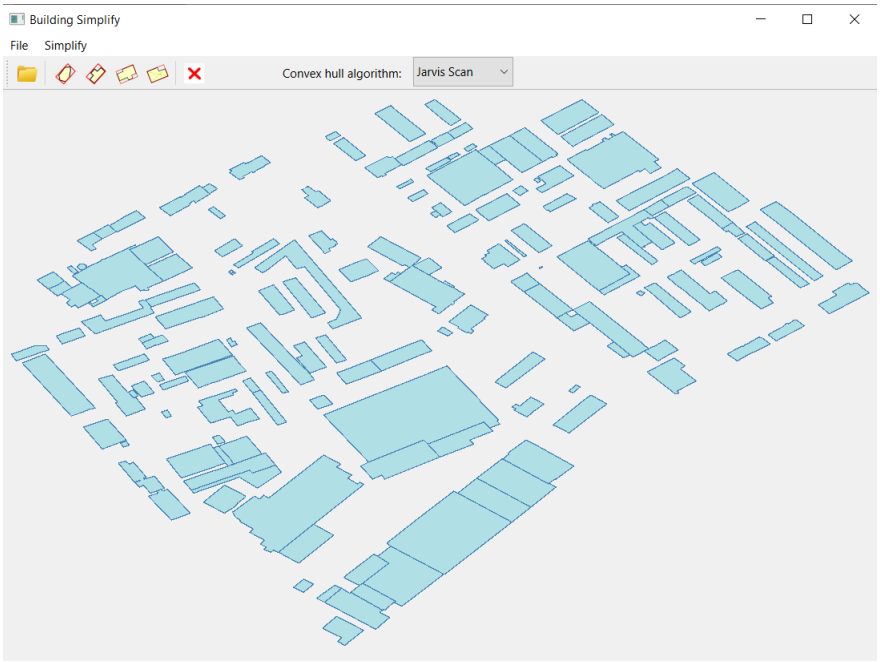
3 Aplikace

Aplikace pro generalizaci budov při zachování jejich orientace byla vytvořena v prostředí *Qt Creator* a jsou v ní implementovány výše popsané algoritmy - *Minimum Area Enclosing Rectangle*, *Wall Average*, *Longest Edge* a *Weighted Bisector*.

Po spuštění aplikace přes soubor *mainform.py* je otevřeno okno, jehož většinu tvoří prázdná plocha pro vykreslení polygonů. V horní části se pak nachází menu s jednotlivými funkcemi. V záložce *File* je možné pomocí funkce *Open* vyvolat dialogové okno a vybrat vstupní data ve formátu *.shp*, která jsou následně vykreslena (viz obr. 1). Možnost *Exit* pak slouží k zavření aplikace.

Nahrané polygony budov je možné generalizovat s využitím funkcí v záložce *Simplify*. Konkrétně je možné využít algoritmů *Minimum Area Enclosing Rectangle*, *Wall Average*, *Longest Edge* a *Weighted Bisector*. V pravé části menu je pomocí rozbalovacího seznamu možné vybrat, zda má být pro tvorbu konvexní obálky u *Minimum Area Enclosing Rectangle* použit algoritmus *Jarvis Scan* nebo *Graham Scan*. Možností *Clear* jsou pak vymazány nahrané polygony a jejich generalizace.

Popsané funkce je možné vyvolat i pomocí ikon umístěných pod záložkami menu, v pořadí zleva se jedná o funkce *Insert file*, *Minimum Area Enclosing Rectangle*, *Wall Average*, *Longest Edge*, *Weighted Bisector* a *Clear*. Ikony odpovídají jednotlivým funkcím v záložkách menu a po umístění kurzoru myši na ikonu je vypsán i název odpovídající funkce.



Obrázek 1: Ukázka aplikace

4 Dokumentace

Program byl vytvořen v prostředí PyCharm v programovacím jazyce Python s využitím *Qt Creator* a modulu *pyshp*, který umožňuje práci se shapefily. Program se skládá ze souborů *mainform.py*, *draw.py* a *algorithms.py*, které názvem odpovídají příslušným třídám. Ve složce *icons* se pak nachází sedm obrázkových souborů využitých pro ikony jednotlivých funkcí.

4.1 Třída MainForm:

Tato třída slouží pro konfiguraci uživatelského rozhraní aplikace a jeho propojení na metody definované v ostatních třídách. Třída obsahuje šest metod.

Metody *setupUi* a *retranslateUi* byly vytvořeny automaticky na základě vytvořeného prostředí v *Qt Creator*. V první z nich je mimo jiné provedeno napojení jednotlivých položek menu a tlačítek na připravené metody.

Metoda *openFile* ukládá do proměnných *width* a *height* aktuální šířku a výšku okna pro vykreslování polygonů. Následně volá metody *loadData* a *rescaleData* třídy *Draw*, čímž dochází k načtení vstupních dat a jejich vykreslení.

Metoda *createCH* má jako vstupní argument polygon a na základě uživatelem zvoleného algoritmu vytváří jeho konvexní obálku. Pomocí aktuálního indexu rozbalovacího seznamu (comboBox) je kontrolováno, který algoritmus je vybrán a na jeho základě je volána metoda *jarvisScan* nebo *grahamScan* třídy *Algorithms*. Metoda následně vrátí vytvořenou konvexní obálku *ch*.

Metody *simplifyMinEnclosingRectangle*, *simplifyWallAverage*, *simplifyLongestEdge* a *simplifyWeightedBisector* zajišťují generalizaci nahraných polygonů. Všechny nejprve pomocí metody *getPolygons* z třídy *Draw* získají seznam vykreslených polygonů. Následně prochází všechny polygony a pomocí odpovídající metody z třídy *Algorithms*, které předává aktuální polygon, získává obdélník generalizující danou budovu. Tu následně metodou *setER* předává třídě *Draw*. Po generalizaci všech polygonů je vyvolána metoda na překreslení okna.

Metoda *clearCanvas* slouží k vymazání nahraných budov a jejich generalizovaných polygonů a volá metodu *cleanCanvas* třídy *Draw*.

4.2 Třída Draw:

Tato třída zajišťuje grafické rozhraní aplikace a obsahuje sedm metod. Inicializační metoda má dva poziční argumenty a dochází v ní k inicializaci pěti proměnných. Proměnná *features* sloužící pro ukládání objektů z načteného shapefilu je inicializována jako *None*. Dále je inicializován seznam *min_max* pro ukládání minimálních a maximálních souřadnic polygonů a proměnná *no_data*. Proměnná *polygons* je seznam sloužící pro ukládání vykreslovaných polygonů a seznam *er* pro ukládání generalizovaných polygonů.

Metoda *loadData* slouží k načtení vstupních dat. Nejprve dochází k vyvolání dialogového okna, ze kterého je získána cesta k souboru se vstupními daty. Pokud je okno zavřeno bez vybrání souboru, dochází k aktualizaci proměnné *no_data* na hodnotu *True* a ukončení metody. V opačném případě jsou funkcemi modulu *pyshp* získány jednotlivé objekty ze vstupního souboru. Na závěr dochází k projdutí všech polygonů a jejich bodů a získání minimální a maximální X a Y souřadnice.

Metoda *rescaleData* slouží k úpravě souřadnic bodů jednotlivých polygonů, aby je bylo možné vykreslit v okně aplikace. Na vstupu má argumenty *width* a *height*, které odpovídají aktuální šířce a výšce okna pro vykreslení polygonů. Pokud je proměnná *no_data* nastavena na hodnotu *True*, dojde k vytvoření jednoho polygonu, který se nachází mimo vykreslovací okno, čímž je zabráněno pádu aplikace při nezvolení cesty k souboru se vstupními daty.

V opačném případě je inicializován seznam polygonů podle počtu objektů proměnné *features*. Tyto objekty jsou následně procházeny a všem jejich bodům jsou na základě minimálních a maximálních souřadnic a velikosti vykreslovacího okna přepočítány souřadnice. Po úpravě souřadnic dochází k vytvoření bodu typu *QPointF* a jeho přidání do příslušného polygonu.

Metoda *paintEvent* má na vstupu argument *QPaintEvent*. Tato metoda slouží k vykreslení budov a jejich generalizovaných polygonů. Pomocí *for* cyklu jsou procházeny všechny budovy a je jim nastavena jednotná barva a následně jsou vykresleny. Stejným způsobem jsou vykresleny i jejich generalizované polygony uložené v seznamu *er*. Jeho obsah je na závěr vymazán, aby mohlo dojít opětovné generalizaci budov.

Metoda *setER* má na vstupu polygon typu *QPolygonF*, který přidává do seznamu generalizovaných budov.

Metoda *getPolygons* předává vytvořený seznam obsahující všechny nahrané budovy.

Metoda *cleanCanvas* slouží k vyčištění vykreslovacího okna, čehož je docíleno vymazáním obsahu seznamů obsahující budovy a jejich generalizované polygony a vyvoláním metody pro překreslení okna.

4.3 Třída Algorithms

Tato třída v sobě implementuje algoritmy pro generalizaci budov. Metoda *get2LinesAngle* má na vstupu čtyři body typu *QPointF* tvořící dvě linie, mezi kterými počítá úhel. Nejprve dochází k výpočtu vektorů, jejich skalárního součinu a norem obou vektorů. Následně je spočítán argument pro funkci *cosinus*, který je zaokrouhlen na hodnotu 1 nebo -1, pokud je větší, respektive menší. Metoda pak vrací vypočítaný úhel.

Metoda *getPointLinePosition* slouží k určení polohy bodu vůči přímce a na vstupu má analyzovaný bod a dva body definující přímku, všechny typu *QPointF*. Nejprve dochází k výpočtu vektorů a následně determinantu matice tvořené těmito vektory. Pokud je determinant kladný, bod leží v levé polorovině od přímky a metoda vrátí hodnotu 1. V případě záporného determinantu bod leží v pravé polorovině a dochází k návratu hodnoty 0. Pokud bod leží na přímce, vyjde nulový determinant a metoda pak vrací hodnotu -1.

Metoda *getLength* má na vstupu dva body typu *QPointF* a vrací vypočítanou vzdálenost mezi těmito body.

Metoda *jarvisScan* má na vstupu polygon typu *QPolygonF* a implementuje v sobě tvorbu konvexní obálky pomocí algoritmu *Jarvis Scan*, který je detailněji popsaný v kapitole *Popis a rozbor problému*. Metoda vrací vytvořenou konvexní obálku typu *QPolygonF*.

Metoda *grahamScan* má obdobně jako předchozí metoda na vstupu polygon typu *QPolygonF* a vrací jeho vytvořenou konvexní obálku taktéž typu *QPolygonF*. K tvorbě konvexní obálky však využívá algoritmus *Graham Scan*, který je opět podrobněji popsán v kapitole *Popis a rozbor problému*.

Metoda *rotate* má na vstupu polygon typu *QPointF* a úhel ve formátu *float* a slouží k otočení vybraného polygonu o daný úhel. Metoda vrací zrotovaný polygon ve formátu *QPolygonF*.

Metoda *minMaxBox* má na vstupu polygon typu *QPointF* a slouží k nalezení minimálních a maximálních hodnot jeho souřadnic a k vytvoření obdélníku, jehož vrcholy jsou definovány těmito hodnotami. Metoda pak vrací vytvořený obdélník a jeho rozlohu.

Metoda *computeArea* má na vstupu polygon typu *QPointF* a vrací vypočítanou rozlohu tohoto polygonu.

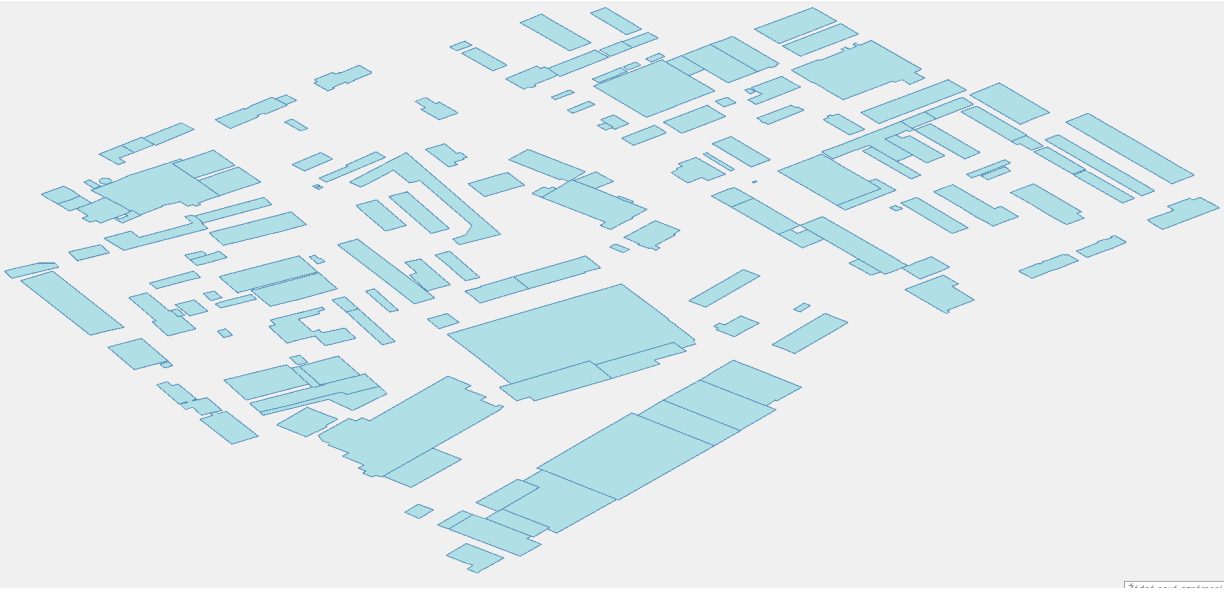
Metoda *resizeRectangle* má na vstupu Enclosing Rectangle *er* a polygon *pol* představující původní budovu, oba typu *QPolygonF*. Metoda počítá poměr ploch původní a generalizované budovy a na jeho základě *er* zmenšuje nebo zvětšuje tak, aby jeho plocha odpovídala ploše původní budovy. Metoda vrací zmenšený nebo zvětšený polygon typu *QPolygonF*.

Metody *minAreaEnclosingRectangle*, *wallAverage*, *longestEdge* a *weightedBisector* mají všechny na vstupu

polygon typu *QPolygonF* a vrací jeho generalizovanou podobu opět typu *QPolygonF*. Metody v sobě implementují odpovídající algoritmy pro generalizaci budov, konkrétně Minimum Area Enclosing Rectangle, Wall Average, Longest Edge a Weighted Bisector, které jsou podrobně popsány v kapitole *Popis a rozbor problému*.

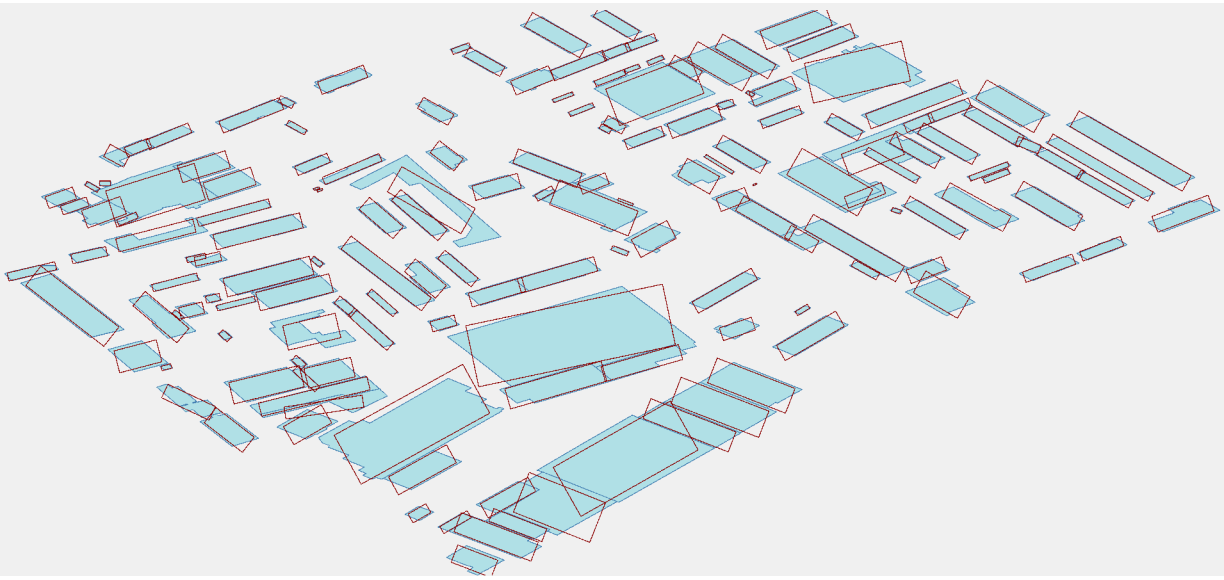
5 Výsledky

Generalizace budov pomocí jednotlivých algoritmů byla testována na třech datasetech, z nichž každý obsahuje přibližně sto budov. Dataset použitý v ukázkách je představován sídlištěm v Kroměříži (obr. 2).

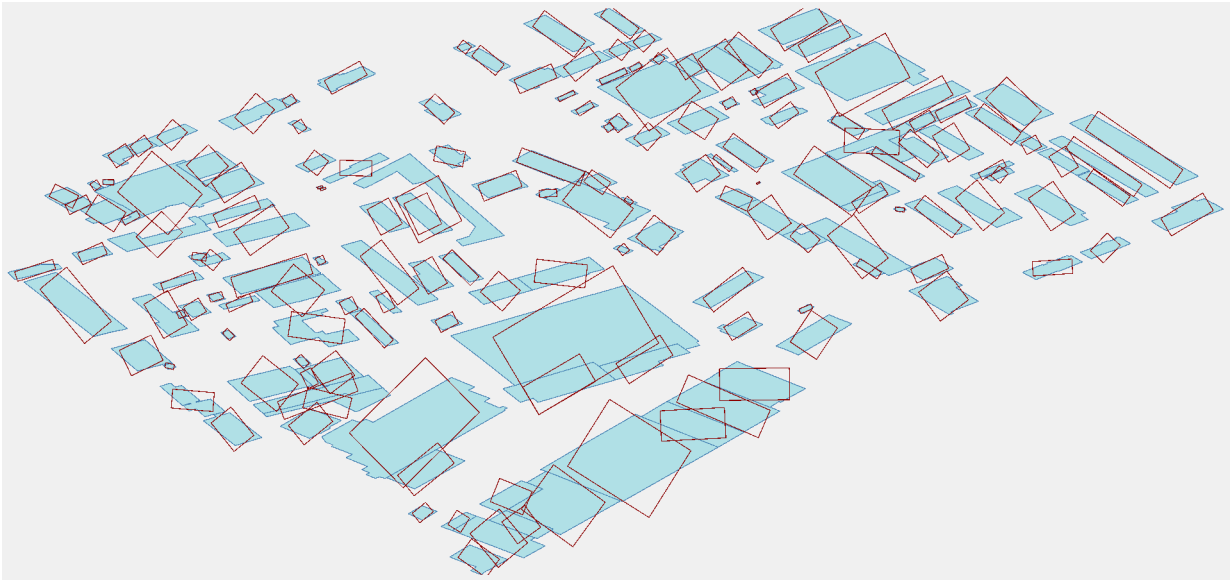


Obrázek 2: Ukázka vstupních dat - sídliště Kroměříž

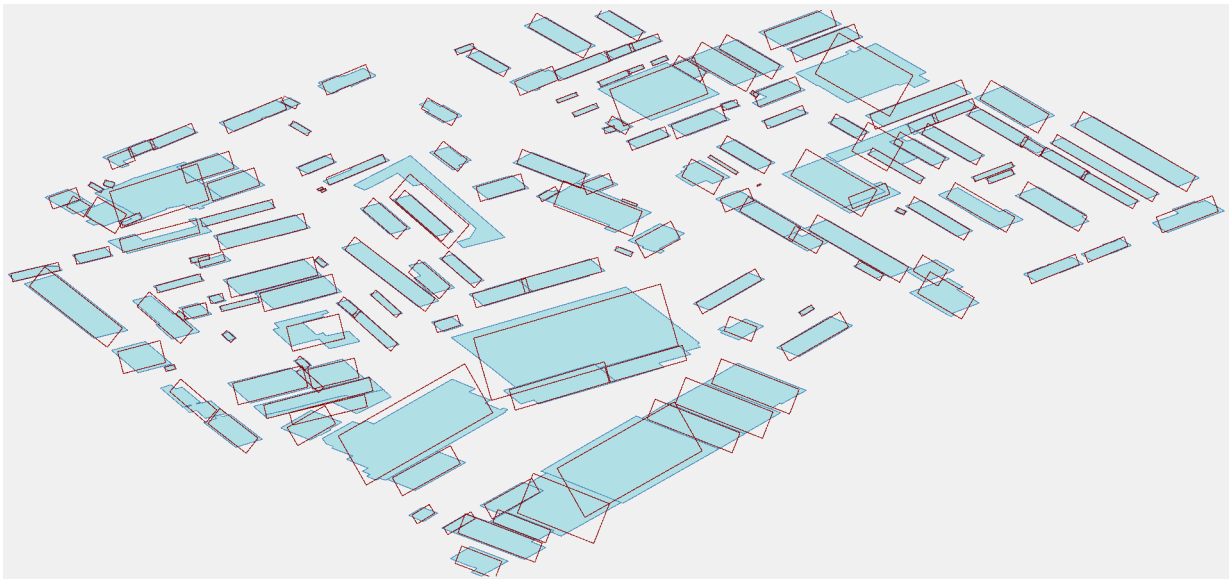
Na obrázcích 3–6 je možné porovnat výsledky generalizace celého datasetu všemi implementovanými metodami. Na základě vizuálního porovnání dosáhl nejlepších výsledků algoritmus *Minimum Area Enclosing Rectangle*. Generalizované budovy zde téměř ve všech případech kopírují hlavní směry budovy a výsledný obdélník svým natočením odpovídá původní budově. Podobně dobrých výsledků dosáhl i algoritmus *Longest Edge*. U některých budov je však patrný nedostatek tohoto algoritmu, kdy nejdelší hrana nemusí vždy kopírovat hlavní směr budovy. To je patrné hlavně u budov obdélníkového tvaru protáhlých v jednom směru, které jsou ale členěny na více hran a nejdelší celistvá hrana je pak orientována jiným směrem. Algoritmus *Weighted Bisector* dosahuje již výrazně horších výsledků. U části budov nedopovídá natočení generalizovaného polygonu směru původní budovy a u některých obdélníkových budov je jejich generalizace spíše čtvercová. Tyto nedostatky se ještě výrazněji projevují u algoritmu *Wall Average*, který dosáhnul nejhorších výsledků.



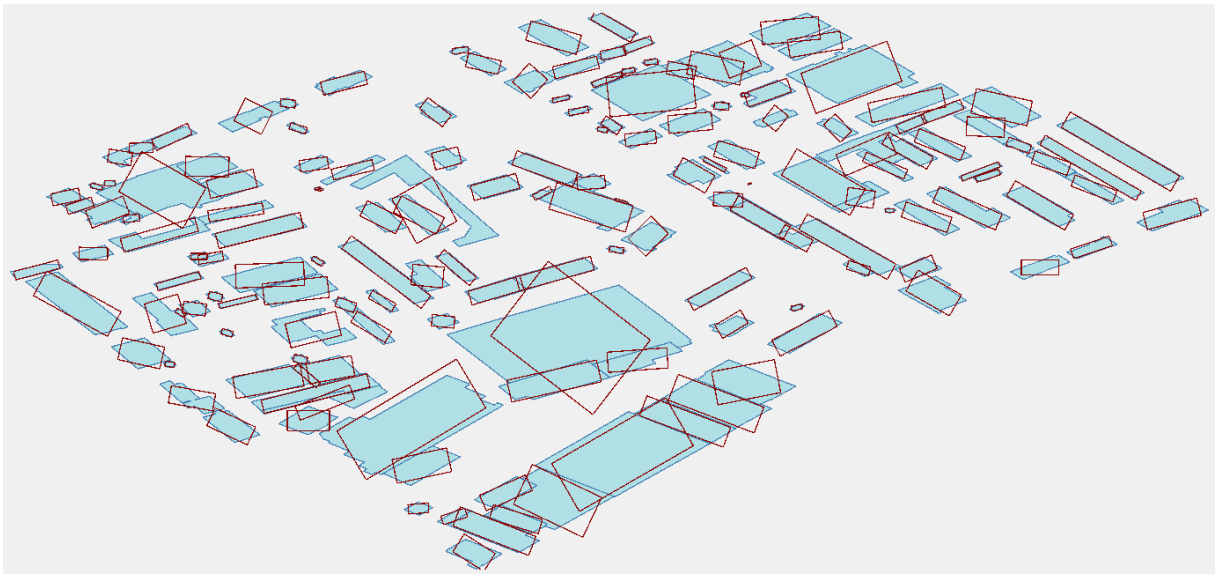
Obrázek 3: Ukázka generalizovaných budov – Minimum Area Enclosing Rectangle



Obrázek 4: Ukázka generalizovaných budov – Wall Average

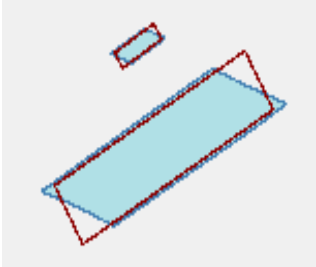


Obrázek 5: Ukázka generalizovaných budov – Longest Edge

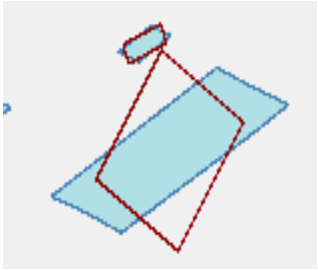


Obrázek 6: Ukázka generalizovaných budov – Weighted Bisector

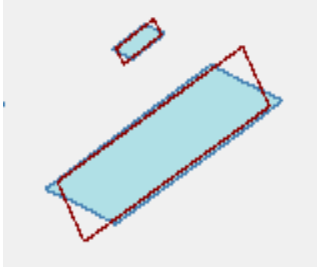
Výsledky jednotlivých algoritmů byly porovnány i na konkrétních tvarech budov. Obrázky 7-10 představují generalizovanou budovu obdélníkového tvaru. V souladu s celkovým hodnocením poskytují nejlepší výsledky algoritmy *Minimum Area Enclosing Rectangle* a *Longest Edge*, které tvar původní budovy dobře kopírují. U algoritmů *Wall Average* a *Weighted Bisector* se pak liší poměr stran polygonu a i natočení je oproti původní budově rozdílné.



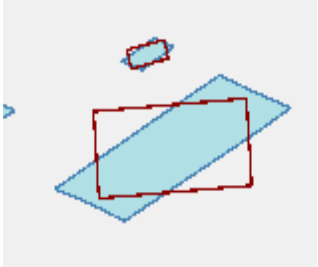
Obrázek 7: Pravidelná budova – Minimum Area Enclosing Rectangle



Obrázek 8: Pravidelná budova – Wall Average

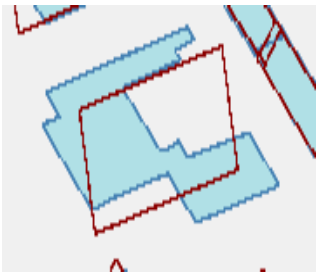


Obrázek 9: Pravidelná budova – Longest Edge

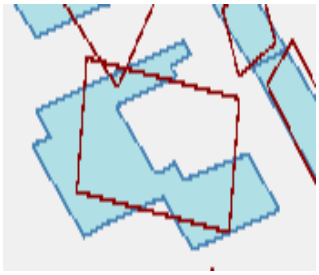


Obrázek 10: Pravidelná budova – Weighted Bisector

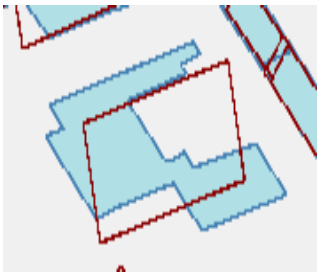
U budovy nepravidelného tvaru (obr. 11–14) dosáhly algoritmy *Minimum Area Enclosing Rectangle*, *Longest Edge* i *Weighted Bisector* podobného výsledku, který dobře aproximuje původní tvar budovy. Výsledek algoritmu *Wall Average* tvarem odpovídá výsledkům předchozích algoritmů, ale je jinak natočený a nekopíruje tak hlavní směry původní budovy.



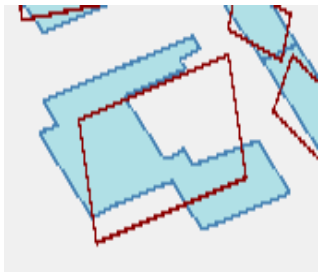
Obrázek 11: Nepravidelný tvar budovy – Minimum Area Enclosing Rectangle



Obrázek 12: Nepravidelný tvar budovy – Wall Average

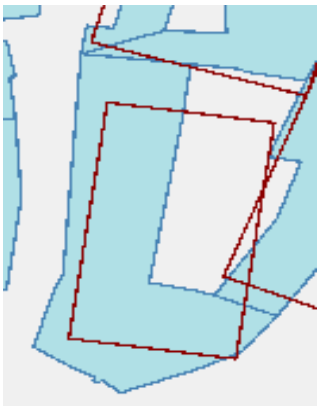


Obrázek 13: Nepravidelný tvar budovy – Longest Edge

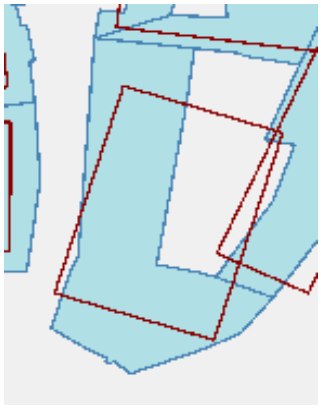


Obrázek 14: Nepravidelný tvar budovy – Weighted Bisector

Posledním porovnávaným typem je budova ve tvaru L (obr. 15–18). Zde opět dosahují nejlepších výsledků algoritmy *Minimum Area Enclosing Rectangle* a *Longest Edge*. Dobrého výsledku dosáhl i algoritmus *Wall Average*, kde opět dochází k mírnému natočení oproti vizuálnímu hlavnímu směru budovy. Nejhoršího výsledku pak dosáhl algoritmus *Weighted Bisector*, kde je natočení vůči hlavnímu směru budovy výrazné a generalizovaný polygon se také blíží spíše čtvercovému tvaru.

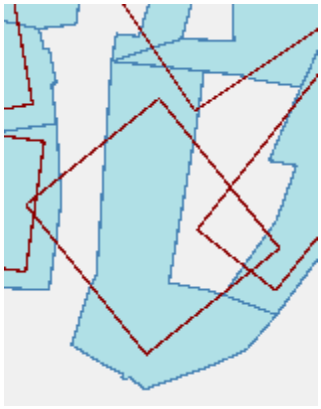


Obrázek 15: Budova ve tvaru L – Minimum Area Enclosing Rectangle



Obrázek 16: Budova ve tvaru L – Wall Average

Obrázek 17: Budova ve tvaru L – Longest Edge



Obrázek 18: Budova ve tvaru L – Weigted Bisector

Jednotlivé algoritmy byly porovnány mezi sebou na základě vizuální interpretace pro dané generalizované budovy. Výsledná úspěšnost pro 3 datasety je uvedena v tabulce 1. Stejně jako v předchozích případech dopadl nejlépe algoritmus *Minimum Area Enclosing Rectangle*, pak *Longest Edge*. Horších výsledků dosáhnul algoritmus *Wall Average* a jako nejhorší skončil *Weighted Bisector*.

dataset \ algoritmus	MAER	Wall Average	Longest Edge	Weighted Bisector
centrum_stare_mesto.shp	90 %	85 %	83 %	60 %
kostely_stare_mesto.shp	91 %	86 %	89 %	75 %
sidliste_kromeriz.shp	84 %	62 %	91 %	76 %
průměr	88,33 %	77,66 %	87,66 %	70,33 %

Tabulka 1: Úspěšnost jednotlivých algoritmů

6 Závěr

V rámci této úlohy byla vytvořena aplikace, která generalizuje budovy na základě uživatelem zvoleného algoritmu (*Minimum Area Enclosing Rectangle*, *Wall Average*, *Longest Edge* a *Weighted Bisector*). Dále je možné vybrat si mezi dvěma algoritmy pro konstrukci konvexní obálky. Jsou jimi *Jarvis Scan* a *Graham Scan*.

Možným zlepšením aplikace by bylo lepší ošetření fungování algoritmu *Weighted Bisector*, kde pro některé typy budov nedochází ke splnění podmínky, aby úhlopříčka nekřížila žádnou hranu polygonu. V případě nesplnění této podmínky pak nedochází k vytvoření žádné úhlopříčky pro danou budovu a generalizace končí pádem programu. Tomu bylo zamezeno tím, že v případě nevytvoření úhlopříček pro daný polygon jsou do výsledných dvou úhlopříček přiřazeny první dvě strany polygonu.

7 Zdroje

přednášky z předmětu *Algoritmy počítačové kartografie*, dostupné z: http://web.natur.cuni.cz/ bayertom/ index.php/teaching/algoritmy-pocitacove-kartografie

ARNON, S., D., GIESELMANN, P., J. (1983): A Linear Time Algorithm for the Minimum Area Rectangle Enclosing a Convex Polygon, Department of Computer Science Technical Reports, Purdue University.

DUCHENE, C. et al. 2003: Quantitative and qualitative description of building orientation.

FELKEL, P. (2020): Convex Hulls. Přednáškové texty z předmětu Výpočetní geometrie, FEL ČVUT.

LI, Z. a kol. (2004): Automated building generalization based on urban morphology and Gestalt theory. International Journal of Geographical Information Science, 18, 5, 513–534.

PUNT, E., WATKINS, D. (2010): User-directed generalization of roads and buildings for multi-scale cartography. 13th Workshop of the ICA commission on Generalisation and Multiple Representation.

REGNAULD, N. (1998): Généralisation du bâti: Structure spatiale de type graphe et représentation cartographique.Thèse de doctorat, Laboratoire d'Informatique de Marseille.