|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE  FAKULTA STAVEBNÍ, OBOR GEODÉZIE A KARTOGRAFIE  KATEDRA GEOMATIKY | | | | | |
| název předmětu  Algoritmy digitální kartografie a GIS | | | | | |
| název úlohy  Generalizace budov | | | | | |
| školní rok  2021/22 | studijní skup.  60 | číslo zadání  - | zpracoval, email  Jakub Šimek, Jan Kučera | datum  7.11.21 | klasifikace |

Obsah

[1. Zadání 3](#_Toc93931436)

[2. Bonusové úlohy 3](#_Toc93931437)

[3. Popis a rozbor problému 4](#_Toc93931438)

[4. Popis algoritmů 4](#_Toc93931439)

[4.1 Hledání konvexní obálky 4](#_Toc93931440)

[4.2 Generalizace budov 5](#_Toc93931441)

[5. Problematické situace a jejich rozbor 8](#_Toc93931442)

[6. Vstupní data 9](#_Toc93931443)

[Import textového souboru 9](#_Toc93931444)

[7. Výstupní data 11](#_Toc93931445)

[8. Vzhled aplikace 12](#_Toc93931446)

[9. Dokumentace 13](#_Toc93931447)

[1. Třída *Algorithms* 13](#_Toc93931448)

[2. Třída *CSV* 14](#_Toc93931449)

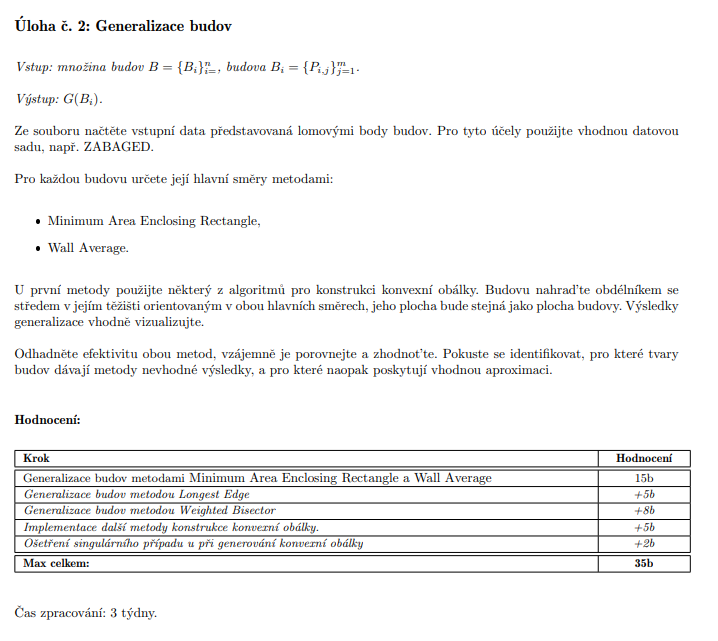
[3. Třída *Draw* 14](#_Toc93931450)

[4. Třída *Widget* 14](#_Toc93931451)

[10. Závěr 14](#_Toc93931452)

[11. Zdroje 15](#_Toc93931453)

## Zadání



Obrázek 1 - zadání

## Bonusové úlohy

Ke standartnímu zadání byly zpracovány tyto bonusové úlohy:

1. Generalizace budov metodou Longest Edge
2. Generalizace budov metodou Weighted Bisector
3. Implementace další metody konstrukce konvexní obálky – *Graham Scan*

## Popis a rozbor problému

Na vstupu máme množinu budov , kde budova .

Konvexní obálka konečné množiny představuje konvexní mnohoúhelník s nejmenší plochou.

Množina je konvexní právě tehdy, když celá spojnice dvou libovolných bodů náležících dané množině leží uvnitř množiny .

Pro každou budovu hledáme její generalizaci .

Generalizací se rozumí zjednodušení tvaru budovy za účelem redukce objemu dat.

## Popis algoritmů

### Hledání konvexní obálky

#### Algoritmus Jarvis Scan

První metodou pro generování konvexní obálky je metoda Jarvis Scan.

V prvním kroku je potřeba nejprve nalézt pivota :

Bod je přidán do konvexní obálky .

Dále je vybrán bod tak, aby přímka dána body a byla rovnoběžná s osou . Pak je vždy v cyklu přidáván do konvexní obálky bod s maximálním úhlem .

#### Implementace algoritmu Jarvis Scan

1. Nalezení pivota ,
2. Inicializuj:
3. Opakuj, dokud :
4. ,

#### Algoritmus Graham Scan

Nejdříve najdeme pivota . Bod je prvním bodem konvexní obálky. Pro všechny vstupní body spočteme úhel měřený od rovnoběžky s osou s vrcholem v bodě . Množinu vstupních bodů pak setřídíme podle velikosti . Tím vznikne star-shaped polygon.

Do zásobníku přidáme bod a první bod ze star-shaped polygonu. Dále je testován další bod v pořadí. Za předpokladu, že je splněna podmínka levotočivosti, je bod přidán do zásobníku. Pak je testován 4. bod. Jestliže je splněna podmínka levotočivosti, je také přidán do zásobníku. Jestliže není podmínka splněna, je poslední bod smazán ze zásobníku a testuje čtvrtý bod z bodu druhého. Po vykonání cyklu je v zásobníku konvexní obálka.

#### Implementace algoritmu Graham Scan

1. Nalezení pivota
2. Setřídění dle . Index odpovídá setříděnému pořadí.
3. Nalezení bodů se stejným a vymazání bodu bližšího ke .
4. Inicializuj
5. , (indexy posledních dvou prvků )
6. Opakuj pro
7. Pokud vlevo od

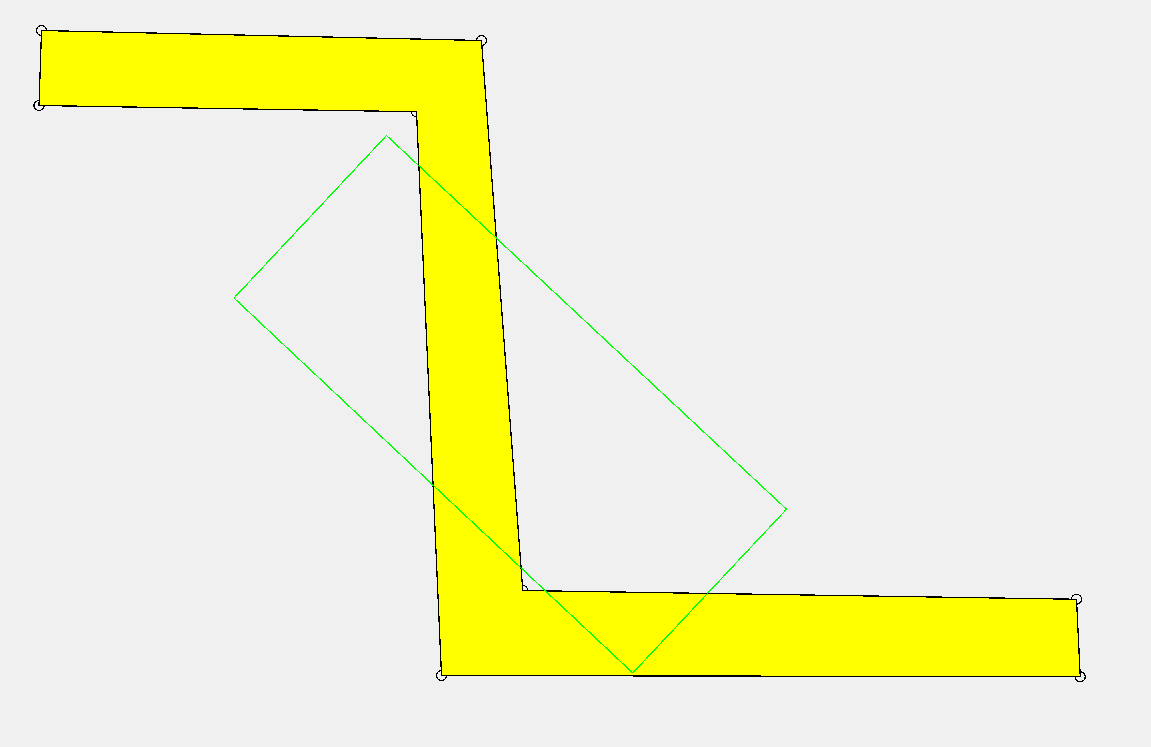
10. Jinak

### Generalizace budov

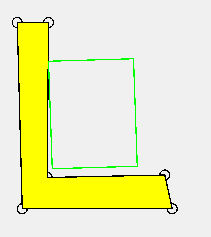
#### Metoda Minimum Area Enclosing Rectangle

Tato metoda slouží k zjištění hlavního směru budovy. Tento směr je určen směrem delší strany ohraničujícího obdélníku, který má minimální plochu.

U tohoto algoritmu nastávají problémy u budov tvaru L a Z, jak je vidět na obrázku.



Obrázek 2 – ukázka chybné generalizace



Obrázek 3– ukázka chybné generalizace

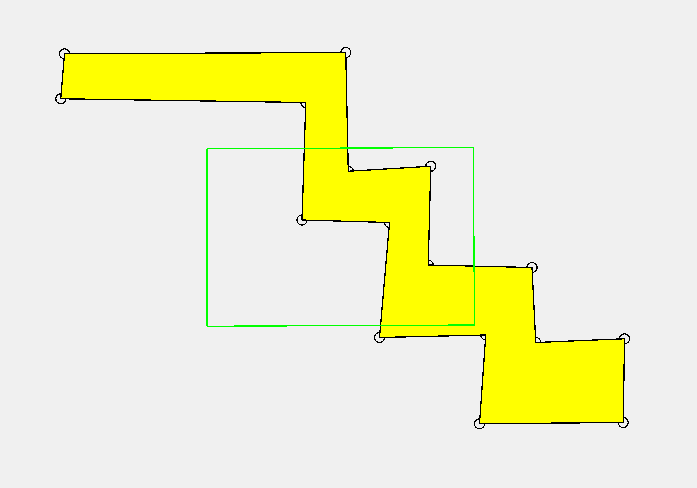
#### Implementace algoritmu Minimum Area Enclosing Rectangle

1. Nalezení
2. Inicializuj
3. Opakuj pro každou hranu obálky :
4. Spočti směrnici hrany
5. Otoč o :
6. Najdi a urči
7. Pokud

#### Metoda Longest Edge

Algoritmus pro detekci hlavního směru budovy. Hlavním směrem budovy je jednoduše nejdelší hrana polygonu.

Tato metoda má nevýhodu, že nejdelší hrana nemusí ve skutečnosti reprezentovat hlavní směr budovy (viz obrázek). Proto není vhodná pro atypické tvary budov.



Obrázek 4– ukázka chybné generalizace

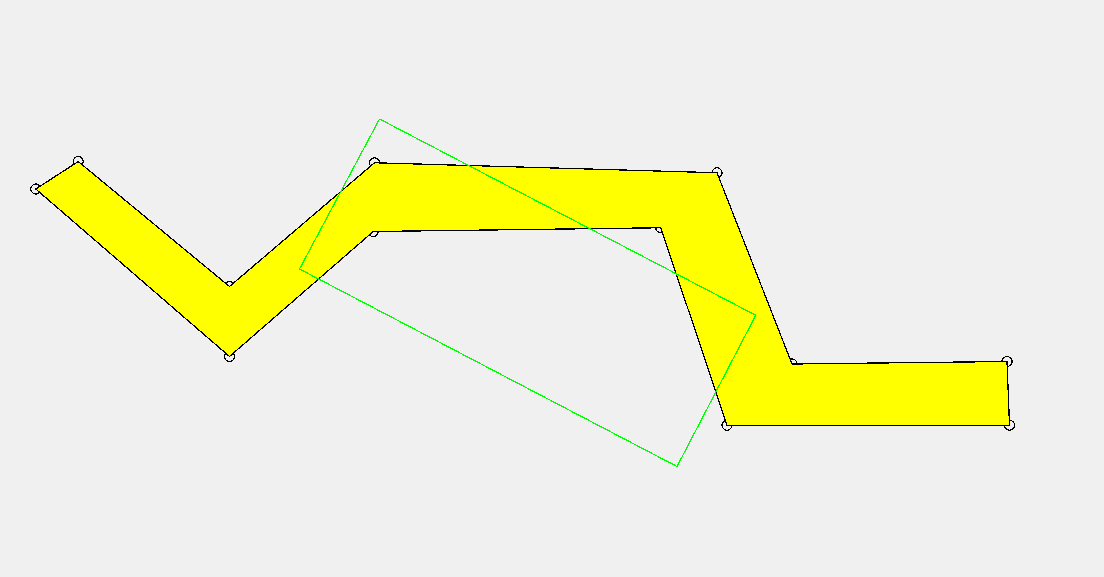
#### Implementace metody Longest Edge

1. Inicializace vektoru dvojic délka hrany/směrnice přímky
2. Opakuj pro všechny body budovy:

5. Seřazení vektoru dvojic podle velikosti
6. Uložení – poslední prvek ve vektoru dvojic
7. Vytvoření *enclosing rectangle*

#### Wall Average

Pro každou stranu budovy je spočtena směrnice. Na ní je pak aplikována metoda, která slouží k nalezení zbytku po dělení. Výsledný směr natočení budovy je pak dán váženým průměrem s váhou rovnou délce příslušné strany.



Obrázek 5 – ukázka chybné generalizace

#### Implementace metody Wall Average

1. Inicializace

– směr natočení budovy

– obvod budovy

- směrnice první hrany budovy

1. Cyklus přes všechny body budovy:

4. - rozdíl směrnic

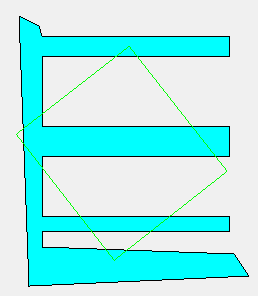


9. – vážený průměr
10. Vytvoření *enclosing rectangle*

#### Metoda Weighted Bisector

Tento algoritmus slouží pro detekci hlavního směru budovy. Hlavním směrem budovy je vážený průměr směrníků dvou nejdelších úhlopříček v polygonu. Váhami jsou délky úhlopříček.

Obecně dává dobré výsledky. Přesto může někdy špatně odhadnout celkovou orientaci budovy (viz obrázek).



Obrázek 6– ukázka chybné generalizace

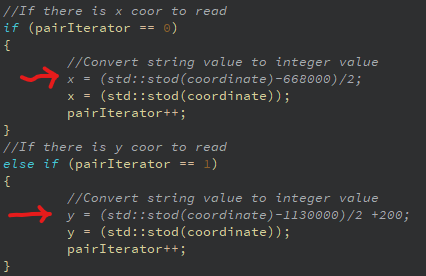
#### Implementace budovy Weighted Bisector

Body 1 až 5 jsou stejné jako u metody *Longest Edge*.

1. Uložení posledních dvou dvojic délka/směrnice ze srovnaného vektoru podle velikosti hrany do proměnných
2. Vážený průměr směru podle vzorce
3. Vytvoření *enclosing area*

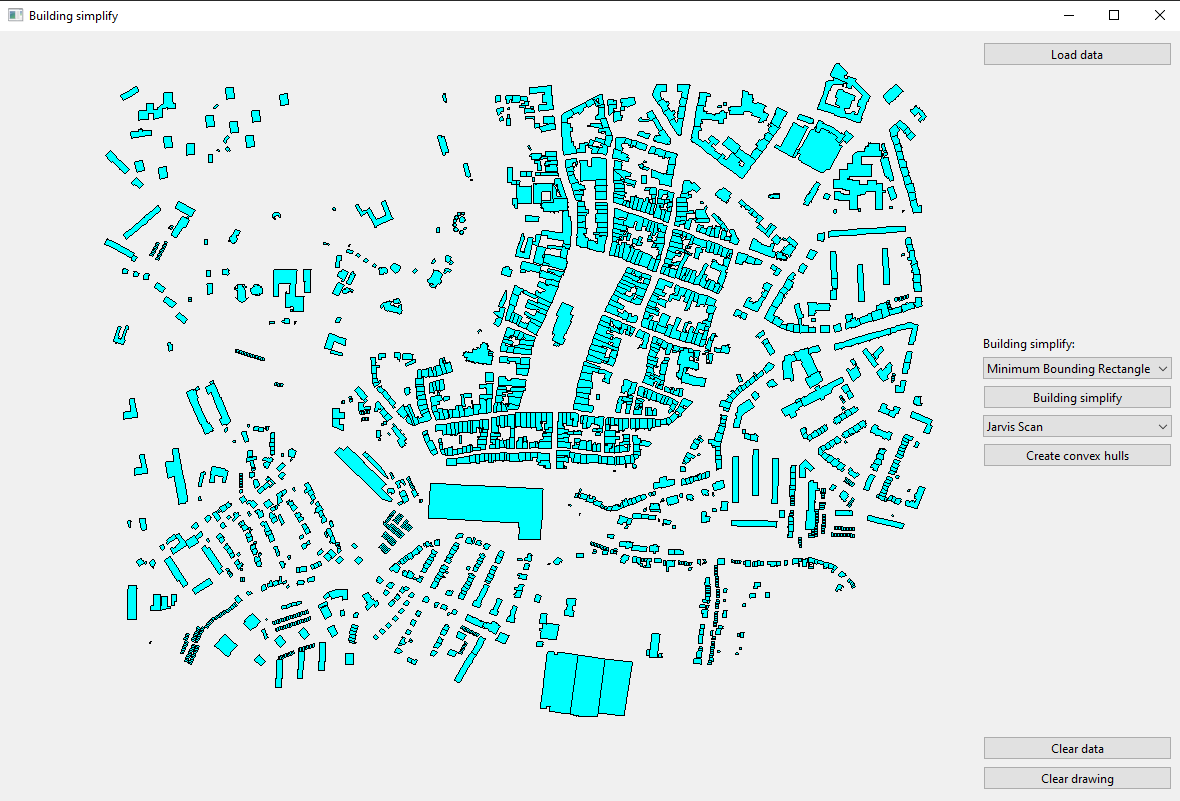
## Problematické situace a jejich rozbor

Jedním z velkých problémů je souřadnicový systém načítaných dat. Widget *Canvas* implementovaný v aplikaci má souřadnice pouze v 1. kvadrantu a jedná se o pixelové souřadnice. Klasický monitor je schopný zobrazit objekty v přibližném rozmezí souřadnic *x* a *y* cca [0, 900]. Widget je samozřejmě natahovací, ale objekty např. v souřadnicích S-JTSK tu rozhodně bez transformace nebudou vidět. Nemluvě o souřadnicích, které nebudou v pravoúhlém XY systému. Pro objekty z Jihlavy byla v kódu zavedena primitivní transformace jen pro účely funkčností načítací nadstavby reálných objektů.



Obrázek 7

Ale je jasné, že takový způsob je nesmyslný aplikovat. Proto je zde uveden jen na ukázku. Takový výstup by byl na následujícím obrázku (část Jihlavy).



Obrázek 8

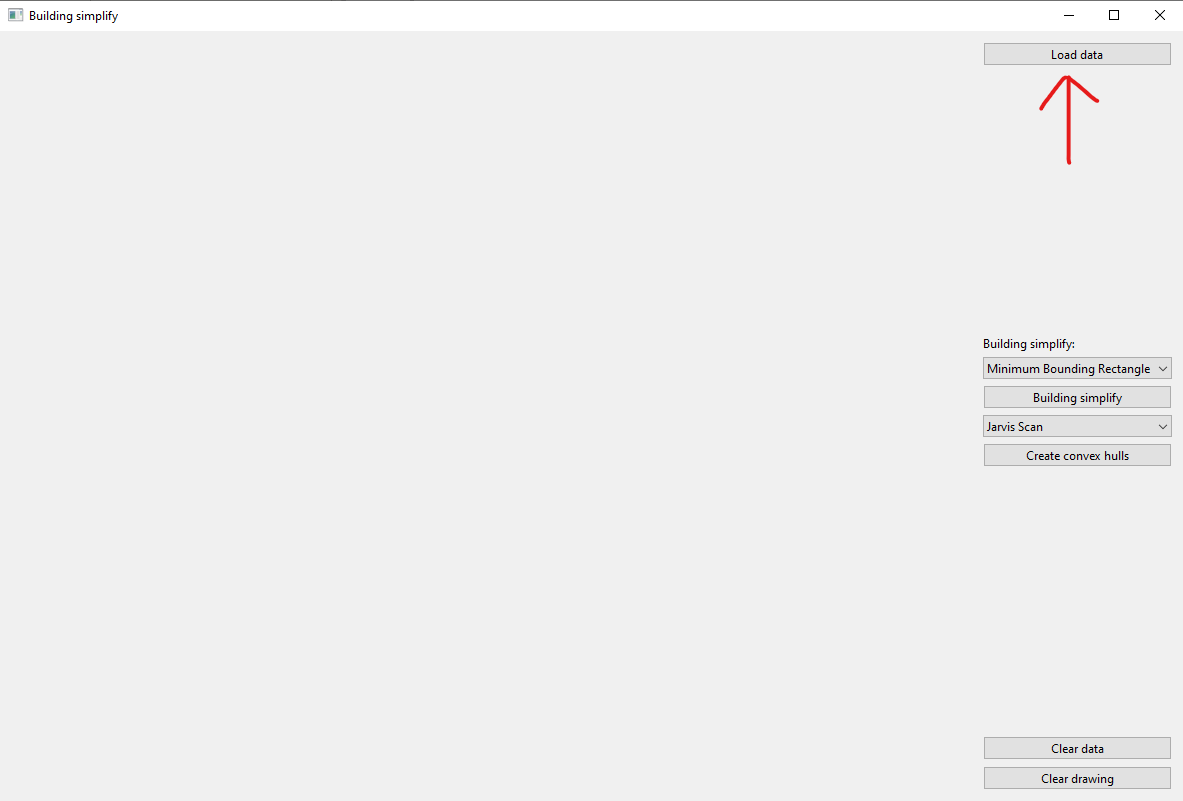
Dalšími problémy jsou čistota souřadnic exportovaných dat. Je potřeba, aby načtená data splňovala kritéria Simple features. I když jsou splněna, tak po transformaci do *Canvas* mohou opět vzniknout problémy narušující tato kritéria. Tyto problémy jsou pak promítnuté do nefunkčnosti algoritmů (hlavně Graham scan).

## Vstupní data

Data do aplikace můžeme importovat dvěma způsoby. První způsob je zadání polygonu uživatelem klikáním do grafického okna. Druhým způsobem je pak import polygonů z textového souboru, ve kterém jsou uloženy polygony ve špagetové reprezentaci.

### Import textového souboru

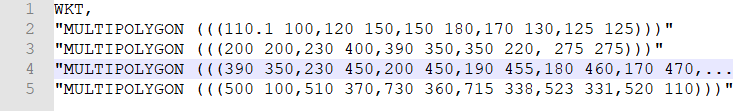
Pro načítaní dat bylo do widgetu vloženo tlačítko *Load data*.



Obrázek 9

Po rozkliknutí tlačítka se objeví Windows dialog pro výběr souborů.

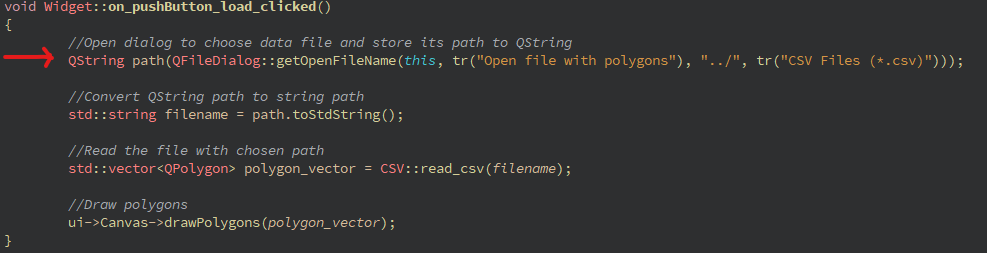
Formát vstupních dat nebyl předem zadán. Implementace načítání souborem je provedena s uvážením špagetového modelu exportovaného ze softwaru QGIS formou WKT. Jedná se o CSV soubor. Program předpokládá, že se na každém řádku nachází WKT *Multipolygon* obsahující sled jednotlivých jeho bodů. Příklad je na následujícím obrázku.



Obrázek 10

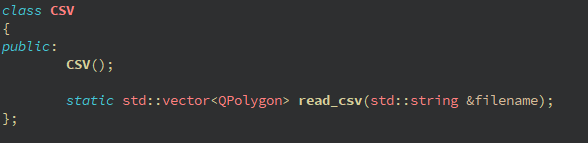
Řádky se mohou lišit v souřadnicích a v jejich množství. Nicméně struktura musí zůstat stejná. Tedy na prvním řádku je z QGIS vypsaná hlavička a pak následují řádky se souřadnicemi ve WKT. Souřadnicový řádek musí obsahovat klíčové slovo *Multipolygon*, následovat musí 3 kulaté závorky, poté sled souřadnic *(x1 y1, x2 y2, x3 y3, …)* a ty jsou opět zakončeny trojicí kulatých závorek. Celý řádek je v uvozovkách.

Po kliknutí na tlačítko *Load* data se do do datové struktury QString uloží řetězec obsahující cestu k souboru vstupních dat získanou z dialogu Windows průzkumníka.



Obrázek 11

Čtení dat bylo realizováno statickou metodou v třídě CSV, jejíž vstupem je do klasického stringu převedená cesta k souboru s uvedeného QStringu.



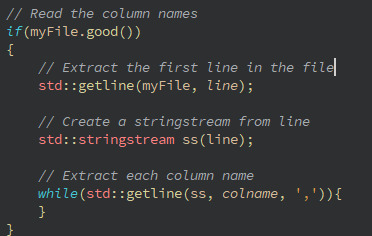
Obrázek 12

Metoda nejdříve zkontroluje, zdali je soubor správně načtený.



Obrázek 13

Poté přečte hlavičku souboru (pouze 1. řádek).



Obrázek 14

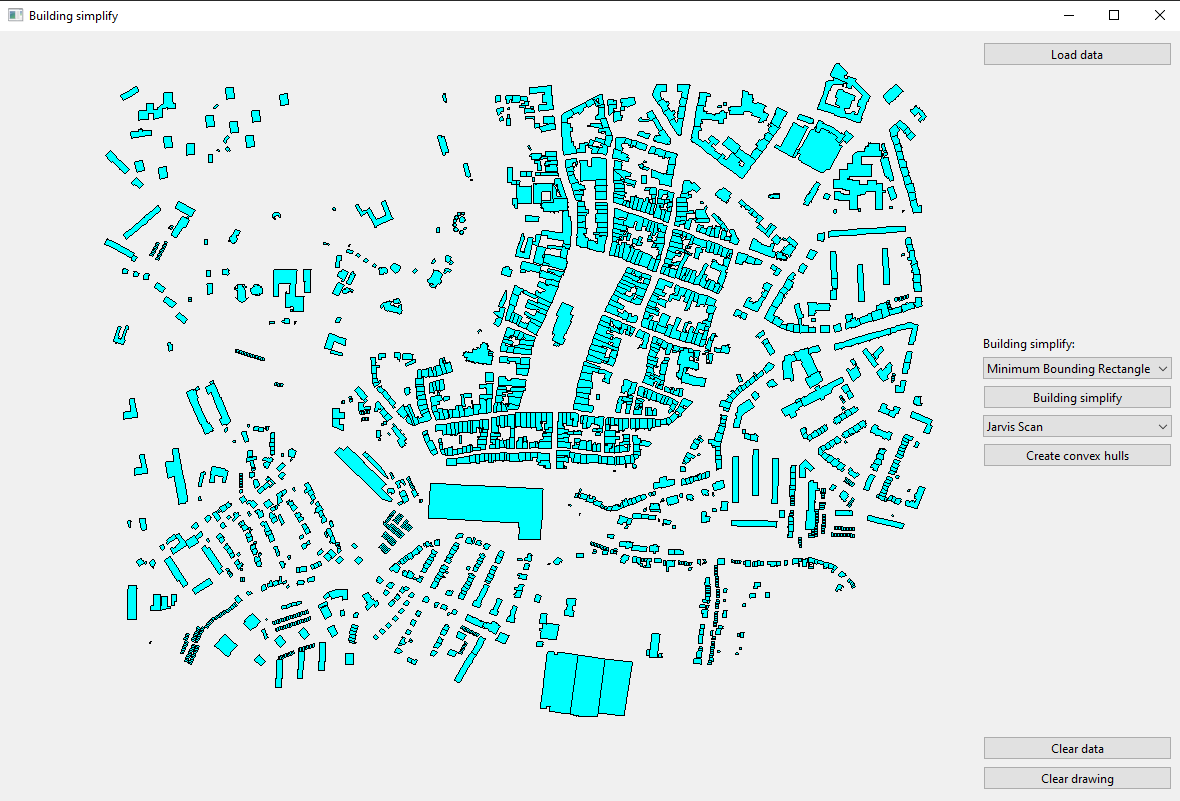
Poté metoda bude projíždět každý řádek zvlášť a bude ukládat do struktury QPolygon všechny body z aktuálního řádku. Z každého řádku odstraní metodami *erase* a *getline* veškeré nečíselné znaky a pomocí *pair iterátoru* postupně z řádku uloží souřadnice každého bodu. Zde je hlavní část načítacího kódu.



Obrázek 15

Nakonec se postupně všechny polygony vykreslí na *Canvas* metodou *drawPolygons* ze třídy *Draw*.

Ukázka načteného souboru dat.



Obrázek 16

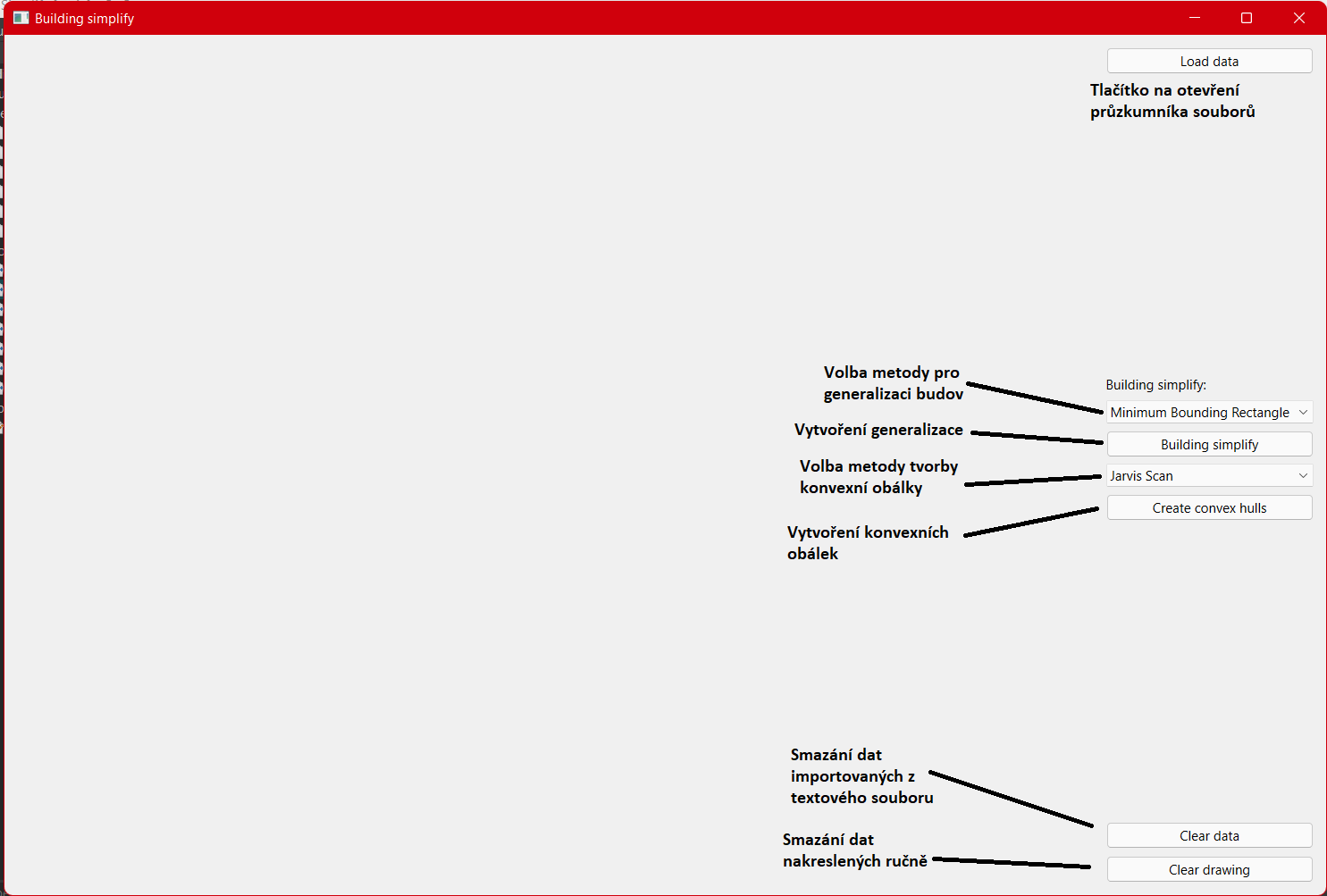
## Výstupní data

Veškeré výstupy jsou vykreslovány do okna *Canvas*.

## Vzhled aplikace

Po spuštění aplikace se otevře okno *Building simplify.* Úvodní plocha obsahuje 5 tlačítek třídy *QPushButton*, a to: *pushButton\_load*, *pushButton*, *pushButton\_createHulls*, *pushButton\_clear\_data* a *pushButton\_2*. Tlačítka jsou uvedena postupně odshora dolů, podle obrázku.

Dále jsou na úvodní obrazovce dvě rolovací menu třídy *QComboBox*, a to: *comboBox* a *comboBox\_HullsMethods*.



Obrázek 17

## Dokumentace

### Třída *Algorithms*

1. *double get2LinesAngle(QPoint &p1, QPoint &p2, QPoint &p3, QPoint &p4)*

* spočte úhel mezi dvěma vektory

1. *QPolygon cHull (std::vector <QPoint> &points)*

* Vytvoří konvexní obálku pomocí metody *Jarvis Scan*

1. *std::vector <QPoint> rotate(std::vector <QPoint> &points, double sigma)*

* rotace vektoru bodů o úhel *sigma* kolem osy *z*

1. *std::tuple<std::vector<QPoint>, double> minMaxBox(std::vector <QPoint> &points)*

* nalezne nejmenší ohraničující obdélník množiny bodů se stranami rovnoběžnými s osami *x* a *y*

1. *QPolygon minAreaEnclosingRectangle(std::vector <QPoint> &points)*

* Vytvoří ohraničující obdélník množiny bodů s nejmenší plochou metodou *Minimum Area Enclosing Rectangle*

1. *QPolygon wallAverage(std::vector <QPoint> &points)*

* Vytvoří ohraničující obdélník množiny bodů s nejmenší plochou metodou *Wall Average*

1. *double LH(std::vector <QPoint> &points)*

* určí plochu polygonu pomocí L'Huillierových vzorců

1. *std::vector <QPoint> resizeRectangle(std::vector <QPoint> &points, std::vector <QPoint> &er)*

* změní velikost vstupního obdélníku

1. *QPolygon weightedBisector(std::vector <QPoint> &points)*

* Vytvoří ohraničující obdélník množiny bodů s nejmenší plochou metodou *Weighted Bisector*

1. *QPolygon longestEdge(std::vector <QPoint> &points)*

* Vytvoří ohraničující obdélník množiny bodů s nejmenší plochou metodou *Longest Edge*

1. *int getPointLinePosition(QPoint &a,QPoint &p1,QPoint &p2)*

* analyzuje vzájemný vztah bodu a přímky

1. *QPolygon Algorithms::cHullGraham(std::vector <QPoint> &points)*

* vrací konvexní obálku určenou metodou *Graham Scan*

1. *double distance(QPoint p1, QPoint p2)*

* *vrací vzdálenost dvou bodů*

### Třída *CSV*

1. *vector<QPolygon> read\_csv(string &filename)*

* čte vstupní *csv* soubor a ukládá polygon do struktury *vector<QPolygon>*

### Třída *Draw*

1. *void paintEvent(QPaintEvent \*event)*

* vykreslí polygony na *Canvas*

1. *void mousePressEvent(QMouseEvent \*event)*

* vrátí souřadnice kurzoru po kliknutí na *Canvas*

1. *void clearDrawing()*

* vyčistí *Canvas*

1. *void drawPolygons(vector<QPolygon> &pols)*

* vykreslí polygony na *Canvas*

1. *void clearData()*

* vyčistí *Canvas*

### Třída *Widget*

1. *void on\_pushButton\_2\_clicked()*

* vyčistí *Canvas*

1. *void on\_pushButton\_clear\_data\_clicked()*

* vyčistí *Canvas*

1. *void on\_processPoints()*

* vytvoří ohraničující obdélník

1. *void on\_pushButton\_load\_clicked()*

* načte data ze souboru

1. *void createHulls(std::vector <QPoint> &points)*

* vytvoří konvexní obálku

1. *void on\_pushButton\_createHulls\_clicked()*

* vytvoří konvexní obálky

## Závěr

Ošetřování záležitostí Simple features a souřadnicových systémů by zabralo moc času a není to hlavní úlohou této práce, proto na tyto úkony nebyla věnována dostatečná pozornost. Je tedy doporučeno nahrávat data v pixelových souřadnicích v rozmezí *x* a *y* cca [0,900] v závislosti na velikosti okna. Jinak se objekty nezobrazí.

Výsledkem je aplikace *Building Simplify* s grafickým uživatelským rozhraním, která je uvolněna pod licencí GNU GPL. Aplikace je psána v jazyce C++. Pro grafické uživatelské rozhraní byl použit framework QT.

Aplikace umožňuje nahrát polygony z textového souboru, případně si naklikat vlastní polygon.

Aplikace má implementovány dvě metody pro nalezení konvexní obálky: *Graham* a *Javis Scan*. Pro nalezení generalizace budovy byly implementovány 4 metody:

1. Minimum Area Enclosing Rectangle
2. Wall Average
3. Longest Edge
4. Weighted Bisector

|  |  |
| --- | --- |
| **Metoda** | **Úspěšnost zjednoduššení** |
| Minimum area enclosing rectangle | 100% |
| Wall average | 90% |
| Weighted bisector | 80% |
| Longest edge | 90% |

## Zdroje

*[1] Konvexní obálka množiny bodů* [online]. [cit. 2021-11-05]. Dostupné z: https://web.natur.cuni.cz/~bayertom/images/courses/Adk/adk4.pdf

Jakub Šimek, Jan Kučera