ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA STAVEBNÍ, OBOR GEODÉZIE A KARTOGRAFIE
KATEDRA GEOMATIKY

název předmětu

Algoritmy digitální kartografie a GIS

název úlohy

Množinové operace s polygony

školní rok	studijní skup.	číslo zadání	zpracoval, email	datum	klasifikace
2021/22	60	-	Jakub Šimek, Jan Kučera	7.12.21	

1. Zadání

Úloha č. 4: Množinové operace s polygony

Vstup: množina n polygonů $P=\{P_1, ..., P_n\}$.

Výstup: množina m polygonů $P' = \{P'_1, ..., P'_m\}$.

S využitím algoritmu pro množinové operace s polygony implementujte pro libovolné dva polygony $P_i, P_j \in P$ následující operace:

- Průnik polygonů $P_i \cap P_j$,
- Sjednocení polygonů $P_i \cup P_j$,
- Rozdíl polygonů: $P_i \cap \overline{P}_j$, resp. $P_j \cap \overline{P}_i$.

Jako vstupní data použijte existující kartografická data (např. konvertované shape fily) či syntetická data, která budou načítána z textového souboru ve Vámi zvoleném formátu.

Grafické rozhraní realizujte s využitím frameworku QT.

Při zpracování se snažte postihnout nejčastější singulární případy: společný vrchol, společná část segmentu, společný celý segment či více společných segmentů. Ošetřete situace, kdy výsledkem není 2D entita, ale 0D či 1D entita.

Pro výše uvedené účely je nutné mít řádně odladěny algoritmy z úlohy 1. Postup ošetření těchto případů diskutujte v technické zprávě, zamyslete se nad dalšími singularitami, které mohou nastat.

Hodnocení:

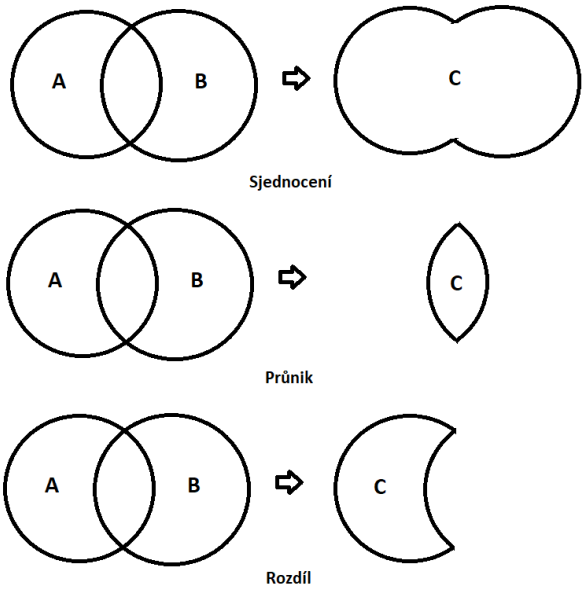
Krok	**Hodnocení**
Množinové operace: průnik, sjednocení, rozdíl	20b
Konstrukce offsetu (bufferu)	*+10b*
Výpočet průsečíků segmentů algoritmem Bentley & Ottman	*+8b*
Řešení pro polygony obsahující holes (otvory)	*+6b*
Max celkem:	**44b**

2. Bonusové úlohy

V úloze nebyly vypracovány bonusové úlohy.

3. Popis a rozbor problému

Při práci v GIS velmi často potřebujeme mít definované množinové operace s polygony. Základní množinové operace jsou: sjednocení, průnik a rozdíl.
Výsledkem těchto operací mohou být různé odvozené vrstvy důležité pro další analýzy.
Nechť $A\ \{p_i\}, B\ \{p_j\}$ jsou polygony tvořeny jejich lomovými body. Pak polygon $C\ \{p_k\}$ je výsledek množinové operace.
Grafické znázornění množinových operací:



Obrázek 1 - množinové operace

4. Popis algoritmů

Algoritmus sestává z několika kroků: výpočet průsečíků a jejich setřídění, ohodnocení vrcholů A, B dle pozice B, A, výběr vrcholů dle hodnocení, vytvoření hran, a nakonec spojení hran do oblastí.

1) Výpočet průsečíků A, B a jejich setřídění

Pro výpočet všech průsečíků množin A, B byl využit naivní algoritmus. To znamená, že se zkontrolují všechny dvojice hran z obou množin, zdali se kříží či nikoliv. Jestliže průsečík existuje, jsou spočteny jeho souřadnice.
Průsečík se spočte následujícím způsobem:
Nejdříve se spočtou směrové vektory:

$$\vec{u} = (x_2 - x_1, y_2 - y_1),$$
$$\vec{v} = (x_4 - x_3, y_4 - y_3),$$
$$\vec{w} = (x_1 - x_3, y_1 - y_3).$$

Z nich se následně spočtou koeficienty:

$$k_1 = v_x w_y - v_y w_x,$$
$$k_2 = u_x w_y - u_y w_x,$$
$$k_3 = v_y u_x - v_x u_y.$$

A nakonec koeficienty:

$$\alpha = \frac{k_1}{k_3},$$
$$\beta = \frac{k_1}{k_3}.$$

V případě, že $\alpha, \beta \in < 0,1 >$, pak průsečík existuje a můžeme určit jeho souřadnice:

$$x_b = x_1 + \alpha u_x,$$
$$y_b = y_1 + \alpha u_y.$$

Dalším krokem je rozhodnutí, zda průsečík zařadit nebo nezařadit do polygonu. K tomuto účelu je použita metoda *processIntersection* třídy *Algorithms*, která má následující implementaci:

1. $if\ (|t| > e\ \&\&\ ||t| - 1| > e)$: //$t = \alpha, \beta$
2. $i \leftarrow i + 1$ //Inkrementuj pozici
3. $P \leftarrow (b, i)$ //Přidej průsečík na pozici i+1

Pro implementaci algoritmu byla vytvořena nová třída *QPointFBO*, s rodičovskou třídou *QPointF*, uchovávající hodnoty α, β. Algoritmus pro hledání průsečíků má následující podobu:

1. $for\ (i = 0, i < n, i++)$: //Cyklus přes polygon A
2. $M = map(double, QPointFBO)$ //Inicial. mapy pro průsečíky
3. $for(j = 0, j < m, j++)$: //Cyklus pro polygon B
4. $if\ b_{ij} = (p_i, p_{(i+1)\%m}) \cap (q_j, q_{(i+1)\%m}) \neq \emptyset$ //Existuje průsečík?
5. $M[\alpha_i] \leftarrow b_{ij}$ //Přidat průsečík do mapy
6. $processIntersection(b_{ij}, \beta, B, j)$
7. $if(\|M\| > 0)$ //Jestliže byl nalezen alespoň 1
8. $for(\forall m \in M)$: //Cyklus přes všechny průsečíky
9. $b \leftarrow m.second$ //Získání druhé hodnoty z páru
10. $processIntersection(b, \alpha, A, i)$

2) Ohodnocení vrcholů A, B dle pozice vůči B, A

Ohodnocením vrcholů se rozumí zjištění, jakou polohu zaujímá daný vrchol vůči druhému polygonu. Zda leží vně, na hranici nebo uvnitř polygonu. Metoda *Winding Number Algorithm* pro tuto operaci byla vytvořena v jedné z předcházejících úloh.
Pro informaci o poloze bodu vůči polygonu byl vytvořen datový typ *TPointPolygonPosition* obsahující tři stavy: *Inner*, *Outer* a *On*.
Tato metoda třídy *QPointFBO* má název *setEdgePositions*. Je volána dvakrát, nejdříve Pro polygon A vůči polygonu B a poté obráceně. Je implementována následovně:

1. $for(i = 0, i < n, i++)$:
2. $m_x = (p_A[i].x() + p_A[i+1].x())/2$
3. $m_y = (p_A[i].y() + p_A[i+1].y())/2$
4. $getPositionWinding((m_x, m_y), B)$
5. Uložení pozice počátečního vrcholu hrany

3) Výběr vrcholů dle hodnocení

Dalším krokem je výběr hran podle zadaného kritéria podle stavů datového typu *TPointPolygonPosition*.
V následující tabulce jsou popsány situace, které mohou nastat.

Operace	A	B
Průnik	Inner	Inner
Sjednocení	Outer	Outer
A-B	Outer	Inner
B-A	Inner	Outer

4) Vytvoření hran

Posledním krokem je vytvoření hran výsledného polygonu. Ten se vytvoří průchodem všech vrcholů s ohodnocením podle tabulky z předchozí kapitoly.
Metoda třídy *Algorithms* se jmenuje *selectEdges* a má následující implementaci:

1. $for(i - 0, i < n, i++)$:
2. $if\ (pozice\ vrcholu == pozadovana\ operace)$:
3. $vytvoren\ hrany\ z\ aktualniho\ bodu\ a\ bodu + 1$
4. $pridani\ hrany\ do\ vektoru\ hran$

5. Optimalizace algoritmu

Pro četnější množinu polygonů může být výhodou zbytečně nepočítat vzdálené polygony bez žádného vzájemného kontaktu. Proto byla do procesu zavedena funkce počítající Bounding Box (dále jen BB) polygonů. Jedná se jednoduše o obdélník ohraničení minimálními a maximálními souřadnicemi x a y. Existuje-li styk BB obou polygonů, bude se dále pokračovat ve výpočetně náročnější množinové operaci. Tato optimalizace je realizována funkcí *BBoxIntersection* ve třídě *Algorithms*.

6. Vstupní data

Data lze do aplikace dostat dvěma způsoby. Prvním je klikáním do okna *Canvas*. Po určení prvního polygonu uživatel stiskne tlačítko *Switch A/B* a určí druhý polygon.
Dalším možným způsobem jak nahrát data je přes tlačítko *Load Data*. Soubor musí být ve formátu CSV.

6.1 Načtení ze souboru CSV

Pro tento vstup se nachází v aplikaci tlačítko $Load\ data$. Po kliknutí na toto tlačítko se otevře $Windows$ průzkumník souborů, ve kterém uživatel zvolí cestu k vstupnímu soboru.
Vstupní soubor musí být v textovém souboru ve formátu CSV. Požadavkem je, aby na prvním řádku byla hlavička souboru "WKT". Ukázka prvních dvou řádků souboru s vysvětlením:

```
WKT, formát souboru        Y1                X1                Y2                X2                Y3                X3
"MULTIPOLYGON (((668859.630091865 1130247.13127784,668859.920107736 1130247.06126134,668865.38009172 1130265.46126831,
 definice typu geometrie
```

Proces načítání můžeme rozdělit na dvě hlavní části, a to načtení souřadnic z CSV souboru a následná transformace na *Canvas*. Načítáme-li další data a nevymažeme aktuálně načtená, proběhne načtení dat s už existujícími transformačními parametry. Pokud od sebe nejsou data příliš vzdálená, budou viditelná spolu se stávajícími načtenými daty. Pokud budou data daleko od sebe nemusí být nově načítaná data na objektu *Canvas* vidět a bude potřeba nejdříve objekt vyčistit tlačítkem *Clear All*.

6.1.1 Načtení ze souboru

Princip této části je vysvětlen v úvodu této podkapitoly, kde je stěžejním prvkem, aby data byla v pravoúhlém systému XYZ a měla požadovaný WKT formát, jinak načtení dat nemusí proběhnout správně.

6.1.2 Transformace dat

Jelikož data mohou být v různých souřadnicových systémech, je nutné je transformovat, pokud jsou v pravoúhlém systému. Transformační klíč obsahuje pouze dva parametry. Je to posun a měřítko.

6.1.2.1 Měřítko

Měřítko je spočteno poměrem šířky (resp. dély) zobrazovacího okna (Canvasu) a šířky (resp. dély) datasetu (minmax boxu).

```
//Get size ratio for transformation to canvas
double x_ratio = canvas_width/dataset_width;
double y_ratio = canvas_height/dataset_height;
```

Aby nedošlo k deformaci poměru stran, je z uvedených poměrů *x* a *y* rozměrů vzat do transformace menší z nich.

```
//Get proper scale for whole dataset visibility without deformation
double scale;
if (x_ratio < y_ratio)
    scale = x_ratio;
else
    scale = y_ratio;
```

6.1.2.2 Posun

Posun je spočten posunem počátku datasetu vůči zobrazovacímu oknu. Počátky jsou v obou případech definovány levým minimálními souřadnicemi, tedy vzhledem k orientaci souřadných systémů se jedná o horní levé rohy.

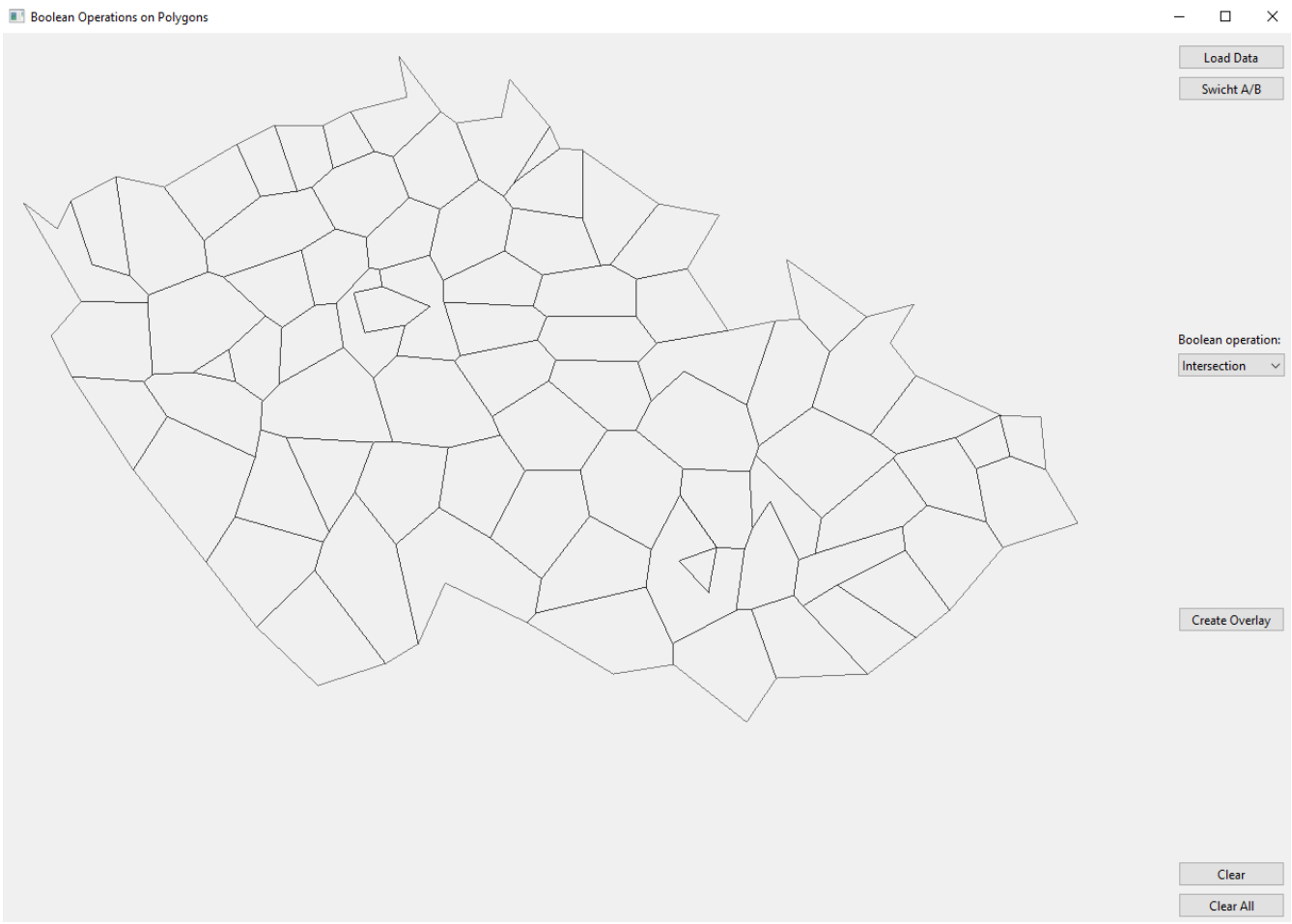
```
//Get translation parameter for transformation
double x_trans = x_min - x_left_top;
double y_trans = y_min - y_left_top;
```

6.1.2.3 Výsledná transformace a vliv posunutého počátku Canvasu

Transformace je realizována funkcí *transformPoints* z třídy *Algorithms*. Zde je potřeba zmínit zvláštní pozici počátku Canvasu, který je lokalizován na souřadnicích [x,y] = [11,11]. Tento posun je počátku je v transformaci potřeba uvážit.

```
//Transform polygon coorinates by basic transformation based on minmax box of dataset
//x_min, x_max, y_min, y_max represent boundaries of dataset minmax box
std::vector<QPoint3D> points_transformed;

for (QPoint3D p : points_3d)
{
    //Translation with slight offset due to canvas origin set on [11,11] coors
    double dx = p.x()-trans_x-offset_x;
    double dy = p.y()-trans_y-offset_y;

    //Data scaling
    double ddx = dx*(scale/1.1);
    double ddy = dy*(scale/1.1);

    //Translate data back to visible part of Canvas
    double x0 = ddx + offset_x;
    double y0 = ddy + offset_y;

    points_transformed.push_back(QPoint3D(x0, y0, p.getZ()));
}

//Compute transformation key
return points_transformed;
```

Jak je na obrázku vidět, data jsou nejdříve posunuta (parametr *trans*) s uvážením posunu počátku (parametr *offset*). Poté jsou délkově redukována měřítkem (parametr *scale*) a nakonec pro viditelnost dat znovu posunuta o ofset počátku.



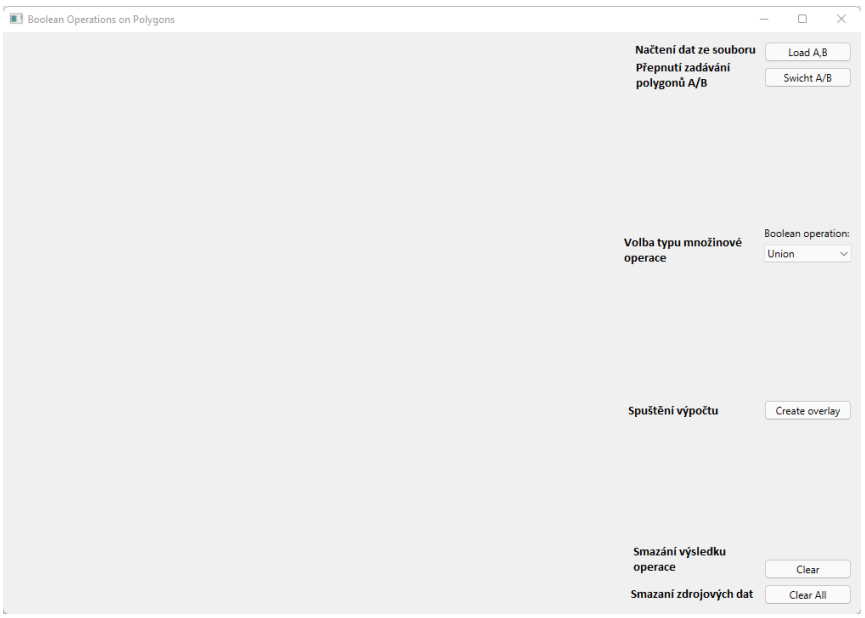
Ukázka načtení dat

7. Výstupní data

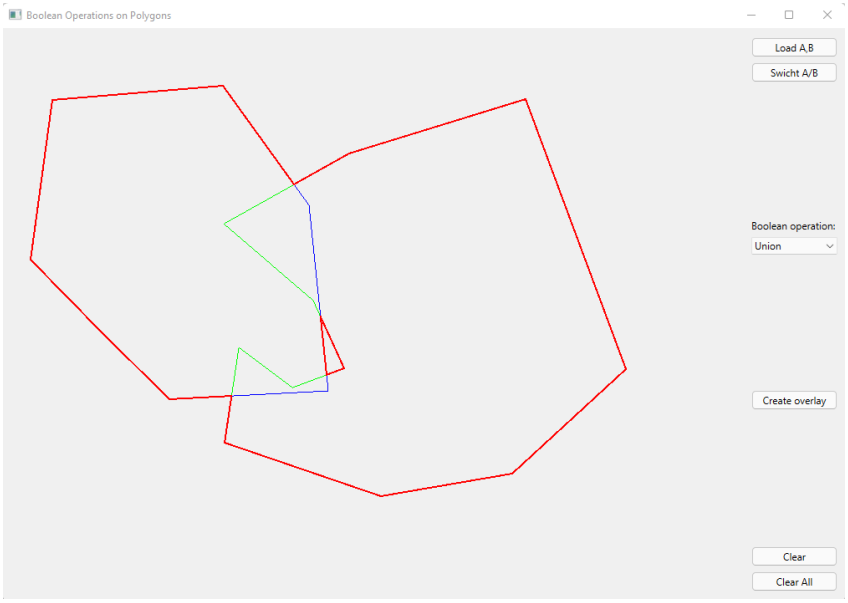
Výstupními daty se rozumí grafický výstup do *Canvas* v levé části UI.

8. Vzhled aplikace

V této kapitole bude aplikace ukázaná včetně jejich výstupů. Polygon A je modrý, polygon B je zelený a výsledek je červený.



Obrázek 2 - vzhled aplikace po spuštění



Obrázek 3 - výsledek operace Union

Obrázek 4 - výsledek operace Intersection

Obrázek 5 - výsledek operace Difference A-B

Obrázek 6 - výsledek operace Difference B-A

9. Dokumentace

1. Třída *Algorithms*

1. TPointLinePosition getPointLinePosition(QPointFBO &a,QPointFBO &p1,QPointFBO &p2);
   - Analyzuje vzájemnou polohu bodu a line
2. double get2LinesAngle(QPointFBO &p1, QPointFBO &p2, QPointFBO &p3, QPointFBO &p4);
   - vrátí úhel dvou linií
3. TPointPolygonPosition getPositionWinding(QPointFBO &q, TPolygon &pol);
   - Analyzuje vzájemnou polohu bodu a polygonu
4. std::tuple<QPointFBO,T2LinesPosition> get2LinesIntersection(QPointFBO &p1, QPointFBO &p2, QPointFBO&p3, QPointFBO &p4);
   - vrátí průsečík dvou přímek
5. void updatePolygons(TPolygon &A, TPolygon &B);

- Aktualizuje vrcholy polygonu pomocí vzájemných průniků

6. void processIntersection(QPointFBO &b, double t, int &index, TPolygon &P);
   - přidání průsečíku do polygonu
7. void setEdgePositions(TPolygon &A, TPolygon &B);
   - Nastaví polohy hran polygonu A do B
8. void selectEdges(TPolygon &P, TPointPolygonPosition pos, TEdges &edges);
   - vybere hranu podle zadané operace
9. TEdges createOverlay(TPolygon &A, TPolygon &B, TBooleanOperation &op);
   - Vytvoří výsledek z polygonů
10. std::vector<TPolygon> transformPolygons(std::vector<TPolygon> &polygons, double &trans_x, double &trans_y, double &scale, int &offset_x, int &offset_y);
    - transformuje polygon na Canvas
11. bool BBoxIntersection(TPolygon &A, TPolygon &B);
    - vrací hodnotu zda se protínají či jen dotýkají ohraničující obdélníky

2. Třída *CSV*
   a) *vector<QPolygon> read_csv(string &filename)*
      - čte vstupní *csv* soubor
   b) std::vector<QPoint3D> getCSVPolygons(std::vector<std::vector<std::string>> &csv_content, double &x_min, double &x_max, double &y_min, double &y_max);
      - vrací vektor QPoint3D

3. Třída *Draw*
   1. void *paintEvent*(QPaintEvent *event);
      - vykresluje data na Canvas
   2. void *mousePressEvent*(QMouseEvent *event);
   3. void switchSource(){addA = !addA;}
      - prohazuje kreslení polygonů A/B
   4. void drawPolygon(TPolygon &pol, QPainter &qp);
      - vykresluje polygon na Canvas
   5. TPolygon getA(){*return* A;}
   6. TPolygon getB(){*return* B;}
   7. void setEdges(TEdges &edg){res = edg;}
   8. void clear(){res.clear();}
   9. void clearAll(){A.clear(); B.clear(); res.clear(); polygons.clear();}
   10. double getScale(){*return* scale;}
   11. double getTransX(){*return* trans_x;}
   12. double getTransY(){*return* trans_y;}
   13. int getDeltaX(){*return* offset_x;}
   14. int getDeltaY(){*return* offset_y;}
   15. void setScale(double &scale_){scale = scale_;}
   16. void setTrans(double &trans_x_, double &trans_y_){trans_x = trans_x_; trans_y = trans_y_;}
   17. void setOffsets(int &offset_x_, int &offset_y_){offset_x = offset_x_; offset_y = offset_y_;}
   18. void drawCSVPolygons(std::vector<TPolygon> &polygons);
   19. void setCSVPolygons(std::vector<TPolygon> &polygons_){polygons.insert(polygons.end(), polygons_.begin(), polygons_.end());}
   20. std::vector<TPolygon> getPolygons(){*return* polygons;}
   21. int polygonsSize(){*return* polygons.size();}

4. Třída *Widget*
   1. void on_pushButton_clicked();
   2. void on_pushButton_2_clicked();
   3. void on_pushButton_3_clicked();

4. void on_pushButton_4_clicked();
5. void on_pushButton_LoadA_clicked();

10. Závěr

Výsledkem je aplikace *Boolean Operations* 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 nebo si naklikat vlastní. Po spuštění vrací aplikace výsledek množinové operace do grafického okna.

11. Zdroje

[1] Operace s uzavřenými oblastmi v GIS [online]. [cit. 2021-01-10]. Dostupné z: https://web.natur.cuni.cz/~bayertom/images/courses/Adk/adk9.pdf

Jakub Šimek, Jan Kučera