# ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

FAKULTA STAVEBNÍ, OBOR GEODÉZIE A KARTOGRAFIE
KATEDRA GEOMATIKY

Název předmětu

ALGORITMY DIGITÁLNÍ KARTOGRAFIE A GIS

| Úloha | Název úlohy |
|---|---|
| 4 | Množinové operace s polygony |

| Akademický rok | Semestr | Studijní skupina | Vypracoval | Datum | Klasifikace |
|---|---|---|---|---|---|
| 2018/2019 | 3. | 60 | Janovský Michal<br>karving47@gmail.com | 28. 12. 2018 | |

# 1 Obsah



# 2 Seznam obrázků

# 1 Zadání úlohy

**Úloha č. 4: Množinové operace s polygony**

Vstup: $množina\ n\ polygonů\ P = \{p_1, \ldots\ldots, p_n\}$
Výstup: $množina\ m\ polygonů\ P' = \{p'_1, \ldots\ldots, p'_n\}$

S využitím algoritmu pro množinové operace s polygony implementujte pro libovolné dva polygony následující operace:

- Průnik polygonů
- Sjednocení polygonů
- Rozdíl polygonů

Jako vstupní data použijte existující kartografická data 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 výsledky 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.

Povinná část úlohy:

- Množinové operace: průnik, sjednocení, rozdíl

Bonusové úlohy:

- Konstrukce bufferu
- Výpočet průsečíků segmentů algoritmem Bentley & Ottman
- Řešení pro polygony obsahující holes

# 2 Údaje o bonusových úlohách

V rámci úlohy byl pokus o implementaci bonusové úlohy konstrukce bufferu, nicméně tato implementace není zcela funkční a z časových důvodů byla opuštěna.

# 3 Popis a rozbor problému

Cílem úlohy je tvorba aplikace, která je schopná nad dvěma polygony provádět základní množinové operace. V rámci úlohy byly naprogramovány operace průnik, sjednocení a rozdíl polygonů A a B.


*Obrázek 1 - ukázka množinových operací*

# 4 Popisy algoritmů formálním jazykem

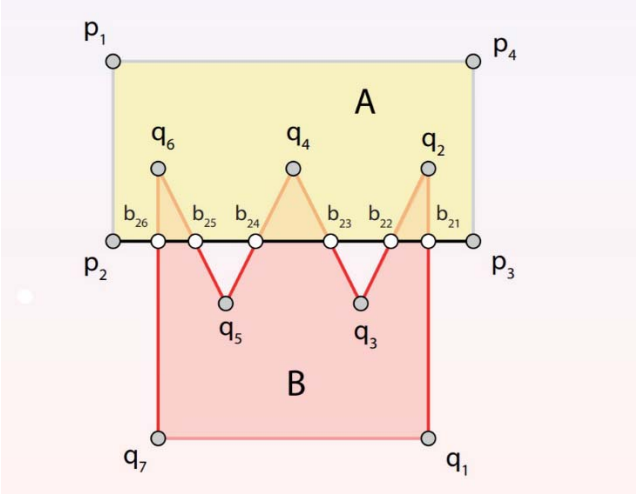
Celý algoritmus byl rozdělen do několika dílčích algoritmů.

- Určení průsečíků A a B a jejich seřazení
- Ohodnocení vrcholů podle pozice vůči druhému polygonu
- Podle ohodnocení výběr vrcholů a vytvoření fragmentů
- Z fragmentů vytvoření výsledných polygonů

Algoritmy použité v této úloze, byly implementovány v programovacím jazyce C++ v prostředí Qt Creator. Množinové operace, které byly rozděleny do dílčích algoritmů, jsou popsány níže.

## 4.1 Výpočet průsečíků + setřídění + update

Pomocí metody Get2LinesPosition byl hledán průsečík hran polygonů. Průsečíky byly uloženy do proměnné typu mapa s klíčem α,β a hodnotou průsečíku. Po nalezení průsečíku se aktualizuje seznam bodů polygonu za využití parametru α,β aby byly body polygonu ve správném pořadí. Nakonec pomocí Winding algoritmu je určena poloha středu hrany vzhledem k druhému polygonu.




*Obrázek 2 - Nalezení průsečíků množin A a B*

### 4.1.1 Implementace metody

Pro všechna i: $for\ (i = 0; i < n; i++)$

Vytvoření mapy: $M = map < double, QPointFB >$

Pro všechna j: $for\ (j = 0; j < m; j++)$

Pokud existuje průsečík: $if\ \left(b_{ij} = \left(p_i, p_{(i+1)\%n}\right) \cap \left(q_j, q_{(j+1)\%m}\right) \neq 0\right)$

Přidání do mapy: $M[\alpha_i] \leftarrow b_{ij}$

Zpracování prvního průsečíku: $ProcessIntersection\left(b_{ij}, \beta, B, j\right)$

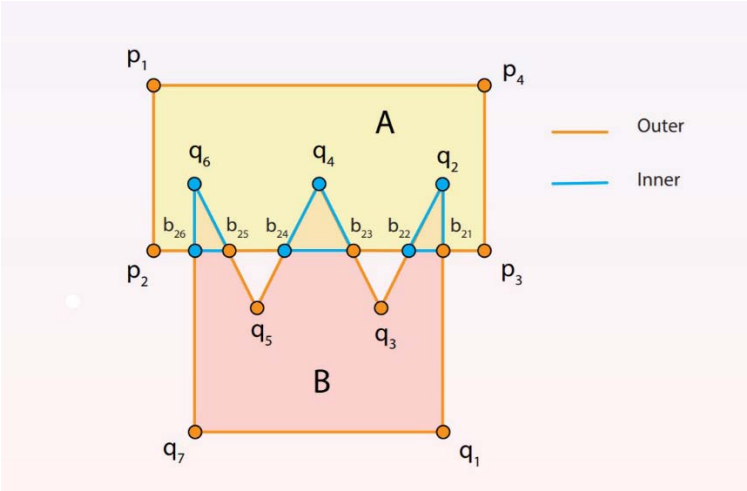
Při nalezení průsečíku: $if(\|M\| > 0)$

Procházení všech průsečíků: $for\ \forall m\ \in M$

Zpracování aktuálního průsečíku: $ProcessIntersection(b, \alpha, A, i)$

## 4.2 Ohodnocení vrcholů

Pro možnost ohodnocení vrcholů vůči jednotlivým polygonům byl vytvořen nový datový typ, který nabývá hodnot podle toho, zda se nachází uvnitř, na hranici či mimo polygon.




*Obrázek 3 - ohodnocení hran*

### 4.2.1 Implementace metody

Pro všechna i: $for\ (i = 0; i < n; i++)$

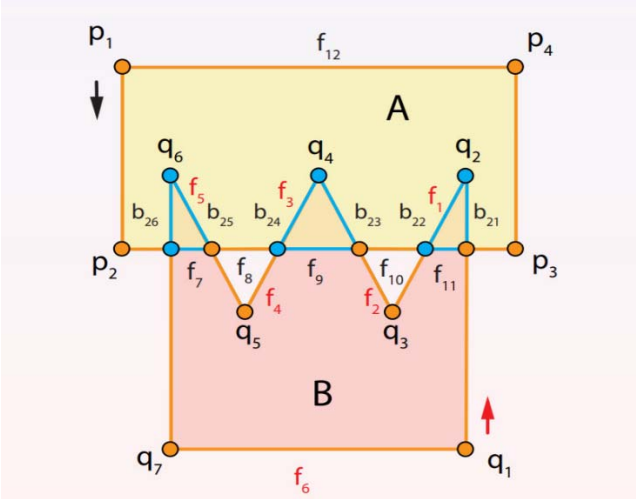
Výpočet středu hrany: $M = (p_i(x, y) + p_{i+1}(x, y))/2$

Poloha M vůči druhé množině: $pos = getPositionWinding(p, B)$

$p_i[pos] = pos$

## 4.3 Vytvoření fragmentů

Vrcholy se stejným ohodnocením byly přidány do fragmentu spolu s pozicí, na níž se nachází. Každý fragment začíná průsečíkem a končí prvním bodem s jiným ohodnocením. Fragment má opačné pořadí vrcholů (CW). Nakonec jsou fragtmenty spojeny do jednotlivých oblastí.




*Obrázek 4 - Fragmenty*

### 4.3.1 Implementace metody

Dokud $P[i]$ není průsečík s orientací g: $g(P[i]) \neq g \vee P[i] \neq inters$)

$i \leftarrow i + 1$

Žádný bod s touto orientací neexistuje: $if\ (i \equiv n)\ return$

Uložení startovního indexu prvního průsečík $i_s \leftarrow i$

$do$

Vytvoření prázdného fragmentu: $f = \emptyset$

Při nalezení fragmentu: $if\ (createFragmentFromVertices(i_s, P, g, if\ ))$

Swapování prvků je-li potřeba: $f.reverse()$

Přidání fragmentu do mapy s klíčem počátečního bodu: $F[f[0]] \rightarrow f$

Přejdi k dalšímu bodu přes index: $i \leftarrow (i+1)\%\, m$

Opakování dokud se nevrátí zpět k počátečnímu průsečíku: $while\ (i \neq i\_s)$

Následně byla vytvořena ještě jedna funkce pro tvorbu fragmentů, do níž vstupuje index startovního bodu, polygon P, orientace g, index vrcholů i a již vytvořený fragment f.

Bod není průsečíkem s orientací g: $if\ g(P[i]) \neq g \vee P[i] \neq inters \rightarrow return\ FALSE$

Nekonečný cyklus: $for\ (;;)$

Přidání bodu do fragmentu: $f \leftarrow P[i]$

Následující bod ze seznamu: $i \leftarrow (i+1)\%n$

Při navrácení ke startovnímu bodu: $if\ (i \equiv i\_s) \rightarrow return\ FALSE$

Při nalezení prvního bodu s rozdílnou orientací: $if\ (g(P[i]) \neq g)$

Přidání bodu do seznamu a úspěšné ukončení: $f \leftarrow P[i] \rightarrow return\ TRUE$

## 4.4 Vytvoření oblastí z fragmentů

Následně je nutné projít všechny fragmenty a sestavit z nich oblasti. Vstupem do funkce jsou vzniklé fragmenty F a výstupem seznam polygonů C.

### 4.4.1 Implementace metody

Pro všechna f: $for\ \forall f \in F$

Vytvoření prázdného polygonu: $P \leftarrow \emptyset$

Nalezení startovního bodu fragmentu: $s \leftarrow f.first$

Při nezpracování fragmentu: $if\ (!f.second.first)$

Přidání polygonu do seznamu: $C \leftarrow P$

Inicializace následujícího bodu: $QPointFB\ n \leftarrow s$

Nekonečný cyklus k procházení všech fragmentů: $for(;;)$

Nalezení navazujícího fragmentu: $f \leftarrow F.find(n)$

Při neexistenci fragmentu s takovýmto počátečním bodem: $if(f \equiv F.end) \rightarrow return\ FALSE$

Fragment označen za zpracovaný: $f.second.first \leftarrow TRUE$

Následující bod: $n \leftarrow f.second.second.back()$

Přidání bez počátečního bodu: $P \leftarrow f.second.second - \{f.second.second[0]\}$
Při vrácení se na začátek: $if\ (n \equiv s) \rightarrow return\ TRUE$

### 4.5 Výsledné množinové operace

Po vytvoření zmíněných dílčích algoritmů jsou funkce postupně volány.

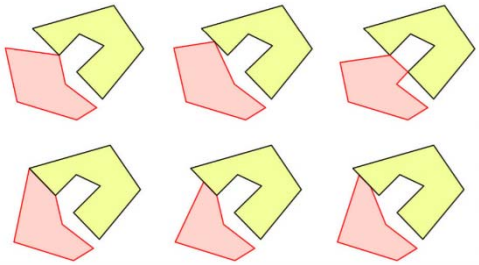
Nastavení správné orientace obou polygonů. $A.switchOr();\ B.switchOr()$
Výpočet průsečíků A, B: $ComputeIntersections(A, B)$
Určení polohy vrcholů vůči oblastem: $setPositions(A, B)$
Tvorba mapy fragmentů: $map\ F$
Určení pozice: $pos = (oper \equiv Intersection \vee oper \equiv DifBA ? Inner : Outer)$
Prohození: $swap = (oper \equiv DifAB) : TRUE : FALSE$
Tvorba fragmentů: $createFragments(A, pos, swap, F)$
Propojení fragmentů: $mergeFragments(A, B, C)$

## 5 Problematické situace a jejich rozbor

### 5.1 Singulární případy

Mezi singulární případy patří případy, kdy při množinových operacích nevzniká polygon, ale pouze úsečka či bod. Tyto případy jsou problematické, jelikož návratovým tipem algoritmu je polygon.

Problematické situace:
1) Společný vrchol
2) Více společných vrcholů
3) Vrchol na hraně polygonu
4) Společná část hrany
5) Společná celá hrana
6) Totožné polygony
7) Polygony s otvory



*Obrázek 5 - Problematické situace*

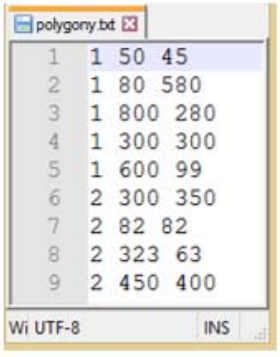
Při použití množinových operací nad těmito situacemi mohou vzniknout například tyto problémy:
Při použití operace UNION se v některých situacích polygony nesloučí
UNION → 1,3, 4,5 → vzniká prázdná množina namísto sloučení polygonů

Při použití operace INTERSECTION vznikají bod nebo úsečka, které se nikterak neuloží, jelikož nejsou typu polygon
INTERSECTION → 1,3 → bod
INTERSECTION → 4,5 → úsečka

Dalším problémem je zaokrouhlování a vyladění algoritmu pro zjištění vztahu bod-linie a linie-linie, kde může dojít k nesprávnému vyhodnocení vztahu dvou entit, což může vést ke vzniku výše popsaných situací. Tyto problémy však nebyly z časových důvodů řešeny a jsou blíže popsány v kapitole 10 Závěr.

## 6 Vstupní data, formát vstupních dat, popis

Primárními vstupními daty jsou dva polygony naimportované z textového souboru. V souboru jsou uložené souřadnice bodů jednotlivých polygonů s informací, ke kterému polygonu bod patří. Sekundárně se dají polygony „naklikat“ nebo se dají „naklikáním“ přidat body k již existujícím polygonům. Dále se jen zadává, jakou množinovou operaci chceme provést.

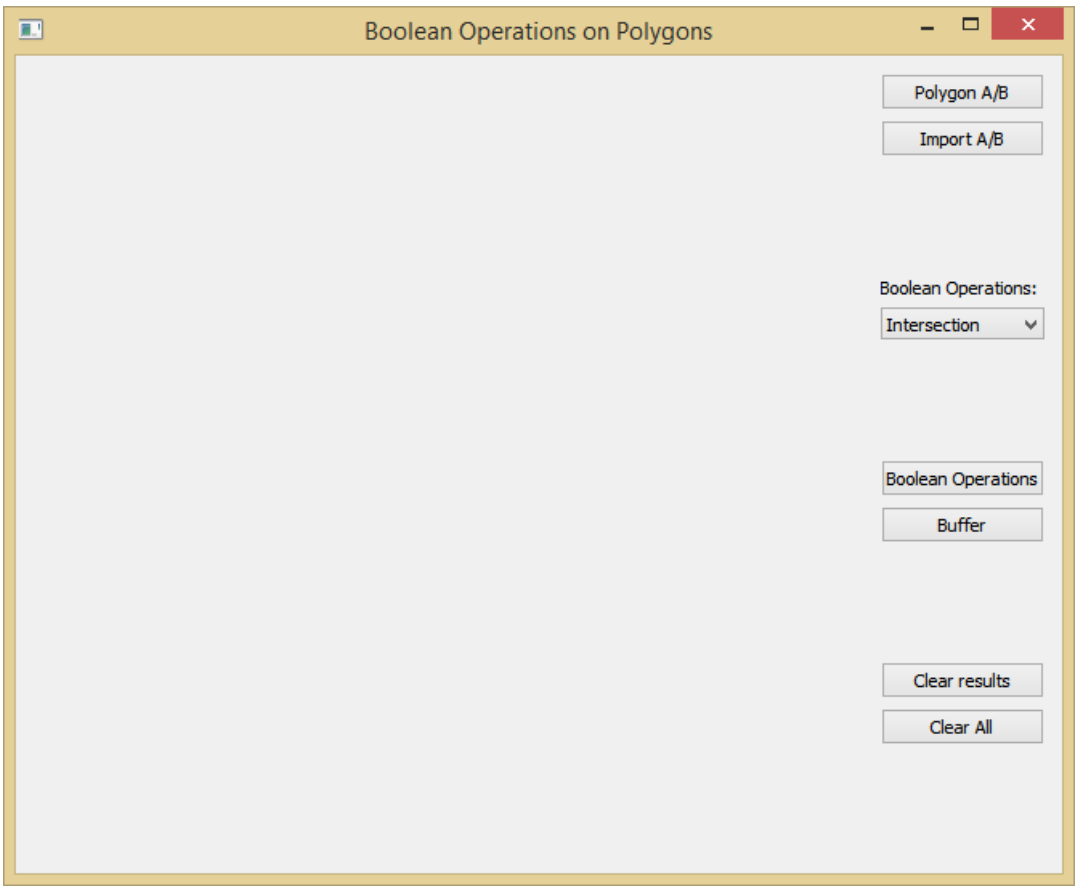



*Obrázek 6 - Vstupní data*

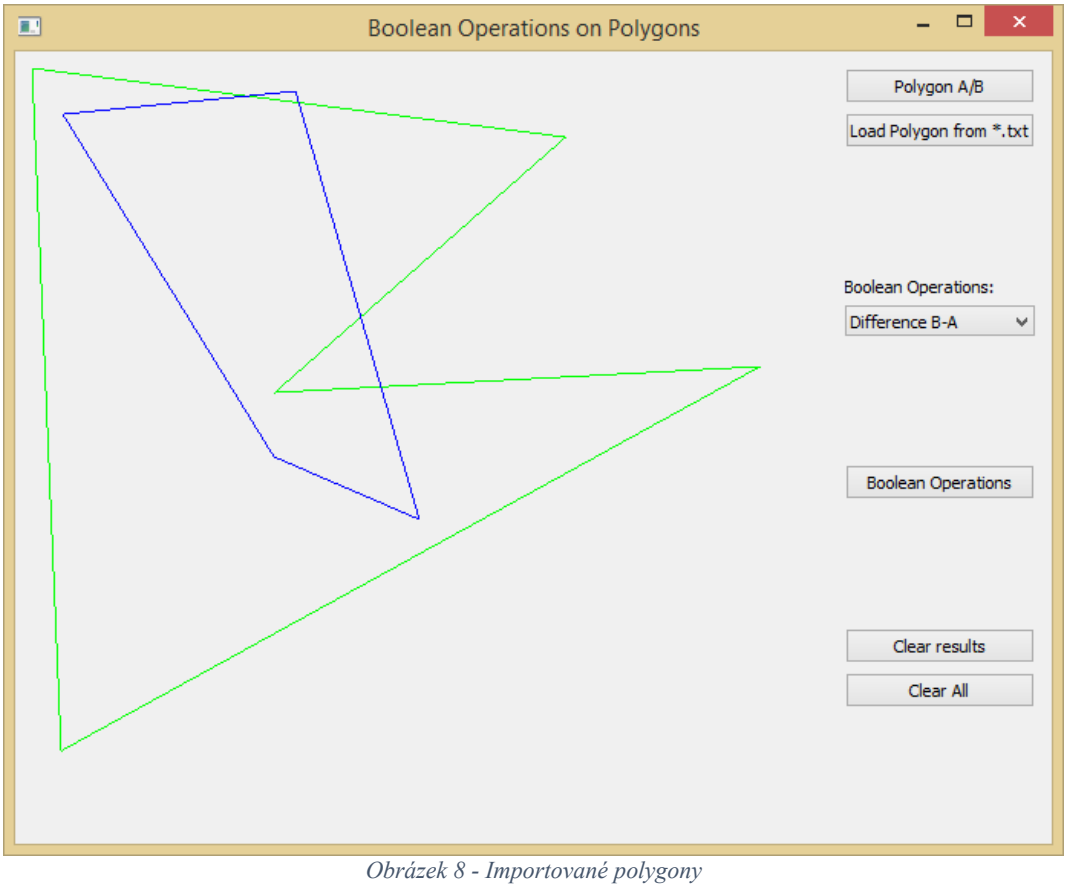
## 7 Výstupní data, formát výstupních dat, popis

Výstupem z programu je zobrazení nově vzniklých polygonů, které vzniknou aplikováním množinových operací na dva naimportované polygony z textového souboru (popř. z „naklikaných“ polygonů).

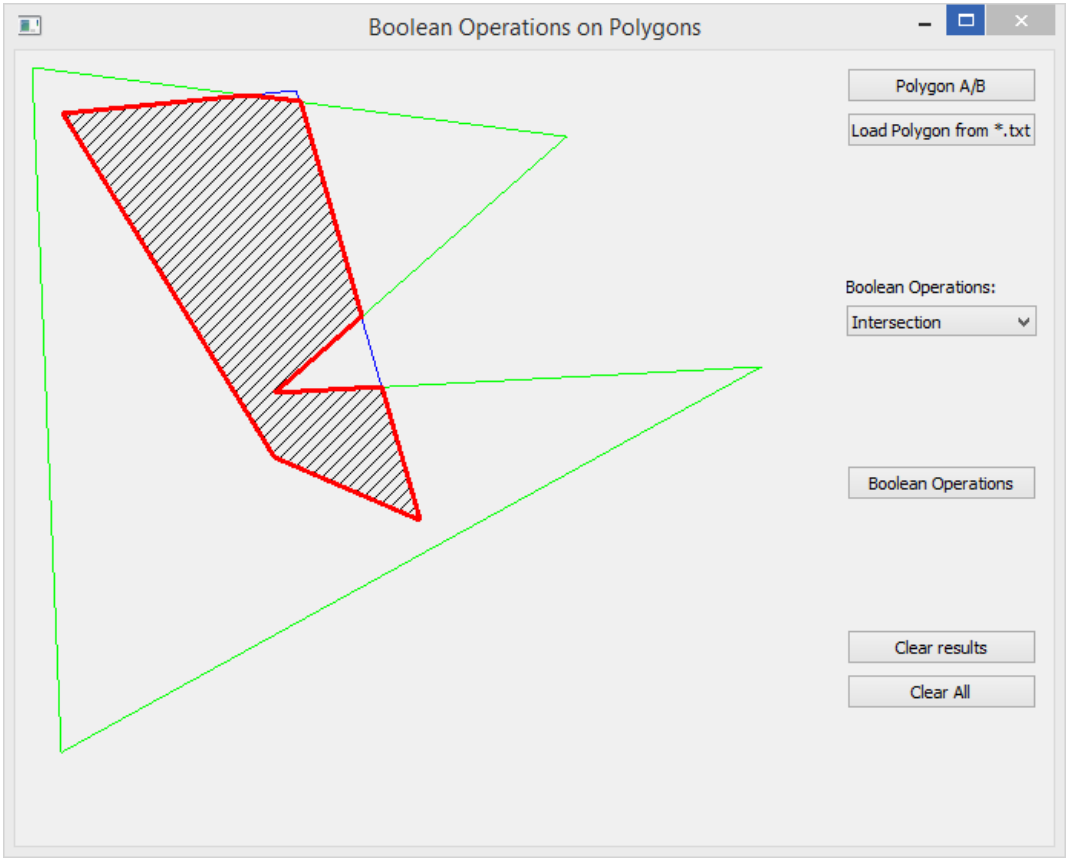
# 8 Printscreen vytvořené aplikace



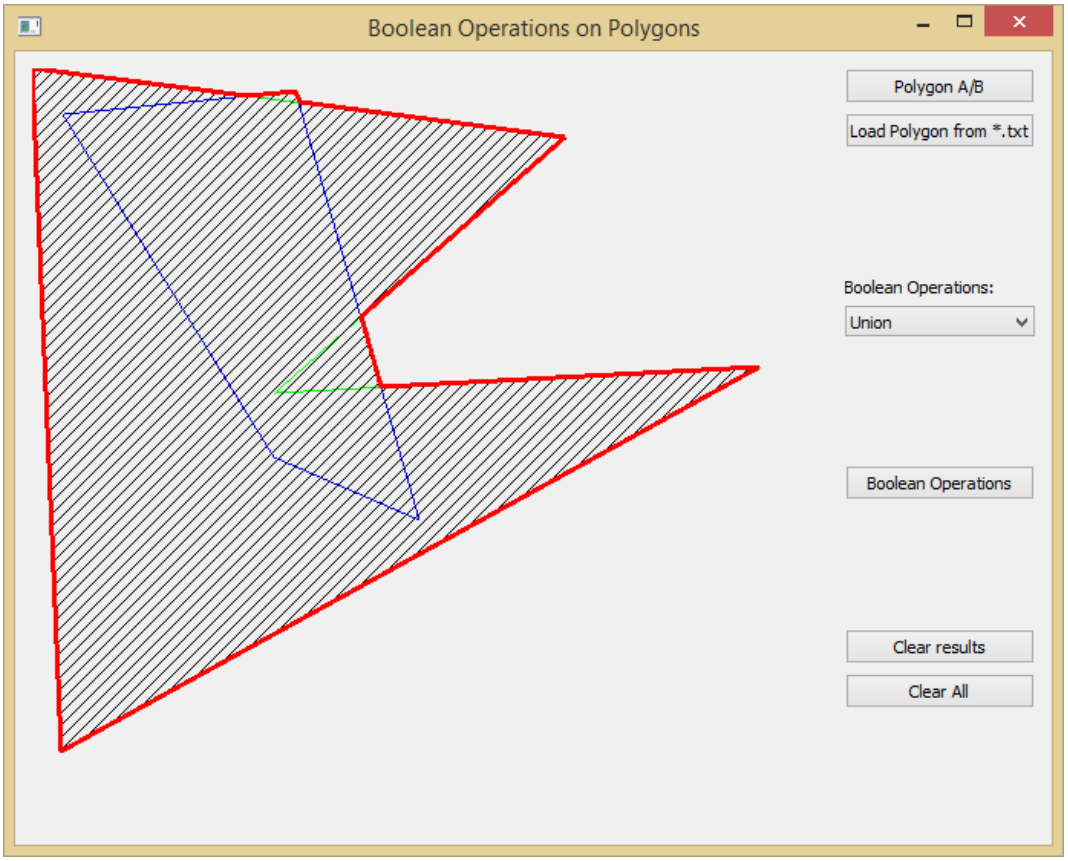

*Obrázek 7 - Program po spuštění*



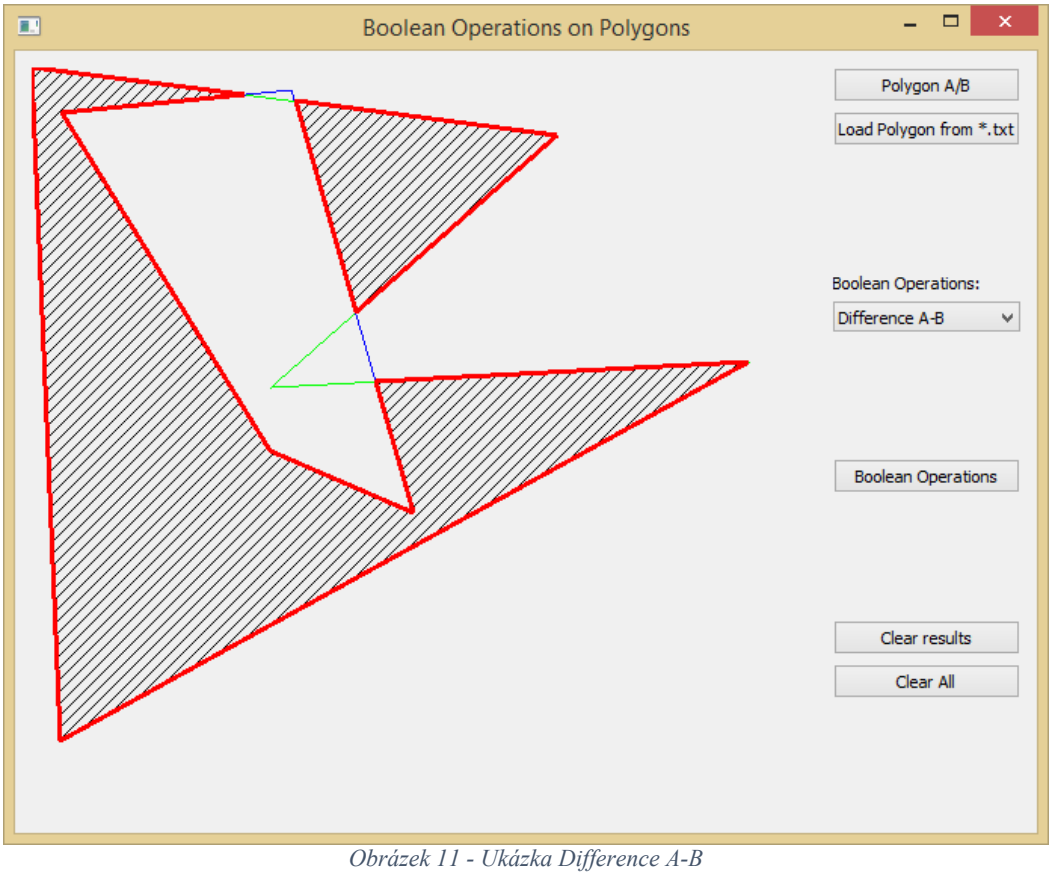

*Obrázek 8 - Importované polygony*



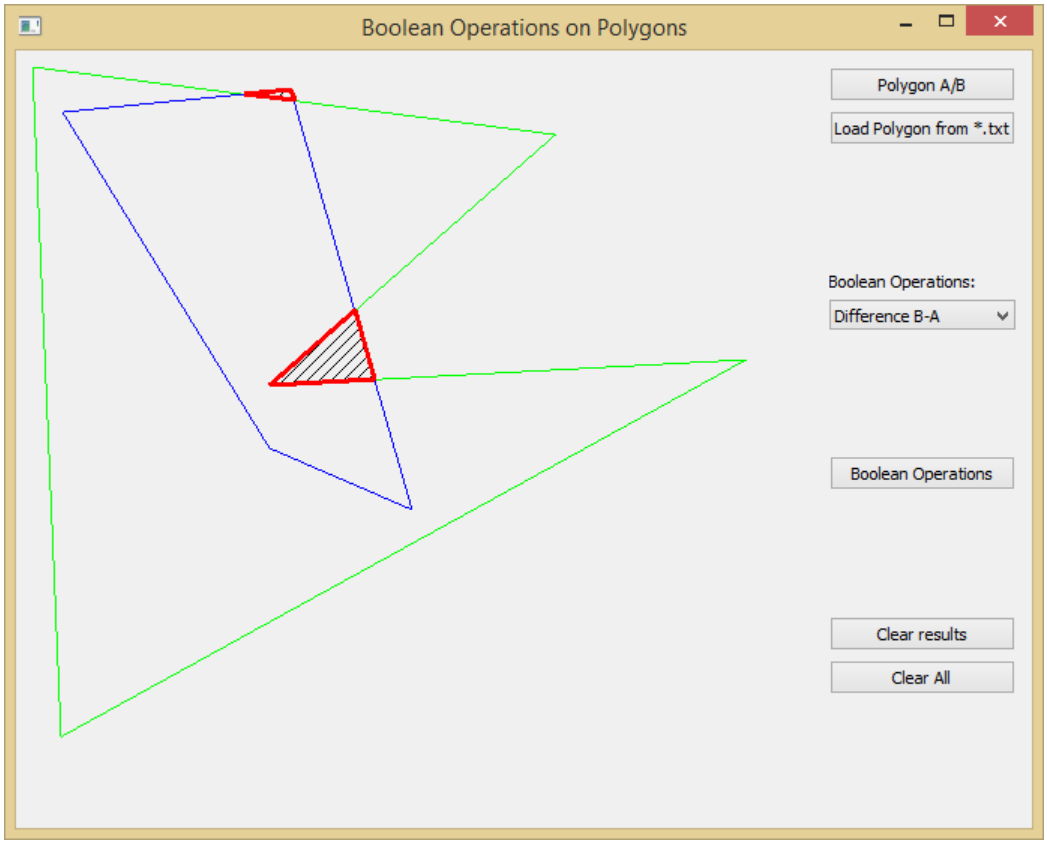

*Obrázek 9 - Ukázka Intersection*




*Obrázek 10 - Ukázka Union*




*Obrázek 11 - Ukázka Difference A-B*




*Obrázek 12 - Ukázka Difference B-A*

# 9 Dokumentaci: popis tříd, datových položek a jednotlivých metod

## 9.1 Třída Algorithms

Třída obsahující veškeré početní výkony potřebné k provedení množinových operací.

### 9.1.1 Metody třídy Algorithms

```
static TPointPolygon getPositionWinding(QPointFB q, std::vector<QPointFB> pol);
```
vrací vztah bodu k polygonu: INSIDE, OUTSIDE, ON

```
static TPointLinePosition getPointLinePosition(QPointFB &q, QPointFB &a, QPointFB
&b);
```
vrací vztah bodu k přímce: LEFT, RIGHT, COL

```
static double get2LinesAngle(QPointFB &p1,QPointFB &p2,QPointFB &p3, QPointFB &p4);
```
vrací úhel mezi 2 úsečkami

```
static T2LinesPosition get2LinesPosition(QPointFB &p1,QPointFB &p2,QPointFB &p3,
QPointFB &p4, QPointFB &intersection);
```
vrací vztah dvou přímek: PARALLEL, COLINEAR, INTERSECTION, NONINTERSECTING

```
static void computePolygonIntersections(std::vector<QPointFB> &p1,
std::vector<QPointFB> &p2);
```
spočte průsečíky polygonů a přidá je do polygonů

```
static void processIntersection(QPointFB &b, double t, std::vector<QPointFB> &poly,
int &i);
```
vkládá body do polygonu

```
static void setPositions (std::vector<QPointFB> &pol1,std::vector<QPointFB> &pol2);
```
nastaví bodu polygonu hodnotu pos

```
static void createFragments(std::vector<QPointFB> &pol, TPointPolygon pos, bool swap,
std::map <QPointFB, std::pair<bool,  std::vector<QPointFB> > > &fragments);
```
vytvoří fragmenty o stejné hodnotě posit a uloží je do hashovací tabulky

```
static bool createFragmentFromVertices (int i_start, std::vector<QPointFB> &pol,
TPointPolygon pos, int &i, std::vector<QPointFB> &fr);
```

```
static void mergeFragments(std::map<QPointFB, std::pair<bool, std::vector<QPointFB> >
> &FR, std::vector<std::vector<QPointFB> > &res);
```
sjednotí fragmenty a uloží je do vektoru polygonů

```
static bool createPolygonFromFragments(QPointFB &start, std::map<QPointFB,
std::pair<bool, std::vector<QPointFB> > > &FR, std::vector<QPointFB>  &pol);
```

```
static double getPolygonOrientation(std::vector<QPointFB> &pol);
```
vrací hodnotu plochy a podle znaménka určí orientaci

```
static std::vector<std::vector<QPointFB> > BooleanOper(std::vector<QPointFB> &A,
std::vector<QPointFB> &B, TBooleanOperation oper);
```
provádí nad polygony metodu INTERSECTION, UNION, DIFAB, DIFBA

```
static void resetIntersections(std::vector<QPointFB> &A);
```

```
static std::vector<QPointFB> lineOffset(QPointFB &p1,QPointFB &p2, double d, double
delta);
static std::vector<std::vector<QPointFB> > lineOffset(std::vector<QPointFB> &pol,
double d, double delta);
static void sampleArc(QPointFB &s, double r, double fi_s, double fi_e, double delta,
std::vector<QPointFB> &pol);
static std::vector<std::vector<QPointFB> > polygonOffset(std::vector<QPointFB> &pol,
double d, double delta);
```
Slouží k tvorbě bufferu, momentálně nefunkční

## 9.2 Třída Draw

Slouží k vykreslení polygonů, bufferu a nahrávání polygonů z txt souboru.

### 9.2.1 Datové položky třídy Draw

```
std::vector<QPointFB> polA;
std::vector<QPointFB> polB;
vstupní polygony

std::vector<std::vector<QPointFB> > res;
výsledné polygony množinových operací

bool ab;
výběr polygonu pro kreslení bodů

std::vector<std::vector<QPointFB> > buff;
polygony obsahující buffer
```

### 9.2.2 Metody třídy Draw

```
void paintEvent(QPaintEvent *e);
metoda pro kreslení

void drawPol(std::vector<QPointFB> &pol, QPainter &painter);
vykreslení polygonů

void mousePressEvent(QMouseEvent *e);
vkládání bodů do polygonů kliknutím do okna

void setAB(){ab = !ab;}
nastavení aktivního polygonu pro mousePressEvent

void clearAll();
smaže vše z okna

void clearResults();
smaže výsledky z okna

void setRes(std::vector<std::vector<QPointFB> > result){res = result;}
nastavení výsledků množinových operací

void setA(std::vector<QPointFB> polA_){polA = polA_;}
void setB(std::vector<QPointFB> polB_){polB = polB_;}
nastavení polygonů A a B

std::vector<QPointFB> getA(){return polA;}
std::vector<QPointFB> getB(){return polB;}
navrácení polygonů A a B

void setBuff(std::vector<std::vector<QPointFB> > buff_) {buff=buff_;}
nastavení bufferu

static void importPolygons(std::string &path, std::vector<QPointFB> &A,
std::vector<QPointFB> &B,  QSizeF &canvas_size);
import polygonů z TXT souboru
```

## 9.3 Třída Types

Třída definující nové návratové hodnoty pro použité algoritmy

## 9.4 Třída Widget

### 9.4.1 Sloty třídy Widget

```
void on_pushButton_clicked();
```
nastavení aktivního polygonu pro kreslení

```
void on_pushButton_4_clicked();
```
provedení **clearAll**

```
void on_pushButton_2_clicked();
```
provede výpočet a zobrazení výsledků množinových operací

```
void on_pushButton_3_clicked();
```
provedení **clearResults**

```
void on_pushButton_5_clicked();
```
provede výpočet a zobrazení bufferu

```
void on_Import_clicked();
```
importuje polygony A a B z TXT souboru

## 9.5 Třída QPointFB

Odvozená třída bodů rozšířená o nové parametry použité pro provedení množinových operací.

### 9.5.1 Datové položky třídy QPointFB

```
double alfa;
double beta;
```
koeficienty určující zda bod leží na přímce A nebo B

```
bool inters;
```
určuje, zda je bod průsečíkem

```
TPointPolygon pos;
```
Určuje polohu bodu k danému polygonu

### 9.5.2 Metody třídy QPointFB

```
double getAlfa(){return alfa;}
double getBeta(){return beta;}
bool getInters(){return inters;}
TPointPolygon getPosition(){return pos;}
```
Navrací hodnoty bodu třídy QPointFB

```
void setAlfa(double alfa_){alfa = alfa_;}
void setBeta(double beta_){beta = beta_;}
void setInters(bool inters_){inters = inters_;}
void setPosition(TPointPolygon pos_){pos = pos_;}
```
Nastavují hodnoty bodu třídy QPointFB

```
bool operator < (const QPointFB &p) const{return this -> x() < p.x();}
```
přetížený operátor pro porovnávání bodů třídy QPointFBpodle x

# 10 Závěr, možné či neřešené problémy, náměty na vylepšení

## 10.1 Závěr

Byla vytvořena aplikace, která dokáže provádět nad dvěma polygony množinové operace: průnik, sjednocení a rozdíl. Polygony, se kterýma se pracuje, jsou nahrávány z textového souboru. Problémem však jsou singulární případy, kdy program nevrací správné výsledky. Tento problém může být způsobený špatným odladěním části kódu s Winding algoritmem, chybami v zaokrouhlování či jinými chybami v kódu, které nebyly nalezeny.

Krom singulárních případů uvedených v zadání se mohou vyskytnout případy, kdy výsledkem může být bod či linie, kde celý algoritmus funguje pouze s výstupy typu polygon, tedy tyto výstupy (bod, linie) se nikterak neuloží/nezobrazí.

## 10.2 Náměty na vylepšení

Hlavním námětem na vylepšení je samozřejmě ošetření singulárních případů, které z časových důvodů nebylo provedeno. Dalším vylepšením by bylo zprovoznění konstrukce bufferu, které není zcela funkční a bylo tedy z finální verze odstraněno.