Algoritmy v digitální kartografii

Množinové operace s polygony

Zimní semestr 2018/2019
(oprava: 17. 1. 2019)

Tereza Kulovaná
Markéta Pecenová

Obsah

1 Zadání **2**

2 Popis a rozbor problému **3**

3 Algoritmy **3**
3.1 Výpočet průsečíků 4
3.1.1 computePolygonIntersections 4
3.1.2 setPositions 5
3.2 Fragmenty 5
3.2.1 createFragments 6
3.2.2 mergeFragments 7
3.3 BooleanOper 8

4 Vstupní data **8**

5 Výstupní data **9**

6 Problematické situace **9**

7 Aplikace **9**

8 Dokumentace **13**
8.1 Algorithms 13
8.2 Draw 18
8.3 QPointFB 19
8.4 Types 19
8.5 Widget 20

9 Závěr **22**

10 Zdroje **23**

1 Zadání

Zadání úlohy bylo staženo ze stránek předmětu 155ADKG.

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**

Čas zpracování: 2 týdny

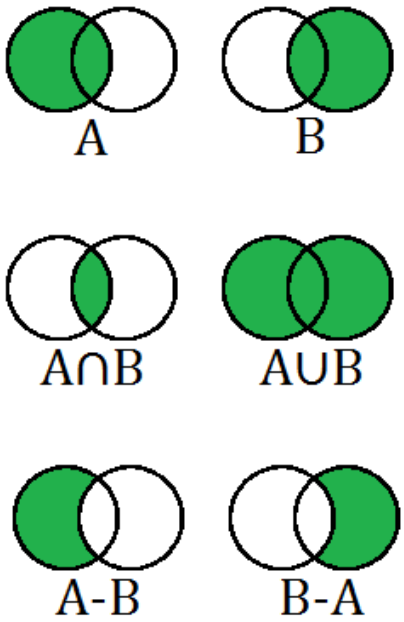
V rámci této úlohy nebyly implementovány žádné bonusové úlohy.

2 Popis a rozbor problému

Úloha **Množinové operace s polygony** se zabývá vytvořením aplikace, která nad libovolnými dvěma vstupními polygony provede zvolenou množinovou operaci.

Mějme vstupní polygony A a B. Množinové operace, které nad nimi lze vykonat, jsou následující:

1. Průnik (Intersection): $\longrightarrow A \cap B$
2. Sjednocení (Union) $\longrightarrow A \cup B$
3. Rozdíl (Difference) $\longrightarrow A \backslash B$ nebo $B \backslash A$



Obrázek 1: Ukázka množinových operací [Zdroj]

3 Algoritmy

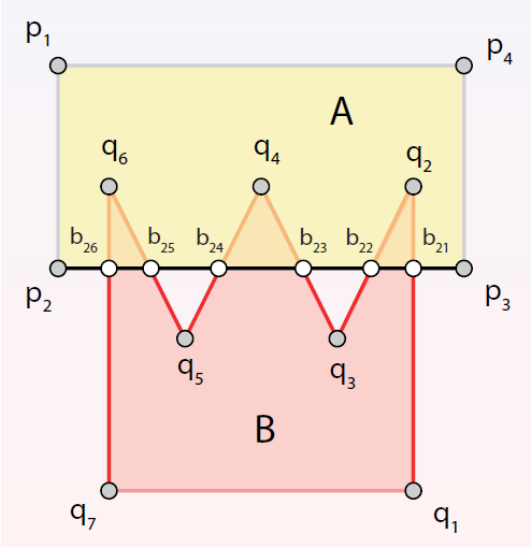
Tato kapitola se zabývá popisem algoritmů, které byly v aplikaci implementovány. Vzhledem k jejich složitosti je popis výsledného algoritmu rozdělen do jednotlivých fází. Výsledný algoritmus, který spojuje všechny kroky dohromady, se nazývá *BooleanOper*. Vstupní polygony A a B jsou reprezentovány kruhovými seznamy s orientací proti směru hodinových ručiček (CCW). Body, které tvoří oba polygony, mají nově vytvořený datový typ `QPointFB`.

3.1 Výpočet průsečíků

3.1.1 computePolygonIntersections

Průsečíky b_{ij} jsou ukládány do proměnné M datového typu `map`, která funguje na principu hashování. Spolu s výsledkem je ukládán i tzv. klíč, který na něj odkazuje. Za klíč byl zvolen parametr α.

Po nalezení každého nového průsečíku je nutné aktualizovat stávající seznamy bodů obou polygonů. K tomu slouží lokální procedura *ProcessIntersection*. Parametr t reprezentuje směrnice α nebo β, parametr i index daného bodu.



Obrázek 2: Průsečíky segmentů [Zdroj]

Zjednodušený zápis algoritmů lze zapsat způsobem uvedeným níže:

Postupně pro všechny $p_i \in A$:
- Vytvoř mapu M
- Postupně pro všechny $p_j \in B$:
  - Podmínka ($\exists$ průsečík b_{ij}):
    - Přidej: M$[\alpha_i] \leftarrow b_{ij}$
    - Zpracuj průsečík pro B: *ProcessIntersection*(b_{ij}, β, B, j)
- Podmínka ($M \neq \emptyset$):
  - Postupně pro všechny $b_{ij} \in M$:
    - Zpracuj průsečík pro A: *ProcessIntersection*(b_{ij}, α, A, i)

Lokální procedura *ProcessIntersection*(b, t, P, i):

1. Podmínka ($|t| < \epsilon$):

   Průsečíkem je počáteční bod: $P[i] \leftarrow inters$

2. Podmínka ($(|t-1| < \epsilon)$):

   Průsečíkem je koncový bod: $P[(i+1)\%n] \leftarrow inters$

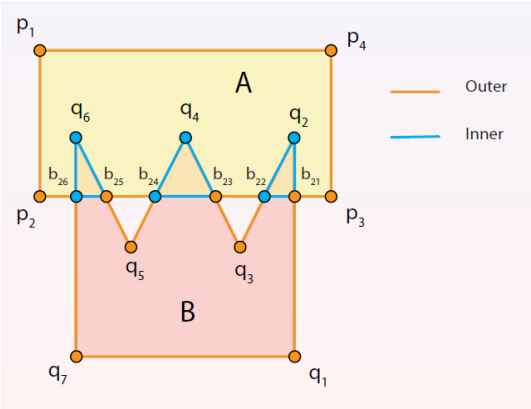
3. Jinak:

   Inkrementace: $i = i + 1$

   $P \leftarrow (b, i)$

3.1.2 setPositions

Všechny vrcholy polygonu A, resp. B (včetně nalezených průsečíků) jsou následně ohodnoceny, zda leží uvnitř, vně nebo na hraně polygonu B, resp. A. Pro všechny hrany e_i jsou vypočteny jejich středy $\bar{p}$, pro které se následně určuje jejich pozice (ohodnocení g) vůči druhému polygonu. Tato informace je uložena do počátečního bodu hrany e_i do parametru pozice. K určení pozice bodu vůči polygonu byl použit algoritmus *GetPositionWinding* z úlohy č. 1. Nejprve jsou zpracovány všechny hrany prvního polygonu a stejný postup je analogicky aplikován i pro druhý polygon.



Obrázek 3: Ohodnocení hran polygonů [Zdroj]

Zjednodušený zápis algoritmů lze zapsat způsobem uvedeným níže:

Postupně pro všechny $p_i \in A$:

$\bar{p} = \frac{p_i(x,y)+p_{i+1}(x,y)}{2}$

pozice = $GetPositionWinding(\bar{p}, B)$

$p_i[pozice]$ = pozice

3.2 Fragmenty

Sousedící vrcholy se stejným ohodnocením jsou uloženy do samostatných fragmentů f. Seznam bodů každého fragmentu začíná průsečíkem a končí bodem s jiným ohodnocením. Pro odlišení je orientace vrcholů ve fragmentech po směru hodinových ručiček (CW).

3.2.1 createFragments

Do metody vstupuje polygon P o velikosti n, ohodnocení vrcholů g a seznam fragmentů F. Algoritmus obsahuje lokální proceduru *createFragmentFromVertices*, která ze sousedících bodů o stejném ohodnocení vytváří fragment f. i_s je index počátečního bodu fragmentu, i je index přidávaného bodu a g ohodnocení.

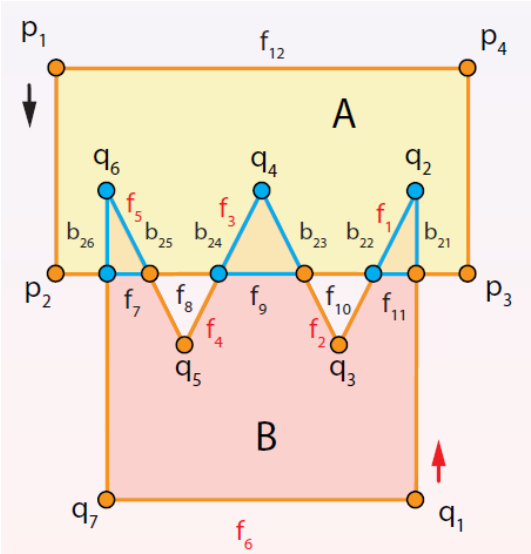
Zjednodušený zápis algoritmů lze zapsat způsobem uvedeným níže:

1. Inicializace: $i = n - 1;\ i_s = -1$
2. Dokud $(i > 0)$
   Podmínka $(P[i] = \text{průsečík} \wedge g(P[i]) = g)$
   $i_s = i;\ i - -$
3. Podmínka $(i_s < 0) \rightarrow$ žádný bod neexistuje, ukonči proces
4. Inicializace: $i = i_s$
5. Proveď:
   Podmínka $(P[i] = \text{průsečík} \wedge g(P[i]) = g)$
   Vytvoř fragment $f = \emptyset$
   Podmínka (*createFragmentFromVertices* vytvořen)
   Je-li potřeba, prohoď orientaci
   Přidej f do seznamu fragmentů F: $F[f[0]] \leftarrow f$
   Jinak inkrementace: $i = (i + 1)\%n$

   Dokud $(i \neq i_s)$

Lokální procedura *createFragmentFromVertices*(i_s, P, g, i, f):

Opakuj:
Přidej bod do fragmentu: $f \leftarrow P[i]$
Inkrementace: $i = (i + 1)\%n$
Podmínka $(i = i_s) \rightarrow$ return FALSE
Podmínka $(g(P[i]) \neq g)$
Přidej bod do fragmentu: $f \leftarrow P[i]$
Return TRUE



Obrázek 4: Fragmenty [Zdroj]

3.2.2 mergeFragments

Algoritmus spojuje jednotlivé fragmenty f do výstupních polygonů. Metoda má na vstupu seznam fragmentů F a seznam polygonů C, do kterého jsou ukládány výsledné polygony. Metoda nejprve spojí jednotlivé fragmenty do oblastí a ty jsou následně lokální procedurou *createPolygonFromFragments* převedeny na polygony. Počáteční bod fragmentu je značen jako s, n značí následující bod.

Postupně pro všechna $f \in F$:

Vytvoř: $P = \emptyset$

Najdi startovní bod fragmentu: $s = f.first$

Podmínka (f nebyl ještě zpracován)

Podmínka (*createPolygonFromFragments*)

Přidej: $C \leftarrow P$

Lokální procedura *createPolygonFromFragments*(s, F, P):

Inicializace: $n = s$

Opakuj:

Nalezni další fragment: $f = F.find(n)$

Podmínka (fragment s daným počátečním bodem $\nexists$) $\rightarrow$ return FALSE

Označ fragment za zpracovaný: $f.second.first \leftarrow$ TRUE

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

Přidej fragment bez počátečního bodu: $P \leftarrow f.second.second - f.second.second[0]$

Podmínka ($n = s$) $\rightarrow$ return TRUE

3.3 BooleanOper

Tento algoritmus spojuje všechny výše uvedené kroky. Na vstupu jsou polygony A a B a typ zvolené množinové operace.

1. Podmínka (orientace $A \vee B \neq$ CCW) $\rightarrow$ prohoď orientaci
2. *computePolygonIntersections(A, B)*
3. *setPositions(A, B)*
4. Vytvoř mapu fragmentů F
5. Zvolení ohodnocení: $pos1 = (oper \equiv intersection \vee oper \equiv DiffAB?Inner : Outer)$
   $pos2 = (oper \equiv intersection \vee oper \equiv DiffBA?Inner : Outer)$
6. Prohození: $swap1 = (oper \equiv DiffAB? : true : false)$
   $swap2 = (oper \equiv DiffBA? : true : false)$
7. *CreateFragments (A, pos1, swap1, F)*

   CreateFragments (B, pos2, swap2, F)
8. *MergeFragments (A,B,C)*

4 Vstupní data

Seznam bodů vstupních polygonů je uložen v textovém souboru *polygons.txt*. Pro vykreslení polygonů v aplikaci je nutno tento soubor do aplikace nahrát pomocí tlačítka *Load polygon file*. K vygenerování souřadnic bodů byla použita online aplikace ze stránek mobilefish.com. Struktura souboru s polygony je následující:

První řádek: počet bodů n v polygonu A
Sloupec 2: souřadnice X bodu polygonu A
Sloupec 3: souřadnice Y bodu polygonu A
...
(n+2)-tý řádek: počet bodů m v polygonu B
Sloupec 2: souřadnice X bodu polygonu B
Sloupec 3: souřadnice Y daného polygonu B

Po úspěšném/neúspěšném nahrání souboru je uživatel upozorněn hláškou. Uživatel nemůže kliknout na žádné jiné tlačítko, nejsou-li nahrána data (tlačítka jsou zašedivělá). Po nahrání vstupního souboru jsou data vykreslena a uživatel získá možnost zvolit, jaký typ množinové operace bude nad daty prováděn rozbalením nabídky *Boolean Operations*. Výpočet se provede stisknutím tlačítka *Boolean Operations*.

5 Výstupní data

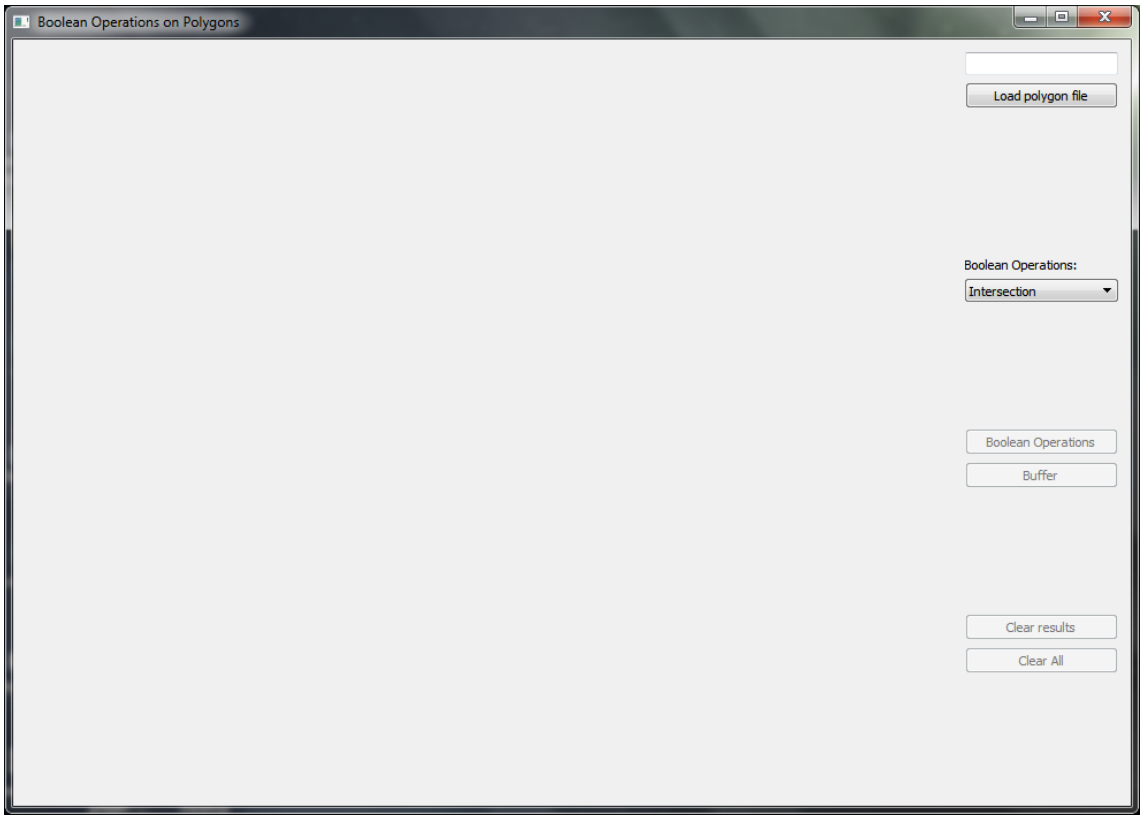
Vstupní polygony a nad nimi provedená množinová operace jsou zobrazeny v grafickém okně aplikace. Polygon A je vykreslen zelenou barvou, polygon B modrou barvou a výsledek operace je zobrazen červeně. Tlačítkem *Clear results* má uživatel možnost smazat výsledek operace při zachování vykreslení původních polygonů. Tlačítko *Clear All* maže veškerá vykreslená data.

6 Problematické situace

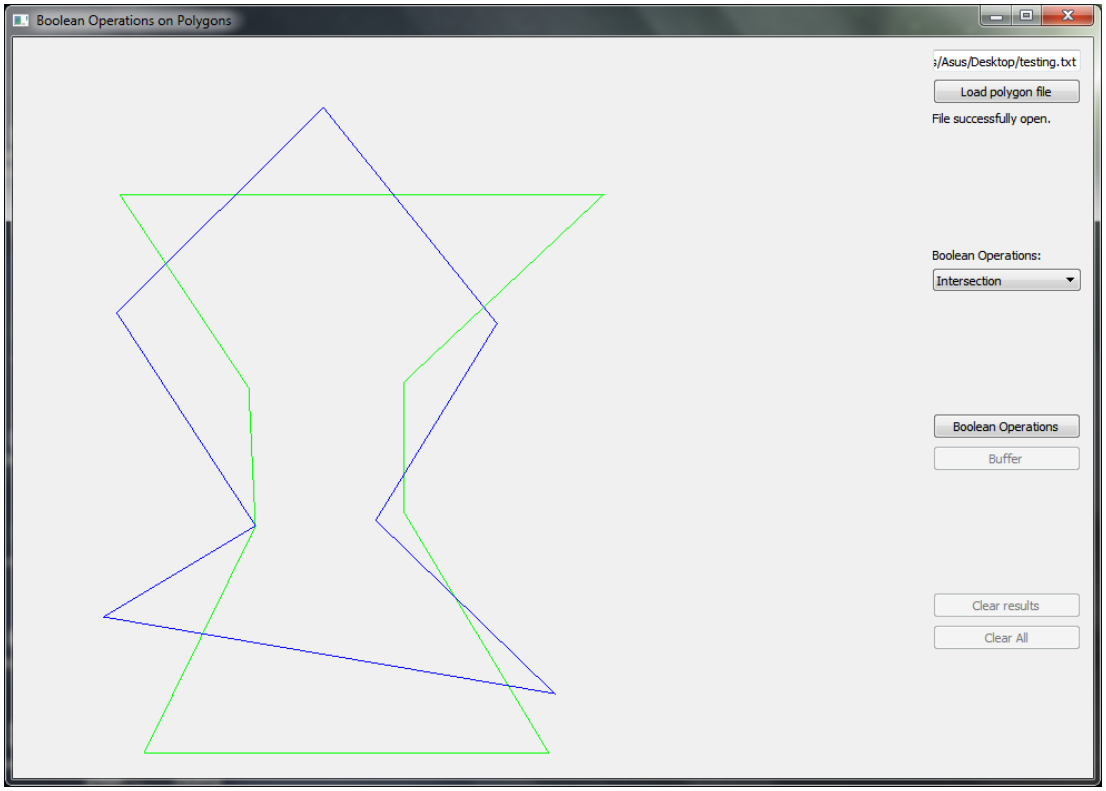
Vzhledem k náročnosti použitých algoritmů může docházet k velkému množství problematických situací. Konkrétně se jedná o případy, dotýkají-li se vzájemně dva polygony pouze svými vrcholy (ať už jedním nebo více) nebo mají-li společnou jednu či více hran. Další problém by nastal, pokud by byly oba polygonu shodné, či se v některém z nich nacházel otvor. Aplikace v nynějším stavu si není schopna poradit ani s jednou výše uvedenou situací.

7 Aplikace

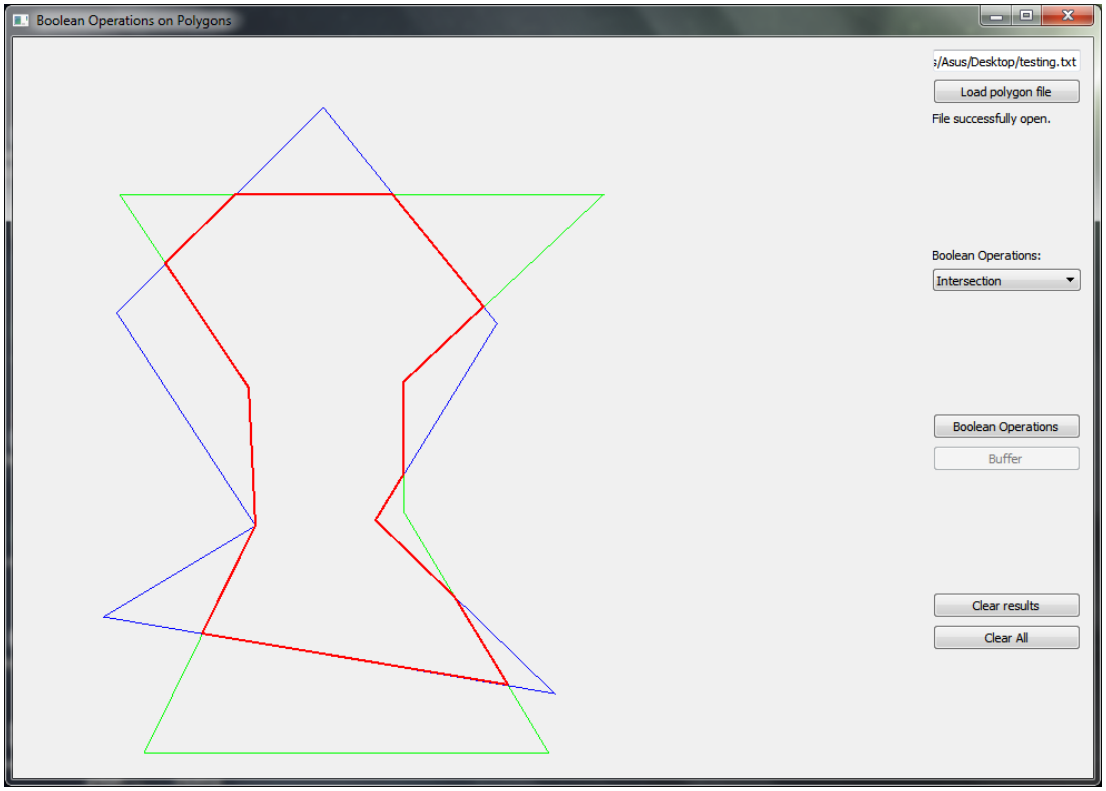
V následují kapitole je představen vizuální vzhled vytvořené aplikace tak, jak ji vidí prostý uživatel.



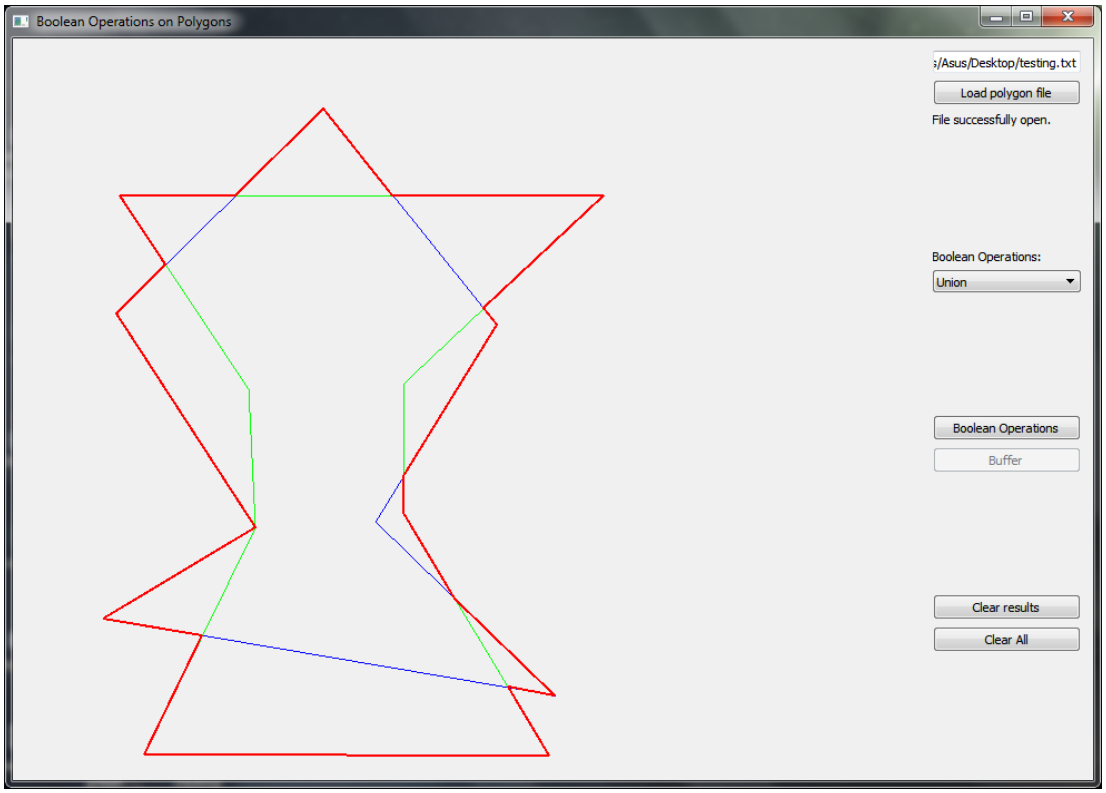
Obrázek 5: Výchozí vzhled aplikace po spuštění



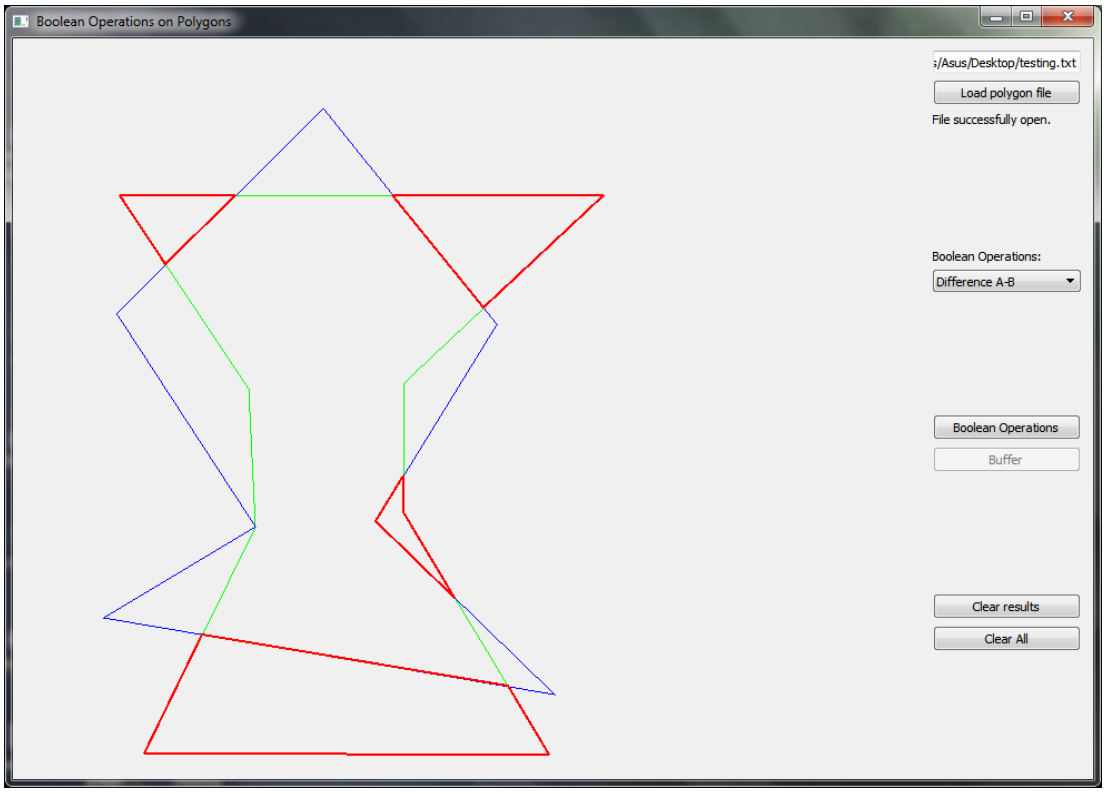
Obrázek 6: Aplikace po nahrání vstupních dat



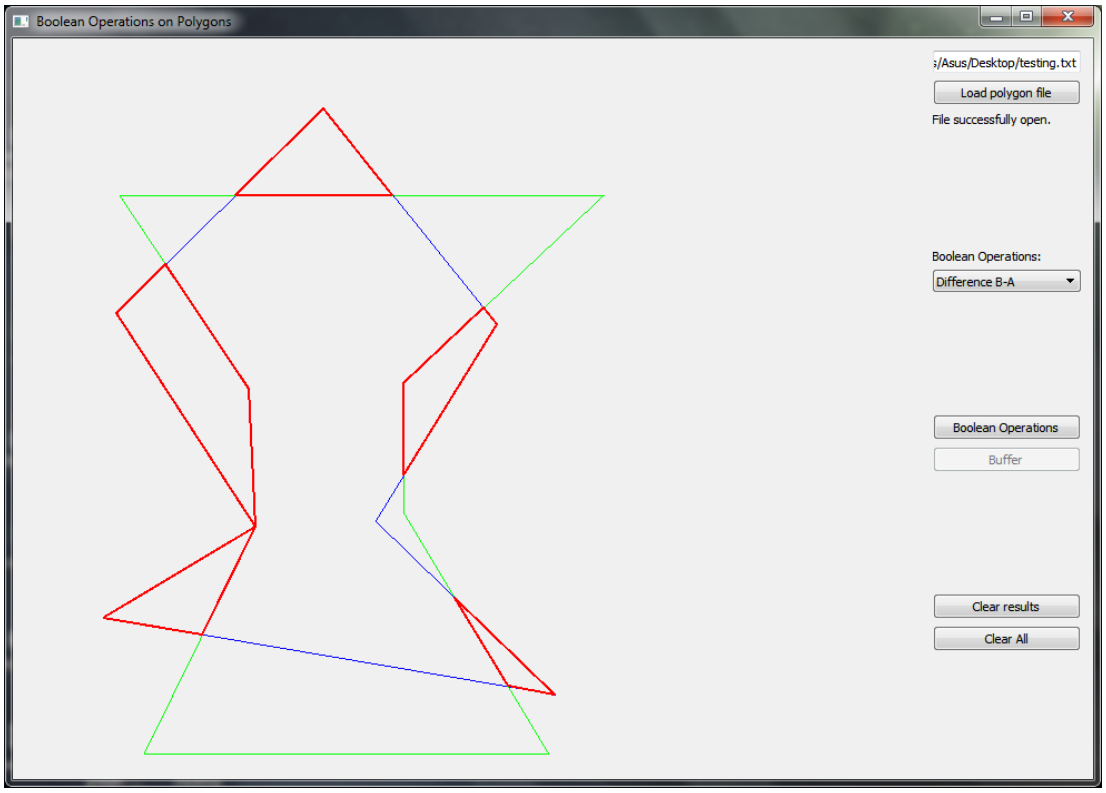
Obrázek 7: Průnik polygonů $A \cap B$



Obrázek 8: Sjednocení polygonů $A \cup B$



Obrázek 9: Rozdíl polygonů $A \backslash B$



Obrázek 10: Rozdíl polygonů $B \setminus A$

8 Dokumentace

Tato kapitola obsahuje dokumentaci k jednotlivým třídám.

8.1 Algorithms

Třída *Algorithms* obsahuje metody pro realizaci množinových operací. Dále obsahuje pomocné algoritmy k jejich fungování.

getPositionWinding

Metoda **getPositionWinding** určuje polohu bodu q vzhledem k polygonu *pol* za použití algoritmu *Winding Number*. Na vstupu je bod q a vektor bodů polygonu třídy `QPointFB`. Návratová hodnota typu `TPointPolygon` vrací polohu bodu q vůči polygonu *pol*.

Input:

- `QPointFB` q
- *vector* <`QPointFB`> *pol*

Output:

- `INSIDE` → q se nachází uvnitř polygonu *pol*
- `OUTSIDE` → q se nachází vně polygonu *pol*
- `ON` → q se nachází na hraně polygonu *pol*

getPointLinePosition

Metoda **getPointLinePosition** určuje polohu bodu q vzhledem k přímce tvořené dvěma body. Na vstupu jsou 3 body typu `QPointFB`, návratová hodnota je nově definovaný typ `TPosition`.

Input:

- `QPointFB` q
- `QPointFB` a
- `QPointFB` b

Output:

- `LEFT` → bod se nachází vlevo od přímky
- `RIGHT` → bod se nachází vpravo od přímky
- `ON` → bod se nachází na přímce

get2LinesAngle

Metoda **get2LinesAngle** počítá úhel mezi dvěma přímkami. Na vstupu jsou 4 body typu `QPointFB`, návratová hodnota typu `double` vrací velikost úhlu v radiánech. Body p_1 a p_2 definují první přímku, zbylé dva body druhou přímku.

Input:

- `QPointFB` p_1
- `QPointFB` p_2
- `QPointFB` p_3
- `QPointFB` p_4

Output:

- `double`

get2LinesPosition

Metoda **get2LinesPosition** určuje vzájemnou polohu dvou přímek. Pokud se přímky protínají, metoda vypočte jejich průsečík $p_{intersection}$. Na vstupu jsou 4 body typu `QPointFB`, návratová hodnota je nově definovaný typ `T2LinesPosition`. Body p_1 a p_2 definují první přímku, další dva body druhou přímku.

Input:

- `QPointFB` p_1
- `QPointFB` p_2
- `QPointFB` p_3
- `QPointFB` p_4
- `QPointFB` $p_{intersection}$

Output:

- `PARALLEL` → přímky jsou rovnoběžné
- `COLINEAR` → přímky jsou kolineární
- `INTERSECTING` → přímky se protínají v průsečíku $p_{intersection}$
- `NONINTERSECTING` → přímky se neprotínají

computePolygonIntersections

Metoda **computePolygonIntersections** počítá průsečíky dvou polygonů A a B. Na vstupu jsou dva vektory bodů polygonů, návratová hodnota je typu `void`.

Input:

- *vector* <`QPointFB`> $polA$
- *vector* <`QPointFB`> $polB$

processIntersection

Metoda **processIntersection** slouží k aktualizování seznamu bodů (tzv. map) polygonu po přidání nově nalezeného průsečíku. Na vstupu jsou čtyři parametry: průsečík b, směrnice přímky t, vektor bodů polygonu *poly* a index i aktuálního bodu. Návratová hodnota je typu `void`.

Input:

- `QPointFB` b
- `double` t
- *vector* <`QPointFB`> *poly*
- `int` i

setPositions

Metoda **setPositions** určuje pozici vrcholů prvního polygonu vzhledem k druhému polygonu, tzv. je ohodnocuje. Informace o pozici, která je datového typu `TPointPolygon`, je ukládána do parametru *pos* daného bodu. Na vstupu jsou dva vektory bodů polygonu, návratová hodnota typu `void`.

Input:

- *vector* <`QPointFB`> $polA$
- *vector* <`QPointFB`> $polB$

createFragments

Metoda **createFragments** vytváří ze sousedních bodů o stejném ohodnocení jednotlivé fragmenty f a ukládá je do seznamu (mapy) fragmentů. Na vstupu je vektor bodů polygonu, ohodnocení (pozice) hledaných bodů, orientace fragmentu a seznam se vzniklými fragmenty. Návratová hodnota je typu `void`.

Input:

- *vector* <`QPointFB`> *pol*

- `TPointPolygon` *pos*
- `bool` *swap*
- *map* <`QPointFB`, *pair* <`bool`,*vector* <`QPointFB`>>> *fragments*

createFragmentFromVertices

Metoda **createFragmentFromVertices** je lokální procedurou metody **createFragments**, která zajišťuje vytváření fragmentů f. Na vstupu je index i_s počátečního bodu fragmentu, vektor bodů polygonu a fragmentu, pozice (ohodnocení) bodů a index i daného bodu. Návratová hodnota typu `bool` vrací, zda byl fragment vytvořen či nikoli.

Input:

- `int` i_s
- *vector* <`QPointFB`> *pol*
- `TPointPolygon` *pos*
- `int` i
- *vector* <`QPointFB`> fr

Output:

- 0 → fragment nebyl vytvořen
- 1 → fragment byl vytvořen

mergeFragments

Metoda **mergeFragments** spojuje fragmenty do jednotlivých oblastí a následně do polygonů, které ukládá do seznamu polygonů. Na vstupu je seznam fragmentů FR a seznam polygonů res. Návratová hodnota je typu `void`.

Input:

- *map* <`QPointFB`, *pair* <`bool`,*vector* <`QPointFB`>>> FR
- *vector* <*vector* <`QPointFB`>> res

createPolygonFromFragments

Metoda **createPolygonFromFragments** je lokální procedurou metody **mergeFragments** a vytváří z fragmentů polygony. Na vstupu je index *start*, seznam fragmentů FR a vektor bodů polygonu. Návratová hodnota typu `bool` vrací, zda byl z fragmentu vytvořen polygon či nikoli.

Input:

- `int` *start*
- *map* <`QPointFB`, *pair* <`bool`,*vector* <`QPointFB`>>> *FR*
- *vector* <`QPointFB`> *pol*

Output:

- 0 → polygon nebyl vytvořen
- 1 → polygon byl vytvořen

getPolygonOrientation

Metoda **getPolygonOrientation** slouží k získání orientace bodů v polygonu za využití l'Huilierových vzorců. Na vstupu je vektor bodů polygonu, návratová hodnota typu `double` vrací plochu polygonu.

Input:

- *vector* <`QPointFB`> *pol*

Output:

- `double` $< 0 \rightarrow$ CW orientace
- `double` $> 0 \rightarrow$ CCW orientace

BooleanOper

Metoda **BooleanOper** spojuje výše uvedené algoritmy. Na vstupu jsou dva vektory bodů polygonu, metoda vrací vektor vektoru bodů `QPointFB`, který obsahuje výsledek zvolené množinové operace.

Input:

- *vector* <`QPointFB`> *polA*
- *vector* <`QPointFB`> *polB*

Output:

- *vector* <*vector* <`QPointFB`>>

resetIntersections

resetIntersections je pomocná metoda, která pro všechny body v polygonu nastaví, že nejsou průsečíky. Na vstupu je vektor bodů polygonu, návratová hodnota je typu `void`.

Input:

- *vector* <`QPointFB`> *pol*

8.2 Draw

Třída *Draw* obsahuje metody, které nahrávají a vykreslují vstupní polygony. Dále zajišťuje vykreslení a smazání všech operací, kterou jsou nad polygony prováděny.

paintEvent

Metoda **paintEvent** vykresluje vstupní polygony a výsledek množinové operace.

drawPol

Metoda **drawPol** vykresluje vstupní polygon. Na vstupu je vektor bodů polygonu.

loadPolygon

Metoda **loadPolygon** slouží k načtení vstupních dat do aplikace. Součástí metody je i kontrola, zda se soubor úspěšně nahrál. Návratová hodnota typu *QString* vrací hlášku, zda byly polygony úspěšně nahrány či nikoli.

setAB

Metoda **setAB** slouží k převedení nahraných polygonů do kreslící plochy.

clearAll

Metoda **clearAll** slouží k vymazání všech vykreslených dat.

clearResults

Metoda **clearResults** slouží k vymazání výsledku množinové operace. Vstupní polygony zůstávají nedotčené.

setRes

Metoda **setRes** slouží k převedení výsledného vektoru vektoru bodů do kreslící plochy.

setA

Metoda **setA** slouží k převedení vektoru bodů polygonu A do kreslící plochy.

getA

Metoda **getA** slouží k získání vektoru bodů polygonu.

setB

Metoda **setB** slouží k převedení vektoru bodů polygonu B do kreslící plochy.

getB

Metoda **getB** slouží k získání vektoru bodů polygonu B.

8.3 QPointFB

Třída *QPointFB* slouží k definování nového datového typu `QPointFB`, který je odvozen od typu `QPointF` a který navíc obsahuje směrnice přímek *alfa* a *beta*, informaci, zda bod je průsečíkem, a polohu bodu vůči druhému polygonu. Defaultně je nastaveno, že bod není průsečíkem, leží na hraně druhého polygonu a hodnoty obou směrnic jsou rovny nule.

getAlfa

Metoda **getAlfa** slouží k získání směrnice *alfa*.

setAlfa

Metoda **setAlfa** slouží k nastavení směrnice *alfa*.

getBeta

Metoda **getBeta** slouží k získání směrnice *beta*.

setBeta

Metoda **setBeta** slouží k nastavení směrnice *beta*.

getInters

Metoda **getInters** slouží k získání informace, zda bod je průsečíkem (*true*) či nikoli (*false*).

setInters

Metoda **setInters** slouží k nastavení informace, zda bod je průsečíkem či nikoli.

getPosition

Metoda **getPosition** slouží k získání pozice (ohodnocení) bodu.

setPosition

Metoda **setPosition** slouží k nastavení pozice (ohodnocení) bodu.

8.4 Types

Třída *Types* slouží k definování nových datových typů výčtového typu.

TPointPolygon

Datový typ **TPointPolygon** definuje polohu bodu q vůči polygonu P.

- `INSIDE` $\rightarrow q \in P$
- `OUTSIDE` $\rightarrow q \notin P$

- `ON` $\rightarrow q$ leží na P

TBooleanOperation

Datový typ **TBooleanOperation** definuje množinovou operaci, která je nad polygony A a B prováděna.

- `INTERSECTION` $\rightarrow A \cap B$
- `UNION` $\rightarrow A \cup B$
- `DIFFAB` $\rightarrow A \backslash B$
- `DIFFBA` $\rightarrow B \backslash A$

T2LinesPosition

Datový typ **T2LinesPosition** definuje polohu dvou přímek a a b.

- `PARALLEL` $\rightarrow a \parallel b$
- `COLINEAR` $\rightarrow a = b$
- `INTERSECTING` $\rightarrow a \cap b \neq \emptyset$
- `NONINTERSECTING` $\rightarrow a \cap b = \emptyset$

TPointLinePosition

Datový typ **TPointLinePosition** definuje polohu bodu q a přímky a.

- `LEFT` $\rightarrow$ bod q leží vlevo od přímky a
- `RIGHT` $\rightarrow$ bod q leží vpravo od přímky a
- `COL` $\rightarrow$ bod q leží na přímce a

8.5 Widget

Metody třídy *Widget* slouží pro práci uživatele s aplikací. Metody na vstupu nemají žádné parametry a návratové hodnoty jsou typu `void`.

on_load_button_clicked

Metoda **on_load_button_clicked** načítá data z textového souboru. Uživatel sám vyhledává cestu k požadovanému souboru.

on_operation_button_clicked

Metoda **on_operation_button_clicked** nad vstupními polygony provede zvolenou množinovou operaci.

on_clear_res_button_clicked

Metoda **on_clear_button_clicked** vymaže výsledek množinové operace z kreslící plochy.

on_clear_all_button_clicked

Metoda **on_contours_button_clicked** vrací aplikaci do výchozí polohy smazáním všeho, co bylo vykresleno.

9 Závěr

V rámci úlohy *Množinové operace s polygony* byla vytvořena aplikace, která nad dvěma vstupními polygony aplikuje zvolenou množinovou operaci. Implementace algoritmů byla často dost náročná a nad naše síly. Nepodařilo se nám tedy naplnit všechny cíle této úlohy, jako je například ošetření singulárních případů či dodělání bufferu, který byl vytvořen na poslední hodině, ale který obsahoval chybu, kterou se nám nepodařilo nalézt. V rámci opravy úlohy byly doplněny obrázky ilustrující princip fungování dílčích algoritmů a doplněna kapitola *Problematické situace*.

Do budoucna by jistě bylo vhodné ošetřit právě singulární případy, např. mají-li oba polygony právě jeden společný vrchol nebo celou hranu. Dále by se dal ošetřit případ, má-li jeden (nebo oba) polygony v sobě díru (pro představu, vypadají jako donut).

10 Zdroje

1. *BAYER, Tomáš.* Množinové operace s polygony [online][cit. 17. 1. 2019]. Dostupné z: https://web.natur.cuni.cz

2. *Elementary Set Theory* [online] [cit. 4. 1. 2019]. Dostupné z: http://www.efgh.com/

3. *Record XY mouse coordinates* [online][cit. 5. 1. 2019]. Dostupné z: https://mobilefish.com/