

Algoritmy v digitální kartografii

Geometrické vyhledávání bodu

Zimní semestr 2018/2019

Tereza Kulovaná
Markéta Pecenová

Obsah

1	Zadání	2
2	Popis a rozbor problému	3
3	Algoritmy	3
3.1	Ray Crossing Algorithm	3
3.2	Winding Number Algorithm	4
4	Problematické situace	5
5	Vstupní data	5
6	Výstupní data	5
7	Aplikace	6
8	Dokumentace	9
8.1	Algorithms	9
8.1.1	getPositionRay	9
8.1.2	getPositionWinding	9
8.1.3	getPointLinePosition	10
8.1.4	getTwoVectorsAngle	10
8.2	Draw	11
8.2.1	paintEvent	11
8.2.2	mousePressEvent	11
8.2.3	clearCanvas	11
8.2.4	fillPolygon	11
8.2.5	loadPolygon	11
8.2.6	getPoint????	12
8.3	Widget	12
8.3.1	writeResult	12
8.3.2	on_load_button_clicked	12
8.3.3	on_analyze_button_clicked	12
8.3.4	on_clear_button_clicked	12
9	Závěr	13
10	Zdroje	14

1 Zadání

Vstup: Souvislá polygonová mapa n polygonů $\{P_1, \dots, P_n\}$, analyzovaný bod q .

Výstup: $P_i, q \in P_i$.

Nad polygonovou mapou implementujete následující algoritmy pro geometrické vyhledávání:

- Ray Crossing Algorithm (varianta s posunem těžiště polygonu).
- Winding Number Algorithm.

Nalezený polygon obsahující zadaný bod q graficky zvýrazněte vhodným způsobem (např. vyplněním, šrafováním, blikáním). Grafické rozhraní vytvořte s využitím frameworku QT.

Pro generování nekonvexních polygonů můžete navrhnout vlastní algoritmus či použít existující geografická data (např. mapa evropských států).

Polygony budou načítány z textového souboru ve Vámi zvoleném formátu. Pro datovou reprezentaci jednotlivých polygonů použijte špagetový model.

Hodnocení:

Krok	Hodnocení
Detekce polohy bodu rozlišující stavy uvnitř, vně na hranici polygonu.	10b
Ošetření singulárního případu u Winding Number Algorithm: bod leží na hraně polygonu.	+2b
Ošetření singulárního případu u obou algoritmů: bod je totožný s vrcholem jednoho či více polygonů.	+2b
Zvýraznění všech polygonů pro oba výše uvedené singulární případy.	+2b
Algoritmus pro automatické generování nekonvexních polygonů.	+5b
Max celkem:	21b

Čas zpracování: 2 týdny.

V rámci této úlohy byly implementovány bonusové úlohy č. 1-3. Poslední bonusovou úlohu, generování nekonvexních polygonů, tato aplikace neposkytuje.

2 Popis a rozbor problému

Úloha *Geometrické vyhledávání bodu* se zabývá vytvořením aplikace, která umožní uživateli zjistit polohu jím zvoleného bodu q vzhledem k příslušnému mnohoúhelníku. Jako vhodné řešení bylo vzhledem k náročnosti problému zvoleno opakované určování polohy bodu q a mnohoúhelníku.

Existují dva základní druhy mnohoúhelníků (pro účely této úlohy je nazýváme polygony), konvexní a nekonvexní. Konvexní polygon je takový polygon, jehož všechny diagonály leží uvnitř polygonu a žádná z nich neprotíná jeho hranu. Konkávní polygon tuto podmínku nesplňuje. Pro představu je níže přiložen obrázek obou druhů polygonů.



Obrázek 1: Ukázka konvexního (vlevo) a konkávního polygonu (vpravo) (zdroj)

Z výše uvedeného vyplývá, že bod q může vůči polygonu P zaujímat 4 pozice:

1. Bod q se nalézá uvnitř polygonu P .
2. Bod q se nalézá vně polygonu P .
3. Bod q se nalézá na hraně polygonu P .
4. Bod q se nalézá ve vrcholu polygonu P .

Pro účely této aplikace byly zvoleny výpočetní algoritmy *Ray Crossing* a *Winding Number*, jejichž princip je vysvětlen v následující kapitole.

3 Algoritmy

Tato kapitola se zabývá popisem algoritmů, které byly v aplikaci implementovány.

3.1 Ray Crossing Algorithm

Prvním zvoleným algoritmem je tzv. *Ray Crossing Algorithm* neboli *Paprskový algoritmus*. Svůj název získal po metodě, jež využívá pro nalezení řešení polohy bodu vůči polygonu. Tento algoritmus je primárně využíván pro konvexní polygony, lze ho však zobecnit a využít ho i pro nekonvexní polygony.

Mějme polygon P a daný bod q . Z bodu q ved'me libovolný počet polopřímek (paprsků). Princip algoritmu je založen na vyhodnocení počtu průsečíků k , které vzniknou protnutím paprsků vedených z bodu q s hranami polygonu P . Pro k mohou nastat dvě situace:

1. Hodnota k je rovna lichému číslu \rightarrow bod q se nachází uvnitř polygonu P .
2. Hodnota k je rovna sudému číslu \rightarrow bod q se nachází vně polygonu P .

Základní varianta algoritmu neošetřuje problémové situace, které mohou během výpočtu nastat. Konkrétně se jedná o situace, kdy je bod q totožný s jedním z vrcholů polygonu P nebo pokud bod q leží na jedné z hran polygonu P . Pro eliminaci těchto tzv. singularit je vhodné použít modifikovanou variantu algoritmu, která redukuje souřadnice bodů polygonu k bodu q . OŠETŘENÍ SINGULARITY

Zjednodušený zápis takto modifikovaného algoritmu lze zapsat způsobem uvedeným níže:

1. Načtení bodů polygonu p_i , počet průsečíků $k = 0$
2. Postupně pro všechny p_i opakuj kroky I–IV:
 - I. Redukce souřadnic bodu p_i k bodu q :
$$x'_i = x_i - x_q$$

$$y'_i = y_i - y_q$$
 - II. Podmínka $(y'_i > 0) \& \& (y'_{i-1} \leq 0) \parallel (y'_{i-1} > 0) \& \& (y'_i \leq 0)$
 - III. Je-li podmínka splněna: $x'_m = \frac{x'_i y'_{i-1} - x'_{i-1} y'_i}{y'_i - y'_{i-1}}$
 - IV. Splněno $(x'_m > 0) \rightarrow k = k + 1$
3. Výpočet $k \% 2$
4. Vyhodnocení k (liché $k \rightarrow q$ náleží P ; sudé $k \rightarrow q$ nenáleží P)

3.2 Winding Number Algorithm

Druhý algoritmus použitý v aplikaci je tzv. *Winding Number Algorithm* neboli *Metoda ovýjení*, který je vhodný pro nekonvexní polygony. Princip tohoto algoritmu je založen na součtovém úhlu ω .

Mějme polygon P a bod q , na kterém stojí pozorovatel. Nachází-li se q uvnitř P , pak pozorovatel, který by si přál postupně vidět všechny vrcholy polygonu, se musí otočit celkem o 2π . Výsledkem algoritmu je pak tzv. Winding number ω , které říká, o kolik otáček se pozorovatel otočil:

$$\Omega = \frac{1}{2\pi} \sum_{i=1}^n \omega_i$$

Zde se hodí zdůraznit, že záleží na zvoleném směru otáčení. Otáčí-li se pozorovatel ve směru chodu hodinových ručiček, uhly se sčítají. V opačném směru se od sebe uhly odečítají a ω by vyšlo záporné. Během výpočtů je také nutno zavést určitou toleranci ϵ , která pokrývá chyby způsobené zaokrouhlováním. Z výše uvedených vztahů vyplývá:

1. $w = 2\pi \rightarrow q$ se nachází uvnitř P

2. $w < 2\pi \rightarrow q$ se nachází vně P

Mezi singularity pro tento algoritmus patří pouze případ, je-li $q \approx p_i$.

OŠETŘENÍ SINGULARITY

Zjednodušený zápis algoritmu:

1. Načtení bodů polygonu, úhel $\omega = 0$, tolerance $\epsilon = 1e - 10$
2. Postupně pro všechny orientované trojice p_i, q, p_{i+1} opakuj kroky I-III:
 - I. Výpočet úhlu $\omega_i = \angle p_i, q, p_{i+1}$
 - II. Podmínka (q je vlevo) $\rightarrow \omega = \omega + \omega_i$
 - III. Jinak $\omega = \omega - \omega_i$
3. Podmínka $(|\omega - 2\pi|) < \epsilon \rightarrow q$ náleží P
4. Jinak q nenáleží P

4 Problematické situace

5 Vstupní data

Aplikace požaduje dva druhy vstupních dat:

1. soubor daných polygonů
2. daný bod q

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

První řádek: počet polygonů v souboru

Sloupec 1: číslo polygonu, jehož součástí daný bod je

Sloupec 2: souřadnice X daného bodu polygonu

Sloupec 3: souřadnice Y daného bodu polygonu

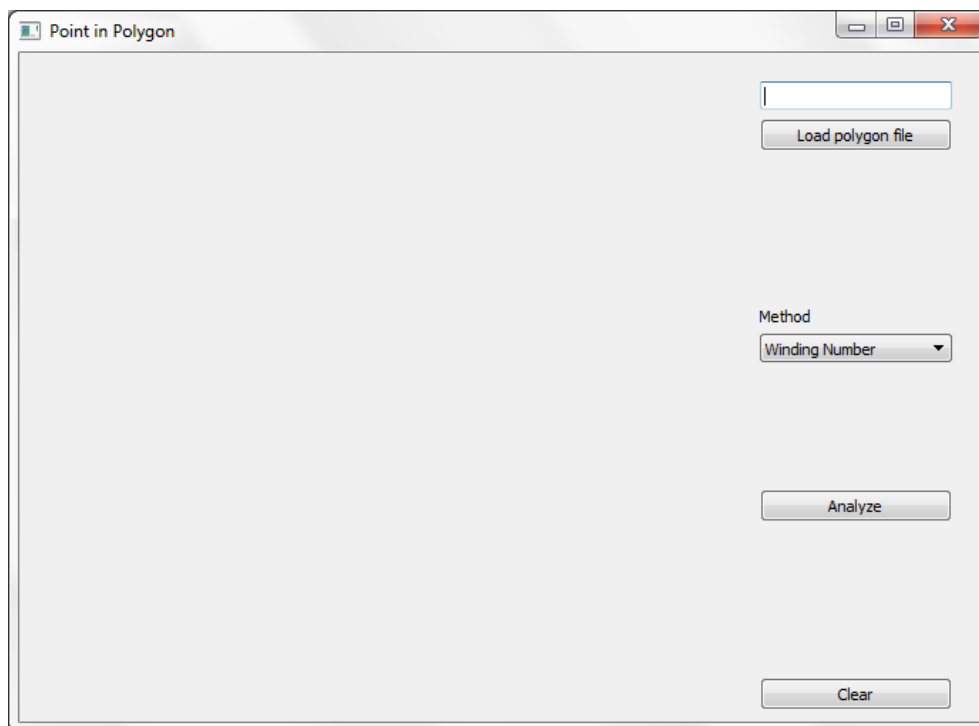
Bod q není součástí textového souboru, do aplikace vstupuje na základě ručního zadání uživatelem. Pro zadání bodu je nutné v aplikaci kliknout levým tlačítkem myši do okna s polygony.

6 Výstupní data

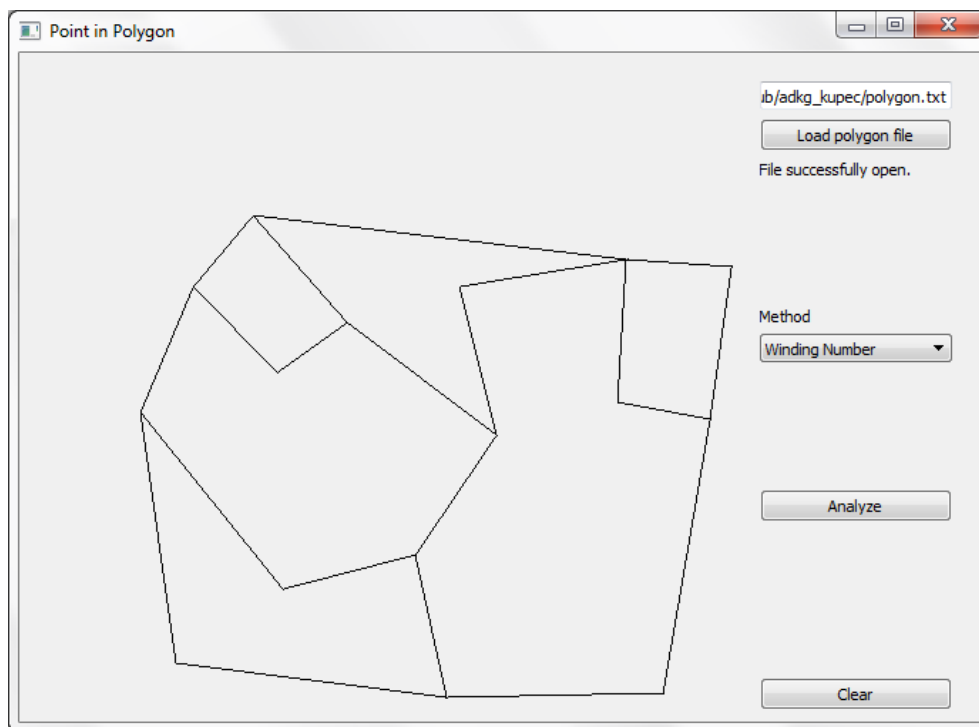
Výstupem úlohy je vypsání v grafickém okně aplikace, v jaké poloze se analyzovaný bod vůči polygonům nachází. Polygony, kterým bod náleží, jsou barevně zvýrazněny.

7 Aplikace

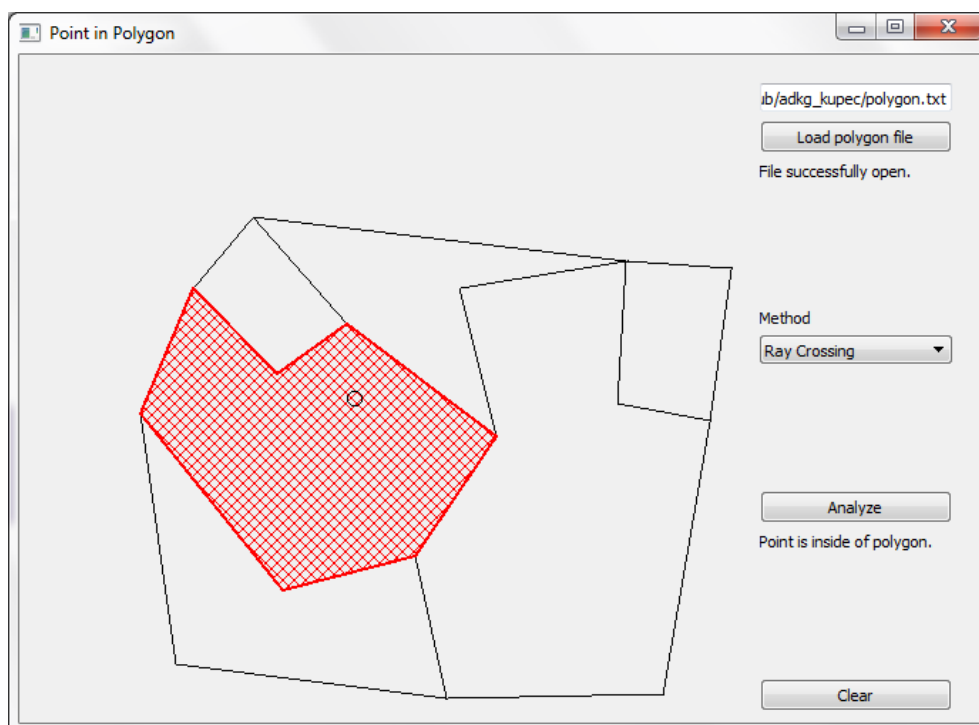
V následující kapitole je představen vizuální vzhled vytvořené aplikace tak, jak ji vidí prostý uživatel.



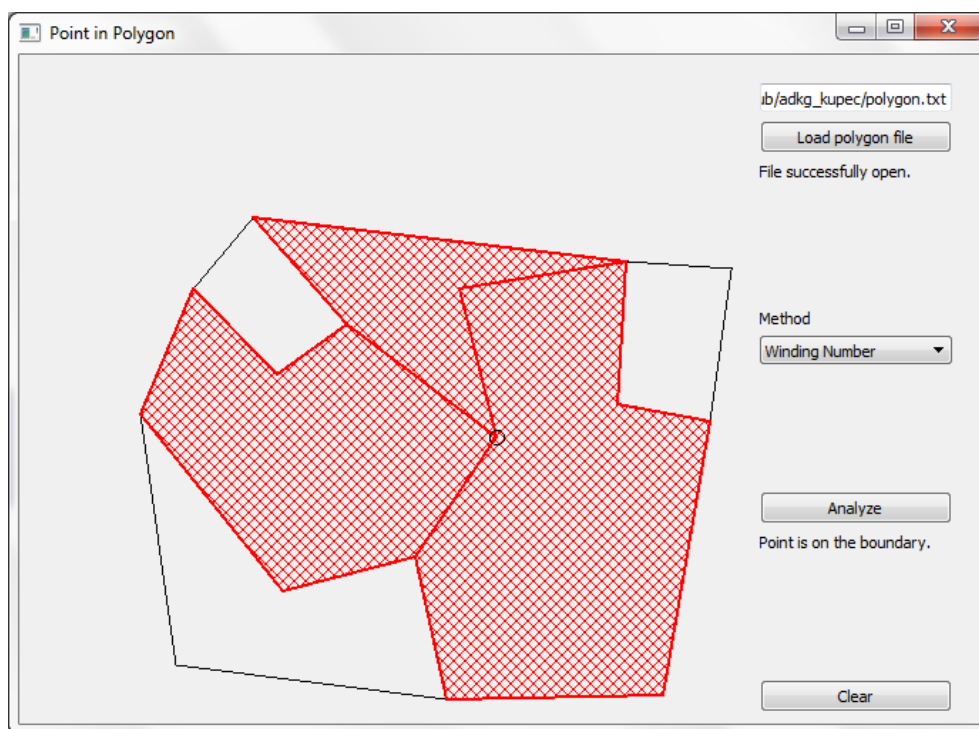
Obrázek 2: Aplikace ve výchozím stavu



Obrázek 3: Aplikace po nahrání polygonů



Obrázek 4: Výstup při použití *Ray Crossing Algorithm*



Obrázek 5: Výstup při použití *Winding Number Algorithm* pro q ležící ve více vrcholech polygonů

8 Dokumentace

Tato kapitola obsahuje dokumentaci k jednotlivým třídám.

8.1 Algorithms

Třída *Algorithms* obsahuje metody, které určují polohu zvoleného bodu q vzhledem k polygonu. Dále obsahuje pomocné metody pro výpočet úhlu mezi třemi body a na zjištění polohy bodu vzhledem k přímce.

8.1.1 getPositionRay

Metoda **getPositionRay** určuje polohu bodu q vzhledem k polygonu za použití algoritmu *Ray Crossing*. Na vstupu je bod třídy *QPoint* a vektor bodů polygonu. Návrátová hodnota je typu *int*.

Input:

- *QPoint* q
- *std :: vector<QPoint>* pol

Output:

- 0 \rightarrow bod se nachází vně polygonu
- 1 \rightarrow bod se nachází uvnitř nebo na hraně polygonu

8.1.2 getPositionWinding

Metoda **getPositionWinding** určuje polohu bodu q vzhledem k polygonu za použití algoritmu *Winding Number*. Na vstupu je bod třídy *QPoint* a vektor bodů polygonu. Návrátová hodnota je typu *int*.

Input:

- *QPoint* q
- *std :: vector<QPoint>* pol

Output:

- 0 \rightarrow bod se nachází vně polygonu
- 1 \rightarrow bod se nachází uvnitř polygonu
- -1 \rightarrow bod se nachází na hraně polygonu

8.1.3 `getPointLinePosition`

Metoda **`getPointLinePosition`** určuje polohu bodu q vzhledem k přímce tvořené dvěma body. Na vstupu jsou souřadnice x a y pro 3 body typu *double*, návratová hodnota je typu *int* podle výsledku.

Input:

- *double* x_q
- *double* y_q
- *double* x_1
- *double* y_1
- *double* x_2
- *double* y_2

Output:

- 0 \rightarrow bod se nachází vpravo od přímky
- 1 \rightarrow bod se nachází vlevo od přímky
- -1 \rightarrow bod se nachází na přímce

8.1.4 `getTwoVectorsAngle`

Metoda **`getTwoVectorsAngle`** počítá úhel mezi dvěma vektory. Na vstupu jsou souřadnice x a y pro 4 body typu *double*, návratová hodnota typu *double* vrací velikost úhlu v radiánech.

Input:

- *double* x_1
- *double* y_1
- *double* x_2
- *double* y_2
- *double* x_3
- *double* y_3
- *double* x_4
- *double* y_4

Output:

- *double*

8.2 Draw

Třída *Draw* obsahuje metody, které slouží k vykreslení polygonů, analyzovaného bodu *q* a výstupních dat.

8.2.1 paintEvent

Metoda **paintEvent** vykresluje polygony a analyzovaný bod *q*. Návrátová hodnota je typu *void*.

Input:

- QPaintEvent *e

8.2.2 mousePressEvent

Metoda **mousePressEvent** slouží k načtení souřadnic bodu *q*. Návrátová hodnota je typu *void*.

Input:

- QMouseEvent *e

8.2.3 clearCanvas

Metoda **clearCanvas** slouží k vymazání všech vykreslených dat. Metoda neobsahuje žádné proměnné na vstupu a návratová hodnota je typu *void*.

8.2.4 fillPolygon

Metoda **fillPolygon** barevně vyšrafuje polygon, ve kterém se nachází analyzovaný bod *q*. Návrátová hodnota je typu *void*.

Input:

- *std :: vector<std :: vector<QPoint>>* poly_fill

8.2.5 loadPolygon

Metoda **loadPolygon** slouží k nahrání bodů jednotlivých polygonů do aplikace. Součástí metody je i kontrola, zda se soubor úspěšně nahrál, zda vůbec obsahuje nějaké polygony a zda jsou polygony tvořeny aspoň 3 body. Návrátová hodnota je typu *QString* vrací hlášku, zda byly polygony úspěšně nahrány.

Input:

- const char* path

Output:

- QString

8.2.6 `getPoint????`

Metoda `getPoint`

Input:

-

Output:

-

8.3 `Widget`

Metody třídy **`Widget`** slouží pro práci uživatele s aplikací. Až na jednu výjimku nemá žádná metoda nic na vstupu, návratové hodnoty všech metod jsou typu *void*.

8.3.1 `writeResult`

Metoda **`writeResult`** vrací polohu bodu q vzhledem k polygonu na základě vstupní hodnoty typu *int*.

Input:

- `int`

8.3.2 `on_load_button_clicked`

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

8.3.3 `on_analyze_button_clicked`

Metoda **`on_analyze_button_clicked`** vypisuje na obrazovku polohu bodu q vzhledem k polygonům po zvolení požadovaného algoritmu a kliknutí na tlačítko *Analyze*.

8.3.4 `on_clear_button_clicked`

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

9 Závěr

V rámci úlohy *Geometrické vyhledávání bodu* byla vytvořena aplikace, která určuje polohu analyzovaného bodu q vzhledem k polygonu. V důsledku časové zaneprázdněnosti obou autorek nebyly implementovány všechny bonusové úlohy a některé části kódu mohly být řešeny lépe.

Jedná se například o použití tříd `QPoint` a `QPolygon`, které na vstupu mají hodnoty typu *integer* a které mohly být nahrazeny třídami `QPointF` a `QPolygonF`, jež na vstupu mají hodnoty typu *float*, což je pro práci se souřadnicemi bodů praktičtější. Dále jako ne zrovna nejšťastnější řešení hodnotíme množství vstupních hodnot, které mají na vstupu metody *getPointLinePosition* a *getTwoVectorsAngle* třídy **Algorithms**. Vhodné by nahradit jednotlivé souřadnice třídou `QPoint`, resp. `QPointF`.

Úloha přinesla i pozitivní přínos v tom směru, že se autorky mohly potrénoval v psaní kódu v jazyce C++ a oprášit své znalosti psaní v prostředí LaTeX.

10 Zdroje

1. *BAYER, Tomáš. Geometrické vyhledávání bodů* [online][cit. 24. 10. 2018].
Dostupné z: <https://web.natur.cuni.cz>
2. *What is concave & convex polygon?* [online][cit. 23. 10. 2018].
Dostupné z: <https://www.nextgurukul.in>
3. *Record XY mouse coordinates* [online][cit. 23. 10. 2018].
Dostupné z: <https://mobilefish.com/>