Zadání

Vstup: Souvislá polygonová mapa n polygonů {P1, ..., Pn}, analyzovaný bod q.

Výstup: Pi , q ∈ Pi.

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.

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

Čas zpracování: 2 týdny.

Popis a rozbor problému

Zjistit, zda bod leží nebo neleží uvnitř polygonu, je problém, který řeší tato úloha. S tímto problémem je možné se setkat v momentě zjišťování orientace určitého bodu vůči ostatním plošným objektům.

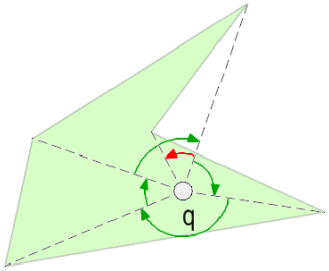
Tento problém je například řešen v GIS při označení (selectu) určitého polygonu kurzorem. Je nutné rozpoznat, v jakém polygonu se kurzor nachází, aby bylo možné označit správný polygon. Popřípadě jestli se nachází kurzor na hranici více polygonů, dochází k označení všech těchto polygonů.

Pro řešení tohoto problému je nutné aplikovat rychlý algoritmus, jelikož dnešní datasety obshaují často až miliony polygonů.

Princip řešení spočívá v opakovaném určování polohy hledaného bodu q vzhledem k polygonu. Tento problém je možné řešit několika algoritmy. Jednoduší *Ray Crossing Algorithm* nebo *Winding Algorithm* a složitější *Slabs Method* nebo *Metoda trapezoidálních map*, která využívá binární stromy.

Popisy algoritmů formálním jazykem

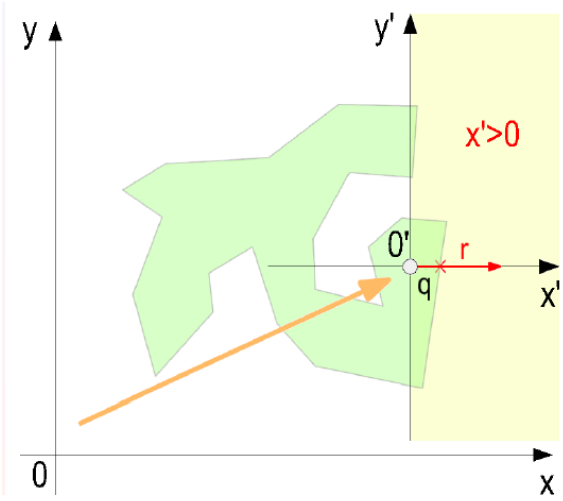
- **Winding Algorithm**
    - Princip algoritmu spočívá v opakovaném počítání úhlu mezi zjišťovaným bodem a dvěma po sobě jdoucími body polygonu. Tyto úhly se sčítají a pokud se výsledný úhel rovná 2π neboli otočení o 360°, tak se daný bod nachází uvnitř polygonu. Princip vychází z úvahy, že pokud se nacházím v místnosti a chci se podívat do každého jejího rohu, musím se otočit o 360°. U tohoto algoritmu je důležité zjišťovat vzájemnou polohu mezi spojnicí hledaného bodu (q) a bodu polygonu (p_i) a dalšího bodu polygonu (p_{i+1}). Zjištění polohy bodu p_{i+1} je klíčové, aby bylo možné zjistit, jestli úhel přičíst k výsledné sumě nebo odečíst. Viz. obrázek číslo 1.



Obr. 1 znázornění otáčení při winding algoritmu

Zdroj: (Bayer 2018)

- **Ray Crossing Algorithm**
    - Princip tohoto algoritmu spočívá v určování počtu průsečíků. Daným bodem (q) vedu horizontální přímku, která rozdělí polygon na horní a dolní část a počítám počet průsečíků hranic polygonu s přímkou. Pokud je počet průsečíků sudý, bod se nachází vně polygonu, pokud je lichý, nachází se uvnitř. Problém nastává, když přímka prochází vrcholem nebo vede přes kolineární hrany. Důsledkem toho, se počet průsečíků liší od skutečného počtu. Tento problém je řešen následovně. Algoritmus pracuje s průsečíky pouze v prvním kvadrantu souřadnicového systému, kde bod q je jeho středem se souřadnicemi (0,0). Průsečíky jsou započítávány pouze tehdy, pokud linie, která protíná přímku, začíná nebo končí na přímce a vede na horní část nebo linie vede přes přímku (tzn. linie začíná v horní a končí v dolní části a naopak). Viz. obrázek číslo 2.



Obr. 2 ukázka algoritmu Ray Crossing

Zdroj: (Bayer 2018)

Problematické situace a jejich rozbor

Jedním z problémů je již zmíněná situace s vrcholy a kolineárními hranami v algoritmu Ray Crossing viz. *Popis algoritmů formálním jazykem.*

Další problematická situace u algoritmu Ray Crossing nastává při situaci, kdy je bod na hraně polygonu. Tato situace se dá řešit několika způsoby. V mojí aplikaci je tento problém řešen pomocí determinantu, kdy znaménko determinantu určuje, zda bod leží vpravo či vlevo od úsečky. Pokud je determinant 0, bod leží na přímce. Teoreticky je toto řešení jednoduché a účinné, ale v praxi se bohužel setkává s problémem. Pokud získávám souřadnice kliknutím myši, tak dostávám celočíselné souřadnice. Při výpočtu determinantu tedy v mnoha případech 0 nikdy nevyjde. Řešení tohoto problému nacházím v zadávání x a y souřadnic přes

klávesnici s desetinnými místy. Takto je možné se determinantu 0 alespoň přiblížit. Stále to ale pro většinu případů nestačí a je nutné výsledný determinant porovnávat s určitým intervalem namísto 0. Jelikož hodnoty determinantu vycházejí často v řádech tisíců, je možné zvolit interval například od 50 do -50. Vybrat správné rozpětí intervalu je klíčové, jelikož při velkém intervalu může docházet v určitých případech k vyhodnocení, že bod se nachází na hraně, ale ve skutečnosti je viditelně mimo.

Problematika winding algoritmu byla nastíněna již v popisu tohoto algoritmu. Jedná se o opakované zjišťování polohy bodů viz. *Popis algoritmů formálním jazykem*. Tato problematika se dá elegantně řešit výpočtem již zmíněného determinantu.

Vstupní data, formát vstupních dat, popis

Vstupními daty musí být alespoň jeden polygon a bod, jehož poloha bude určována vzhledem k polygonu. Aplikace nabízí možnost generování náhodných polygonů, použití předdefinovaných polygonů a nahrání vlastního textového souboru s polygony. Počet bodů generovaných polygonů je vybrán náhodně. Polygony mají nejméně čtyři body a nejvíce třicet bodů. Předdefinované polygony jsou dva polygony, které mají společnou hranu. Nahrávání polygonů z textového souboru má následovný formát. Na prvním řádku textového souboru musí být napsaný celkový počet polygonů. Na druhém řádku souřadnice x bodů prvního polygonu oddělené čárkou a na řádku třetím y souřadnice prvního polygonu oddělené čárkou. Na dalších řádcích se zápis 2. a 3. řádku opakuje pro další polygon. První souřadnice x a první souřadnice y reprezentuje první bod polygonu. Souřadnice musí být seřazeny tak, aby byl výsledný polygon korektní. Viz. soubor test.txt.

Bod je možné vytvořit kliknutím kurzorem myši nebo zapsáním x a y souřadnice do připravených okének a následně bod vytvořit tlačítkem. Při vytváření bodu pomocí myši, jsou souřadnice bodu vždy celočíselné, zatímco do okének je možné klávesnicí zapsat souřadnice i s desetinnými místy.

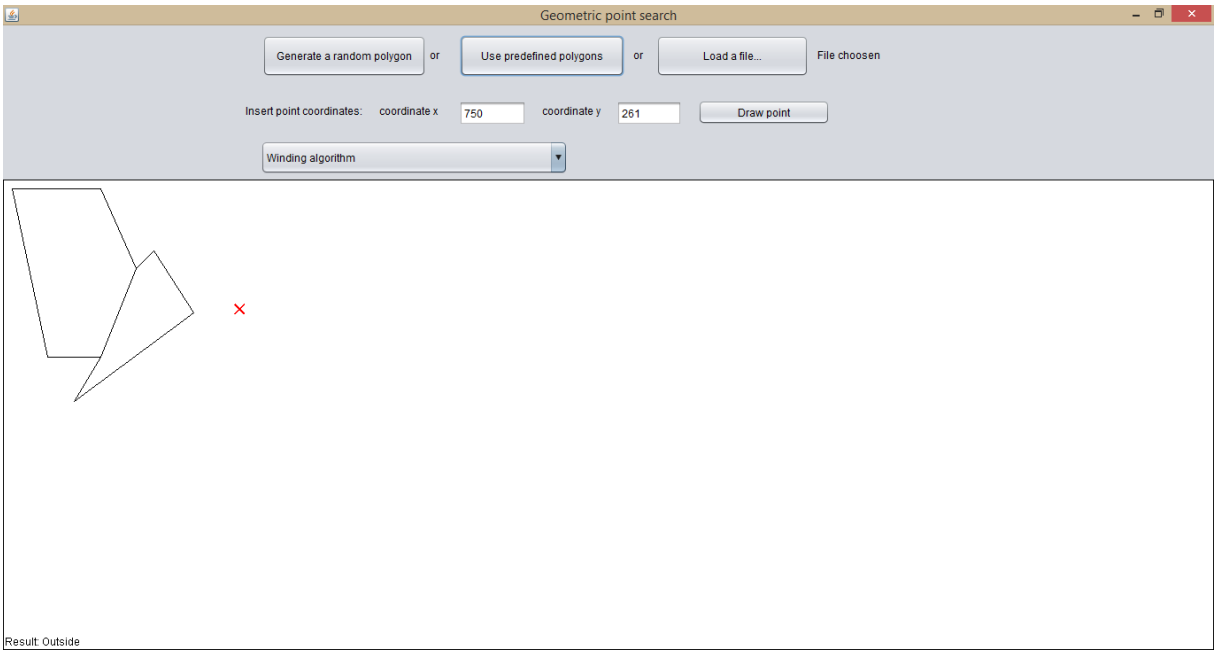
Výstupní data, formát výstupních dat, popis

Výstupní data mají pouze grafickou podobu. Jedná se o barevné znázornění výsledku výše popsaných algoritmů. Pokud se bod nachází uvnitř polygonu, tak je polygon vybarven zeleně.

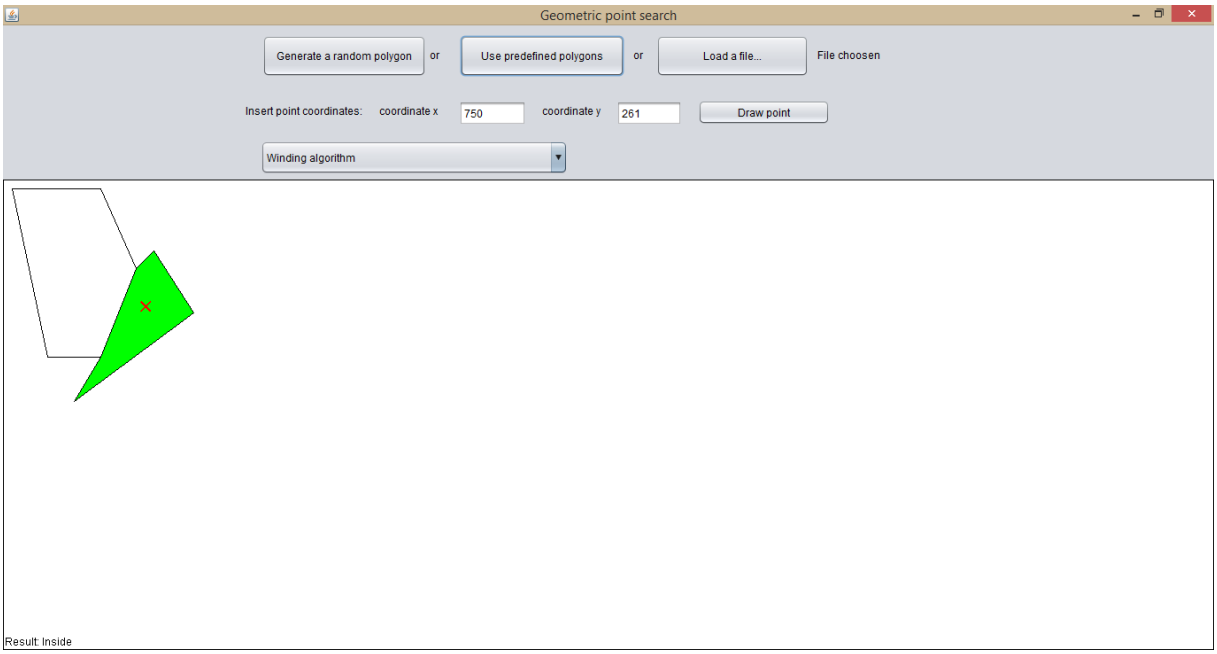
Pokud se bod nachází na hraně nebo je bod identický s vrcholem polygonu, zvýrazní se zeleně obrys polygonu. Bod je vždy reprezentován jako červený křížek.

Printscreen vytvořené aplikace

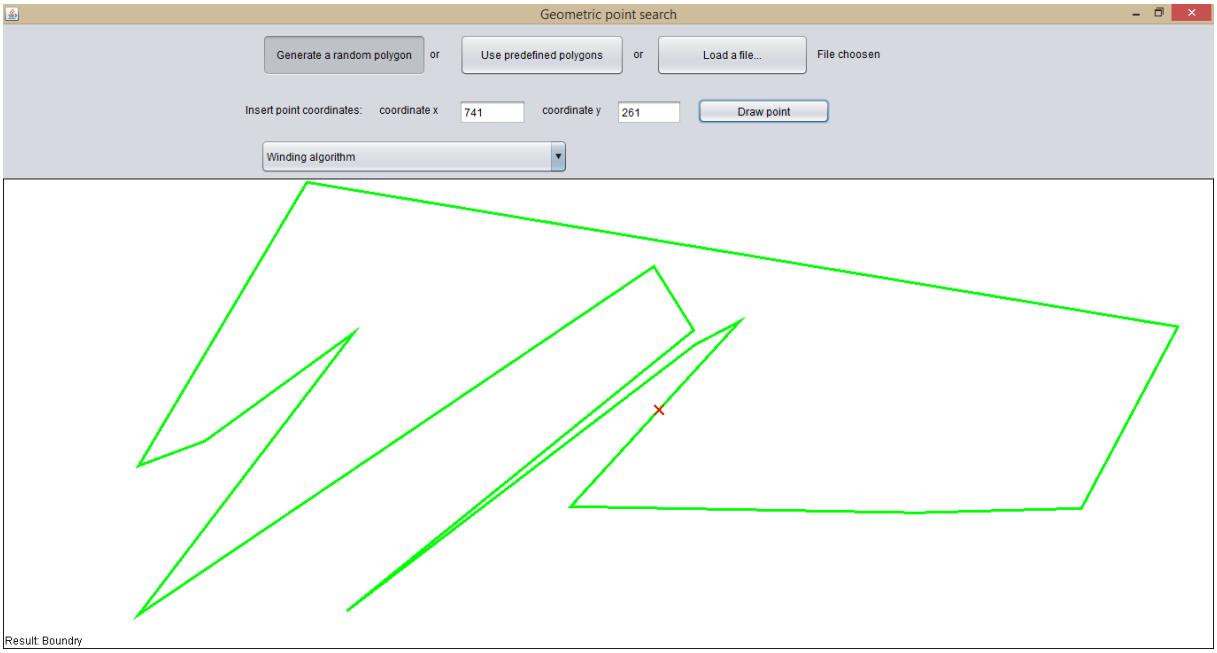
Na obrázku číslo 3 je možné vidět grafické rozhraní aplikace a předdefinované polygony s bodem umístěným vně. Obrázek číslo 4 znázorňuje stejnou situaci, jen bod se nachází uvnitř polygonu. Na obrázku číslo 5 je vidět náhodně vygenerovaný polygon s bodem na hraně vytvořeným pomocí klávesnice. Obrázek číslo 6 znázorňuje nahrané polygony z textového souboru a bod identický s vrcholem, který sdílejí všechny tři polygony.



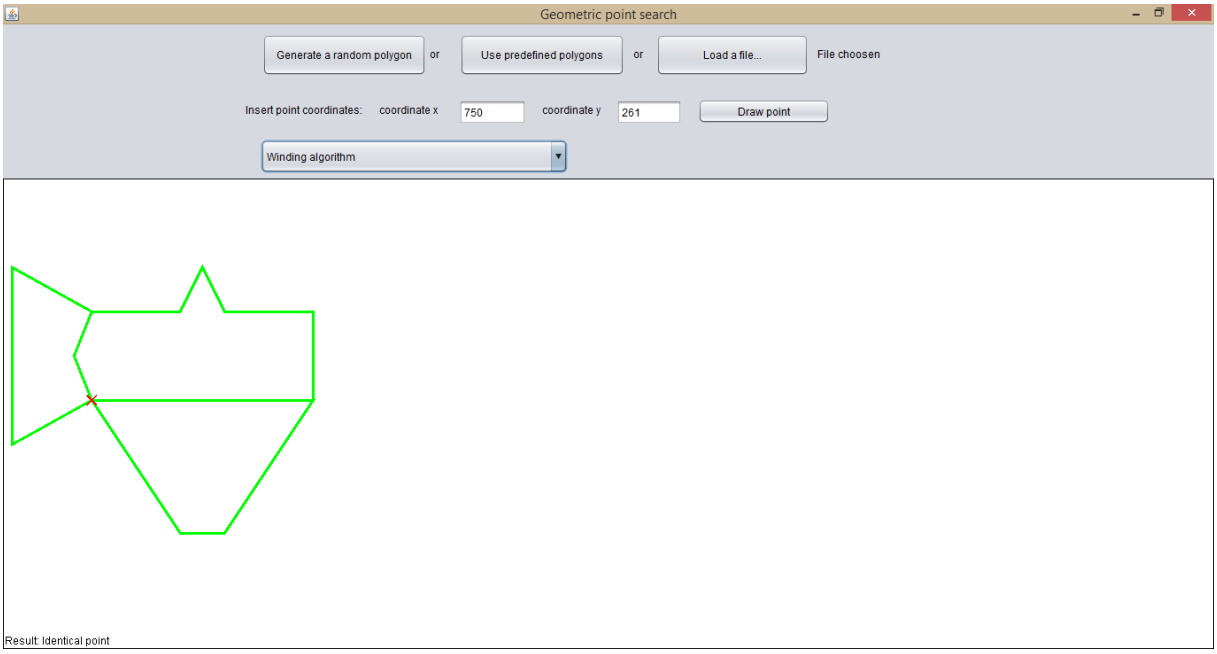
Obr. 3 bod vně



Obr. 4 bod uvnitř



Obr. 5 bod na hraně



Obr. 6 bod identický s vrcholem

Dokumentaci: popis tříd, datových položek a jednotlivých metod

Aplikace se celkově skládá z pěti tříd. Třída *Algorithm*, která reprezentuje veškeré algoritmy a výpočty, které se v aplikaci dějí. Třída má jedinou proměnnou, která je datového typu enum a reprezentuje výsledek algoritmů, které určují polohu bodu vůči polygonu. Třída má celkem

deset metod, ale některé reprezentují pouze krátké výpočty, jako metoda *dotProd*, která počítá skalární součin nebo metoda angle, která počítá úhel mezi vektory. Hlavními metodami této třídy jsou metody *windingalg, rayCrossingAlg* a *deter. Windingalg a rayCrossingAlg* reprezentující stejnojmenné algoritmy a metoda *deter, která* počítá determinant, který se používá pro zjištění orientace mezi bodem a přímkou. Dalšími důležitými metodami jsou metody *polygonmaker a intersect*, které vytváří polygon z náhodně vygenerovaných bodů. Metoda *polygonmaker* volá v cyklu metodu intersect, která zkoumá, zda se dané dvě linie polygonu kříží, či nikoliv. Pokud se žádná linie nekříží s žádnou další linií, cyklus končí a z bodů je možné vytvořit polygon. Pokud se najde linie, která se kříží s jinou linií, listy souřadnic bodů jsou náhodně promíchány a proces se opakuje.
Druhá třída je třída *DrawPanel*, která se stará o veškeré vykreslování na plátno. Tato třída má několik proměnných, například má dvě proměnné datového typu ArrayList reprezentující souřadnice x a y polygonu, dále má proměnnou datového typu Polygon [], která reprezentuje pole polygonů apod. Veškeré proměnné je možné najít ve zdrojovém kódu. V této třídě se nachází opět několik metod. Nejdůležitější metoda této třídy je předdefinovaná metoda *paintComponent,* která vykresluje jednotlivé polygony a bod. Dále se stará o vizualizaci výsledků zvoleného algoritmu. Další důležitou metodou je metoda *fromMouseClicked*, která při kliknutí myši nad plátnem naplní proměnnou reprezentující bod souřadnicemi x a y kurzoru myši a spustí nové vlákno. Vlákno, které reprezentuje třetí třída *Worker* rozšiřující třídu *SwingWorker.* V tomto vláknu se volá metoda ve třídě *Algortihm* a dochází k provedení veškerých výpočtů. Jakmile výpočty skončí, zavolá se předdefinovaná metoda třídy *DrawPanel paintComponent.*
Čtvrtá třída *GUI* reprezentuje grafické rozhraní aplikace. Většina proměnných v této třídě reprezentuje tlačítka, textové zprávy a textová pole v grafickém rozhraní. Kromě metody *main*, která spouští celé grafické rozhraní je v této třídě několik důležitých metod. Metoda *CBActionPerformed*, která nastaví algoritmus pro vyhodnocování vzájemné polohy bodu a polygonu. Tato metoda se spustí pokaždé, když se změní hodnota v comboboxu. Metoda *predefActionPerformed* se spustí po kliknutí na tlačítko "Use predefined polygons" a vytvoří dva předdefinované polygony a následně zavolá metodu DrawPanelu paintComponent. Metoda *ButtonActionPerformed* se spustí po kliknutí na tlačítko "Draw point" a na základě zadaných souřadnic v textových polích vytvoří bod. Následně spustí nové vlákno, které je popsané výše. Metody *polygongenActionPerformed* a *loadbutActionPerformed* se stejně jako

ostatní metody ActionPerformed spustí po kliknutí na příslušná tlačítka. Tyto metody nastaví proměnnou datového typu boolean a spustí nové vlákno, které reprezentuje pátá třída *GenerateWorker.* Toto vlákno provádí operace na základě nastavené proměnné datového typu boolean. Pokud se spustila metoda *polygongenActionPerformed,* ve vláknu dochází ke generování náhodných bodů a následně k vytvoření polygonu z těchto bodů. Pokud se spustila metoda *loadbutActionPerformed*, ve vláknu dochází k otevření textového souboru, čtení jednotlivých řádků a z načtených hodnot dochází k vytvoření polygonů. Po dokončení dané operace se ve vláknu zavolá metoda DrawPanelu paintComponent.

Závěr, možné či neřešené problémy, náměty na vylepšení

Jediným problémem, který jsem již zmiňoval je určování bodu na hraně polygonu. S tím jsem si víceméně poradil, ale domnívám se, že existuje elegantnější řešení, než na které jsem přišel já. Moje řešení spočívá v porovnávání determinantu mnou náhodně vybraným intervalem na místo s 0.

Vylepšit by se určitě dal algoritmus pro generování náhodných polygonů. Tento algoritmus jsem vytvořil vlastními silami, bez použití literatury a již vymyšlených algoritmů na internetu. Tudíž mi šlo především o funkčnost tohoto algoritmu, než o jeho rychlost.

Závěrem si myslím, že aplikace je povedená a splňuje zadání na 100 %. Zdrojový kód je anglicky okomentován a nahrán i se zbytkem souborů na Githubu.

Seznam literatury

BAYER, T. 2018. *Geometrické vyhledávání: Ray algoritmus. Winding algoritmus. Lichoběžníkové (trapezoidální) mapy* [elektronický zdroj]. Praha. Dostupné také z: https://web.natur.cuni.cz/~bayertom/index.php/teaching/algoritmy-pro-tvorbu-digit-map