

155ADKG: GEOMETRICKÉ VYHLEDÁVÁNÍ BODU

V Praze 23.10.2018

Michael Kala, Anna Zemánková

Oprava dne 3.11.2018

1. Zadání

Navrhněte aplikaci s grafickým rozhraním, která určí polohu bodu vůči souvislé polygonové mapě. (Toto zadání nevyžaduje bližší specifikaci).

2. Údaje o bonusových úlohách

V rámci této úlohy byly zpracovány následující bonusy:

- 1) Ošetření singulárního případu u Winding algorithm: bod leží na hraně polygonu.
- 2) Ošetření singulárního případu u obou algoritmů: bod je totožný s vrcholem jednoho či více polygonů.
- 3) Zvýraznění všech polygonů pro oba výše uvedené singulární případy.

3. Popis a rozbor

Mějme v rovině bod q a polygony P_j tvořené vrcholy p_i a hranami σ_{p_i} . Polygony tvoří souvislou mapu. Naším úkolem je určit polohu bodu q vůči polygonové mapě.

Problematika vzájemné polohy bodu a souvislé polygonové mapy byla převedena na problém vzájemné polohy bodu a jednoho polygonu – je postupně určována poloha bodu vůči jednotlivým polygonům mapy (je-li nalezen polygon uvnitř, kterého bod leží, je analýza ukončena) a teprve poté rozhodnuto o vzájemné poloze v rámci celé mapy.

Možné varianty výsledků:

- 1) Bod $q \in P_j \rightarrow$ bod je uvnitř polygonu
- 2) Bod $q \notin P_j \rightarrow$ bod vně polygonu
- 3) Bod $q \in \sigma_{p_i} \rightarrow$ bod leží na hraně polygonu/společné hraně více polygonů
- 4) Bod $q = p_i \rightarrow$ bod q je totožný s vrcholem polygonu/společným vrcholem polygonů

Pro svou jednoduchost byly pro určení polohy bodu vůči polygonu vybrány algoritmy Ray crossing algorithm a Winding algorithm.

4. Popis algoritmů

4.1 Ray Crossing algorithm

Mějme v rovině bod q a polygon P_j tvořený vrcholy p_i .

Bodem q jsou vedeny polopřímky libovolným směrem. Každá polopřímka n -krát protne hrany polygonu P_j . Jestliže je n :

- 1) Liché \rightarrow bod $q \in P_j$
- 2) Sudé \rightarrow Bod $q \notin P_j$

Je třeba ošetřit případy, kdy je přímka vedená bodem q kolineární s hranou polygonu nebo je bod q totožný s vrcholem polygonu/leží na hraně polygonu. Více v kapitole 5.

Implementace:

1. *Inicializace inters = 0*
2. *Opakuj : Pro $\forall p_i \in P$*
3. $x'_i = x_i - x_q$
4. $y'_i = y_i - y_q$
5. *if $(y'_i > 0) \&\& (y'_{i-1} \leq 0) \parallel (y'_{i-1} > 0) \&\& (y'_i \leq 0)$*
 - a. $x'_m = (x'_i y'_{i-1} - x'_{i-1} y'_i) / (y'_i - y'_{i-1})$
 - b. *if $(x'_m > 0)$ pak inters = inters + 1*
6. *if $(inters \% 2) \neq 0$ pak $q \in P$*
7. *else $q \notin P$*

4.2 Winding algorithm

Mějme v rovině bod q a polygon P_j tvořený vrcholy p_i .

Určíme úhly $\omega_i(p_i, q, p_{i+1})$.

$$\cos \omega_i = \frac{\vec{u}_i \cdot \vec{v}_i}{\|\vec{u}_i\| \cdot \|\vec{v}_i\|} ; \text{ kde } \vec{u}_i = (q, p_i), \vec{v}_i = (q, p_{i+1})$$

Pokud je úhel orientován proti směru hodinových ručiček, má kladné znaménko, pokud je orientován po směru hodinových ručiček, znaménko je záporné. Nakonec je určena suma všech úhlů.

Je-li $\sum \omega_i$ rovna:

- 1) 360° - Bod $q \in P_j$
- 2) 0° - Bod $q \notin P_j$
- 3) Jinak - Bod $q \in \sigma_{P_j}$ nebo Bod $q = p_i$

Implementace:

1. *Inicializace $\omega = 0$, tolerance ε*
2. *Opakuj pro \forall trojici p_i, q, p_{i+1} :*
3. *Urči orientaci o_i bodu q ke straně $\overline{p_i, p_{i+1}}$*
4. *Urči úhel $\omega_i = \angle p_i, q, p_{i+1}$*
5. *if q vlevo od $\overline{p_i, p_{i+1}}$, pak $\omega = \omega + \omega_i$*
6. *else $\omega = \omega - \omega_i$*
7. *if $(|\omega - 2\pi| < \varepsilon)$, pak $q \in P$*
8. *else $q \notin P$*

5. Problematické situace

Ray Crossing algorithm

Je-li polopřímka vedená bodem q kolineární s hranou polygonu, dochází k tomu, že počet průsečíků polopřímky s hranou polygonu je sudý, přestože je bod q uvnitř polygonu a naopak.

Aby ke kolizi nedocházelo, je a zaveden místní souřadnicový systém tak, že:

- 1) Počátek soustavy je vložen do bodu q
- 2) Osa x' je rovnoběžná s osou x
- 3) Osa y' je kolmá na osu x'

a řešení omezeno pouze na hrany polygonu, jejichž jeden bod leží "pod" osou x' a druhý "nad" osou x' .

Pro ošetření případu, kdy je bod q totožný s vrcholem polygonu nebo leží na hraně polygonu, je určena délka hrany polygonu a poté součet vzdáleností mezi bodem q a vrcholy hrany. Pokud jsou tyto vzdálenosti totožné, leží bod na hraně polygonu, nebo je totožný s vrcholem polygonu.

$$(q \in \sigma_{p_i}) \cup (q = p_i) \leftrightarrow ||\sigma_{p_i}|| = ||q, p_i|| + ||q, p_{i+1}||$$

Winding algorithm

Ošetření případu, kdy je bod q totožný s vrcholem polygonu nebo leží na hraně polygonu:

$$\sum \omega \neq 360^\circ \ \&\& \ \omega \neq 0^\circ \rightarrow q \in \sigma_{p_i}$$

Vstupní data

6.1 polygonová mapa

Polygonová mapa je importována formou textového souboru *.txt, který obsahuje vždy v tomto pořadí:

- 1) počet polygonů
- 2) počet bodů v prvním polygonu
- 3) souřadnice X a Y (s desetinnou tečkou, pokud se nejedná o celé číslo) bodů prvního polygonu
- 4) počet bodů v druhém polygonu, souřadnice X a Y bodů druhého polygonu atd.

Př.: Vstupní soubor se 2 polygony, kde první polygon obsahuje 3 body a druhý polygon 4 body bude vypadat následovně:

2

3 12.2 4.1 8.3 11.4 6.8 7.9

4 1.1 2.2 3.3 4.4 5.5 6.6 7 8

Vše může být zapsáno do jednoho řádku, odděleno mezerami nebo jako v příkladu výše může každý útvar začínat na novém řádku.

6.2 Bod q

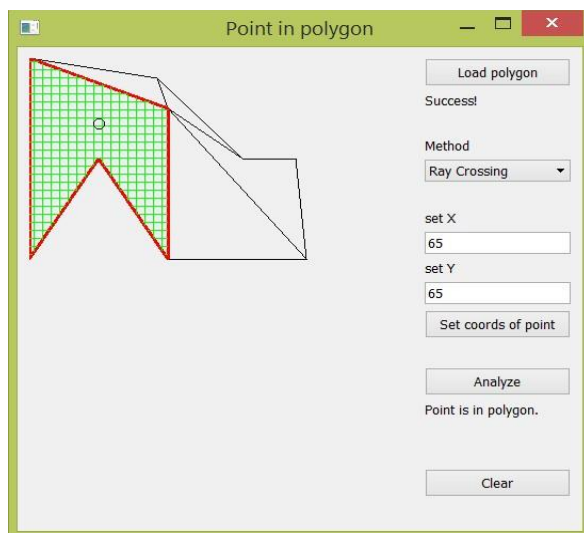
Souřadnice bodu, jehož polohu vůči polygonům chceme zjistit, do aplikace vstupují interaktivně – zadáním souřadnic (s desetinnou tečkou, pokud se nejedná o celé číslo) nebo kliknutím myši do kanvasu.

6. Výstupní data

V grafickém rozhraní aplikace se po dokončení výpočtu vypíše výsledek:

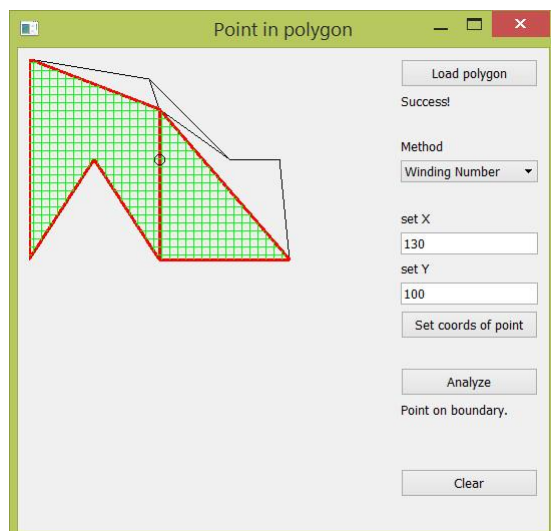
Point in polygon / Point on boundary / Point is out a dojde k vybarvení dotčených polygonů.

Pokud je výsledek “*Point in polygon*”, vybarví se polygon, uvnitř kterého bod leží.

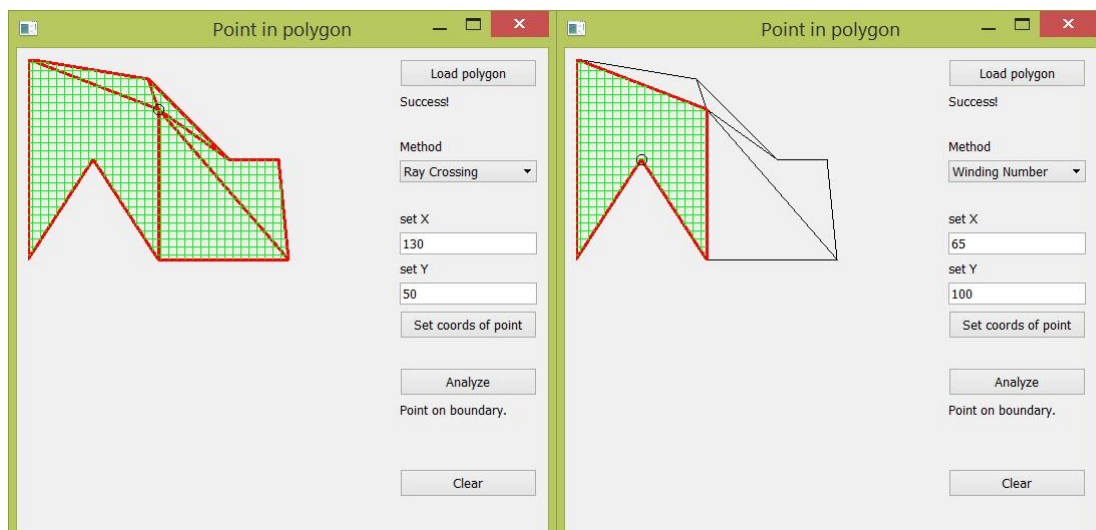


Obr. 1 Point in polygon

V případě, že je výsledek “*Point on boundary*” vybarví se ty polygony, na jejichž společné hraně bod leží nebo v případě, že je bod q zároveň vrcholovým bodem jednoho či více polygonů, vybarví se ty polygony, na jejichž hranici se vrcholový bod nachází.

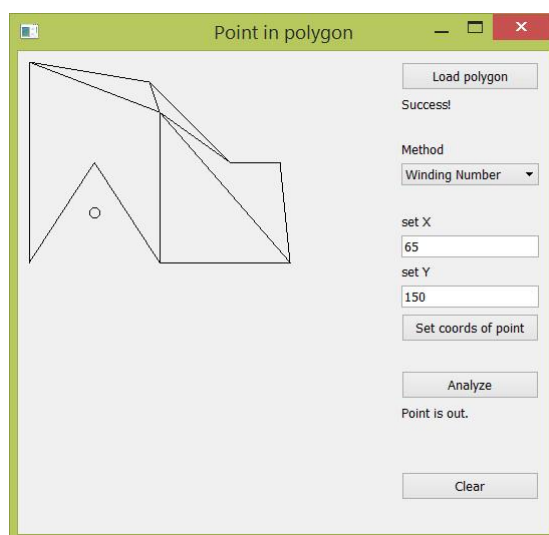


Obr. 2 Point on boundary – společná hrana



Obr. 3, 4 Point on boundary – vrcholový bod

Pokud je výsledek “Point is out” nevybarví se žádný polygon.

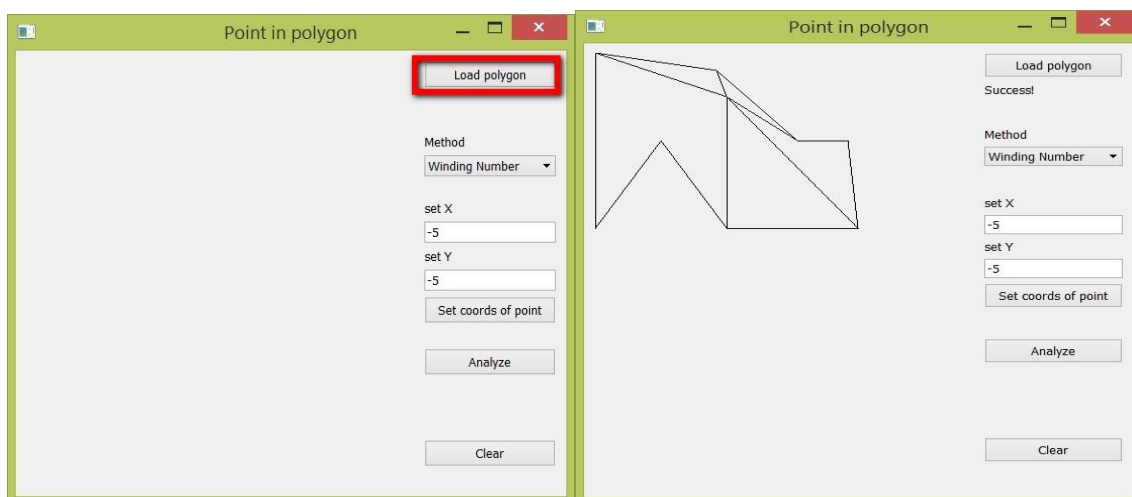


Obr. 5 Point is out

7. Ukázka vytvořené aplikace

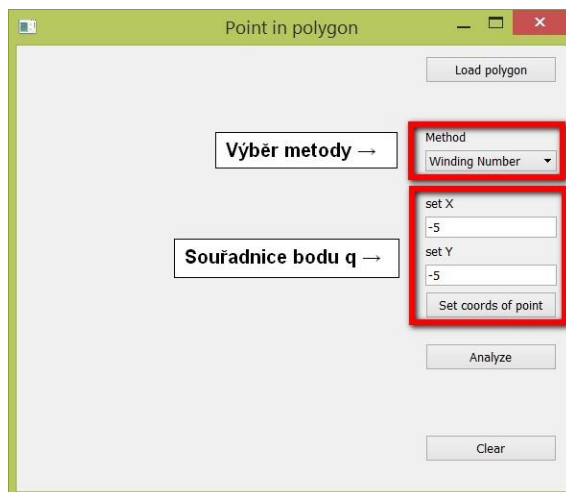
Načtení polygonů

Soubor se souřadnicemi polygonů je do aplikace importován stisknutím tlačítka *“Load polygon”* a výběrem souboru. Jestliže vše proběhlo v pořádku, vypíše se pod toto tlačítko zpráva *“Success!”*



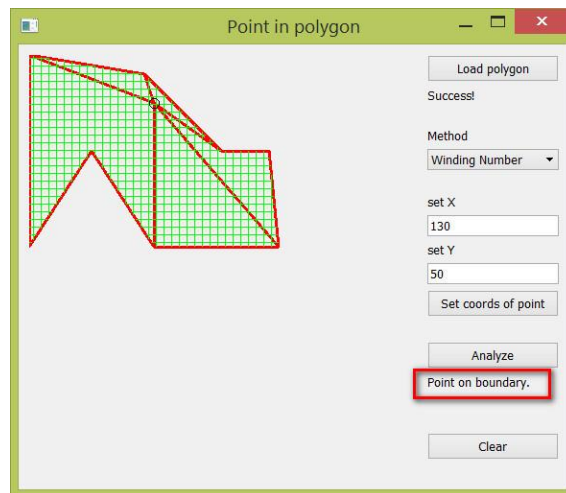
Obr. 6, 7 – Načtení polygonů

do políček *“Set” X* a *“Set Y”* a následným kliknutím na tlačítko *“Set coords of point”*. Poté se zobrazí bod reprezentovaný kružnicí. Následně je třeba ještě vybrat v rozbalovacím menu metodu.



Obr. 8 – Vložení souřadnic bodu q a výběr metody

Následným stisknutím tlačítka *Analyze* je proveden výpočet a zobrazeny výsledky.



Obr. 9 – Výstupy

8. Dokumentace

Třídy

1) Algorithms

- Ve třídě Algorithms jsou staticky implementovány výpočetní algoritmy pro určení polohy bodu vůči polygonu a jejich pomocné metody.
- Metoda **getPositionRay**
 - Tato metoda slouží k určení polohy bodu vůči polygonu pomocí Ray crossing algoritmu. V metodě je navíc implementováno určení, zda-li se bod nachází na hraně polygonu případně v některém z vrcholů. Její návratový typ je **integer**.
 - Vstup
 - **QPointF q** – bod, jehož poloha vůči polygonu se určuje
 - **std::vector<QPointF> pol** – polygon, vůči němuž se určuje poloha bodu q
 - Výstup
 - **-1** – bod se nachází na hranici nebo ve vrcholu daného polygonu
 - **0** – bod se nachází mimo daný polygon
 - **1** – bod se nachází uvnitř daného polygonu
- Metoda **getPositionWinding**
 - Tato metoda slouží k určení polohy bodu vůči polygonu pomocí Winding algoritmu. Její návratový typ je **integer**.
 - Vstup
 - **QPointF q** – bod, jehož poloha vůči polygonu se určuje
 - **std::vector<QPointF> pol** – polygon, vůči němuž se určuje poloha bodu q
 - Výstup
 - **-1** – bod se nachází na hranici nebo ve vrcholu daného polygonu
 - **0** – bod se nachází mimo daný polygon
 - **1** – bod se nachází uvnitř daného polygonu
- Metoda **getPointLinePosition**
 - Tato metoda je pomocnou pro metodu **getPositionWinding**. Slouží k určení polohy bodu vůči přímce. Návratovou hodnotou je **integer**.

- Vstup
 - **double x, y** – souřadnice určovaného bodu
 - **double x1, y1, x2, y2** – souřadnice bodů přímky
- Výstup
 - **-1** – bod se nachází na přímce
 - **0** – bod se nachází vpravo od přímky
 - **1** – bod se nachází vlevo od přímky
- Metoda **getTwoVectorsAngle**
 - Tato metoda je druhou pomocnou pro metodu **getPositionWinding**. Slouží k určení úhlu mezi 2 přímkami. Její návratovou hodnotou je **double**.
 - Vstup
 - **double x1, y1, x2, y2** – body určující první přímku
 - **double x3, y3, x4, y4** – body určující druhou přímku
 - Výstup
 - úhel mezi 2 přímkami

2) Draw

- Třída **draw** slouží k vykreslení zjišťovaného bodu a nahaných polygonů, dále k výplni polygonů, jichž se bod dotýká resp. polygonu, ve kterém bod leží. V konstruktoru třídy jsou nastaveny souřadnice zjišťovaného bodu **q** na výchozí hodnoty (-5,-5). Třída dědí od třídy **QWidget**.
- Členské proměnné
 - **QPointF q** – souřadnice zjišťovaného bodu, výchozí hodnoty jsou nastaveny v konstruktoru, dále se hodnoty mění buď stisknutím tlačítka myši nad vykreslovacím plátnem nebo přímým vstupem z klávesnice
 - **std::vector<std::vector<QPointF>> poly_pol** – vektor, ve kterém jsou uloženy vektory obsahující body jednotlivých polygonů
 - **std::vector<int> analysis_results** – vektor, ve kterém jsou uloženy výsledky analýzy polohy bodu vůči jednotlivým polygonům, pozice v tomto vektoru odpovídá pozici polygonů ve vektoru **poly_pol**
- Metoda **paintEvent**
 - Tato metoda slouží k vykreslení zjišťovaného bodu, nahaných polygonů a k vybarvení polygonů, jichž se bod dotýká resp. polygonu, ve kterém bod leží v závislosti na datech ve vektoru **analysis_results**. Volá se při zavolání **repaint()**, jež je členskou metodou třídy **QWidget**. Návratovým typem je **void**.
 - Vstup
 - **QPaintEvent *e**
- Metoda **mousePressEvent**
 - Tato metoda reaguje na stisknutí tlačítka myši nad vykreslovacím plátnem. Volá metodu **setPointCoords**. Návratovým typem je **void**.
 - Vstup
 - **QMouseEvent *e**
- Metoda **setPointCoords**

- Tato metoda slouží ke změně souřadnic bodu (členské proměnné) **q** a k překreslení. Jejím návratovým typem je **void**.
- Vstup
 - **double x, y** – nové souřadnice bodu **q**
- Metoda **clearCanvas**
 - Tato metoda slouží k vymazání obsahu vykreslovacího plátna, tedy k vymazání obsahu členské proměnné **poly_pol**, **analysis_results**, k nastavení souřadnic bodu **q** na výchozí hodnoty a k překreslení. Metoda nemá žádné vstupní hodnoty, jejím návratovým typem je **void**.
- Metoda **loadPolygon**
 - Tato metoda slouží k načtení polygonů z textového souboru a k jejich uložení do seznamu polygonů **poly_pol**. Metoda kontroluje správnost vstupního souboru. Jejím návratovým typem je **QString**.
 - Vstup
 - **std::string path** – cesta k souboru
 - Výstup
 - **QString msg** – zpráva obsahující informaci o správnosti načtení souboru
- Metoda **getQ**
 - Tato metoda slouží k získání souřadnic bodu **q**. Metoda nemá žádnou vstupní hodnotu, jejím návratovým typem je **QPointF**.
 - Výstup
 - **QPointF q** – souřadnice bodu **q**
- Metoda **getPol**
 - Tato metoda slouží k získání polygonu ze seznamu polygonů **poly_pol** v závislosti na vstupním indexu. Jejím návratovým typem je **std::vector<QPointF>**.
 - Vstup
 - **int index** – index chtěného polygonu
 - Výstup
 - **std::vector<QPointF> poly_pol[index]** – daný polygon
- Metoda **fillPolygon**
 - Tato metoda slouží k přiřazení výsledků analýzy do členské proměnné **analysis_results** a k překreslení. Jejím návratovým typem je **void**.
 - Vstup
 - **std::vector<int> analysis_results** – vektor s výsledky analýzy

3) Widget

- Tato třída slouží ke komunikaci s GUI. Třída dědí od třídy **QWidget**. Všechny metody, které slouží jako sloty k signálům z GUI nemají žádné vstupní hodnoty a jejich návratovým typem je **void**.
- Metoda (slot) **on_clear_button_clicked**
 - Tato metoda reaguje na signál zmáčknutí tlačítka **Clear**. Volá metodu **clearCanvas()** z třídy **Draw**, maže výstup s informací o proběhlé analýze.
- Metoda (slot) **on_analyze_button_clicked**
 - Tato metoda reaguje na signál zmáčknutí tlačítka **Analyze**. V závislosti na uživatelsky zvoleném algoritmu zavolá příslušnou metodu z třídy **Algorithms**.
- Metoda (slot) **on_load_button_clicked**

- Tato metoda reaguje na signál zmáčknutí tlačítka **Load polygon**. Otevírá dialog s adresáři a ukládá cestu zvoleného souboru s body polygonů, kterou dále posílá do metody **loadPolygon** třídy **Draw**.
- Metoda (slot) **on_set_coords_button_clicked()**
 - Tato metoda reaguje na signál zmáčknutí tlačítka **Set point coords**. Volá metodu **setPointCoords** z třídy **Draw** se vstupními hodnotami získanými uživatelským vstupem.
- Metoda **writeAnalysisResult**
 - Tato metoda vypisuje zprávu s informací o výsledku analýzy. Jejím návratovým typem je **void**.
 - Vstup
 - **int result** – výsledek analýzy (-1/0/1)
 - **bool &write_result** – určuje zda-li již byla vypsána zpráva s informací o poloze bodu v/na hranici polygonu

9. Závěr

10.1 Závěrečné zhodnocení

Programování je zábava. Je to ještě větší zábava v tom smyslu, že se tím člověk vlastně učí. Škoda, že se nám nepodařilo vymyslet algoritmus na generování nekonvexních polygonů, strávili jsme nad tím několik hodin a napsali a následně smazali několik desítek řádek kódu. Ale o tom to je, baví nás vymýšlet algoritmy a nevádí nám, že to někdy nedokážeme (a taky doufáme, že nám to ukážete vy). A ne, není to myšleno ironicky.

10.2 Náměty na vylepšení

Algoritmus pro generování nekonvexních polygonů

- Autorům se nepodařilo vymyslet a naimplementovat kvalitní algoritmus, který by generoval nekonvexní polygony.
 - Navrhovaný algoritmus byl následující:
1. Vstup počet polygonů - uživatelské určení počtu polygonů
 2. Deklarace seznamu polygonů
 3. Pro první polygon
 - 3.1) Vygenerování náhodného počtu bodů tak, že u každé nové hrany se testuje, zda-li nekříží žádnou již vytvořenou (pomocí implementované metody **getTwoVectorsAngle** z třídy **Algorithms** a vypočtení případného průsečíku a otestování, zda-li neleží na testované hraně)
 - 3.2) Vytvoření obálky, do které je vložený celý polygon
 - 3.3) Vložení polygonu do seznamu polygonů.
 4. Pro každý další polygon
 - 4.1) Určení náhodného startovního bodu z obálky a náhodného počtu po sobě jdoucích bodů z obálky, které budou součástí nového polygonu (počínaje startovním bodem), zapsání těchto bodů do seznamu bodů nového polygonu

- 4.2) Vygenerování náhodného počtu bodů, kterými se doplní vybrané body z obálky, u každé nové hrany se testuje křížení s hranami obálky a s nově vzniklými hranami. Dále se testuje, zda-li nový bod neleží uvnitř obálky (toto se testuje pouze u prvního přidávaného bodu, jelikož ten jediný by mohl splňovat podmínku nekřížení a být uvnitř obálky) pomocí například Winding algoritmu. Zapsání nových bodů do seznamu bodů nového polygonu.
 - 4.3) Vymazání bodů z **1.** z obálky (krom krajních), přidání bodů z **2.** do obálky.
 - 4.4) Vložení nového polygonu do seznamu polygonů.
- tento algoritmus je časově velmi náročný vzhledem k testování podmínek až po vložení bodu, krok vkládání jednoho bodu může být opakování nekonečněkrát
 - místo, kam vkládat nový bod by bylo třeba určit předem
 - Dalším návrhem bylo vygenerování náhodného počtu náhodně rozmístěných bodů, dále výběr jednoho z vygenerovaných bodů, výpočet úhlů mezi rovnoběžkou s osou x a spojnicí s každým dalším bodem, seřazení bodů podle velikosti úhlu a spojení v tomto pořadí. Nicméně takto vznikne pouze jeden polygon, problém je v generování a připojení dalších polygonů.
 - Možným řešením by mohlo být využití triangulace.

10. Seznam literatury

[1] QT5 TUTORIAL QPAINTERPATH AND QPOLYGON - 2018. *BogoToBogo* [online]. San Francisco: Golden Gate Ave, 2016 [cit. 2018-10-23]. Dostupné z:
https://www.bogotobogo.com/Qt/Qt5_QPainterPath_QPolygon.php?fbclid=IwAR28EoKBjMQq9W_GPDswcNbP3HwarjqFzDteCQ197wuk0qPHxsxFV5Mha1E