

České vysoké učení technické v Praze
Fakulta stavební

155ADKG: Konvexní obálky

Michael Kala
Anna Zemánková

1 Zadání

Navrhněte aplikaci s grafickým rozhraním, která určí a vizualizuje konvexní obálku množiny bodů. Pro výpočet konvexní obálky použijte algoritmy Jarvis Scan, Quick Hull a Sweep Line.

Pro testování algoritmů užíjte rozložení bodů:

- 1) náhodné
- 2) mřížka
- 3) kružnice

2 Údaje o bonusových úlohách

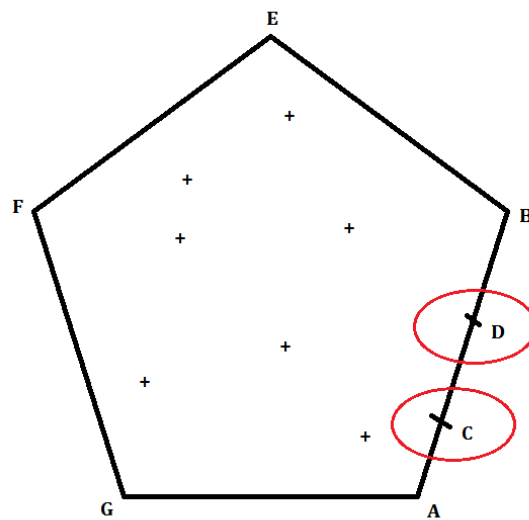
V rámci této úlohy byly zpracovány všechny bonusy kromě generování star-shaped množiny bodů.

3 Popis a rozbor problému

Mějme množinu S bodů $P_i[x_i, y_i]$, naším úkolem je nalézt konvexní obálku κ této množiny.

Konvexní obálka κ je hranice nejmenší konvexní množiny obsahující všechny body. Pokud mezi počátečním a koncovým bodem přímého segmentu konvexní obálky leží více bodů, jde o konvexní obálku, pokud jsou odstraněny, jde o striktně konvexní obálku. Např.:

- Konvexní obálka: A,C,D,B,E,F,G
- Striktně konvexní obálka: A,B,E,F,G



Obrázek 1: Konvexní obálka

4 Popis algoritmů

Pro tuto úlohu byly pro výpočet konvexní obálky vybrány algoritmy Jarvis Scan, Graham Scan, Quick Hull a Sweep Line

4.1 Jarvis Scan

Tento algoritmus nejprve vybere pivota $q : y_{min}$ a následně jej přidá do konvexní obálky, poté je vybrán bod $p_j - 1$ tak, aby spojnice pivota a $p_j - 1$ byla rovnoběžná s osou X $p_j - 1 : x_{min}, y_{min}$. Kvůli numerické stabilitě je vybrána minimální souřadnice X z celé množiny bodů.

Poté je v cyklu vybírán a následně přidáván do konvexní obálky bod $p_j + 1$ tak, aby byl $sphericalangle(p_j - 1, p_j, p_j + 1)$ největší (v prvním kroku $p_j = q$). Cyklus skončí, jakmile dojde opět k pivotovi.

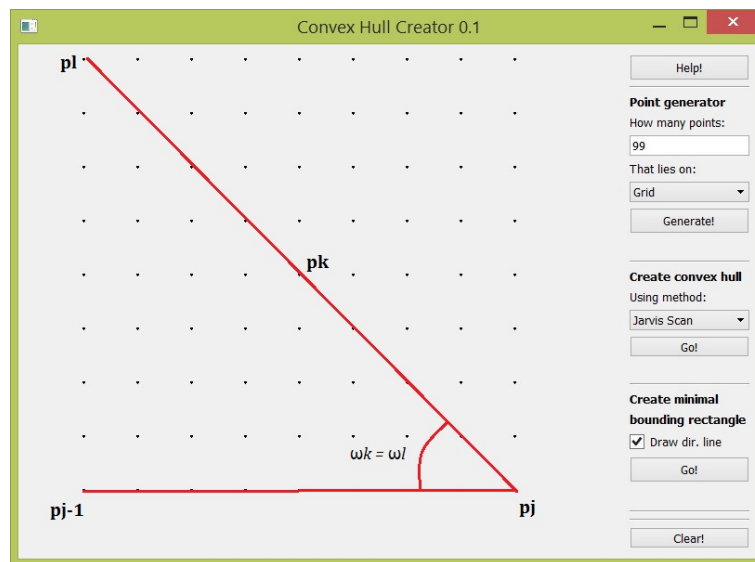
4.1.1 Implementace metody

1. Nalezení pivota q : $q = y_{min}$
2. Přidej $q \rightarrow \kappa$
3. Inicializuj: $p_j - 1 \in X, p_j = q, p_j + 1 = p_j - 1$
4. Opakuj, dokud $p_j + 1 \neq q$:
5. Nalezni $p_j + 1 = \operatorname{argmax}_i i \in P \angle (p_j - 1, p_j, p_i)$
6. Přidej $p_j + 1 \rightarrow \kappa$
7. $p_j - 1 = p_j; p_j = p_j + 1$

4.1.2 Problematické situace

Jestliže se ve vstupních datech vyskytují kolineární body p_k, p_l (např. grid), pak může nastat: $\angle(p_{j-1}, p_j, p_k) = \angle(p_{j-1}, p_j, p_l)$. Je tedy nutné přidat podmínku, aby byl do konvexní obálky zahrnut bod, jehož vzdálenost od bodu p_j je větší.

1. if ($\omega_k = \omega_l$):
2. if ($\|p_j, p_k\| > \|p_j, p_l\|$):
3. $\kappa \leftarrow p_k$
4. else:
5. $\kappa \leftarrow p_l$



Obrázek 2: Kolineární body

4.2 Graham Scan

Nejprve je vybrán pivot q tak, aby z něj bylo vidět na všechny zbylé body. Bodem q je vedena rovnoběžka s osou X a jsou určeny úhly ω od osy X po všechny ostatní body množiny. Následně jsou seříděny podle velikosti ω a pospojovány v tomto pořadí, Vznikne tak star-shaped polygon.

Do zásobníku je přidán bod q a první následující bod, tedy druhý. Poté je testován třetí bod - pokud je splněna podmínka levotočivosti, je přidán do zásobníku. Následně je testován čtvrtý bod. Je-li splněna podmínka levotočivosti, je přidán do zásobníku. Není-li podmínka splněna, je poslední bod v zásobníku smazán a je testován čtvrtý bod z druhého bodu atd. Po skončení cyklu je v zásobníku konvexní obálka množiny bodů.

4.2.1 Implementace metody

1. Nalezení pivotu q : $q = y_{min}$
2. Seřídění $p_i \in S$ dle $\omega_i = \angle(p_i, q, x)$, index j odpovídá jejich seříděnému pořadí
3. Pokud $\omega_k = \omega_l$, vymaž bod p_k, p_l bližší ke q
4. Inicializuj $j = 2, S = \emptyset$
5. $S \leftarrow q, S \leftarrow p_1$ (indexy posledních dvou prvků $p_t, p_t - 1$)
6. Opakuj pro $j < n$:
 7. if p_j vlevo od $p_t - 1, p_t$:
 8. $S \leftarrow p_j$
 9. $j = j + 1$
10. else pop S

4.3 Quick Hull

Algoritmus Quick Hull je rekurzivní a skládá se tedy s lokální a globální procedury.

V lokální proceduře je hledán nejvzdálenější bod od dělicí přímky, který zároveň leží v její pravé polorovině. Spojnici tohoto bodu a koncového bodu dělicí přímky nazýváme první segment a spojnici vyhledaného bodu a počátečního bodu dělicí přímky druhý segment. Aby byly body do konvexní obálky přidány ve správném pořadí, je nejprve provedena rekurze pro první segment, následně přidán vyhledaný bod a poté provedena rekurze pro druhý segment.

V globální proceduře je z množiny S nalezen počáteční p_s (x_{min}) a koncový p_e (x_{max}) bod, které tvoří dělicí přímku. Nejprve je do konvexní obálky přidán koncový bod dělicí přímky, následně je provedena lokální procedura pro horní polorovinu dělicí přímky, přidán počáteční bod dělicí přímky a provedena lokální procedura pro dolní polorovinu. Takto dostaneme body konvexní obálky ve správném pořadí.

4.3.1 Implementace metody - lokální procedura

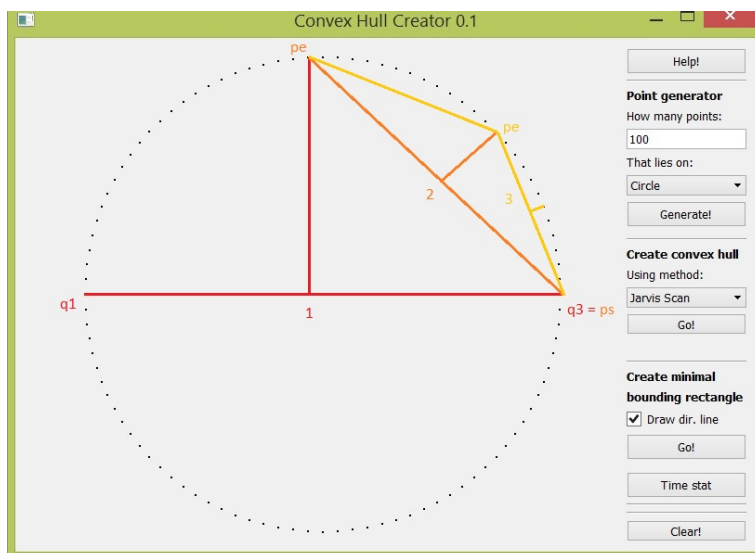
1. Najdi bod $p = \operatorname{argmax}_{p_i \in S} ||p_i - (p_s, p_e)||, p \in \sigma_r(p_s, p_e)$
2. If $p \neq \emptyset$:,
3. QuickHull (s, i, S, κ) // Upper Hull
4. $\kappa \leftarrow p$
5. QuickHull (i, e, S, κ) // Lower Hull

4.3.2 Implementace metody - globální procedura

1. $\kappa = \emptyset, S_U = \emptyset, S_L = \emptyset$
2. $q_1 = x_{min}, q_3 = x_{max}$
3. $S_U \leftarrow q_1, S_U \leftarrow q_3$
4. $S_L \leftarrow q_1, S_L \leftarrow q_3$
5. for $p_i \in S$
6. if $(p_i \in \sigma_l(q_1, q_3))$ $S_U \leftarrow p_i$
7. else $S_L \leftarrow p_i$
8. $\kappa \leftarrow q_3$
9. QuickHull ($1, 0, S_U, \kappa$) // Upper Hull
10. $\kappa \leftarrow q_1$
11. QuickHull ($0, 1, S_L, \kappa$) // Lower Hull

4.3.3 Prolematické situace

Problematická situace při využití algoritmu Quick Hull nastává v okamžiku, kdy je vstupní množina uspořádána do kruhu/kružnice. Spojnice počátečního p_s a koncového bodu p_e je vždy sečna kružnice, dokud do konvexní obálky nejsou zařazeny všechny body kružnice, jak je vidět z obrázku.



Obrázek 3: Quick Hull - kružnice

4.4 Sweep Line

Metoda sweep line pracuje s "přepisováním" předchůdců a následníků a jelikož by se tu až nezdravě opakovala slova přechůdce a následník, samotná implementace metody bude srozumitelnější (jako ostatně vždy).

4.4.1 Implementace metody

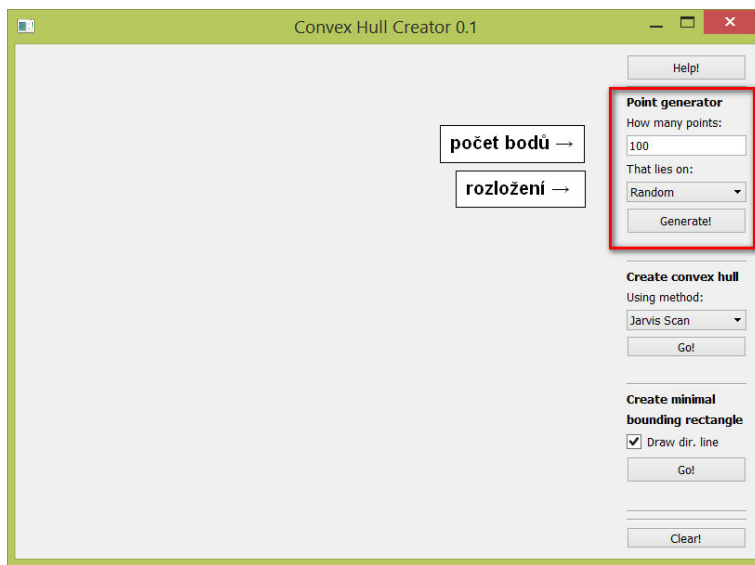
1. Sort $P_s = \text{sort}(P)_{byx}$
2. if $p_3 \in \sigma_L(p_1, p_2)$
3. $n[1] = 2; n[2] = 3; n[3] = 1$
4. $p[1] = 3; p[2] = 1; p[3] = 2$
5. else
6. $n[1] = 3; n[3] = 2; n[2] = 1$
7. $p[1] = 2; p[3] = 1; p[2] = 3$
8. for $p_i \in P_s, i > 3$
9. if $(y_i > y_i - 1)$
10. $p[i] = i-1; n[i] = n[i-1]$
11. else
12. $n[i] = i-1; p[i] = p[i-1]$
13. $n[p[i]] = i; p[n[i]] = i;$
14. while $(n[n[i]]) \in \sigma_R(i, n[i])$
15. $p[n[n[i]]] = i; n[i] = n[n[i]];$
16. while $(p[p[i]]) \in \sigma_L(i, p[i])$
17. $n[p[p[i]]] = i; p[i] = p[p[i]];$

5 Vstupní data

Vstupními daty je množina bodů, kterou lze zadat dvěma způsoby:

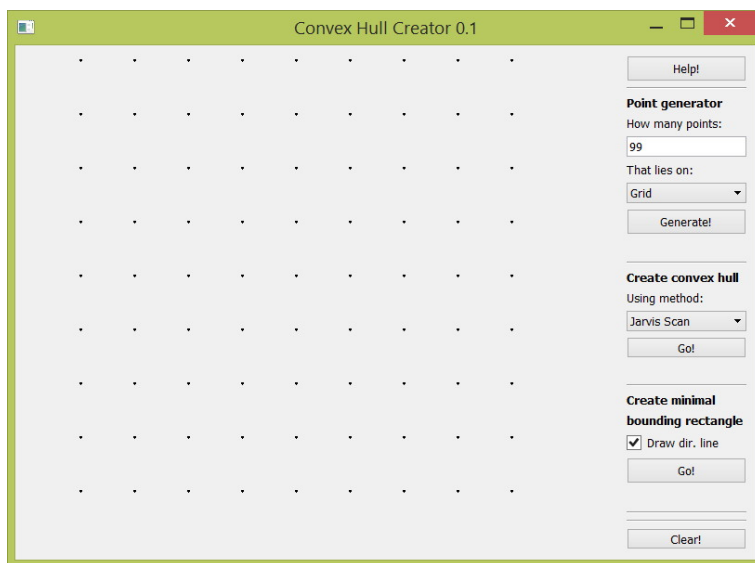
- 1) klikáním na canvas
- 2) použitím generátoru bodů, který je součástí aplikace, přičemž se body generují dle zadaných kritérií.

Do generátoru bodů je třeba zadat počet bodů a následně vybrat v rozbalovacím menu požadované rozmístění - Random/Grid/Circle/Elipse/Square



Obrázek 4: Generátor bodů

V případě, že je zadán nekorektní počet bodů pro dané rozmístění - např. pro čtverec nebo grid, je obrázen vytvořen z nejbližšího menšího možného počtu prvků.

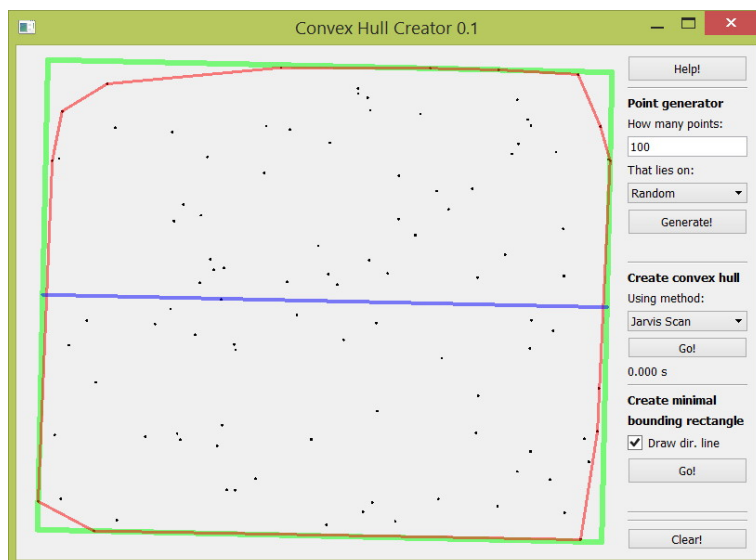


Obrázek 5: Ukázka nekorektního počtu bodů pro zadaný tvar

Pokud jsou body vygenerovány pomocí generátoru, je možné přidat další klikáním na canvas. Při spuštění generátoru jsou všechny dosud existující body vymazány.

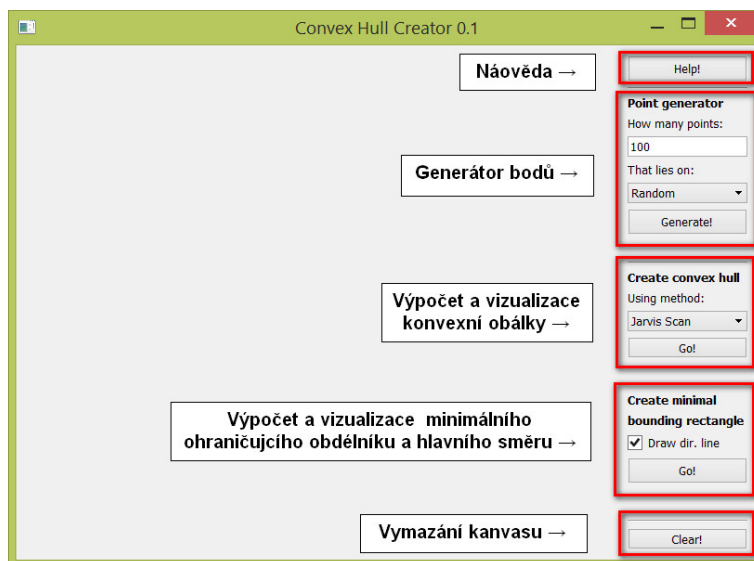
6 Výstupní data

Výsledná data (konvexní obálka/minimální ohraňující obdélník/hlavní směr útvaru) jsou vizualizována v kanvasu aplikace.



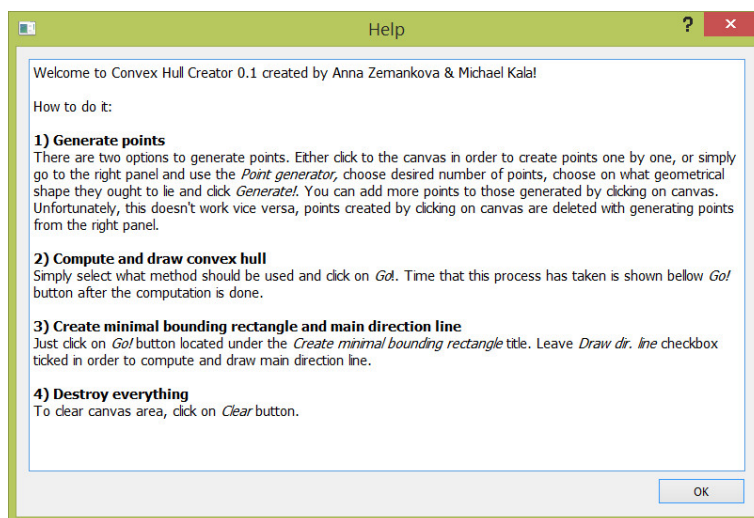
Obrázek 6: Výstupní data

7 Ukázka vytvořené aplikace



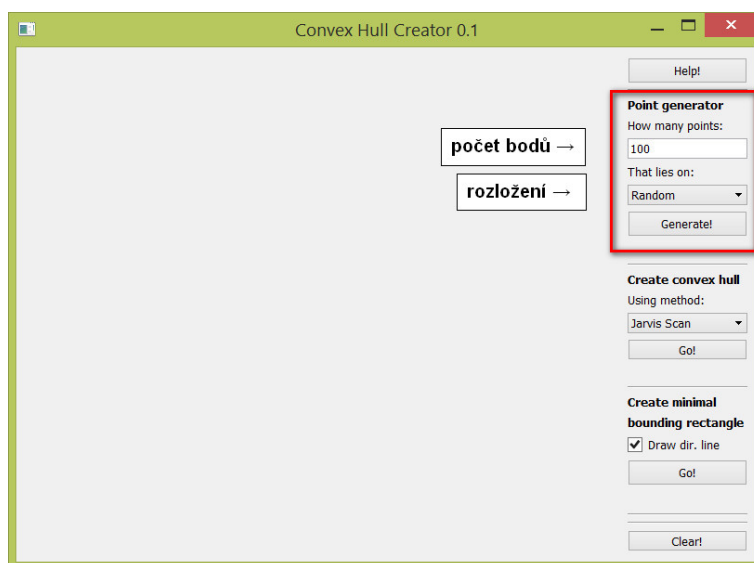
Obrázek 7: Okno aplikace

V Pravé horní části je možnost kliknutí na nápovědu "Help", poté se otevře okno s nápovědou, kde je popsán způsob zadávání vstupních dat a funkcionality aplikace.



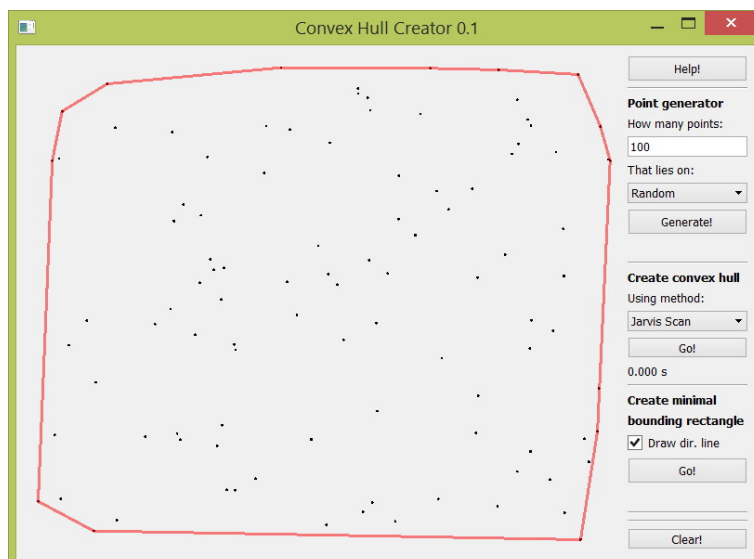
Obrázek 8: Nápověda

Nejprve je třeba zadat vstupní data - množinu bodů.



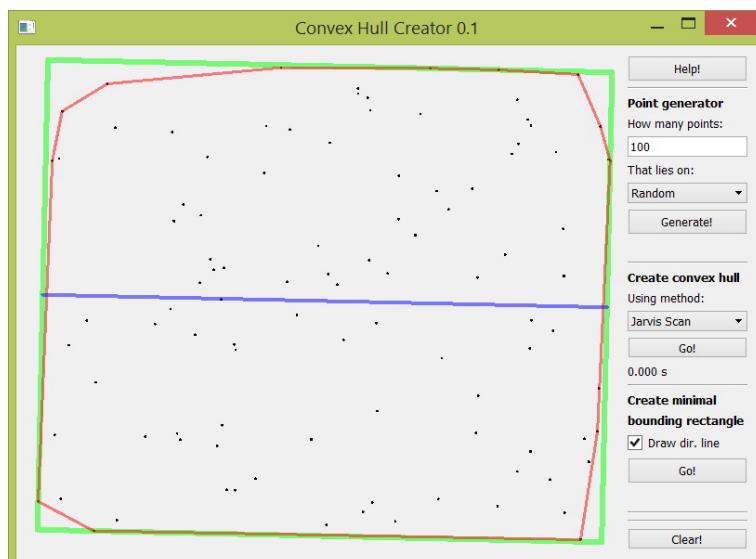
Obrázek 9: Vstupní data

Následně je možné vybrat metodu výpočtu konvexní obálky a kliknutím na tlačítko GO v sekci Create convex hull zahájen výpočet a vizualizace konvexní obálky zadané množiny bodů. Pod tlačítkem Go je vypsán čas, jak dlouho trval výpočet.



Obrázek 10: Výpočet konvexní obálky

Pokud chceme spočítat minimální ohraničující obdélník, lze tak učinit kliknutím na tlačítko Go! v sekci Create inimal bounding rectangle. Pokud je v této sekci zaškrtnuta možnost draw dir. line, vykreslí se také hlavní směr útvaru.



Obrázek 11: Výpočet hlavního směru a nejmenšího ohraničujícího obdélníku

Vše je uvedeno do původního stavu (smazány body i vypočtené výsledky) kliknutím na tlačítko Clear.

8 Dokumentace

8.1 Algorithms

V třídě Algorithms jsou staticky implementovány algoritmy počítající kovexní obálku a minimální ohraničující obdélník (včetně vodící linie).

- Výčtový typ **TPosition**
 - Typ využitý jako návratová hodnota členské metody **getPointLinePosition**.
 - **LEFT = 0**
 - **RIGHT = 1**
 - **ON = 2**
- Metoda **getPointLinePosition**
 - Tato metoda slouží k určení polohy bodu vůči přímce. Návratovou hodnotou je výčtový typ **TPosition**.
 - Vstup
 - * **QPointF &q** - určovaný bod
 - * **QPointF &a, &b** - body přímky
 - Výstup
 - * **LEFT** - bod vlevo od přímky
 - * **RIGHT** - bod vpravo od přímky
 - * **ON** - bod na přímce
- Metoda **getTwoVectorsAngle**
 - Tato metoda slouží k určení úhlu mezi 2 přímkami. Její návratovou hodnotou je **double**.
 - Vstup
 - * **QPointF &p1, &p2** - body první přímky
 - * **QPointF &p3, &p4** - body druhé přímky
 - Výstup
 - * Úhel mezi 2 přímkami
- Metoda **getPointLineDistance**
 - Tato metoda slouží k výpočtu vzdálenosti bodu od přímky. Její návratovou hodnotou je **double**
 - Vstup
 - * **QPointF &q** - určovaný bod
 - * **QPointF &a, &b** - body přímky
 - Výstup

- * Vzdálenost bodu od přímky

- Přetížená metoda **rotateByAngle**

- Tato metoda slouží k rotaci dané množiny o úhel. Jejím návratovým typem je **void**.

- Vstup

- * Přetížení 1

- **std::vector<QPointF> &points** - vektor bodů, jež má být orotován

- **double angle** - úhel, o který má rotace být provedena

- * Přetížení 2

- **QPolygonF &points** - polygon, jež má být orotován

- **double angle** - úhel, o který má rotace být provedena

- * Přetížení 3

- **QLineF &points** - úsečka, jež má být orotována

- **double angle** - úhel, o který má rotace být provedena

- Metoda **getDistance**

- Tato metoda slouží k výpočtu vzdálenosti dvou bodů. Jejím výstupním typem je **double**.

- Vstup

- * **QPointF &a, &b** - body, mezi kterými je vzdálenost počítána

- Výstup

- * Vypočtená vzdálenost

- Metoda **jarvisScanCH**

- Tato metoda slouží k výpočtu konvexní obálky pomocí algoritmu Jarvis Scan. Jejím výstupním typem je **QPolygonF**.

- Vstup

- * **std::vector<QPointF> points** - vektor bodů, kolem nichž má být vytvořena konvexní obálka.

- Výstup

- * Polygon obsahující konvexní obálku.

- Metoda **grahamScanCH**

- Tato metoda slouží k výpočtu konvexní obálky pomocí algoritmu Graham Scan. Jejím výstupním typem je **QPolygonF**.

- Vstup

- * **std::vector<QPointF> points** - vektor bodů, kolem nichž má být vytvořena konvexní obálka.

- Výstup

- * Polygon obsahující kovexní obálku.
- Metoda **quickHullCH**
 - Tato metoda slouží k výpočtu konvexní obálky pomocí algoritmu Quick Hull. Jejím výstupním typem je **QPolygonF**.
 - Vstup
 - * **std::vector <QPointF> points** - vektor bodů, kolem nichž má být vytvořená konvexní obálka.
 - Výstup
 - * Polygon obsahující kovexní obálku.
- Metoda **quickHullLocal**
 - Pomocná metoda k výpočtu konvexní obálky metodou Quick Hull. Jejím výstupním typem je **void**.
 - Vstup
 - * **int s, e** - index počátečního a koncového bodu, ohraničující množinu bodů, pro které je počítána lokální procedura
 - * **std::vector <QPointF> &points** - vektor bodů, kolem nichž má být vytvořená konvexní obálka.
 - * **QPolygonF &poly_ch** - polygon obsahující body konvexní obálky
 - Výstup
 - * Polygon obsahující kovexní obálku.
- Metoda **sweepLineCH**
 - Tato metoda slouží k výpočtu konvexní obálky pomocí algoritmu Sweep Line. Jejím výstupním typem je **QPolygonF**.
 - Vstup
 - * **std::vector <QPointF> points** - vektor bodů, kolem nichž má být vytvořená konvexní obálka.
 - Výstup
 - * Polygon obsahující kovexní obálku.
- Metoda **generatePoints**
 - Metoda pro generování zadaného počtu a tvaru bodů. Jejím výstupním typem je **std::vector <QPointF> points**.
 - Vstup
 - * **QSizeF &canvas_size** - rozměry kreslicího plátna, ze kterých se determinuje rozsah generovaných bodů
 - * **int point_count** - počet bodů, který se má generovat
 - * **std::string shape** - tvar vytvářené množiny bodů (random, grid, na kružnici, na elipse, na čtverci)

- Výstup
 - * Vektor nagenерованých bodů.
- Metoda **minimalRectangle**
 - Metoda pro výpočet minimálního ohraničujícího obdélníku a hlavní linie. Jejím výstupním typem je **void**.
 - Vstup
 - * **QPolygonF &poly_ch** - polygon obsahující konvexní obálku
 - * **QPolygonF &minimal_rectangle** - polygon, do kterého jsou počítány body minimálního ohraničujícího obdélníku
 - * **QLineF &direction** - hlavní linie minimálního ohraničujícího obdélníku (resp. do této proměnné je počítaná)
 - * **bool compute_dir_line** - ukazatel určující zda-li má být počítána hlavní linie minimálního ohraničujícího obdélníku

8.2 Draw

Třída `draw` slouží k vykreslení vygenerovaných (nebo naklikaných) bodů, vypočteného minimálního ohraničujícího obdélníku a hlavní linie minimálního ohraničujícího obdélníku. V této třídě jsou zároveň nagenеровané body zbavené duplicit a vypočtené konvexní obálky se zde omezují na striktní konvexní obálky (vše v metodě **setCH**). Třída dědí od třídy **QWidget**.

- Členské proměnné
 - **std::vector <QPointF> points** - vektor obsahující nagenеровané nebo naklikané body
 - **QPolygonF ch** - polygon obsahující body konvexní obálky
 - **QPolygonF rect** - polygon obsahující body minimálního ohraničujícího obdélníku
 - **QLineF direction** - hlavní linie minimálního ohraničujícího obdélníka
- Metoda **paintEvent**
 - Tato metoda slouží k vykreslení nagenеровaných (nebo naklikaných) bodů, konvexní obálky, minimálního ohraničujícího obdélníka a hlavní linie minimálního ohraničujícího obdélníka. Metoda se volá pomocí metody **repaint()**. Návrátovým typem je **void**.
 - Vstup
 - * **QPaintEvent *e**
- Metoda **mousePressEvent**
 - Metoda sloužící k uložení bodu do členské proměnné **points** určeného kliknutím myši nad kreslícím plátnem. Jejím návrátovým typem je **void**.
 - Vstup
 - * **QMouseEvent *e**
- Metoda **setCH**
 - Tato metoda slouží pro kontrolu duplicity generovaných bodů, pro kontrolu alespoň 3 bodů, k zavolání příslušného algoritmu pro vypočtení konvexní obálky a k omezení konvexní obálky na striktně konvexní obálku. Metoda počítá dobu trvání výpočetních algoritmů. Jejím návrátovým typem je **double**.
 - Vstup
 - * **std::string &selected_algorithm** - uživatelsky vybraný algoritmus pro počítání konvexní obálky
 - Výstup
 - * Čas trvání výpočtu.
- Metoda **setRect**

- Tato metoda slouží pro zavolání algoritmu pro výpočte minimálního ohraničujícího obdélníku a jeho hlavní linie. Jejím návratovým typem je **void**.
- Vstup
 - * **bool draw_dir_line** - uživatelsky nastavený indikátor, zda-li se má vypočítat hlavní linie minimálního ohraničujícího obdélníku
- Metoda **setPoints**
 - Metoda volající algoritmus pro generování bodů daného počtu a tvaru. Jejím návratovým typem je **void**.
 - Vstup
 - * **QSizeF &canvas_size** - rozměr kreslicího plátna pro pozdější určení rozsahu generování bodů
 - * **int count** - počet bodů, jež se má generovat
 - * **std::string &shape** - tvar, do kterého se body mají generovat
- Metoda **clearCanvas**
 - Metoda, která maže obsah kreslicího okna. Jejím návratovým typem je **void**. Do metody nevstupují žádné parametry.

8.3 SortByXAsc, SortByYAsc, SortByAngleAsc

Třídy sloužící jako sortovací kritérium - podle rostoucí souřadnice x resp. souřadnice y (při stejných souřadnicích x resp. y je druhým kritériem druhá souřadnice) a podle rostoucího úhlu mezi body (při stejném úhlu je druhým kritériem vzdálenosti mezi body).

8.4 Widget

Tato třída slouží ke komunikaci s GUI. Třída dědí od třídy `QWidget`. Všechny její metody slouží jako sloty k signálům z GUI, nemají žádné vstupní hodnoty a jejich návratovým typem je `void`.

- Metoda **`on_createCHButton_clicked`** - reaguje na zmáčknutí tlačítka pro vypočtení konvexní obálky, volá metodu **`setCH`** z třídy **`Draw`**, zapisuje čas výpočtu do GUI.
- Metoda **`on_generateButton_clicked`** - reaguje na zmáčknutí tlačítka pro generování bodů, volá metodu **`setPoints`** z třídy **`Draw`**.
- Metoda **`on_clearButton_clicked`** - reaguje na zmáčknutí tlačítka pro vymazání obsahu kreslicího plátna, volá metodu **`clearCanvas`** z třídy **`Draw`**.
- Metoda **`on_createRectButton_clicked`** - reaguje na zmáčknutí tlačítka pro vypočtení minimálního ohraničujícího obdélníka, volá metodu **`setRect`** z třídy **`Draw`**.
- Metoda **`on_helpButton_clicked`** - reaguje na zmáčknutí tlačítka pro volání nápovědy, volá okno s nápovědou **`help_dialog`** z třídy **`HelpDialog`**.

8.5 HelpDialog

Třída sloužící pro vykreslení okna s nápovědou.

9 Přílohy

- Příloha č.1: Testování výpočetních dob algoritmů - "Testovani.pdf"

10 Závěr

Aplikace byla vzhledem ke zpracování bonusů programována tak, že vstupní i výstupní data jsou typu float, při testování doby běhu algoritmů však bylo zjištěno, že při použití datového typu float je výpočet velmi VELMI pomalý, naopak při přepsání kódu do verze pracující s integrem byl výpočet výrazně rychlejší. Bohužel už ale byly vyřešeny bonusy za předpokladu, že vstupní data jsou typu float, a proto jsou nakonec odevzdávány dvě verze - float pro funkcionality aplikace pro menší množiny bodů a integer pro testování doby běhu algoritmů.

11 Zdroje

1. BAYER, Tomáš. Konvexní obálky [online][cit. 21.10.2018].
Dostupné z: <https://web.natur.cuni.cz/~bayertom/images/courses/Adk/adk4.pdf>