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 užijte rozložení bodů:
1) náhodné
2) mřížka
3) kružnice

2 Údaje o bonusových úlohách

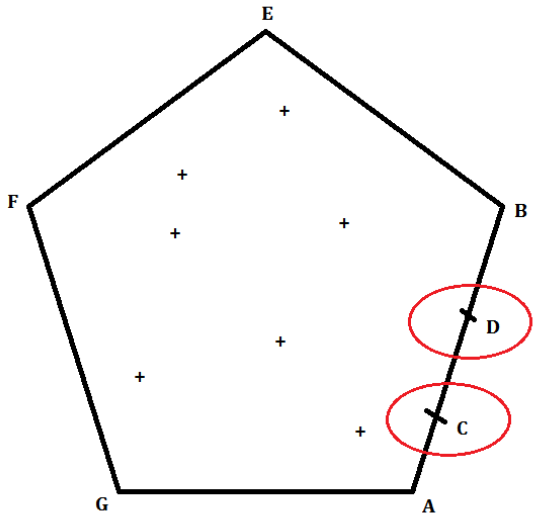
V rámci této úlohy byly zpracovány všechny bonusy kromě generování star-shaped množiny bodů.

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 kovexní 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 algortimy 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 souřadnce X z celé množiny bodů.
Poté je v cyklu vybírán a následně přidáván do konvexní obálky bod $p_j + 1$ tak, aby byl $spherical angle(p_{j-1}, p_j, p_{j+1})$ největší (v prvním kroku $p_j = q$). Cyklus skončí,jakmile dojde opět k pivotovi.

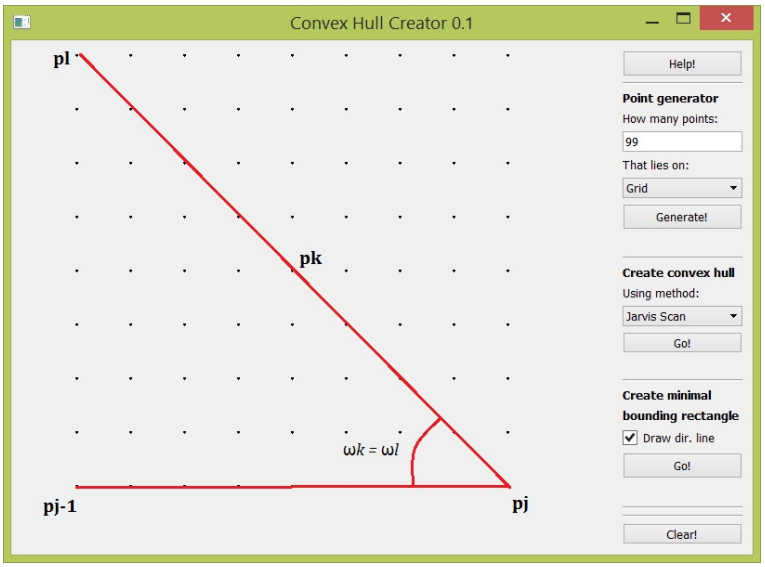
4.1.1 Implementace metody

1. Nalezení pivota q: $q = y_{min}$
2. Přidej $q \longrightarrow \kappa$
3. Inicializuj: $p_{j-1} \in X, p_j = q, p_{j+1} = p_{j-1}$
4. Opakuj, dokud $p_{j+1}! = q$:
5. Nalezni $p_j + 1 = argmax_{pi \in P} \measuredangle(p_{j-1}, p_j, p_i)$
6. Přidej $p_j + 1 \longrightarrow \kappa$
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: $\sphericalangle(p_{j-1}, p_j, p_k) = \sphericalangle(p_{j-1}, p_j, p_l)$. 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. $\kappa \longleftarrow p_k$
4. else:
5. $\kappa \longleftarrow p_l$



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 setříděny 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 zásbníku. Následně je testován čtvrtý bod. Je-li splněna podmínka levotočiosti, 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 množiy bodů.

4.2.1 Implementace metody

1. Nalezení pivota q: $q = y_{min}$
2. Setřídění $p_i \in S$ dle $\omega_i = \measuredangle(p_i, q, x)$, index j odpovídá jejich setříděnému pořadí
3. Pokud $\omega_k = \omega_l$, vymaž bod p_k, p_l bližší ke q
4. Inicialiuj $j = 2, S = \emptyset$
5. $S \longleftarrow q, S \longleftarrow p_1$ (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. $S \longleftarrow p_j$
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 nazývejme první segment a spojnici vyhledaného body 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 dělící 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 $p = argmax_{pi \in S}||p_i - (p_s, p_e)||, p \in \sigma_r(p_s, p_e)$
2. If $p \neq \emptyset$:,
3. QuickHull (s, i, S, κ) // Upper Hull
4. $\kappa \longleftarrow p$
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. $S_U \longleftarrow q_1$, $S_U \longleftarrow q_3$
4. $S_L \longleftarrow q_1$, $S_L \longleftarrow q_3$
5. for $p_i \in S$
6. if $(p_i \in \sigma_l(q_1, q_3))$ $S_U \longleftarrow p_i$
7. else $S_L \longleftarrow p_i$
8. $\kappa \longleftarrow q_3$
9. QuickHull $(1, 0, S_U, \kappa)$ // Upper Hull
10. $\kappa \longleftarrow q_1$
11. QuickHull $(0, 1, S_L, \kappa)$ // Lower Hull

4.3.3 Prolematické situace

Problematická situace při využití algotimu Quick Hull nastává v okamžiku, kdy je vstupní množina uspořádána do kružnice. Spojnice počátečního p_s a koncového bodu p_e je pak vždy sečna kružnice -¿ existuje další bod v pravé polorovině dokud do konvexní obálky nejsou zařazeny všechny body a rekurze bude volána pro každý bod vstupní množiny. V obrázku níže jsou naznačeny první tři kroky, odlišené barvami.

	Porovnání průměrných dob běhů algoritmů									
	1000	10 000	100 000	230 000	360 000	490 000	620 000	750 000	880 000	1 000 000
Jarvis Scan	0,021	0,113	0,144	0,144	0,146	0,144	0,143	0,202	0,222	0,218
Graham Scan	0,002	0,003	0,003	0,003	0,003	0,003	0,003	0,003	0,003	0,004
Quick Hull	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001
Sweep line	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000

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 $P_s = sort(P)$ by x
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]];

4.5 Minimální ohraničující obdélník

Pro výpočet minimálního ohraničující obdélníku byl vybrán následující algoritmus. Na jeho vstupu je konvexní obálka ch.

4.5.1 Implementace metody

1. for $p_{i\%n}, p_{(i+1)\%n} \in ch$ //n = ch.size
2. rotuj ch o $\sphericalangle(|p_{i\%n}, p_{(i+1)\%n}|, ||$ s osou x)
3. najdi nejmenší resp. největší x a y souřadnice, spočti obsah jimi vytvořeného obdélníka, pokud je menší než jakýkoliv doposud vypočtený, ulož ho, ulož úhel, o který bylo rotováno, ulož souřadnice
4. rotuj zpět o úhel z kroku 2
5. rotuj zpět souřadnice obdélníka o uložený úhel

4.6 Striktně konvexní obálka

Jak již bylo řečeno výše, striktně konvexní obálka na přímém segmentu neobsahuje žádné jiné body kromě počátečního a koncového bodu segmentu. Pro odstranění mezilehlých bodů byla vypočtena rovnice přímky prvních dvou bodů. Poté byl do rovnice dosazen bod třetí, pokud ležel na přímce, byl druhý bod odstraněn z konvexnní obálky a testován bod čtvrtý. Neležel-li na přímce, byla vypočtena rovnice přímky druhého a třetího bodu a poté do ní byl dosazen bod čtvrtý atd. Cyklus končí jakmile je ověřen poslední bod konvexní obálky.

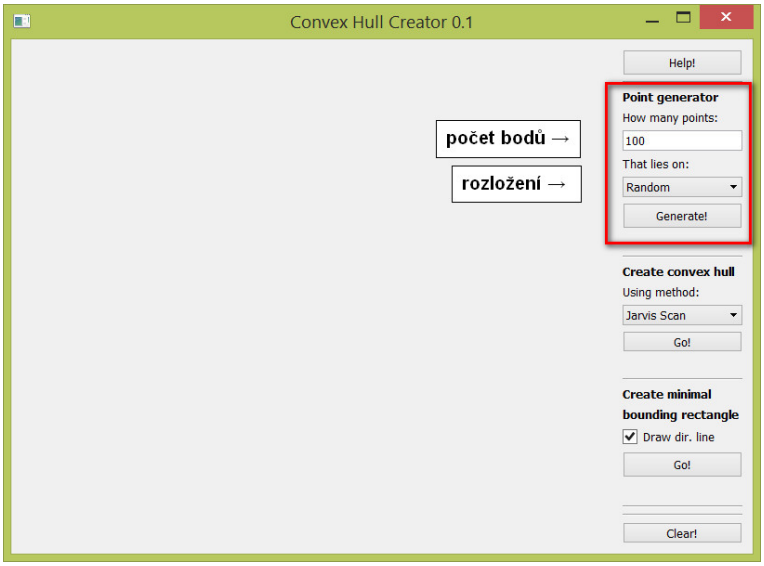
4.6.1 Implementace metody

1. for i = 0, i¡ n, i++ ..... n=pocet bodu mnoziny p
2. a = -(p((i+1)%n).y - p(i%n).y)
   b = p((i+1)%n).x - p(i%n).x ... slozky normaloveho vektoru primky,
   resp. koeficienty a, b obecne rovnice primky ax+by+c=0
3. c = - a * p(i%n).x - b * p(i%n).y ... koeficient c obecne rovnice primky
4. if(a*p((i+2)%n).x + b*p((i+2)%n).y +c = 0):
5. erase p((i+1)%n)
6. i - - , n - -

5 Vstupní data

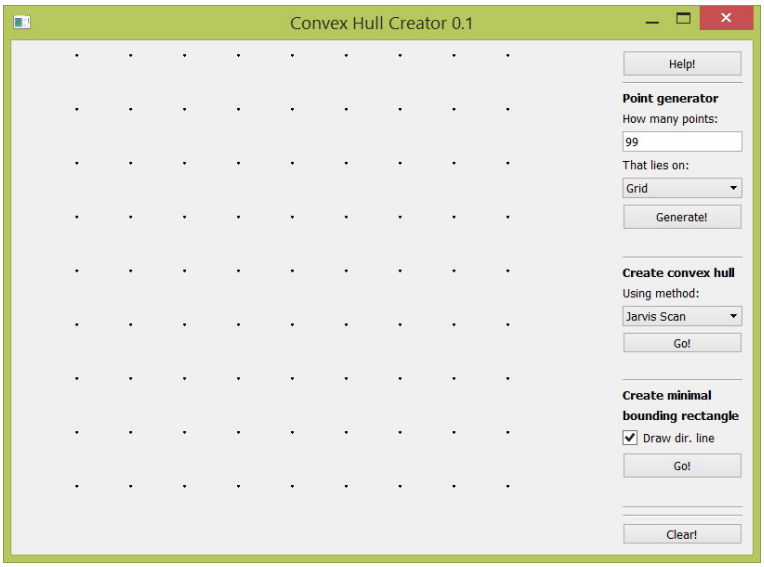
Vstupními daty je množina bodů, kterou lze zadat dvěma způsoby:
1) klikáním na kanvas
2) použitím generátoru bodů, který je součástí aplikace, přičemž se body generují dle zadaných kritérií.

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 obrazen vytvořen z nejbližšího menšího možného počtu prvků.

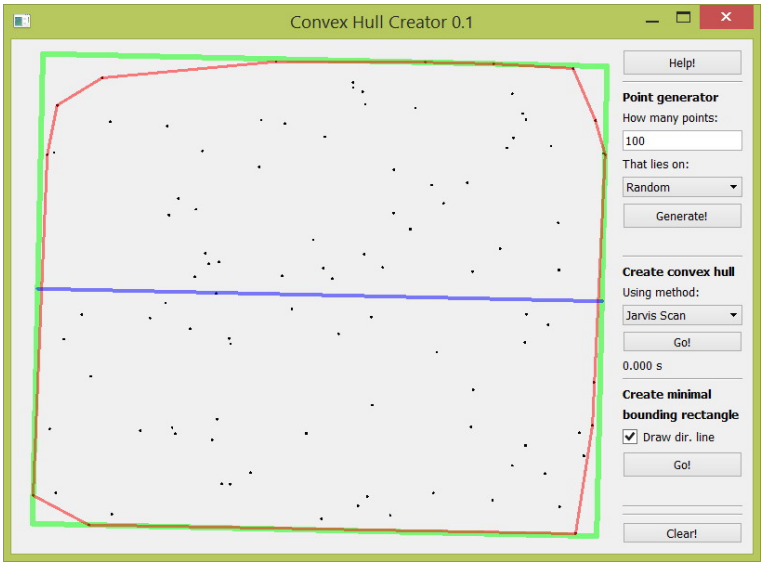


Obrázek 5: Ukázka nekorektního počtu bodů pro zadaný tvar

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

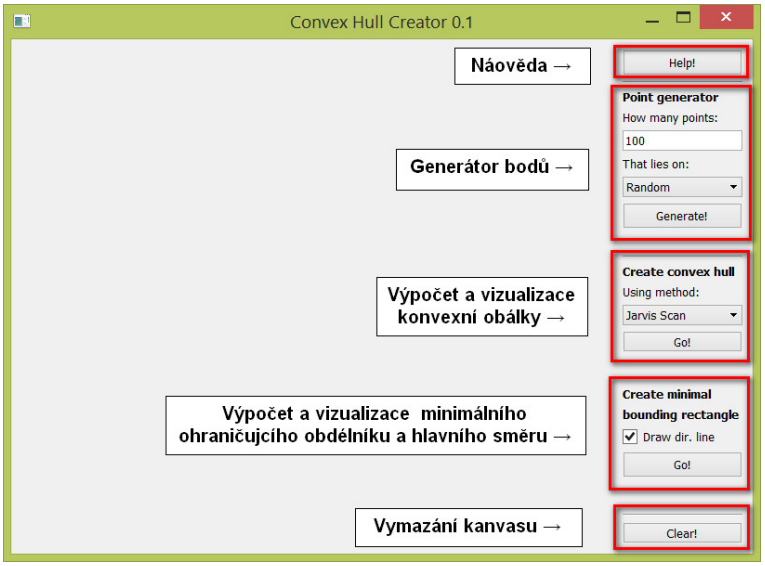
6 Výstupní data

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



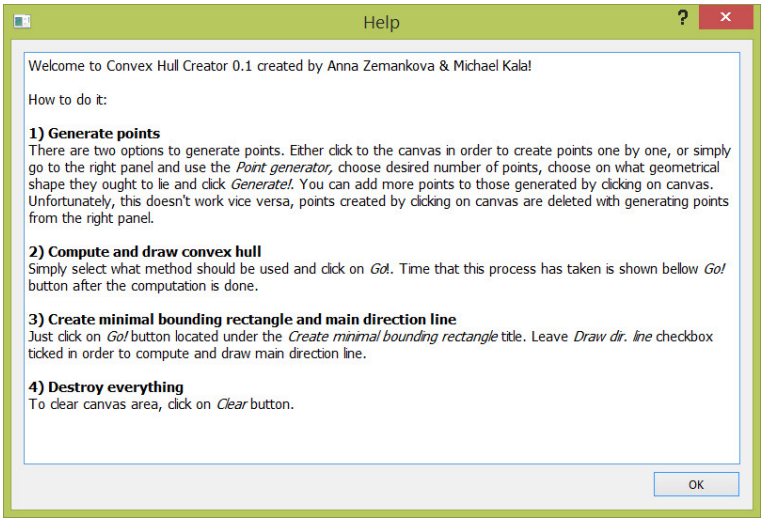
Obrázek 6: Výstupní data

7 Ukázka vytvořené aplikace



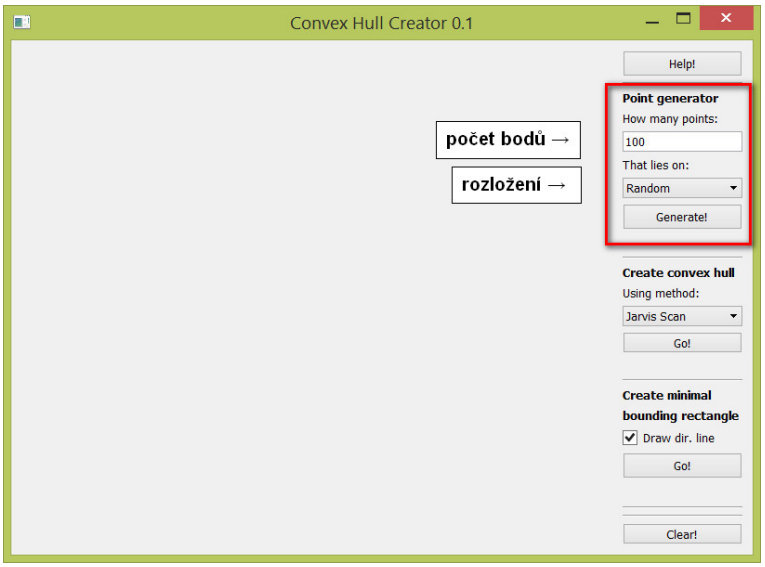
Obrázek 7: Okno aplikace

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.



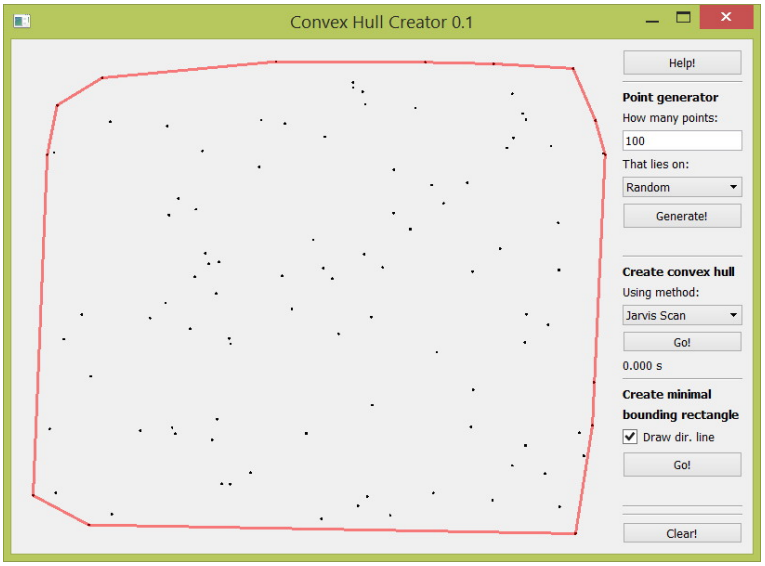
Obrázek 8: Nápověda

Nejprve je třeba zadat vstupní data - množinu bodů.



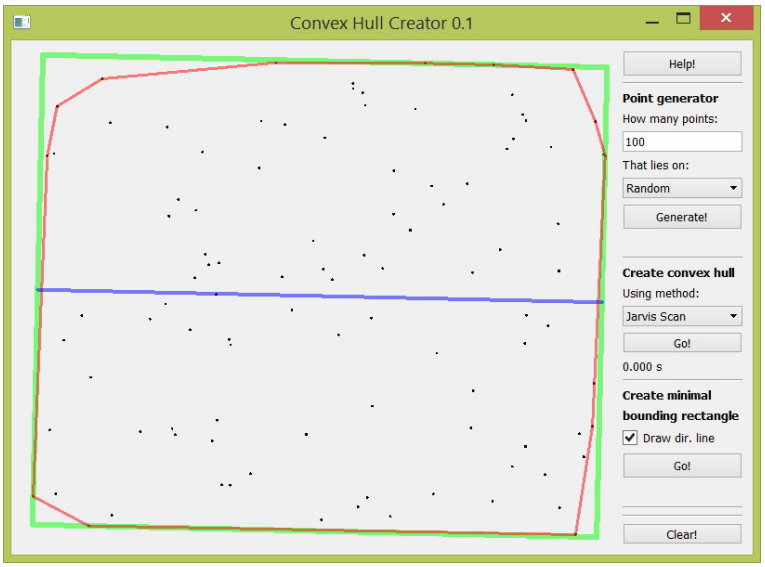
Obrázek 9: Vstupní data

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 vyýsledky) kliknutím na tlačítko Clear.

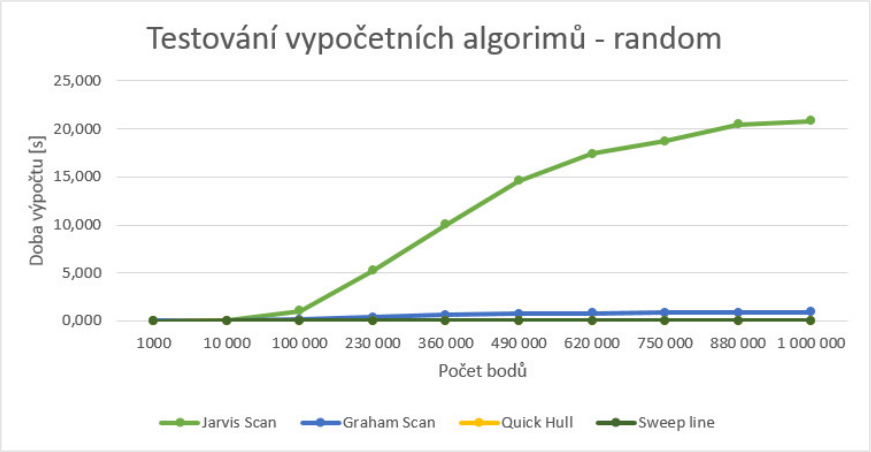
8 Testování výpočetních dob algoritmů

V rámci této úlohy byly testovány výpočetní doby algoritmů pro vsutpní množim o počtu bodů 1 000 - 1 000 000 bodů. Pro každý interval a každý tvar vstupní množiny byl výpočet proveden 10x, následně byl určen průměr a rozptyl.
Na základě těchto testování bylo zjištěno, že algoritmy Sweep Line a Quick Hull jsou univerzální výpočetní algoritmy konvexní obálky pro různé uspořádání množiny bodů a to i pro velké množiny bodů. Graham Scan je oproti nim vždy (kromě kružnice) o řád až dva pomalejší. Jarvis Scan s ostatními algoritmy snad ani nelze srovnávat – je vždy o několik řádů pomalejší. Níže jsou uvedeny výsledné průměrné hodnoty a jejich grafy. Podrobnější záznam testování viz přílohu č. 1 "Testování".

8.1 Rozložení bodů - random

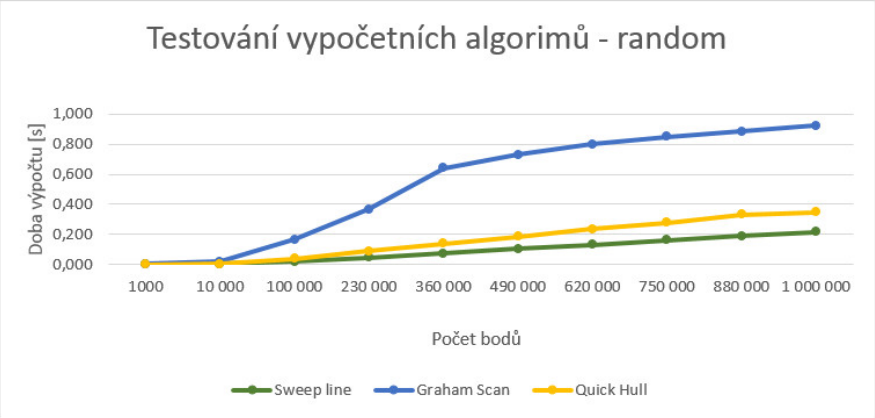
	Porovnání průměrných dob běhů algoritmů									
	1000	**10 000**	**100 000**	**230 000**	**360 000**	**490 000**	**620 000**	**750 000**	**880 000**	**1 000 000**
Jarvis Scan	0,001	0,019	1,070	5,259	10,026	14,573	17,399	18,741	20,483	20,814
Graham Scan	0,002	0,016	0,164	0,366	0,642	0,732	0,801	0,850	0,884	0,923
Quick Hull	0,000	0,001	0,006	0,011	0,013	0,012	0,013	0,014	0,014	0,015
Sweep line	0,000	0,000	0,003	0,007	0,009	0,010	0,011	0,011	0,012	0,012

Obrázek 12: Porovnání průměrné doby výpočtu - random



Obrázek 13: Graf porovnání průměrné doby výpočtu - random

Pro vizuální porovnání algoritmů Quick hull, Sweep line a Graham scan byl vytvořen graf zvlášť.

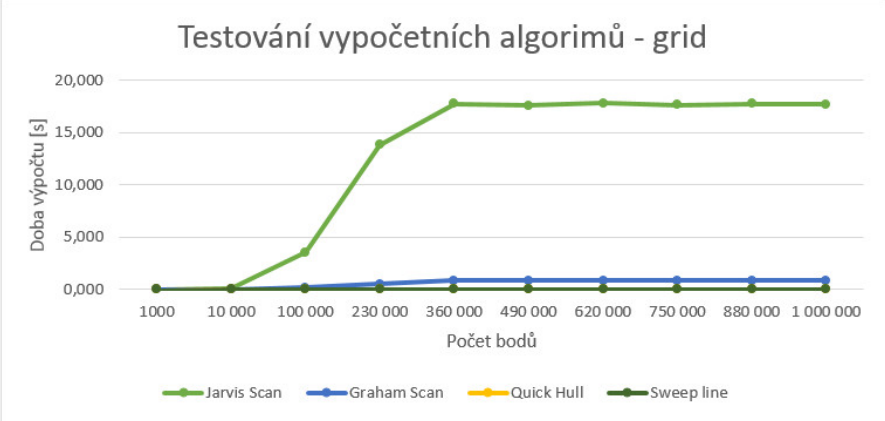


Obrázek 14: Graf porovnání průměrné doby výpočtu - random

8.2 Rozložení bodů - grid

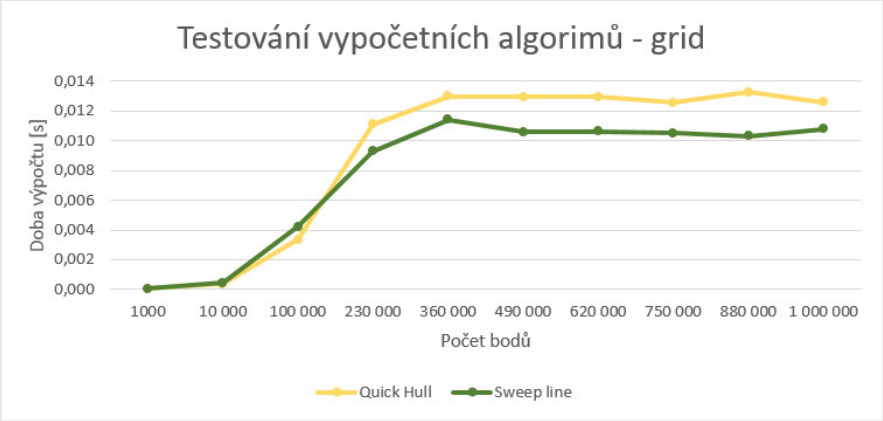
	Porovnání průměrných dob běhů algoritmů									
	1000	**10 000**	**100 000**	**230 000**	**360 000**	**490 000**	**620 000**	**750 000**	**880 000**	**1 000 000**
Jarvis Scan	0,003	0,095	3,575	13,818	17,775	17,628	17,841	17,646	17,787	17,696
Graham Scan	0,001	0,018	0,200	0,535	0,853	0,848	0,848	0,855	0,842	0,842
Quick Hull	0,000	0,000	0,003	0,011	0,013	0,013	0,013	0,013	0,013	0,013
Sweep line	0,000	0,000	0,004	0,009	0,011	0,011	0,011	0,010	0,010	0,011

Obrázek 15: Porovnání průměrné doby výpočtu - grid



Obrázek 16: Graf porovnání průměrné doby výpočtu - grid

Pro vizuální porovnání algoritmů Quick Hull a Sweep Line byl vytvořen graf zvlášť.

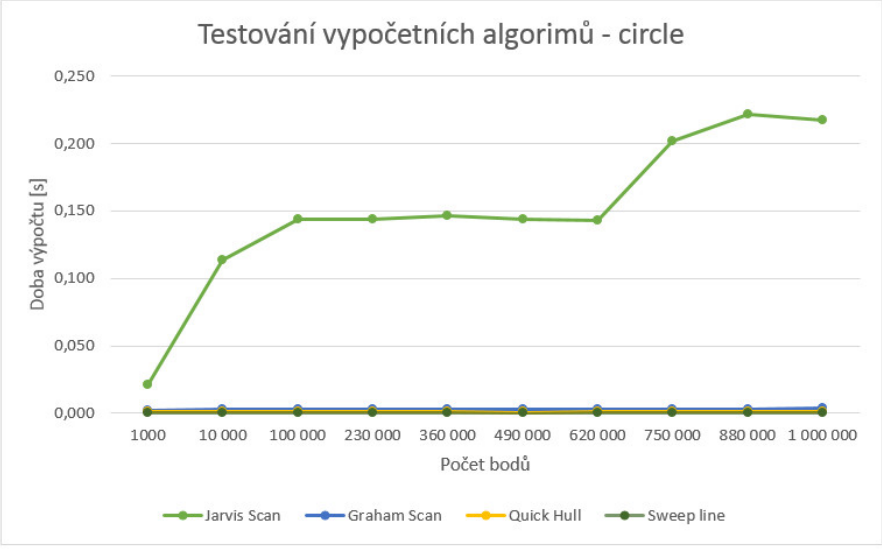


Obrázek 17: Graf porovnání průměrné doby výpočtu - grid

8.3 Rozložení bodů - circle

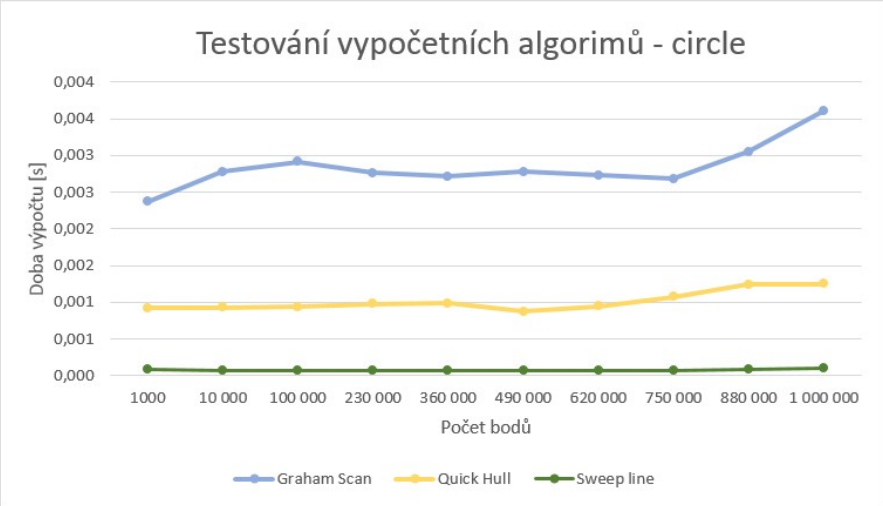
	Porovnání průměrných dob běhů algoritmů									
	1000	**10 000**	**100 000**	**230 000**	**360 000**	**490 000**	**620 000**	**750 000**	**880 000**	**1 000 000**
Jarvis Scan	0,021	0,113	0,144	0,144	0,146	0,144	0,143	0,202	0,222	0,218
Graham Scan	0,002	0,003	0,003	0,003	0,003	0,003	0,003	0,003	0,003	0,004
Quick Hull	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001	0,001
Sweep line	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000

Obrázek 18: Porovnání průměrné doby výpočtu - circle



Obrázek 19: Graf porovnání průměrné doby výpočtu - circle

Pro vizuální porovnání algoritmů Quick Hull, Sweep Line a Graham scan byl vytvořen graf zvlášť.



Obrázek 20: Graf porovnání průměrné doby výpočtu - circle

9 Dokumentace

9.1 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 hodnotu 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. Během výpočtu je ošetřována singularita existence kolineárních bodů v datasetu. 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 **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 vytvořená 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 dělící přímky
    - **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 kreslícího 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 nagenerovaných 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 obdelníku (resp. do této proměnné je počítaná)
        * **bool compute_dir_line** - ukazatel určující zda-li má být počítána hlavní linie minimálního ohraničujícího obdélníku

9.2 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 nagenerované 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 nagenerované 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 nagenerovaných (nebo naklikaných) bodů, konvexní obýlky, minimálního ohraničujícího obdélníka a hlavní linie minimálního ohraničujícího obdélníka. Metoda se volá pomocí metody **repaint()**. Návratovým typem je **void**.
  - Vstup
    - **QPaintEvent *e**
- Metoda **mousePressEvent**
  - Metoda sloužící k uložení bodu do členské proměnné **points** určeného kliknutím myší nad kreslícím plátnem. Jejím návratovým 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 návratovým 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 kreslícího 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 kreslícího okna. Jejím návratovým typem je **void**. Do metody nevstupují žádné parametry.

9.3 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).

9.4 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 kreslícího 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**.

9.5 HelpDialog

Třída sloužící pro vykreslení okna s nápovědou.

10 Přílohy

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

11 Závěr

11.1 Politování se a vysvětlení našeho problému

Aplikace od začátku implementovala body, polygony, linie atd. v typu float (QPointF, QPolygonF, QLineF), v tomto duchu byly psány i všechny algoritmy. Při testování bylo zjištěno, že výpočet takto implementovaných algoritmů trvá VELMI dlouho (v řádu stovek sekund). Proto bylo navrženo (dva dny před odevzdáním), že by bylo vhodné vyzkoušet, jak budou algoritmy rychle počítat při implementaci bodů, polygonů a linií v typu int (QPoint, QPolygon, QLine). Bylo zjištěno, že tato změna (aplikovaná pomocí programu grep (to jsme se jen chtěli pochlubit, že jsme to zmákli přepsat rychle)) algoritmy rapidně urychlí (do řádu sekund až sub-sekund). Z tohoto důvodu (a také z časových důvodů, při testování pomalejší verze počítač zamrzá a člověk u toho musí pořád sedět a odklikávat "Wait", aby program nepadl) jsou uvedeny grafy z testování rychlejší verze. Kdyby všechny algoritmy po tomto přechodu fungovaly správně, ani bychom se neobtěžovali psát tak obsáhlý závěr, ale to by nebyla smůla, aby se to nerozbilo. Především myšlenka rotací tam a zpět u minimálního ohraničujícího obdélníku již nefunguje - orotované souřadnice se zaokrouhlí na celé číslo, při rotaci zpět se opět zaokrouhlí na celé číslo a problém je na světě.
Dále je jakýmsi neidentifikovatelným způsobem rozbitý výpočet Jarvis Scan na kružnici (kód jsme změnili na kód stejný jako z hodiny, kód generování bodů na kružnici máme obdobný jako ostatní skupinky, ale i tak nefunguje úplně správně, neumíme určit problém (a moc nás to mrzí (skutečně, neironicky (to taky není ironie)))). Jelikož jsme strávili spoustu hodin nad co nejoptimálnější implementací těchto algoritmů a samozřejmě jsme to celou dobu testovali pro malé množiny bodů, kdy vše běží rychle, rozhodli jsme se odevzdat 2 verze kódu. Složky jsou pojmenované `src_float` pro původní super vymazlenou verzi a `src_int` pro testovací rozbitou verzi s tím, že si stojíme za super vymazlenou verzí `src_float`, kde funguje vše, ale pro velké množství bodů v některých případech pomaleji (hlavně Jarvis Scan). Jako release odevzdáváme verzi `src_float`. Doufáme, že to takto bude v pořádku.
Ještě bychom se rádi pochlubili šikovnou implementací vymazávání duplicitních bodů při generování množin (nejprve jsme sortovali, a pak porovnávali následující body, uložili vždy poslední stejný, takto nemusíme použít vnořený cyklus a náročnost je lineární a ne kvadratická).

11.2 Návrhy na vylepšení

- Přizpůsobení vykreslených prvků při zvětšování/zmenšování okna
- Úplné vynechání Jarvis Scanu
- Lepší algoritmus na počítání minimálního ohraničujícího obdélníka, nejlépe takový, při kterém se nemusí rotovat celá množina bodů
  - Krom šikovných algoritmů z prezentace by také šla využít analytická geometrie, tj. v podstatě stejný postup jaký jsme použili v našem programu, ale nerotovalo by se:

    - V cyklu by se opět procházely všechny strany, ty by byly proloženy vždy přímkou p.
    - Přímce p by se našel nejvzdálenější bod, vedla by se jím přímka q || p a přímka k kolmá na p a q.
    - Přímce k by se našel nejvzdálenější bod vlevo, jím by se vedla přímka kl kolmá na p a q.
    - Přímce k by se našel nejvdálenější bod vpravo, jím by se vedla přímka kp kolmá na p a q.
    - Našly by se průsečíky přímek p a kl, p a kp, q a kl, q a kp jako souřadnice rohů ohraničujícího obdélníku.
    - Dále by byl postup totožný s postupem v našem programu (testování velikosti obsahu ohraničujícího obdélníku).

- Optimalizace velikosti vykreslovaných značek bodů vzhledem k počtu vykreslených bodů (tak aby byla lépe vidět konvexní obálka).

12 Zdroje

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