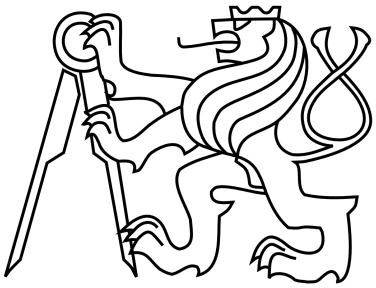
**České vysoké učení technické v Praze**
Fakulta stavební



**155ADKG Algoritmy v digitální kartografii**

# Konvexné obálky a ich konštrukcie

Bc. Lukáš Kettner Bc. Martin Hulín
5.11.2019

# Obsah

# 1 Zadanie

Vytvorte aplikáciu s grafickým rozhraním, ktorá vygeneruje konvexnú obálku podľa zvoleného typu algoritmu. Vstup do aplikácie : množina bodov {p1, . . . , pn} . Výstup aplikácie : konvexná obálka $H$(P).

Nad množinou $P$ implementujte nasledujúce algoritmy pre konštrukciu $H$(P).

- Jarvis Scan
- Quick Hull
- Sweep Line

Vstupné množiny vrátane vygenerovaných konvexných obálok vhodne vizualizujte. Vytvorte grafy pre množiny n $\in \langle$ 1000, 1000 000 $\rangle$ ilustrujúce doby behu algoritmu. Meranie prevádzajte pre rôzne typy množín opakovane 10x a uveďte rozptyl. Namerané údaje usporiadajte do tabuliek.

Taktiež sa zamyslite nad problémom singularít pre rôzne typy vstupných množín a nad možnými optimalizáciami. Zhodnoťte dosiahnuté výsledky. Rozhodnite, ktorá z týchto metód je vzhľadom na časovú náročnosť a typ vstupnej množiny najvhodnejšia.

## 1.1 Bonusové úlohy

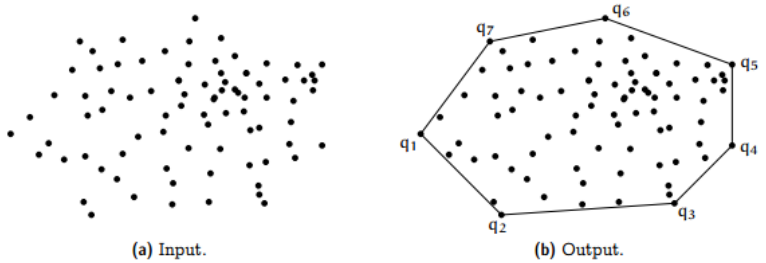
V rámci úlohy sú vypracované tieto bonusové úlohy

- Konštrukcia konvexnej obálky metódou Graham Scan.
- Ošetrenie singulárneho prípadu pro Jarvis Scan
- Konštrukcia striktne konvexných obálok
- Algoritmus pre automatické generovanie konvexných / nekonvexných množín

# 2 Popis a rozbor problému

Konvexná obálka je skupina bodov, ktorých spojením vznikne ohraničenie pre všetky ostatné body množiny. Pre konvexnú obálku platí :

- žiadny bod vstupnej množiny $P$ neleží mimo ohraničenia konvexnej obálky.
- všetky spojnice bodovvstupnej množiny $P$ ležia vnútri konvexnej obálky alebo tvoria jej ohraničenie.
- všetky vnútorné uhly medzi susednými segmentami konvexnej obálky sú menšie ako 180 stupňov.




Obr. 1: Princíp konvexnej obálky

Konvexnú obálku je možné zostrojiť viacerými metódami. V rámci našej práce sme zostrojili algoritmus Jarvis Scan, Quick Hull, Sweep Line a Graham scan.

# 3 Popis použitých algoritmov

## 3.1 Jarvis Scan

Tento algoritmus predstavuje jeden z najpoužívanejích postupov pre tvorbu konvexnej obálky. V algoritme sa zavádza kritérium maximálneho uhlu ($\omega_{max}$). Pri tomto kritériu posudzujeme uhol medzi poslednou stranou konvexnej obálky a úsečkou posledný bod obálky - aktuálny bod. Kritérium hladá taký aktuálny bod, pre ktorý je uhol ($\omega$) maximálny. Postup je nasledovný. V prvom kroku vyberieme pivot q so súradnicou $y_{min}$, tento bod je automaticky súčasťou konvexnej obálky. Zavádzame kritérium maximálneho uhlu ($\omega_{max}$). Keď nájdeme bod odpovedajúci kritériu, prídáme takýto bod do konvexnej obálky. Algoritmus končí v momente keď sa dostaneme opäť k pivotu q.

### 3.1.1 Implementácia metódy

1. Nájdi pivot q. Zoraď body podľa súradnice y. $q : y_{min}$
2. Do konvexnej obálky pridaj $q$
3. Inicializuj $p_{j-1} \in X, p_j = q, p_{j+1} = p_{j-1}$
4. Opakuj pokial $p_{j+1}! = q$
5. Nájdenie bodu pre ktorý je uhol omega maximálny $p_{j+1} = argmax\ p_i \in P \angle(p_{j-1}, p_j, p_i)$
6. Pridaj nájdený bod $p_{j+1}$ do konvexnej obálky
7. Inicializuj $p_{j-1} = p_j, p_j = p_{j+1}$

### 3.1.2 Problematické situácie

Problematickou situáciou sú vyskytujúce sa kolineárne body. V takomto prípade je potrebné ošetriť aby sa do konvexnej obálky dostal najvzdialenejší bod.

### 3.1.3 Implementácia riešenia problému

1. Cyklus $for(inth = 0; h < ch.size() - 1; h + +)$ - prechádame všetky body konvexnej obálky

2. Podmienka totožných súradníc
   $if((ch[h].x() == ch[h+1].x())\&\&(ch[h].y() == ch[h+1].y()$

3. Odstráň bod s totožnými súradnicami $ch.remove(h)$

4. $h--$

## 3.2 Quick Hull

Tento algoritmus uplatňuje pri konštrukcii konvexnej obálky princím rozdeľuj a panuj. Jedná sa rekurzívny algoritmus. Skladá sa z lokálnej a globálnej procedúry. Konvexná obálke je skonštruovaná z dvoch častí - upper hull a lower hull. Deliacou priamkou je spojnica dvoch bodov s extrémnymi súradnicami x (minimum, maximum). S oboma časťami obálky sa počas výpočtu pracuje samostatne. Algoritmus hľadá bod s extrémnymi súradnicami v ose y, samostatne pre hornú a dolnú časť. Výsledná konvexná obálka je pri zachovaní CCW orientácie zložená z dvoch častí automaticky, bez nutnosti spájania.

### 3.2.1 Implementácia metódy

1. Vytvor mnošinu konvexnej obálky, vektor vrchných *upoints* a spodných bodov *lpoints*

2. Zoraď vstupnú množinu bodov podľa súradnice X

3. Nájdi body $q_1$ , $q_3$ s extrémnymi súradnicami X

4. Pridaj $q_1$ , $q_3$ do *upoints* a *lpoints*

5. Rozdeľ všetky body vstupnej množiny podľa kritéria do *upoints* alebo *lpoints*

6. Do konvexnej obálky pridaj bod $q_3$

7. Volaj rekurzívnu funkciu - nájde najvzdialenejší bod v *upoints*

8. Pridaj tento bod do konvexnej obálky

9. Do konvexnej obálky pridaj bod $q_1$

10. Volaj rekurzívnu funkciu - nájde najvzdialenejší bod v *lpoints*

11. Pridaj tento bod do konvexnej obálky

12. Vráť konvexnú obálku.

## 3.3 Sweep Line

Jedná sa o metódu zametacej priamky. Tento algoritmus pracuje na princípe prepisovania pozície predchodcu a následníka i-teho bodu. Popis algoritmu slovne je miestami zložité pochopiť a pri jeho prezentácii pred osobou mimo obor by sme mohli skončiť na psychiatrii. Implemntácia metódy jednoduchšie zobrazí jej fungovanie.

### 3.3.1 Implementácia metódy

1. Zoradenie množiny bodov $P_s$ podľa súradnice 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]];

### 3.3.2 Problematické situácie

Problematická situácia pri algoritme Sweep Line nastáva, ak sa vo vstupnej množine bodov nachádzajú duplicitné body. Tie je potrebné odstrániť a až po ich odstránení určiť veľkosť vektoru bodov vstupnej množiny, vytvoriť zoznam predchodcov, následníkov a v cykle spustiť algoritmus.

### 3.3.3 Implementácia riešenia problému

1. Odstránenie duplicitných bodov po zoradení podľa osy X
   Cyklus $for(unsignedinti = 0; i < points.size()-1; i++)$ - prechádame celý vektor bodov

2. Podmienka totožných súradníc
   $if((points[i].x()! = points[i+1].x())||(points[i].y()! = points[i+1].y()$

3. Ulož bod, ktorý nie je duplicitný $noDuplicitPoints.push\_back(points[i])$

4. $noDuplicitPoints.push\_back(points[points.size() - 1]$

5. $points = noDuplicitPoints$

## 3.4 Graham scan

Grahamov prehľadávací algoritmus využíva kritérium pravotočivosti, pri ktorom posudzuje uhol ($\omega_i$). Množina bodov je zoradená podľa súradnice y. Bod s najmenšou y súradnicou je zvolený za pivota. Z pivota vedieme rovnobežku s osou X. Následne určíme uhol od osi X k ostatným bodom množiny. Tieto uhly je potrebné zotiediť podľa veľkosti. Následne vyberieme bod ($p_i$) pre ktorý je uhol ($\omega_i$) maximálny. Pri pridávani bodu do konvexnej obálky musí byť splnená podmienka ľavotočivosti.

### 3.4.1 Implementácia metódy

1. Zoradenie množiny bodov $P_s$ podľa súradnice y

2. Nájdenie pivota q, $q = min(y_i)$ , q $\rightarrow$ Convex Hull

3. Zoraď body podľa veľkosti uhlu $(\omega_i)$ $\angle(p_{kolinearbyx}, x, p_i)$
4. Vymaž bližší bod ku q $if(\omega_i = \omega_j)$
5. Opakuj pre všetky body j <n
6. Pridaj bod $p_j$ pre ktorý je uhol omega maximálny a je splnená podmienka ľavotočivosti → Convex Hull
7. j = j+1
8. Else pop S

### 3.4.2 Problematické situácie

Problematická situácia pri algoritme Graham Scan nastáva, ak sa vo vstupnej množine bodov nachádzajú body, ktoré zvierajú s osou X a pivotom rovnaký uhol $(\omega_i)$ . Je potrebné určiť vzdialenosť takýchto bodov od pivotu a ponechať v množine bodov len bod s najväčšou vzdialenosťou od pivotu.

### 3.4.3 Implementácia riešenia problému

1. Odstránenie bližších bodov k pivotu pri totožnom uhle $(\omega_i)$
   Cyklus $for(unsignedinti = 0; i < chGS.size() - 2; i++)$ prechádame celú konvexnú obálku
2. Podmienka totožného uhlu $(\omega_i)$
   $if(getPointLineDistance(chGS[i], chGS[i+1], chGS[i+2]) == -1)$
3. Odstráň bližší bod $chGS.erase(chGS.begin() + (i+1))$
4. $i = i - 1$

Odstránenie dupliciných bodov

1. Cyklus $for(inth = 0; h < chGS.size() - 1; h++)$ - prechádame všetky body konvexnej obálky
2. Podmienka totožných súradníc
   $if((chGS[h].x() == chGS[h+1].x())\&\&(chGS[h].y() == chGS[h+1].y()$

3. Odstráň bod s totožnými súradnicami $chGS.remove(h)$

4. $h--$

# 4 Strikne konvexné obálky

Striktne konvexná obálka musí spĺňať podmienku : neobsahovať 3 nasledujúce body na jednej priamke. Pre ošetrenie tohto prípadu sme implementovali funkciu, ktorá takéto body odstránila, taktiež sme vymazali duplicitné body.

# 5 Vstupné dáta - generovanie bodov

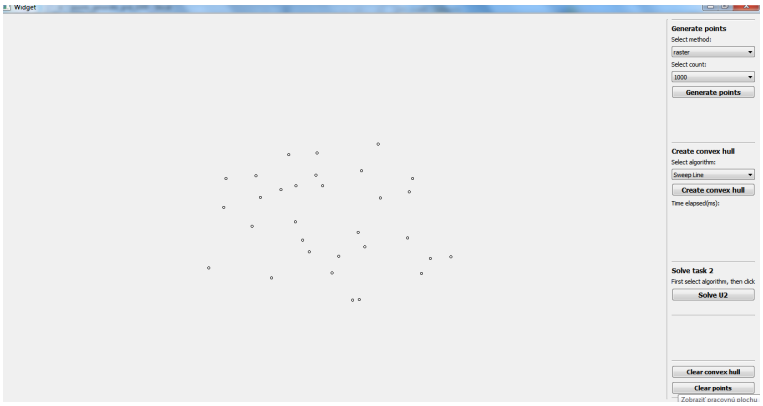
Vstupnými dátami je množina bodov. Táto množina môže byť ručne naklikaná v Canvase alebo automaticky generovaná.

Generovanie je možné previeť do rastru, kruhového tvaru alebo náhodného usporiadania množiny bodov. Oblasť generovania je určená z aktuálnej veľkosti okna aplikácie. Od každej hrany okna je pevne nastavený odstup. Samotné náhodné generovanie je realizované pomocou $std :: random\_device$ s pouižitím generátoru $std :: mt19937$.

Raster je generovaný s pevným krokom pre šírku a dĺžku okna, ktorý sa počíta na základe požadovaného počtu bodov. Náhodné generovanie sa prevádza pomocou príkazu $std :: uniform\_int\_distibution$, ktorý generuje celé čísla v určitom rozsahu. Pre možnosť circle – kruhového generovania bodov sú body generované polárnymi súradnicami s použitím rovnakého náhodného generátoru ako v predchádzajúcom prípade.

## 5.1 Množina bodov vytvorená ručne

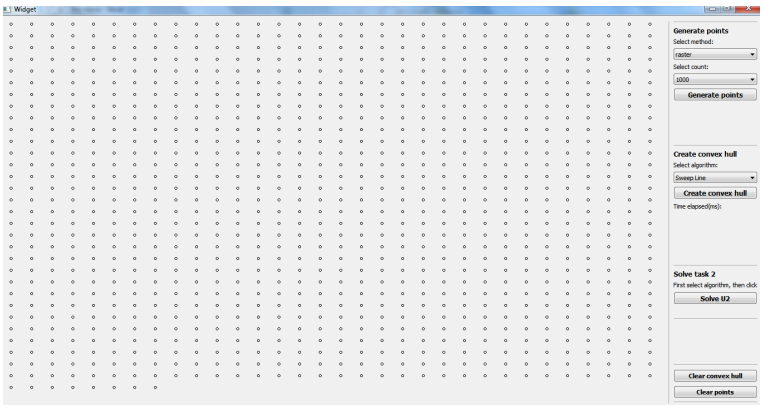
Táto množina vznikne ručným naklikaním bodov priamo v Canvase.



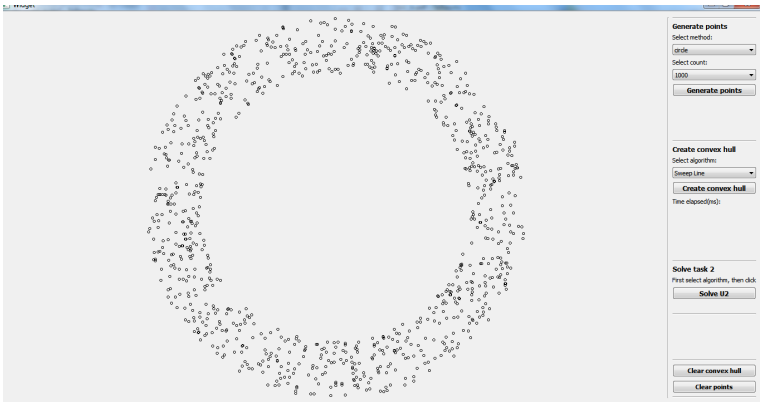
Obr. 2: Manuálne zadané body a vygenerová konvexná obálka

## 5.2 Automaticky generovaná množina bodov

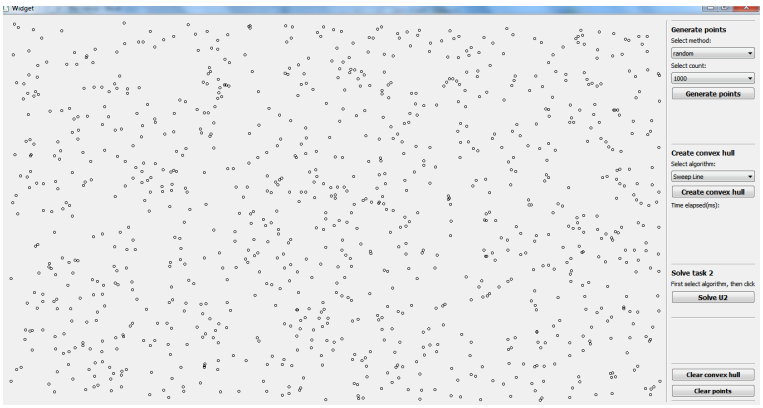
Množina vznikne automatickým vygenerovaním po zadaní požadovaného počtu generovaných bodov. Rozsah generovaných bodov je n $\in \langle$ 1000, 1000 000 $\rangle$ . Tvary generovaných bodov sú raster, kruh, náhodne generované body.




Obr. 3: Automaticky generovaných 1000 bodov v tvare rastru.



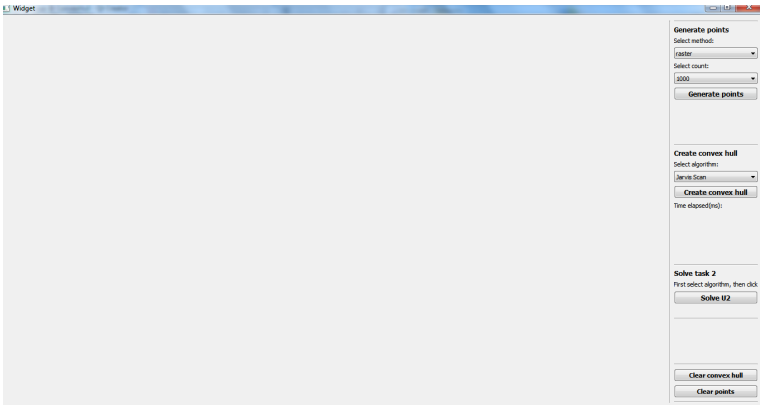

Obr. 4: Automaticky generovaných 1000 bodov v tvare kruhu.




Obr. 5: Automaticky generovaných 1000 bodov v náhodnom tvare.

# 6 Ukážka vytvorenej aplikácie

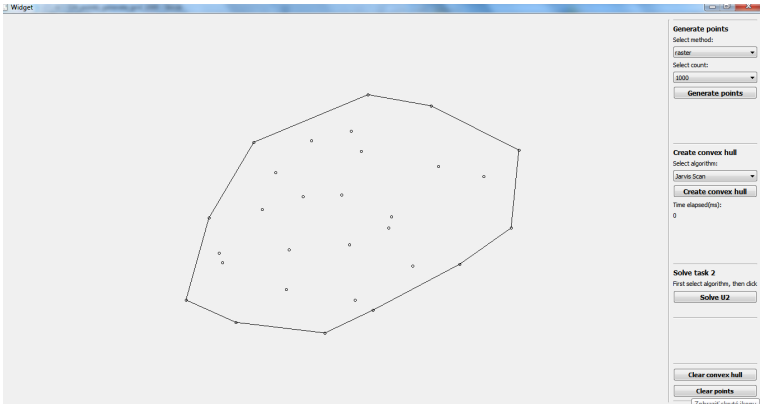
Grafické rozhranie aplikácie, vo svojej pravej časti, obsahuje možnosť generovania počtu bodov v užívateľom zvolenom tvare, vytvorenie konvexnej obálky pomocou zvoleného algoritmu. Pod tlačítkom *Create convex hull* sa nachádza label v ktorom sa zobrazí doba trvania vygenerovania konvexnej obálky. Ďalej grafické rozhranie obsahuje tlačítko *Solve U2*. Po jeho stlačení sa zvoleným algoritmom prevedie generovanie konvexnej obálky automaticky. Toto generovanie prebieha automaticky 10x pre každý typ rozmiestnenia bodov ( raster, kruh, náhodné rozmiestnenie bodov) a počet bodov n $\in \langle$ 1000, 5000, 10 000, 25 000, 50 000, 75 000, 100 000, 250 000, 500 000, 750 000, 1000 000 $\rangle$ . Pre každé generovanie konvexnej obálky je počítaná doba behu, ktorá sa spolu s počtom a typom generovaných bodov, typom algoritmu a poradím opakovania je ukladaná do textového súboru. Tento textový súbor je následne spracovaný v programe R Studio. Výsledné vytvorené grafy s porovnaním časovej náročnosti pri n počte vstupných bodov sú uvedené v kapitole *Grafy a tabuľky*.



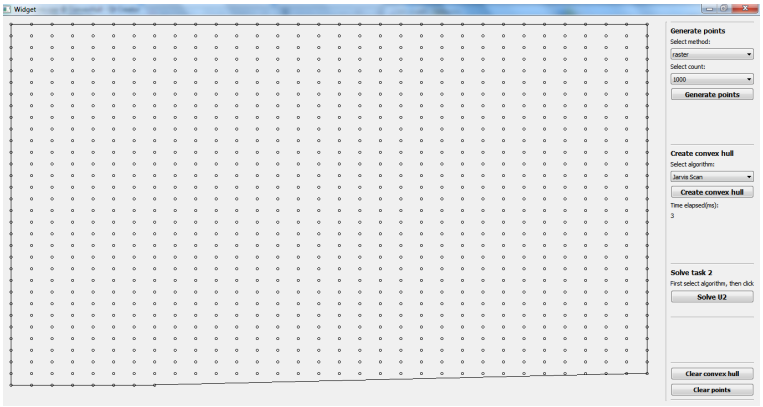

Obr. 6: Ukážka grafického rozhrania aplikácie

# 7 Vykreslenie konvexnej obálky

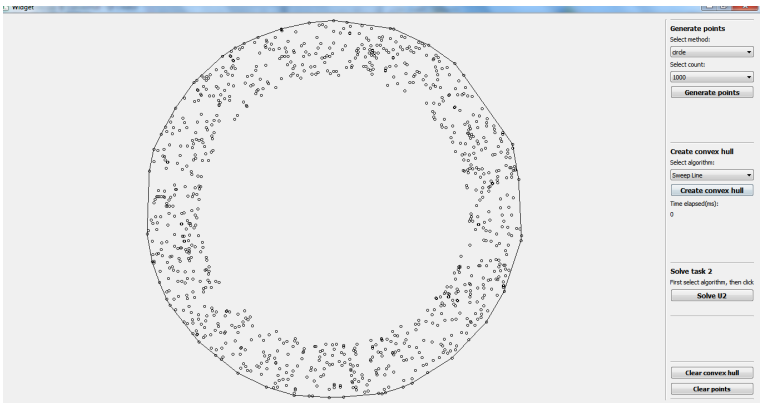
Po automatickom vygenerovaní bodov, prípadne ručnom naklikaní v Canvase , voľbe algoritmu konštrukcie konvexnej obálky a stlačení tlačidla *Create convex hull* dostane užívateľ vygenerovanú konvexnú obálku. Pri genererovaní väčšieho počtu bodov ako $n = 5000$ sa v grafickom rozhraní vykreslí len prvých 5000 bodov, z dôvodu rýchlejšieho a jednoduchšieho vykreslovania. Konvexná obálka sa vygeneruje v plnom rozsahu.



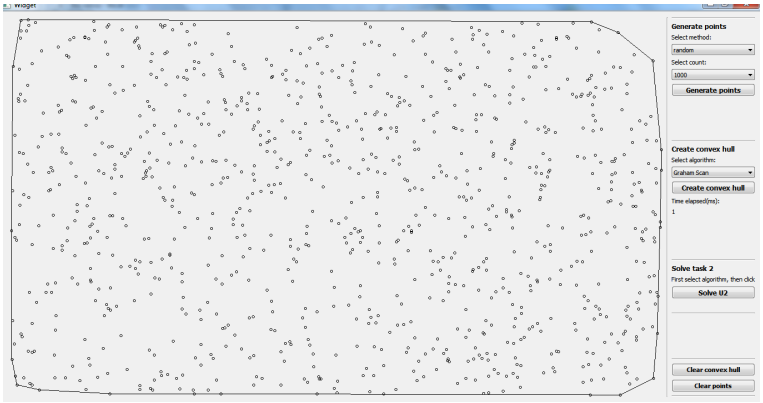

Obr. 7: Manuálne zadané body a vygenerová konvexná obálka



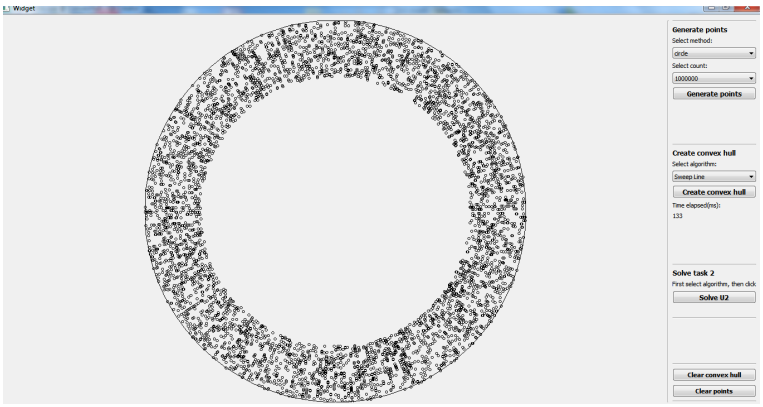

Obr. 8: Automaticky generovaných 1000 bodov v tvare rastru, konvexná obálka bodov.




Obr. 9: Automaticky generovaných 1000 bodov v tvare kruhu, konvexná obálka bodov.



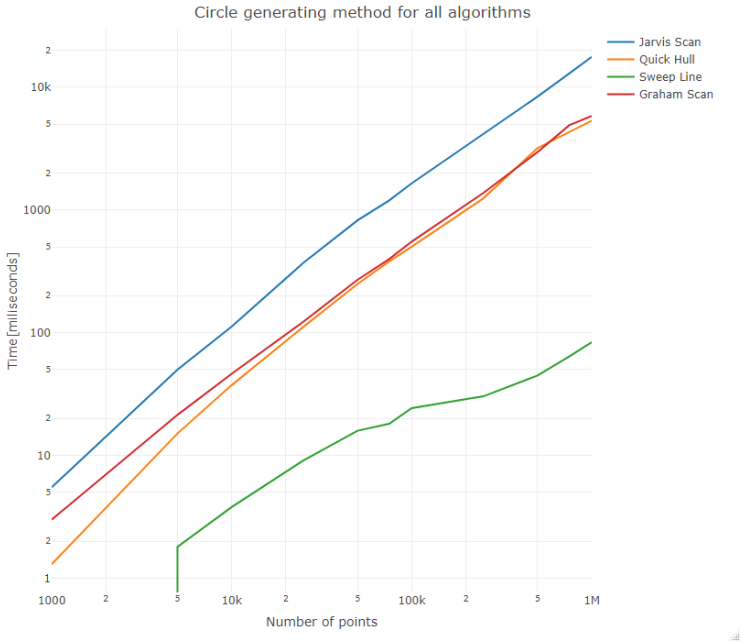

Obr. 10: Automaticky generovaných 1000 bodov v náhodnom tvare, konvexná obálka bodov.



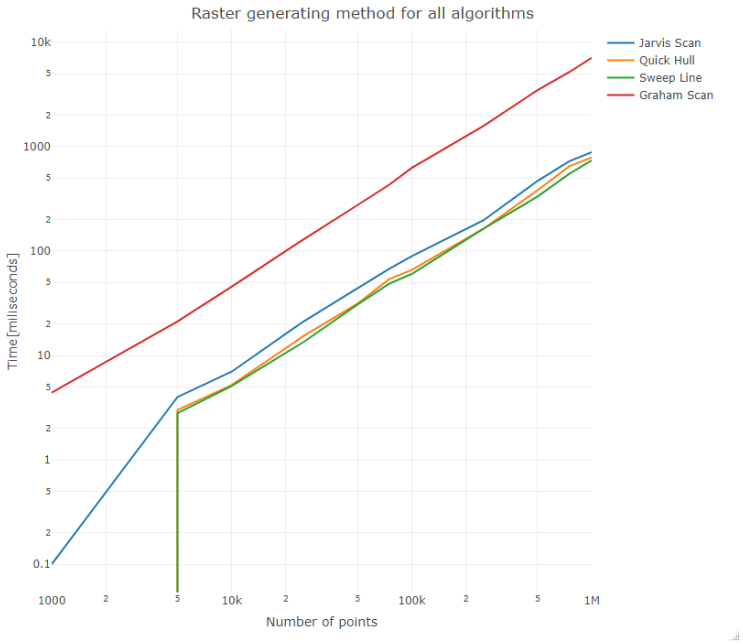

Obr. 11: Automaticky generovaných 1 000 000, konvexná obálka Sweep Line.

# 8 Grafy - doby behu algoritmov

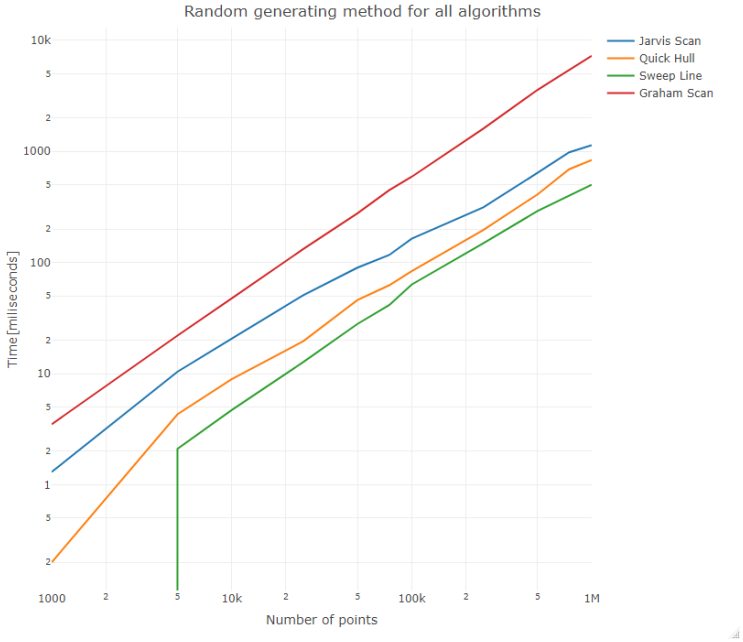
Súčasťou úlohy je porovnanie doby behu jednotlivých algoritmov pre testované množiny bodov generované v tvare kruhu, rastru a náhodne. Prezentácia výsledkov je pomocou grafov vytvorených v programe R Studio. Testovanie bolo vykonané 10x pre každý počet bodov z intervalu a tvar generovanej množiny bodov. Vykonali sme ho v režime Relase.



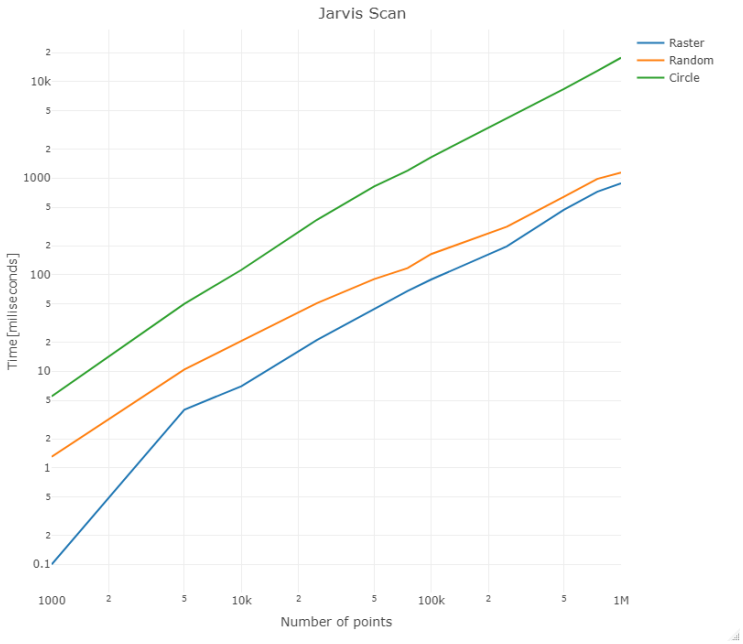

Obr. 12: Doba behu jednotlivých algoritmov pre množinu bodov v tvare kruhu.



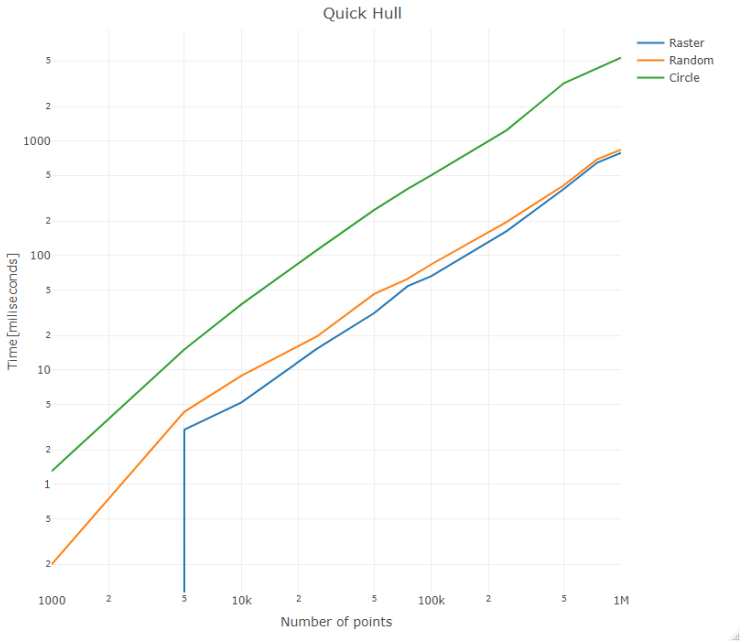

Obr. 13: Doba behu jednotlivých algoritmov pre množinu bodov v tvare rastru.



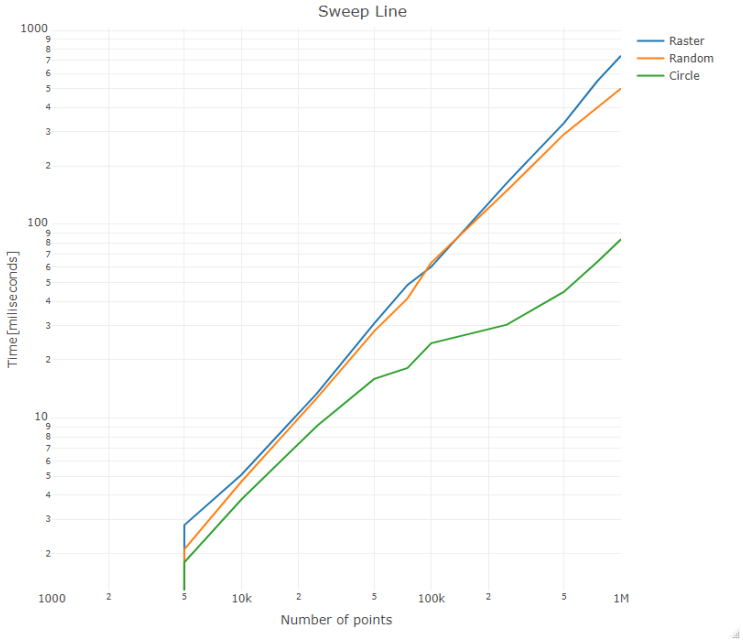
Obr. 14: Doba behu jednotlivých algoritmov pre množinu bodov v náhodnom tvare.



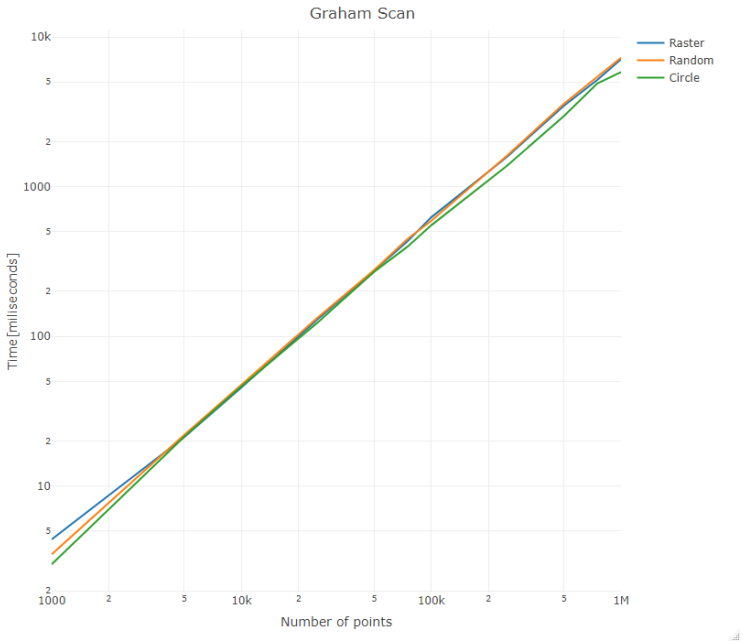

Obr. 15: Doba behu algoritmu Jarvis Scan pre množiny bodov vo všetkých generovaných tvaroch.




Obr. 16: Doba behu algoritmu Quick Hull pre množiny bodov vo všetkých generovaných tvaroch.



Obr. 17: Doba behu algoritmu Sweep Linel pre množiny bodov vo všetkých generovaných tvaroch.




Obr. 18: Doba behu algoritmu Graham Scan pre množiny bodov vo všetkých generovaných tvaroch.

# 9 Tabulky pro jednotlivé metody

Tabuľky s presnou dobou behu jednotlivých algoritmov pre množiny bodov. Tabuľky sú rozdelené po jednotlivých algoritmoch a každá obsahuje priemerný čas generovania konvexnej obálky v jednotkách milisekúnd určený z desiatich pokusov a rozptyl. Rozptyl bol určený ako prostý rozdiel najdlhšieho a najkratšieho času výpočtu obálky z desiatich pokusov.

Tabuľka 1: Jarvis Scan data z výpočtu

| **Jarvis Scan** | | | |
|---|---|---|---|
| points | method | avg elapsed time [ms] | variance [ms] |
| 1000 | raster | 0.1 | 1 |
| 5000 | raster | 4 | 0 |
| 10000 | raster | 7 | 0 |
| 25000 | raster | 21.1 | 1 |
| 50000 | raster | 43.9 | 3 |
| 75000 | raster | 67.6 | 6 |
| 100000 | raster | 89.2 | 3 |
| 250000 | raster | 196.3 | 3 |
| 500000 | raster | 469.5 | 6 |
| 750000 | raster | 722.4 | 4 |
| 1000000 | raster | 882 | 55 |
| 1000 | random | 1.3 | 1 |
| 5000 | random | 10.4 | 1 |
| 10000 | random | 20.6 | 2 |
| 25000 | random | 50.8 | 31 |
| 50000 | random | 89.9 | 10 |
| 75000 | random | 116.8 | 14 |
| 100000 | random | 163.7 | 26 |
| 250000 | random | 313.5 | 42 |
| 500000 | random | 641.3 | 78 |
| 750000 | random | 981.1 | 108 |
| 1000000 | random | 1138.4 | 199 |
| 1000 | circle | 5.5 | 1 |
| 5000 | circle | 49.9 | 5 |
| 10000 | circle | 112.2 | 20 |
| 25000 | circle | 370.3 | 56 |
| 50000 | circle | 821.7 | 88 |
| 75000 | circle | 1195.5 | 32 |
| 100000 | circle | 1647.4 | 14 |
| 250000 | circle | 4167.5 | 86 |
| 500000 | circle | 8373.2 | 452 |
| 750000 | circle | 12913.8 | 833 |
| 1000000 | circle | 17651.2 | 1077 |

Tabuľka 2: Quick Hull data z výpočtu

| **Quick Hull** | | | |
|---|---|---|---|
| points | method | avg elapsed time [ms] | variance [ms] |
| 1000 | raster | 0 | 0 |
| 5000 | raster | 3 | 0 |
| 10000 | raster | 5.2 | 2 |
| 25000 | raster | 15.3 | 4 |
| 50000 | raster | 31.3 | 7 |
| 75000 | raster | 53.8 | 25 |
| 100000 | raster | 65.8 | 15 |
| 250000 | raster | 163 | 43 |
| 500000 | raster | 381.6 | 199 |
| 750000 | raster | 647.3 | 138 |
| 1000000 | raster | 787.4 | 323 |
| 1000 | random | 0.2 | 1 |
| 5000 | random | 4.3 | 1 |
| 10000 | random | 8.9 | 2 |
| 25000 | random | 19.6 | 5 |
| 50000 | random | 46 | 19 |
| 75000 | random | 62.4 | 14 |
| 100000 | random | 83.5 | 16 |
| 250000 | random | 196.5 | 45 |
| 500000 | random | 410.2 | 101 |
| 750000 | random | 691.6 | 202 |
| 1000000 | random | 838.5 | 252 |
| 1000 | circle | 1.3 | 1 |
| 5000 | circle | 15.1 | 1 |
| 10000 | circle | 37.4 | 3 |
| 25000 | circle | 111.4 | 21 |
| 50000 | circle | 249.2 | 60 |
| 75000 | circle | 380.6 | 59 |
| 100000 | circle | 501.8 | 85 |
| 250000 | circle | 1242.6 | 61 |
| 500000 | circle | 3192.4 | 2496 |
| 750000 | circle | 4293.5 | 462 |
| 1000000 | circle | 5331.2 | 618 |

Tabuľka 3: Sweep Line data z výpočtu

| **Sweep Line** | | | |
|---|---|---|---|
| points | method | avg elapsed time [ms] | variance [ms] |
| 1000 | raster | 0 | 0 |
| 5000 | raster | 2.8 | 1 |
| 10000 | raster | 5.1 | 1 |
| 25000 | raster | 13.4 | 4 |
| 50000 | raster | 30.7 | 8 |
| 75000 | raster | 48.6 | 17 |
| 100000 | raster | 60.3 | 17 |
| 250000 | raster | 163.1 | 41 |
| 500000 | raster | 332.3 | 87 |
| 750000 | raster | 547.1 | 201 |
| 1000000 | raster | 739.9 | 243 |
| 1000 | random | 0 | 0 |
| 5000 | random | 2.1 | 1 |
| 10000 | random | 4.7 | 2 |
| 25000 | random | 12.7 | 4 |
| 50000 | random | 28 | 11 |
| 75000 | random | 41.4 | 19 |
| 100000 | random | 63.3 | 18 |
| 250000 | random | 148.7 | 59 |
| 500000 | random | 290.7 | 172 |
| 750000 | random | 398.7 | 248 |
| 1000000 | random | 501.1 | 414 |
| 1000 | circle | 0 | 0 |
| 5000 | circle | 1.8 | 1 |
| 10000 | circle | 3.8 | 3 |
| 25000 | circle | 9.1 | 7 |
| 50000 | circle | 15.9 | 18 |
| 75000 | circle | 18.1 | 30 |
| 100000 | circle | 24.3 | 63 |
| 250000 | circle | 30.3 | 131 |
| 500000 | circle | 44.8 | 274 |
| 750000 | circle | 63.9 | 445 |
| 1000000 | circle | 83.6 | 649 |

Tabuľka 4: Graham Scan data z výpočtu

| **Graham Scan** | | | |
|---|---|---|---|
| points | method | avg elapsed time [ms] | variance [ms] |
| 1000 | raster | 4.4 | 2 |
| 5000 | raster | 21.2 | 1 |
| 10000 | raster | 45.5 | 2 |
| 25000 | raster | 128.9 | 5 |
| 50000 | raster | 275.3 | 47 |
| 75000 | raster | 431.3 | 16 |
| 100000 | raster | 623.9 | 108 |
| 250000 | raster | 1575.6 | 104 |
| 500000 | raster | 3472.2 | 238 |
| 750000 | raster | 5160.9 | 166 |
| 1000000 | raster | 7088.7 | 1641 |
| 1000 | random | 3.5 | 1 |
| 5000 | random | 22 | 0 |
| 10000 | random | 47.5 | 1 |
| 25000 | random | 132.2 | 5 |
| 50000 | random | 277.5 | 11 |
| 75000 | random | 447.2 | 83 |
| 100000 | random | 591.1 | 83 |
| 250000 | random | 1602.9 | 86 |
| 500000 | random | 3579.8 | 1230 |
| 750000 | random | 5402.5 | 2263 |
| 1000000 | random | 7262.5 | 946 |
| 1000 | circle | 3 | 0 |
| 5000 | circle | 21.4 | 1 |
| 10000 | circle | 46.3 | 1 |
| 25000 | circle | 122.7 | 1 |
| 50000 | circle | 269.5 | 38 |
| 75000 | circle | 397.4 | 7 |
| 100000 | circle | 551.6 | 122 |
| 250000 | circle | 1374.7 | 38 |
| 500000 | circle | 2958.7 | 670 |
| 750000 | circle | 4889 | 2046 |
| 1000000 | circle | 5823.7 | 190 |

# 10 Dokumentácia

## 10.1 Trieda Algorithms

Triedu Algorithms sme použili pre deklarovanie funkcií pre výpočtové algoritmy tvorby konvexnej obálky a pre deklaráciu ich pomocných metód.

### 10.1.1 Členské premenné

static QPoint pivot

- bod obsahujúci súradnice pivota

QPoint kolinear X point

- bod obsahujúci súradnice pomocného bodu, kolineárneho s osou X

### 10.1.2 Metódy

getAngle2Vectors

- Slúži k určeniu uhlu medzi dvoma priamkami. Jej návratovou hodnotou je double
- na vstupe má : súradnice bodov $p_1, p_2, p_3, p_4$ určujúcich prvú a druhú priamku
- výstupom je hodnota uhlu medzi priamkami

getPointLinePosition

- Slúži na určenie polohy bodu voči priamke. Jej návratovou hodnotou je integer.
- na vstupe má : súradnice určovaného bodu q , súradnice bodov priamky $p_1$ $p_2$
- na výstupe hodnoty :

  -1 – bod sa nachádza na priamke

  0 – bod sa nachádza vpravo od priamky

  1 – bod sa nachádza vľavo od priamky

getPointLineDistance

- je pomocnou metódou pre metódu positionPointPolygonWinding. Slúži na určenie polohy bodu voči priamke. Jej návratovou hodnotou je double.
- na vstupe má : súradnice určovaného bodu q , súradnice bodov priamky $p_1$ $p_2$
- na výstupe hodnotu vzdialenosti určovaného bodu od priamky.

Distance

- Slúži na porovnanie vzdialenosti dvoch bodov.
- na vstupe má : súradnice bodov $p_1$ a $p_2$
- na výstupe hodnotu vzdialenosti medzi dvoma bodmi.

Angle

- Slúži na porovnanie veľkosti dvoch uhlov.
- na vstupe má : súradnice bodov $p_1$ a $p_2$
- na výstupe väčší uhol.

jarvisScan, qHull, grahamScan, sweepLine

- Metódy slúžia k výpočtu konvexnej obálky podľa zvoleného výpočetného algoritmu. Vstupným typom je QPolygon
- na vstupe je vektror bodov std::vector<QPoint>points
- na výstupe je polygón obsahujúci množinu bodov a konvexnú obálku.

qh

- Metódy slúžia k výpočtu konvexnej obálky podľa algoritmu Quick Hull. Je to pomocná metóda. Vstupným typom je void.
- na vstupe je index počiatočného a koncového bodu deliacej priamky start, end
- std::vector<QPoint>points vektor bodov okolo ktorých vytvárame konvexnú obálku
- QPolygon convexHull ktorý obsahuje body konvexnej obálky
- na výstupe je polygón obsahujúci konvexnú obálku.

## 10.2 Trieda Draw

Trieda Draw slúži ku grafickému vykresleniu množiny modov a konvexnej obálky nad touto množinou.

### 10.2.1 Členské premenné

std::vector<QPoint>points

- vektor bodov okolo ktorých vytvárame konvexnú obálku

QPolygon ch

- polygón obsahujúci body konvexnej obálky

### 10.2.2 Metódy

paintEvent

- slúži k vykresleniu naklikaných a vygenerovaných bodov, vykresleniu konvexnej obálky. Návratovým typom je void.

void mousePressEvent

- slúži k vykresleniu bodu stlačením tlačidla myši, v okamihu stlačenia tlačidla na myši sa uložia súradnice bodu do vektoru points. Návratovým typom je void.

void clearCH

- slúži k vymazaniu konvexnej obálky. Návratovým typom je void.

void clearPoints

- slúži k vymazaniu množiny bodov. Návratovým typom je void.

std::vector<QPoint>getPoints

- vektor, ktorý slúži k vráteniu množiny bodov points.

setCH

- slúži na prevedenie konvexnej obálky do vykresľovacieho okna.

generatePoints

- slúži ku generovaniu množiny bodov. Na vstupe je zadaná metóda, počet bodov, šírka a výška.
- na výstupe je vygenerovaná množina bodov podľa užívateľského zadania.

std::vector<QPoint>generatePointsU2

- slúži ku generovaniu množiny bodov. Na vstupe je zadaná metóda, počet bodov, šírka a výška.
- na výstupe je vygenerovaná množina bodov podľa užívateľského zadania. Generovanie konvexnej obálky prebehne automaticky 10x pre všetky typy tvaru generovanej množiny bodov a počty generovaných bodov v intervale od 1 000 do 1 000 000.

## 10.3 Tridy SortByX, SortByY

Sú to triedy, ktoré obsahujú zoraďovacie kritériá. Pomocou týchto funkcií zoradíme súbor bodov podľa X alebo podľa Y súradnice.

## 10.4 Trieda Widget

Tieda Widget obashuje metódy ktoré sú odkazom na sloty umožňujúce vykonávať príkazy z grafického rozhrania aplikácie. Nemajú žiadne vstupné hodnoty, návratovým typom je void.

### 10.4.1 Metódy

on_pushButton_createCH_clicked

- tlačidlo **Create convex hull** po kliknutí naň sa vygeneruje konvexná obálka množiny bodov

on_pushButton_clearPoints_clicked

- tlačidlo **Clear points** po kliknutí naň sa vymaže množina bodov

on_pushButton_clearCH_clicked

- tlačidlo **Clear convex hull** po kliknutí naň sa vymaže konvexná obálka

on_pushButton_generatePoints_clicked

- tlačidlo **Generate points** po kliknutí naň sa vygenerujú body v zvolenom tvare a počte

on_pushButton_solveU2_clicked

- tlačidlo **Solve U2** po kliknutí naň sa vykoná automatické generovanie konvexnej obálky 10x pre každý typ rozmiestnenia bodov ⟨ raster, kruh, náhodné rozmiestnenie bodov) a počet bodov n ∈ ⟨ 1000, 5000, 10 000, 25 000, 50 000, 75 000, 100 000, 250 000, 500 000, 750 000, 1000 000 ⟩ . Pre každé generovanie konvexnej obálky je počítaná doba behu, ktorá sa spolu s počtom a typom generovaných bodov, typom algoritmu a poradím opakovania je ukladaná do textového súboru

# 11 Záver

Výsledkom úlohy je funkčná aplikácia a grafická prezentácia doby trvania jednotlivých algoritmov pre rôzne množiny bodov. Po vypracovaní úlohy sme došli k nasledujúcemu záveru. Najvhodnejší algoritmus pre generovanie konvexných obálok je Sweep Line. Táto metóda dosiahla najlepšie výsledky vo všetkých testovacích prípadoch. Obzvlásť výrazný časový rozdie oproti ostatným algoritmoml bol pri zväčšujúcej sa množine bodov usporiadanej v kruhovom tvare. Algoritmy Graham Scan a Jarvis Scan sú o poznanie pomalšie, hlavne pri rastri a náhodne generovanej množine bodov. Algoritmus Quick Hull dosahuje pre rastrovú a náhodnú množinu bodov takmer totožnú rýchlosť ako algoritmus Sweep Line, pri kruhovom rozložení bodov je pomalší, obdobne rýchly ako Graham Scan.

V úlohe sme sa neimplementovali generovanie množiny bodov v tvare elipsy a star - shaped rozloženia. Bolo by zaujímavé porovnať doby behov algoritmov aj pre takéto rozloženia množín bodov.

# Zoznam obrázkov



# Zoznam tabuliek