ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

FAKULTA STAVEBNÍ

KATEDRA GEOMATIKY

název předmětu

ALGORITMY V DIGITÁLNÍ KARTOGRAFII

úloha

U2: KONVEXNÍ OBÁLKA

datum	vypracovali
14.11.2020	Bc. Josef Műnzberger, Bc. Martin Hudeček

Úloha č. 2: Konvexní obálky a jejich konstrukce

Vstup: množina $P = \{p_1, ..., p_n\}$, $p_i = [x, y_i]$.

Výstup: $\mathcal{H}(P)$.

Nad množinou P implementujete následující algoritmy pro konstrukci $\mathcal{H}(P)$:

- Jarvis Scan,
- Quick Hull,
- Swep Line.

Vstupní množiny bodů včetně vygenerovaných konvexních obálek vhodně vizualizujte. Pro množiny $n \in < 1000, 1000000 >$ vytvořte grafy ilustrující doby běhu algoritmů pro zvolená n. Měření proveďte pro různé typy vstupních množin (náhodná množina, rastr, body na kružnici) opakovaně (10x) a různá n (nejméně 10 množin) s uvedením rozptylu. Naměřené údaje uspořádejte do přehledných tabulek.

Zamyslete se nad problematikou možných singularit pro různé typy vstupních množin a možnými optimalizacemi. Zhodnoťte dosažené výsledky. Rozhodněte, která z těchto metod je s ohledem na časovou složitost a typ vstupní množiny P nejvhodnější.

Hodnocení:

Krok	**Hodnocení**
Konstrukce konvexních obálek metodami Jarvis Scan, Quick Hull, Seep Line.	15b
Konstrukce konvexní obálky metodou Graham Scan	*+5b*
Konstrukce striktně konvexních obálek pro všechny uvedené algoritmy.	*+5b*
Ošetření singulárního případu u Jarvis Scan: existence kolineárních bodů v datasetu.	*+2b*
Konstrukce Minimum Area Enclosing box některou z metod (hlavní směry budov).	*+5b*
Algoritmus pro automatické generování konvexních/nekonvexních množin bodů různých tvarů (kruh, elipsa, čtverec, star-shaped, popř. další).	*+4b*
Max celkem:	**36b**

Čas zpracování: 3 týdny.

Obsah

1. Popis a rozbor problému

Cílem úlohy je vytvoření konvexní obálky pro libovolnou množinu vygenerovaných bodů. Konvexní obálka představuje nejmenší konvexní mnohoúhelník, který obsahuje všechny body v množině; body se tedy nachází jak uvnitř polygonu, tak v jeho vrcholech či na hranách. Dále platí, že spojíme-li dva libovolné body v množině úsečkou, bude tato úsečka náležet do konvexní obálky (bude ležet uvnitř nebo splývat z jednou z hran polygonu). Konvexní obálky mají široké využití nejen v kartografii, namátkou lze zmínit analýzu tvarů či detekci kolizí dvou objektů.

1.1. Údaje o bonusových úlohách

Mimo hlavní část řešené úlohy (implementace tří algoritmů pro konstrukci konvexních obálek – *Jarvis Scan, Quick Hull* a *Sweep Line*) byla implementována čtvrtá metoda *Graham Scan*, dále byly zavedeny algoritmy pro generování náhodných množin bodů do různých tvarů (elipsa, čtverec) nebo byl ošetřen singulární případ v rámci metody *Graham Scan* při existenci kolineárních bodů v datasetu. U všech uvedených algoritmů byla taktéž zajištěna konstrukce striktně konvexních obálek (odstraněním kolineárních či duplicitních bodů z vytvořené konvexní obálky).

2. Popis použitých algoritmů

Jak již bylo zmíněno výše, aplikace si klade za cíl nejprve vygenerovat množinu bodů podle zadaných kritérií (počet bodů a šablona – náhodná, grid či body na kružnici) a poté sestrojit nad vytvořenou množinou bodů konvexní obálku. Tuto operaci nabízí provést pomocí čtyř různých metod (algoritmy), které se samozřejmě liší přístupem, složitostí a koneckonců i časovou náročností (viz porovnání algoritmů ve části textu věnovaného statistice).

2.1. Jarvis Scan

První popisovaný algoritmus vypracuje konvexní obálku na principu hledání maximálního úhlu. Metoda předpokládá, že v množině vstupních bodů se nevyskytují tři kolineární body (tedy takové tři body, které leží na stejné přímce). Algoritmus není náročný, což nelze tvrdit o jeho časové náročnosti.

Algoritmus si nejprve utřídí všechny body vstupní množiny podle souřadnice Y a jako první bod konvexní obálky zvolí ten s nejmenší souřadnicí Y, obvykle je označován jako pivot. Tímto bodem je vedena rovnoběžka s osou X a zaveden bod *r*, pro který platí, že jeho souřadnice X je menší než souřadnice X pivota (souřadnice Y bodu *r* bude logicky stejná). Dále jsou procházeny všechny ostatní body množiny a měřeny všechny úhly sevřené mezi onou rovnoběžkou s osou X (či, chceme-li, úsečkou mezi bodem *r* a pivotem) a úsečkou mezi pivotem a *i*-tým bodem množiny. Tyto měřené úhly jsou porovnány a do konvexní obálky zařadíme právě ten bod, ke kterému vedla úsečka pod největším úhlem (pokud je více takových bodů, je přidán ten vzdálenější). Nově přidaný *i*-tý bod se stává pivotem a tento proces probíhá analogicky dokud se souřadnice *i*-tého bodu nerovnají souřadnicím pivotu.

2.1.1. Slovní zápis algoritmu

- Setřídění vstupní množiny bodů podle souřadnice Y
- Nalezení pivota q (bod s nejmenší souřadnicí Y)
- Přidání pivota q do konvexní obálky
- Zavedení bodu r na rovnoběžce s osou X procházející pivotem q (q.x() > r.x())
- Inicializace $P_{j-1} \in X, P_j = q, P_{j+1} = P_{j-1}$
- Opakuj pro všechny body množiny, dokud $P_{j+1} \neq q$:
  - Najdi $P_{j+1} = argmax_{\forall p_i \in X} \measuredangle(P_{j-1}, P_j, P_i)$
  - Přidej P_{j+1} do konvexní obálky
  - $P_{j-1} = P_j$

- $P_j = P_{j+1}$

2.2. Qucik Hull

Quick Hull, jak již samotný název napovídá, představuje jednu z nejrychlejších metod konstrukce konvexních obálek. Algoritmus je navíc rekurzivní, což znamená, že v určitých částech kódu volá sám sebe. Jádro této metody tkví v hledání nejvzdálenějších bodů od iniciální úsečky tvořené body s minimální a maximální souřadnicí X (nebo Y, záleží na vkusu). Přímka procházející zvolenými dvěma body s extrémní souřadnicí X rozděluje množinu bodů na horní a dolní polorovinu. Právě v každé z těchto polorovin probíhá ono hledání nejvzdálenějších bodů; jakmile je takový bod nalezen, je přidán do konvexní obálky (v prvním případě spojen se zmíněnými body s nejmenší a největší souřadnicí X), čímž vznikají dvě nové přímky. Zde rekurzivně voláme nastíněný postup (tedy opět hledám nejvzdálenější bod od těchto nových přímek), dokud není hotova konvexní obálka.

2.2.1. Slovní zápis algoritmu

- Setřídění bodů podle souřadnice X, nalezení bodů q_1, q_3 s extrémními hodnotami (min. a max.)
- Zavedení dvou prázdných množin *upper_points* a *lower_points*
- Přidání *q1, q3* do *upper_points* i *lower_points*
- Projdi všechny body vstupní množiny a zjisti, zda *i*-tý bod leží v horní polorovině, pokud ano, přidej *i*-tý bod do *upper_points*, v opačném případě přidej *i*-tý bod do *lower_points*
- Přidání krajního bodu q_3 do konvexní obálky
- Najdi nejvzdálenější bod v horní polorovině od iniciální přímky (vedoucí přes body q_1, q_3), přidej ho do konvexní obálky a rekurzivně opakuj vůči nově vzniklým přímkám
- Přidání krajního bodu q_1 do konvexní obálky
- Rekurzivní hledání nejvzdálenějšího bodu v dolní polorovině

2.3. Sweep Line

Algoritmus v češtině zvaný *Metoda zametací přímky* funguje na pricnipu nkrementální (přírůstkové) konstrukce, v podstatě dělí množinu bodů na dvě části, zpracovanou část doplňuje o nové body z nepracované části. Přitom pracuje s množinami předchůdců a následníků.

Ze všeho nejdřív, podobně jako ostatní metody, setřídí souřadnice podle souřadnice X, přčemž první dva body s minimální souřadnicí X spojí. Poté postupuje na další bod v seřazeném vektoru, propojí s první úsečkou a určí jeho předchůdce a následníky, takto pokračuje do té doby, než vytvoří kompletní konvexní obálku.

2.3.1. Slovní zápis algoritmu

- Setřídění bodů podle souřadnice X
- Tvorba vektorů předchůdců a následovníků
- Počáteční aproximace pomocí dvojúhelníku
  - *n[0] = 1; n[1] = 0;*
  - *p[0] = 1; p[1] = 0;*
- Pro všechny další body testuj $y_i > y_{i-1}$
  - Přeindexování při splnění podmínky
    - *p[i] = pi-1*
    - *n[i] = n[i-1];*
  - Přeindexování při nesplnění podmínky
    - *p[i] = p[i-1]*
    - *n[i] = i-1*
- Přeindexování *n[p[i]] = i; p[n[i]] = i*

- Oprava horní tečny: $while\ (n[n[i]]) \in \sigma_R(i, n[i])$:
  - přeindexování *p[n[n[i]]] = i*
  - *n[i] = n[n[i]]*
- Oprava dolní tečny: $while\ (p[p[i]]) \in \sigma_L(i, p[i])$:
  - přeindexování *n[p[p[i]]] = i*
  - *p[i] = p[p[i]]*
- Z vektoru následovníků sestav konvexní obálku

2.4. Graham Scan

Metoda *graham Scan* pracuje na principu určování CCW orientace trojúhelníku. Nejdříve setřídíme body podle souřadnice Y a za pivota označíme takový bod, který má minimální souřadnici Y. Následně jsou počítány úhly mezi rovnoběžkou s osou X vedenou přes pivot a spojnicemi s jednotlivými body vstupní množiny. Body přetřídíme podle velikosti vypočteného úhlu, v případě shodnosti dvou nebo více úhlů bude uvažován pouze ten nejvzdálenější bod od pivota. Poté probíhá testování CCW orientace na posledních dvou bodech konvexní obálky a následujícím bodě na spojnici pod největším úhlem, dokud není vytvořena konvexní obálka.

2.4.1. Slovní zápis algoritmu

- Setřídění vstupní množiny bodů podle souřadnice Y
- Nalezení pivota q (bod s nejmenší souřadnicí Y)
- Přidání pivota q do konvexní obálky
- Zavedení bodu *r* na rovnoběžce s osou X procházející pivotem q (q.x() > r.x())
- Pro všechny body vstupní množiny spočti úhel sevřený body *r, q,* p_i
- Pro všechny body vstupní množiny spočti úhel sevřený body *r, q,* p_i
- Pro všechny body vstupní množiny spočti vzdálenost od bodu q
- Nové setřídění bodů podle velikosti spočteného úhlu
- V případě stejného úhlu uvažuj pouze vzdálenější bod
- Bod s nejmenší úhlem vlož do konvexní obálky
- Opakuj pro všechny setříděné body podle úhlu
  - pokud následující bod leží vlevo od přímky spojující poslední dva prvky, vlož bod do konvexní obálky
  - v případě, že leží vpravo, odeber poslední bod z konvexní obálky

3. Problematické situace a jejich rozbor

3.1. Singulární případ u metody Graham Scan

Tento algoritmus pracuje s měřenými úhly, resp. na základě jejich velikosti rozhoduje o zařazení bodů do konvexní obálky, z tohoto důvodu je nezbytné, aby byl pro zvolenou velikost úhlu jednoznačně nalezen odpovídající bod. V náhodném množině vygenerovaných bodů sice taková situace nastává náhodně, oproti tomu v jiných šablonách pro generování bodů k tomuto jevu dochází s jistotou; například u bodů generovaných na mřížce či na čtverci.

K ošetření nastíněné kolinearity byly kromě velikosti úhlů počítány i vzdálenosti od pivotu; v případě shodného úhlu by nebyl uvažován bod bližší k pivotu, resp. do konvexní obálky bude přidán právě ten nejvzdálenější.

3.2. Konstrukce striktně konvexních obálek

Z různých důvodů může nastat takový případ, že vytvořená konvexní obálka obsahuje kolineární body (tedy takové body, které leží na jedné z jejích hran či může dojít dokonce k situaci, kdy bude některý z vrcholů konvexní obálky duplicitní.

Výše uvedené problémy byly ošetřeny na konci každého z algoritmů, kdy je volána funkce *strictlyConvexhull*, která duplicitní či kolineární body identifikuje a odstraní, vrátí zpátky tedy striktně konvexní obálku očištěnou od dat, která by byla pro výslednou konvexní obálku zbytečná.

3.3. Duplicity ve vstupní množině bodů u Sweep Line

Metoda *Sweep Line* je citlivá na duplicity vyskytující se ve vstupní množině bodů, z tohoto důvodu algoritmus hned na počátku svého těla duplicitní body jednoduše odstraňuje. Za duplicitní jsou považovány takové body, které od sebe leží blíže, než je povolená tolerance.

4. Vstupní data

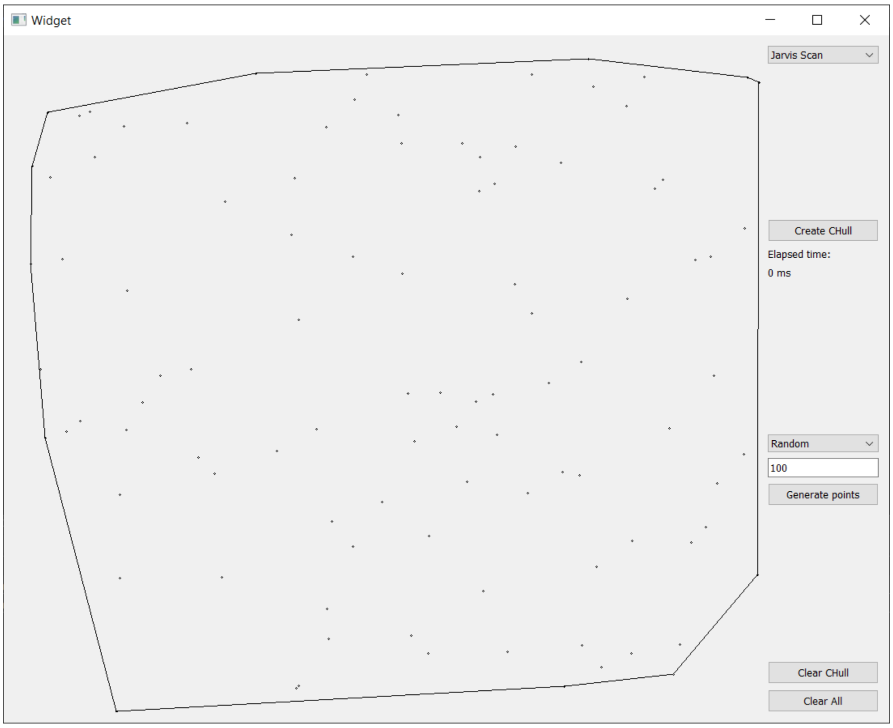
Do programu vstupuje množina generovaných bodů, kterou může uživatel definovat: je nutné zadat počet generovaných bodů a dále je volitelné vybrat šablonu, do které budou body náhodně generovány (body na kružnici, body v mřížce, …), pokud uživatel nenechá defaultně nastavenou možnost *random*.

5. Výstupní data

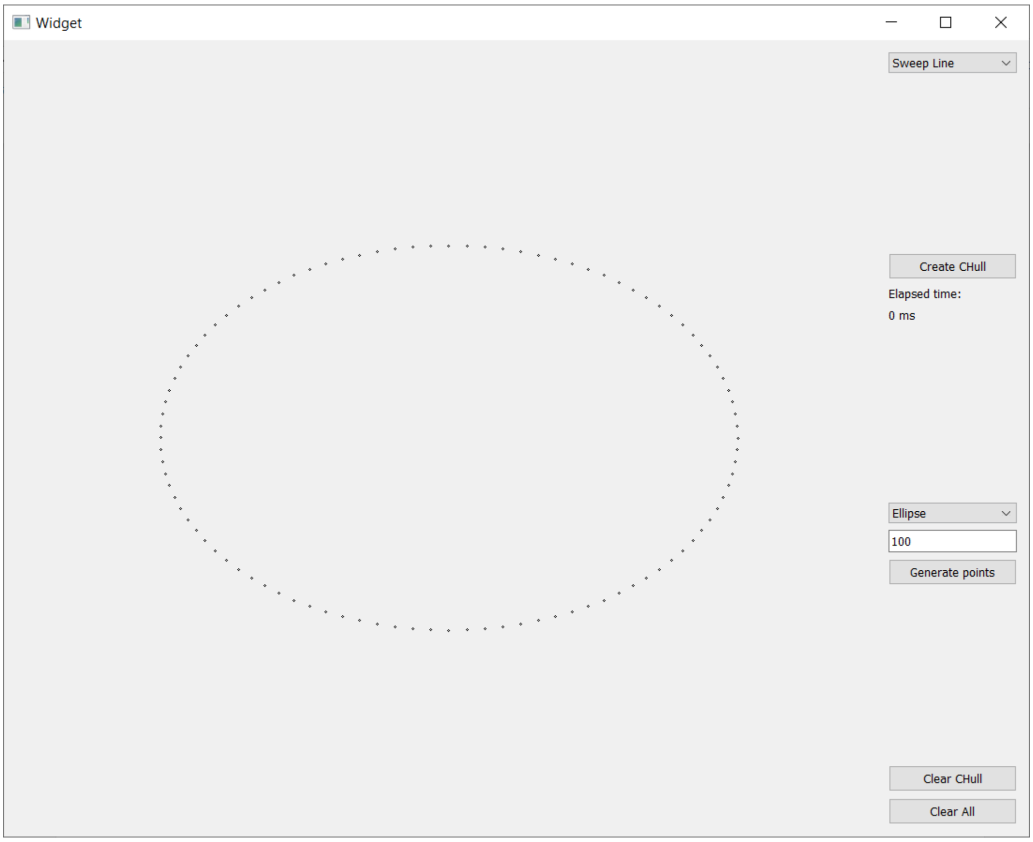
Výstupem programu je grafická aplikace, která sestrojí nad vygenerovanou množinou bodů konvexní obálku. Nabízí k tomu 4 různé algoritmy: *Jarvis Scan, Quick Hull, Sweep Line* a *Graham Scan*. Po vytvoření obálky je možné odečíst dobu běhu algoritmu.

6. Ukázka aplikace

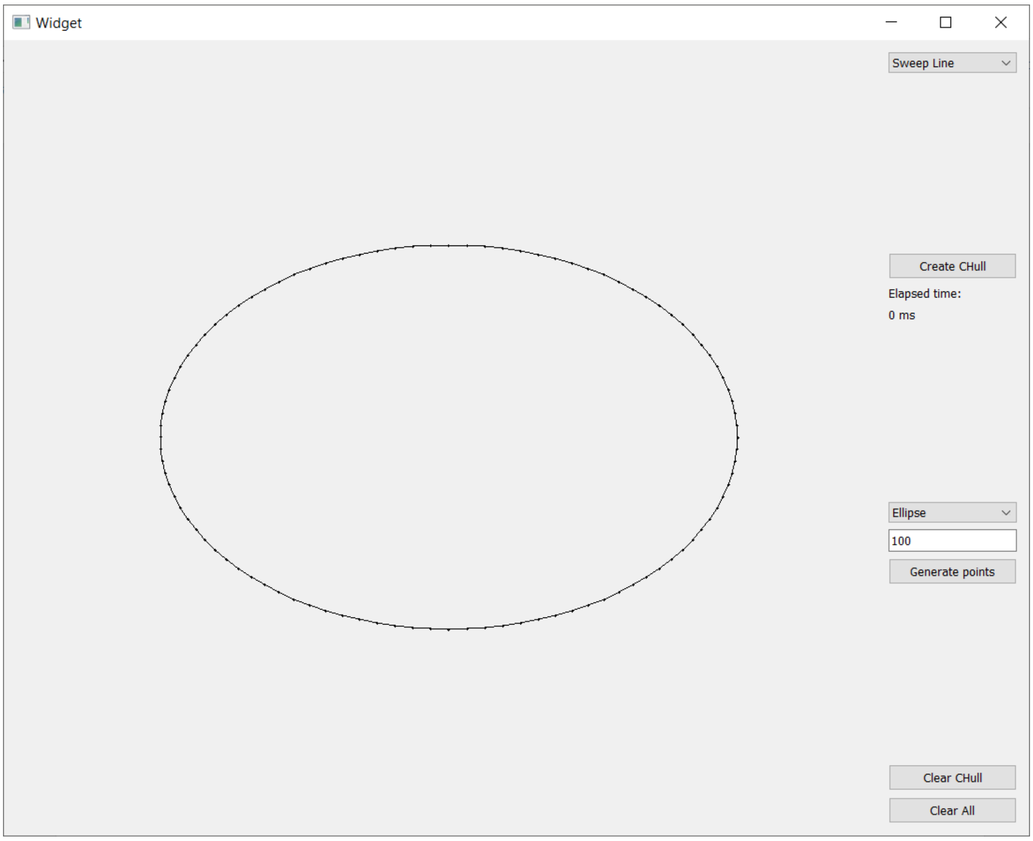
Tato kapitola nabízí přehled několika komentovaných screenshotů chodu aplikace.



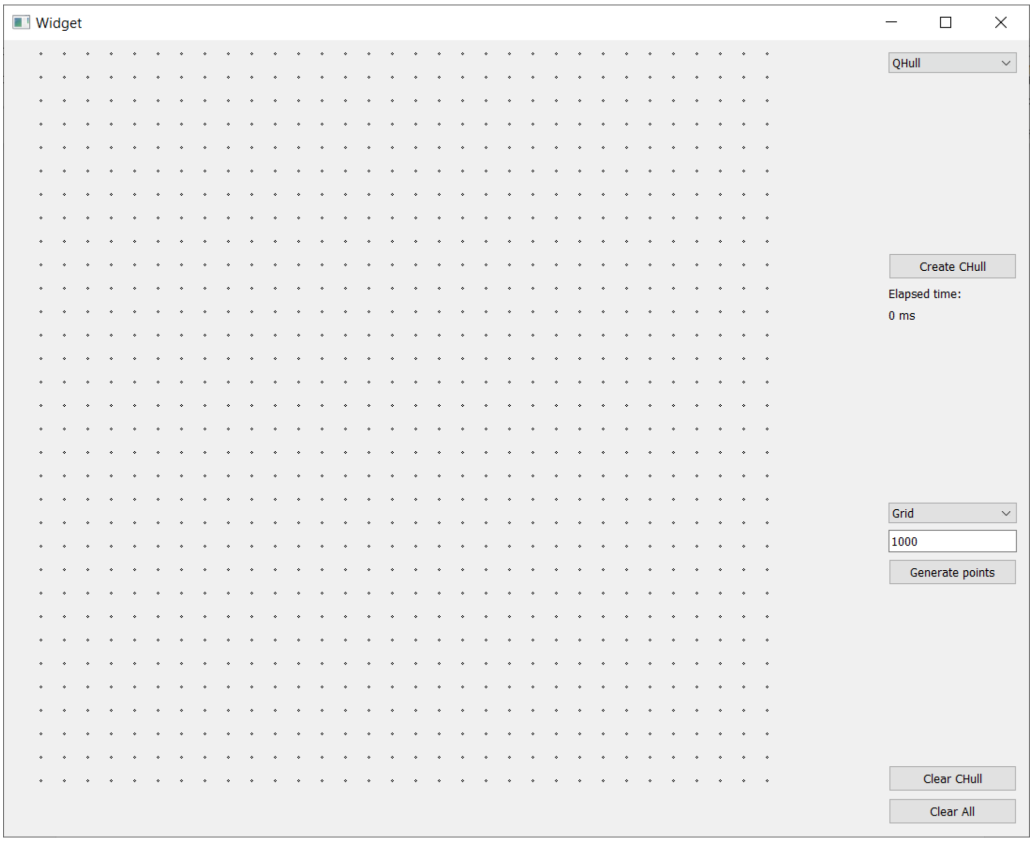
Ukázka sestrojené konvexní obálky metodou *Jarvis Scan* pro 100 náhodně vygenerovaných bodů.



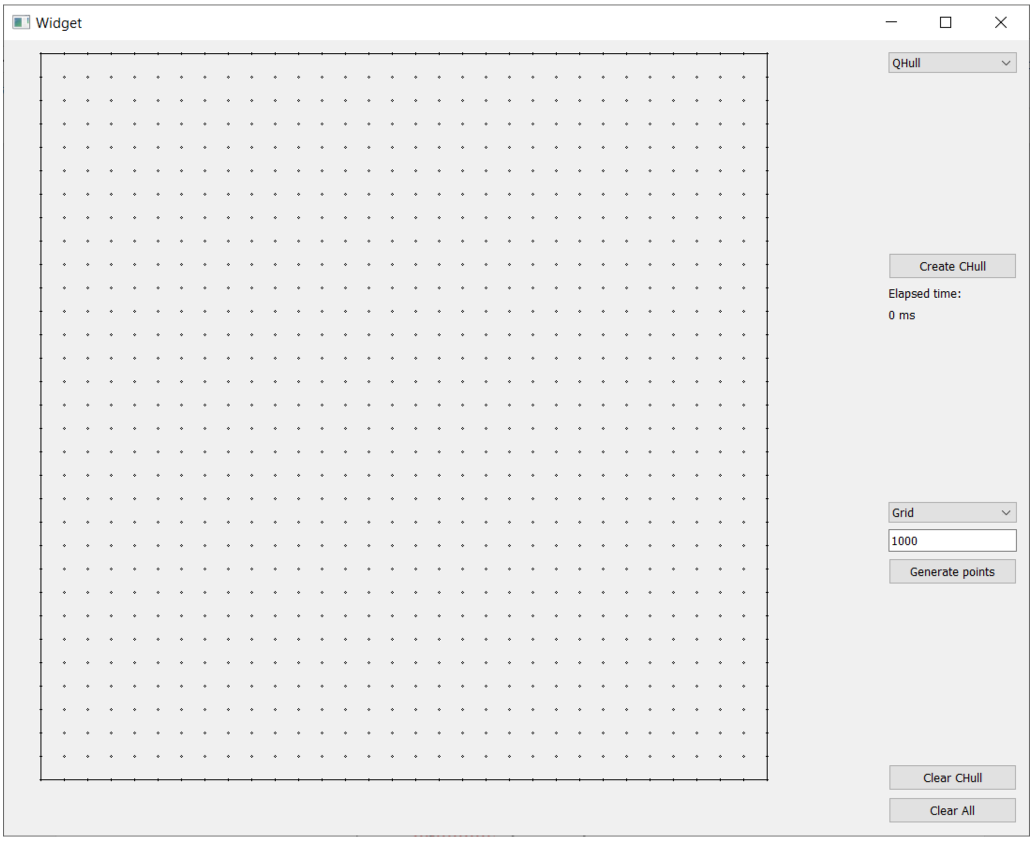
100 bodů vygenerovaných do tvaru elipsy.



Konvexní obálka sestrojená nad body na elipse metodou *Sweep Line.*



1000 bodů vygenerovaných v mřížce.



Konvexní obálka kolem1000 bodů v mřížce pomocí metody *Quick Hull*.

7. Technická dokumentace

Aplikace obsahuje třídy: *Algorithms, Draw, Widget, sortByX, sortByY, sortByAngle, removeByAngle, removeByCoords*. Níže následuje jejich detailnější rozbor.

7.1. Třída Algorithms

Mimo konstruktor zahrnuje třída Algorithms čtyři metody.

int getPointLinePosition(QPoint q, QPoint p1, QPoint p2)

Na začátku si metoda spočítá vzdálenosti *p1q, p2q a p1p2*. Poté zkoumá, zda bod *q* neleží na hraně (nebo v její těsné blízkosti) testováním podmínky, zda absolutní hodnota rozdílu *p1p2 – p1q+p2q* je menší nebo rovna zvolené toleranci. Dalším krokem je zjištění, zda bod *q* není totožný s jedním z vrcholů polygonu, pročež jsou zkoumány absolutní hodnoty rozdílů souřadnic X a Y bodů *q* a *p1*, resp. *p2*. Pokud splňují toleranci, prohlásíme bod *q* totožný s bodem p_i. Dále tato metoda počítá vektory *qp1, p1p2*, z kterých je pak přes determinant určeno, zda bod *q* leží v pravé či levé polorovině.

Metoda tedy vrací 4 možné výsledky: 0 (bod leží v pravé polorovině), 1 (bod leží v levé polorovině), 2 (bod leží na hraně), 3 (bod splývá s vrcholem).

double getAngle(QPoint p1, QPoint p2, QPoint p3, QPoint p4)

Tato funkce počítá úhel mezi dvěma hranami určenými čtyřmi body na vstupu dle známého vzorce:

$$\cos\omega = \frac{\vec{u} \cdot \vec{v}}{|\vec{u}| \cdot |\vec{v}|}$$

Void qh (int s, int e, std::vector<QPoint> &points, Qpolygon &ch)

Na vstupu funkce se zadá 1 0, nebo 0 1, jakožto počáteční přímku od které se funkce odráží. Je to funkce, kterou využíváme v algoritmu qHull. Její fukčnost spočívá v rozeznávání, jestli body jsou součástí konvexní obálky.

double getPointLineDist (QPoint &a, QPoint &p1, QPoint &p2)

Jednoduchá funkce, které spočítá vzdálenost mezi bodem a linií. Tato funkce se používá ve funkci qh

Qpolygon jarvis (std::vector<QPoint> &points)

Metoda pro výpočet konvexní obálky nad vektorem bodů metodou Jarvis Scan. Metoda vrací konvexní obálku s typem Qpolygon.

Qpolygon graham (std::vector<QPoint> &points)

Metoda pro výpočet konvexní obálky nad vektorem bodů metodou Graham Scan. Metoda vrací konvexní obálku s typem Qpolygon.

Qpolygon qHull (std::vector<QPoint> &points)

Metoda pro výpočet konvexní obálky nad vektorem bodů metodou Quick Hull. Metoda vrací konvexní obálku s typem Qpolygon.

Qpolygon sweepLine (std::vector<QPoint> &points)

Metoda pro výpočet konvexní obálky nad vektorem bodů metodou Sweep Line. Metoda vrací konvexní obálku s typem Qpolygon.

Qpolygon strictlyConvexHull (QPolygon &ch)

Metoda přetváří zadaný polygon na striktně konvexní polygon. Dá se využít ve všech předchozích metodách.

7.2. Třída Draw

void mousePressEvent

V závislosti na *draw_mode* je touto metodou vykreslen buď bod *q* nebo polygon.

void paintEvent

Touto metodou je vykreslen bod/ body (zadané ručně či generované).

Std::vector <QPoint> generatePoints (int n, int height, int width)

Tato funkce si vezme z Canvasu výšku a šířku, z LineEditu počet generovaných bodů a náhodně vygeneruje x a y souřadnice pro zadaný počet bodů, odstraní duplicity a body vykreslí v mezích Canvasu.

Std::vector <QPoint> generateGrid (int n, int height, int width)

Tato funkce si vezme z Canvasu výšku a šířku, z LineEditu počet generovaných bodů, bodů musí být více než 4. Zadaný počet bodů funkce zaokrouhlí tak, aby bylo možné vytvořit grid, mřížku bodů, která má stejně sloupců jako řádků, a body vykreslí od levého horního rohu Canvasu (šířka v algoritmu není potřebná.

Std::vector <QPoint> generateCircle (int n, int height, int width)

Tato funkce si vezme z Canvasu výšku a šířku, z LineEditu počet generovaných bodů, definuje centr, bod, kolem kterého se kružnice bodů vykreslí (střed Canvasu) a následně systematicky generuje body na kružnici o poloměru závislém na počtu generovaných bodů.

Std::vector <QPoint> generateEllipse (int n, int height, int width)

Funguje podobně jako kružnice, ale vytváří elipsu.

Std::vector <QPoint> generateSquare (int n, int height)

Pro čtverec byly požity 2 podmínky a to, že počet bodů musí být alespoň 4 a počet bodů musí být dělitelný čtyřmi. Tyto podmínky byly zavedeny kvůli funkčnosti algoritmu, který v každé iteraci tvoří 4 body na čtverci

7.3. removeByAngle

Tato třída porovnává úhly a hledá duplicity.

7.4. removeByCoords.

Tato třída porovnává vzdálenosti mezi jednotlivými body a hledá duplicity, nebo téměř duplicity.

7.5. sortByAngle

tato třída se používá k seřazení vektoru bodů podle úhlu (a vzdálenosti) úhly vždy dvou bodů se zadaným bodem a třídí podle velikosti.

7.6. sortByX

Rovná body podle souřadnice X.

7.7. sortByY

Rovná body podle souřadnice Y

7.8. widget

Void on_pushButton_clicked ()

Spustí se časovač, který měří dobu, za kterou jednotlivé algoritmy vykonají svou práci a podle výběru algoritmu z rolovacího okna spustí danou funkci

Void on_pushButton_2_clicked ()

Vyčistí zobracenou konvexní obálku.

Void on_pushButton_3_clicked ()

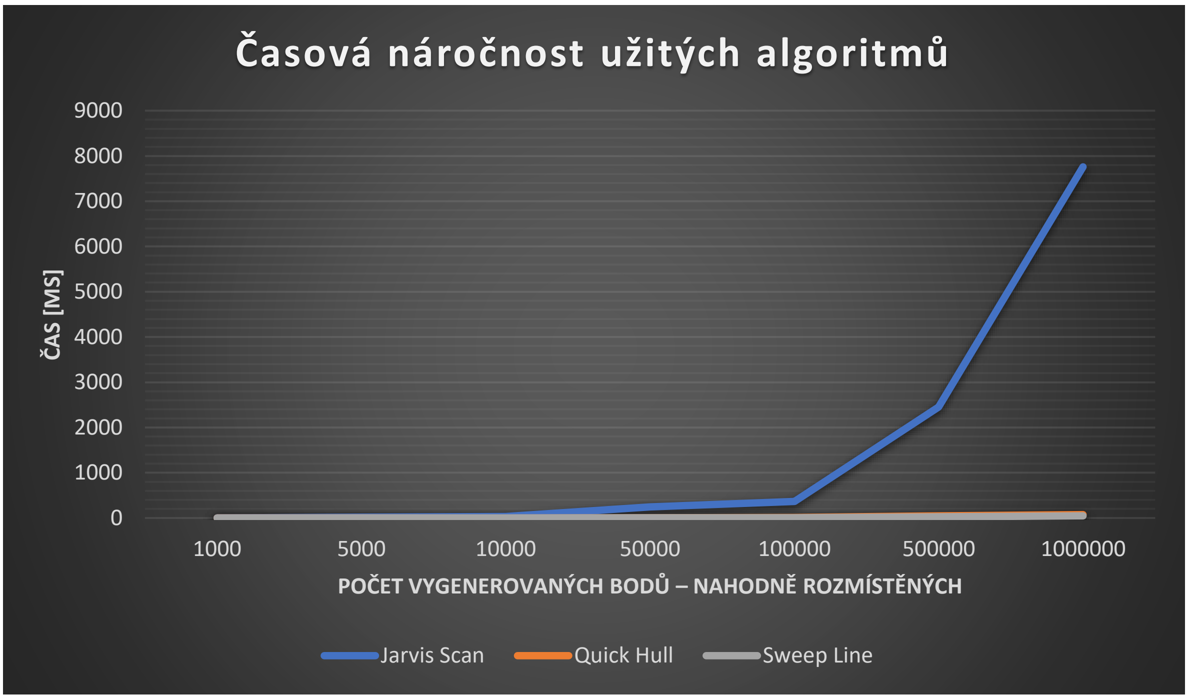
Vyčistí Canvas od všech vstupů

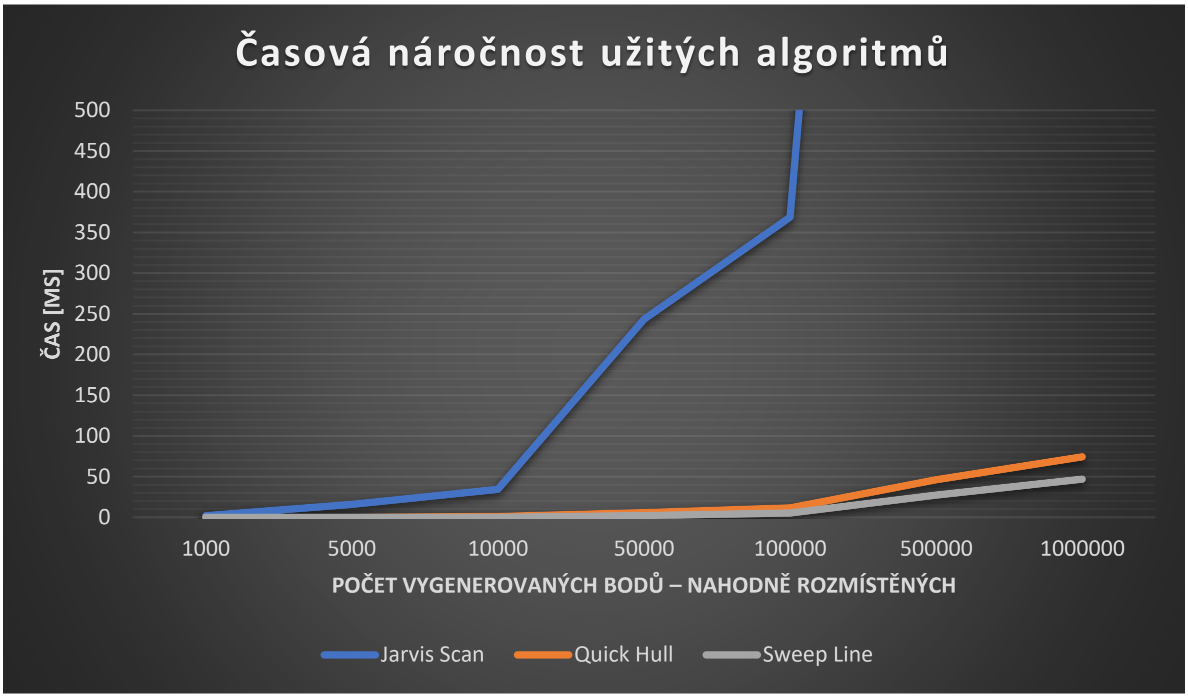
Void on_pushButton_4_clicked ()

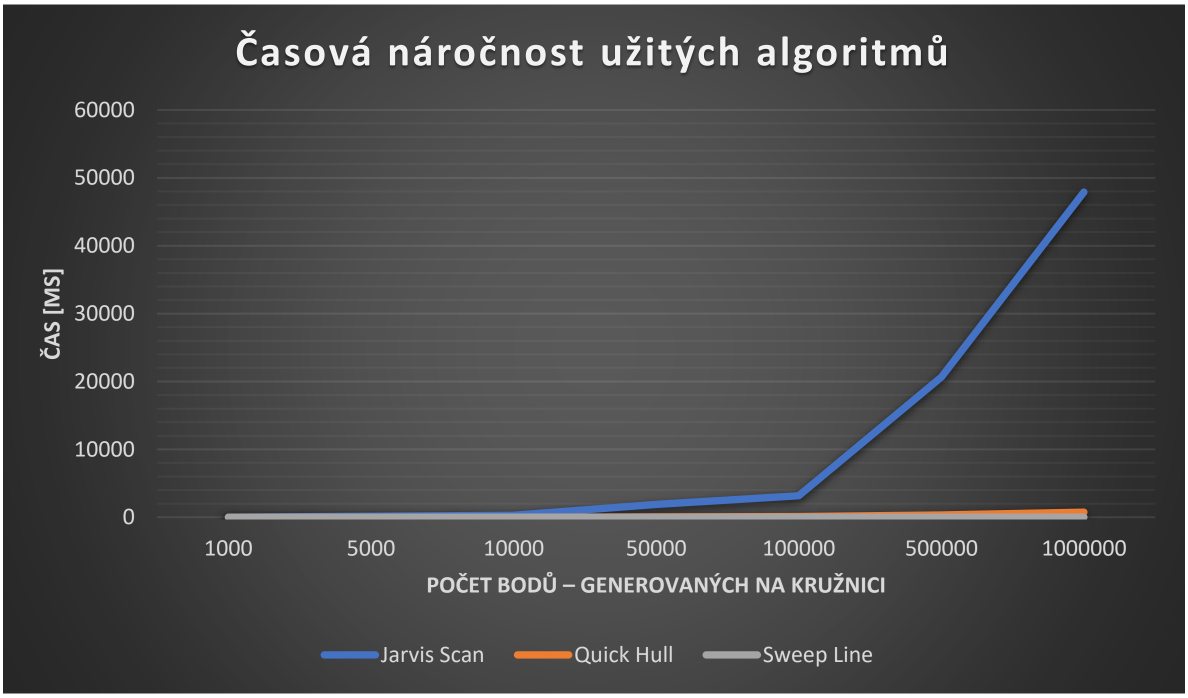
Vezme si z Canvasu výšku a šířku, z LineEdit počet bodů a následně spustí vybranou funkci na generování bodů.

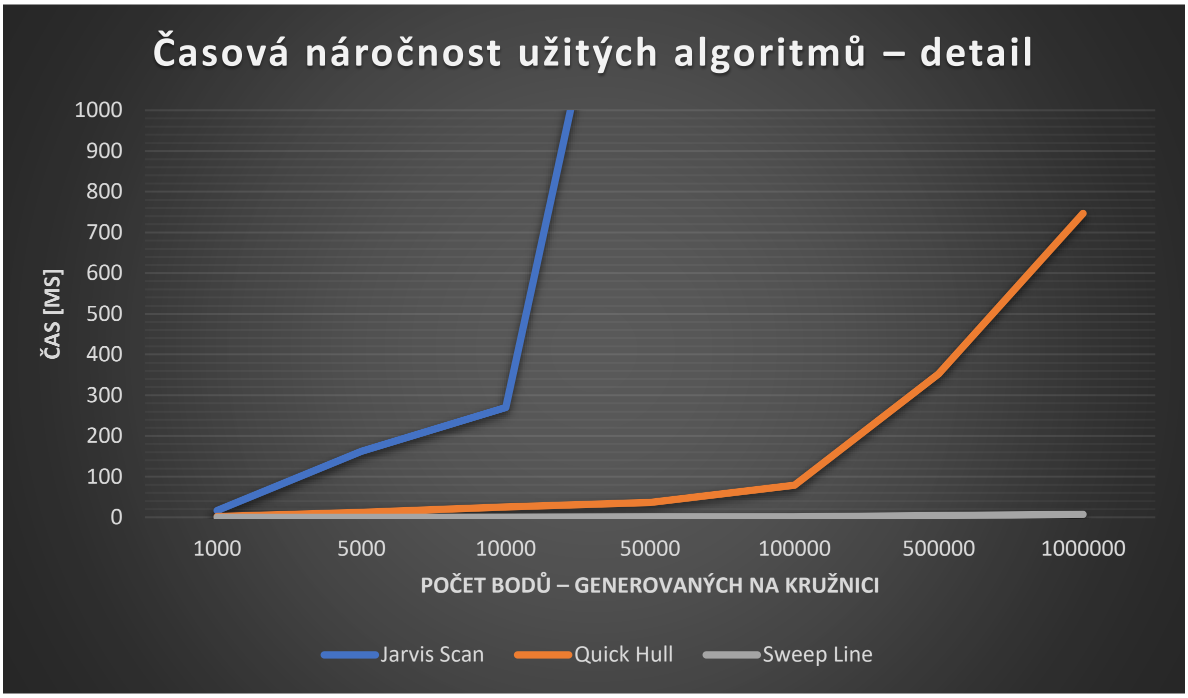
8. Závěr, statistika

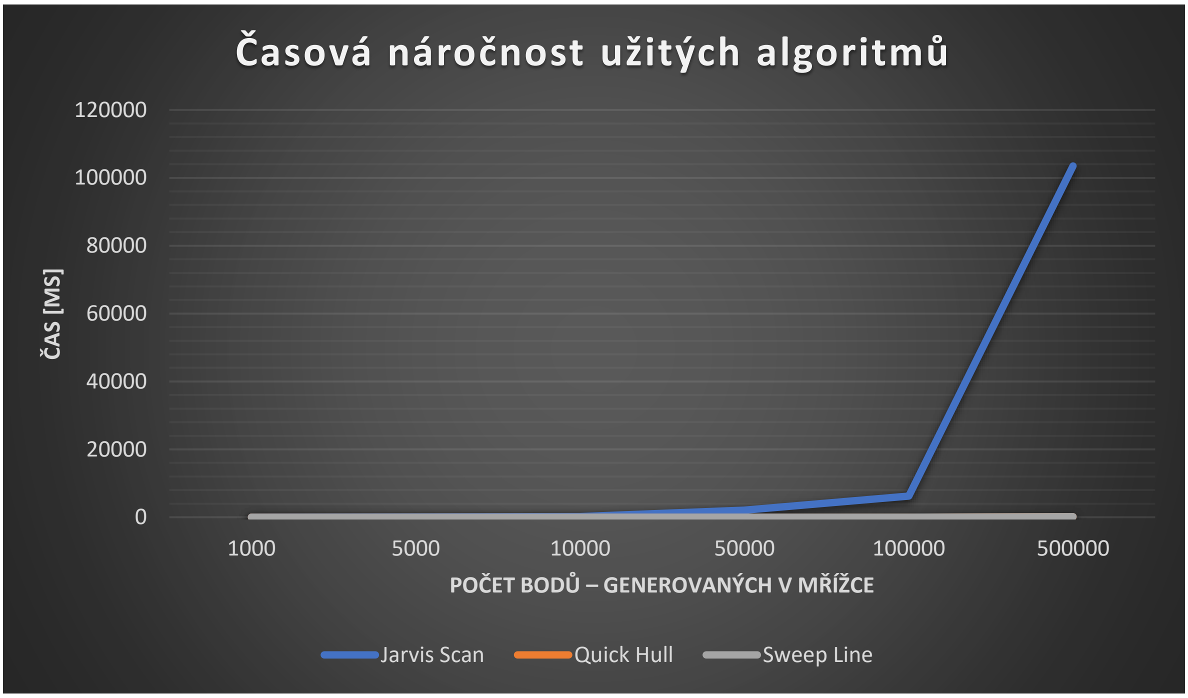
Byla vytvořena solidní aplikace, která nabízí několikero možností, kterak zkonstruovat striktně konvexní obálku. Je zajímavé sledovat časovou náročnost jednotlivých algoritmů v závislosti na typu vstupní množiny bodů (zda je náhodná nebo například v mřížce). Body mohou být zadávány ručně uživatelem, ale ideálně je ke generování bodů určeno tlačítko, které po zadání počtu bodů a šablony vygeneruje vstupní množinu. Na hotovém programu proběhlo testování, jehož výsledky byly zahrnuty v přílohách k této dokumentaci, níže jsou výsledky testování představeny v podobě grafů, pro přehlednost jsou uvedeny pro každou testovanou vstupní množinu dva grafy: první zobrazuje celý průběh testování, druhý se zaměřuje na zajímavý detail.

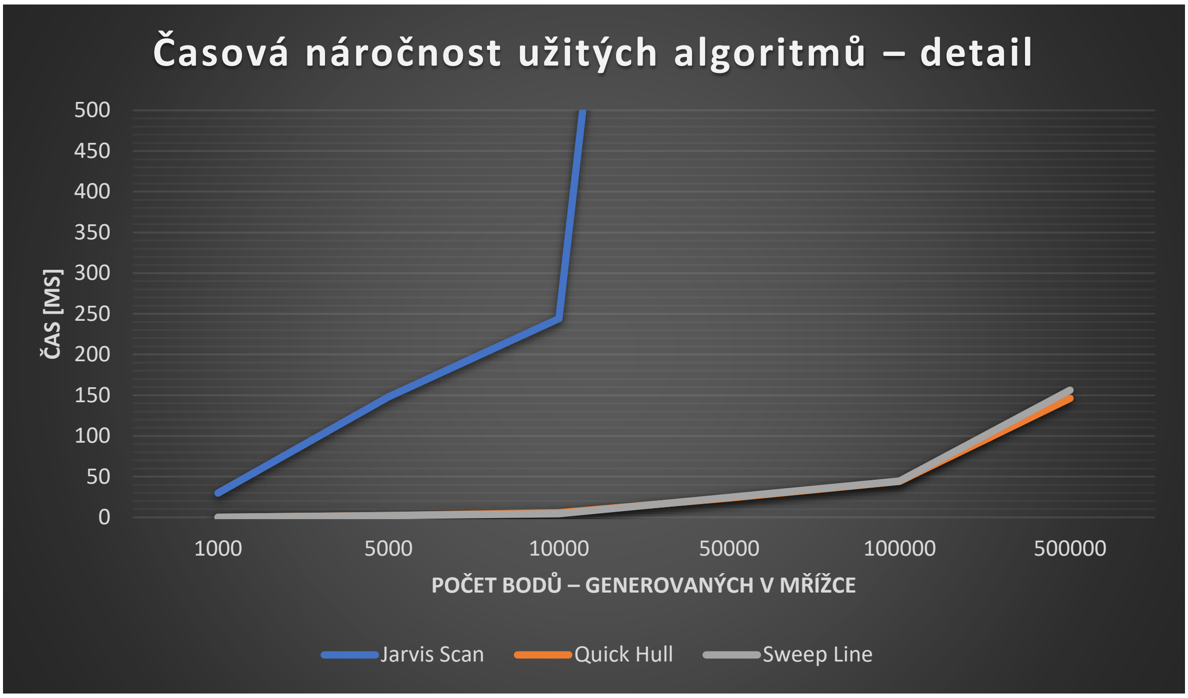



Časová náročnost užitých algoritmů
500
450
400
350
300
250
200
150
100
50
0
ČAS [MS]
1000
5000
10000
50000
100000
500000
1000000
POČET VYGENEROVANÝCH BODŮ – NAHODNĚ ROZMÍSTĚNÝCH
Jarvis Scan
Quick Hull
Sweep Line


Časová náročnost užitých algoritmů
60000
50000
40000
30000
20000
10000
0
ČAS [MS]
1000
5000
10000
50000
100000
500000
1000000
POČET BODŮ – GENEROVANÝCH NA KRUŽNICI
Jarvis Scan
Quick Hull
Sweep Line


Časová náročnost užitých algoritmů – detail
1000
900
800
700
600
500
400
300
200
100
0
ČAS [MS]
1000
5000
10000
50000
100000
500000
1000000
POČET BODŮ – GENEROVANÝCH NA KRUŽNICI
Jarvis Scan
Quick Hull
Sweep Line


Časová náročnost užitých algoritmů
120000
100000
80000
60000
40000
20000
0
ČAS [MS]
1000
5000
10000
50000
100000
500000
POČET BODŮ – GENEROVANÝCH V MŘÍŽCE
Jarvis Scan
Quick Hull
Sweep Line


Časová náročnost užitých algoritmů – detail
500
450
400
350
300
250
200
150
100
50
0
ČAS [MS]
1000
5000
10000
50000
100000
500000
POČET BODŮ – GENEROVANÝCH V MŘÍŽCE
Jarvis Scan
Quick Hull
Sweep Line