ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

FAKULTA STAVEBNÍ, OBOR GEODÉZIE A KARTOGRAFIE

KATEDRA GEOMATIKY

název předmětu

ALGORITMY DIGITÁLNÍ KARTOGRAFIE A GIS

číslo úlohy	název úlohy:			
2	**Konvexní obálky**			
školní rok:	semestr:	zpracovali:	datum:	klasifikace:
2019/20	zimní	**David Němec, Jan Šartner**	5. 11.	

Technická zpráva

Konvexní obálky

1. Zadání

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

Výstup: H(P)

Nad množinou P implementujte následující algoritmy pro konstrukci H(P):

-Jarvis Scan
-Quick Hull
-Sweep line

Vstupní množiny bodů včetně vygenerovaných konvexních obálek vhodně vizualizujte. Pro množiny $n \in \langle 1000, 1000000 \rangle$ vytvořte grafy ilustrující doby běhu algoritmů pro zvolená n. Měření proveďte pro různé typy vstupních množin (náhodná množina, rastr, body na kružnici) opakovaně (10×) 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.

2. Doplňující úlohy

2.1 Konstrukce konvexní obálky metodou Graham Scan

-řešeno

2.2 Konstrukce striktně konvexních obálek pro všechny algoritmy

-řešeno

2.3 Ošetření singulárního případu u Jarvis Scan: existence kolineárních bodů v datasetu

-řešeno

2.4 Konstrukce Minimum Area Enclosing box některou z metod

-řešeno

2.5 Algoritmus pro automatické generování konvexních/nekonvexních množin bodů různých tvarů (kruh, elipsa, čtverec, star-shaped, atd.)

-řešeno

3. Problematika nalezení konvexní obálky a minimálního ohraničujícího obdélníku

3.1 Algoritmus Jarvis Scan

3.1.1 Potřebné vzorce

Výpočet úhlu mezi 2 vektory u, v:

$$\omega = \left|\operatorname{acos}\left(\frac{u * v}{|u| \cdot |v|}\right)\right|$$

3.1.2 Princip

Nejprve je vyhledán tzv „pivot“, neboli bod s nejmenší souřadnicí Y. Dále algoritmus využívá vzorce pro výpočet úhlu mezi dvěma přímkami, které jsou dané vektory. V první iteraci je přidán do konvexní obálky bod, který maximalizuje úhel (bod, pivot, osa x). Následně jsou přidávány body maximalizující úhel (bod, poslední bod v obálce, předposlední bod v obálce), dokud se posledním bodem obálky nestane opět pivot.

3.1.3 Výpočet

1. Inicializace „Konvexní obálky“ H
2. „Pivot“ q = min(y_i)
3. Přidání q do H
4. Inicializace p_{j-1} = „bod na poloose procházející pivotem a rovnoběžné s -x“, $p_j = q$, $p_{j+1} = p_{j-1}$
5. Opakování, dokud $p_{j-1} \neq q$
6. {
7. p_{j+1} = bod, pro který platí, že úhel (p_{j-1}, p_j, p_{j+1}) je maximální
8. Přidání p_{j+1} do H
9. $p_{j-1} = p_j$, $p_j = p_{j+1}$
10. }

3.1.4 Řešení

Ve zdrojovém kódu je algoritmus napsán tak, aby výsledná obálka neobsahovala duplicitní body a byla striktní.

3.2 Algoritmus Quick Hull

3.2.1 Potřebné vzorce

Výpočet pozice bodu vzhledem k linii:

$$\vec{u} = |P_1P_2|$$

$$\vec{v} = |P_1Q|$$

$$t = u \times v$$

-na základě znaménka hodnoty t lze rozhodnout, ve které polorovině, určené analyzovanou hranou, bod Q leží.

3.2.2 Princip

V datasetu jsou vyhledány body s nejnižší a nejvyšší souřadnicí X. Jejich spojnicí jsou rozděleny body do dvou skupin podle toho, ve které polorovině se nachází. Nad oběma polorovinami je provedena rekurzivní procedura. Ta je nejprve provedena nad původními dvěma body, přičemž vyhledá nejvzdálenější bod od jejich spojnice, nacházející se v pravé polorovině (vně dosavadní konvexní obálky), pokud existuje.

3.2.3 Výpočet

1. Metoda QH(hrana h, dataset S, konvexní obálka H):
2. Nalezení bodu $p \in S$ = nejvíce vzdálený bod od h, ležící vpravo od H
3. Když bod existuje
4. {
5. $QH((h_{start}, p), S, H)$
6. Přidání p do H
7. $QH((p, h_{end}), S, H)$
8. }

Celý algoritmus:
1. Inicializace $H = \emptyset$, „Body v horní polorovině“ S_U, „Body v dolní polorovině“ S_L
2. $q_1 = \min(x_i)$, $q_3 = \max(x_i)$
3. Přidání q_1, q_3 do S_U
4. Přidání q_1, q_3 do S_L
5. Pro všechny body v datasetu S
6. {
7. Když bod p_i leží vlevo od přímky p_1, p_3 //Nachází se v horní polorovině
8. Přidání p_i do S_U
9. Jinak
10. Přidání p_i do S_L
11. }
12. Přidání q_3 do H
13. $QH((q_3, q_1), S_U, H)$
14. Přidání q_1 do H
15. $QH((q_1, q_3), S_L, H)$

3.2.4 Řešení

Ve zdrojovém kódu je řešením konvexní obálka, která neobsahuje duplicitní body a je striktní.

3.3 Algoritmus Sweep line

3.3.1 Potřebné vzorce

Výpočet pozice bodu vzhledem k linii (viz. 3.2.1)

3.3.2 Princip

Nejprve jsou body v datasetu setříděny dle souřadnice X. V rámci prvních dvou bodů je inicializována „konvexní obálka“, resp. sekvence bodů. Dále je setříděný dataset postupně procházen a s přibývajícími body se aktualizuje konvexní obálka.

3.3.3 Výpočet

1. Inicializace „seznam indexů následníků“ n = ∅, „seznam indexů předchůdců“ p = ∅
2. Setřídění datasetu podle souřadnice x
3. Když(p_3 leží vlevo od přímky p_1, p_2)
4. {
5. n[1] = 2; n[2] = 3; n[3] = 1
6. p[1] = 3; p[2] = 1; p[3] = 2
7. }
8. Jinak
9. {
10. n[1] = 3; n[2] = 2; n[3] = 1
11. p[1] = 2; p[2] = 1; p[3] = 3
12. }
13. Pro všechny body v datasetu s indexem i větším než 3
14. {
15. Když($y_i > y_{i-1}$)
16. p[i] = i-1; n[i] = n[i-1]
17. Jinak
18. n[i] = i-1; p[i] = p[i-1]
19. n[p[i]] = i; p[n[i]] = i;
20. Dokud bod n[n[i]] leží vpravo od přímky i, n[i]
21. p[n[n[i]]] = i; n[i] = n[n[i]]
22. Dokud bod p[p[i]] leží vlevo od přímky i, p[i]
23. n[p[p[i]]] = i; p[i] = p[p[i]]

3.3.4 Řešení

Na základě zdrojového kódu vykresluje aplikace konvexní obálku, která neobsahuje duplicitní body a obsahuje jinak všechny body na ní ležící. Sekvence bodů je řešena pomocí seznamu předchůdců a následovníků, které jsou aktualizovány s každým nově přidaným bodem. Seznamy se nejprve aktualizují na základě toho, zdali přidaný bod leží nad přímkou rovnoběžnou s osou x procházející posledním zařazeným bodem, nebo pod ní. Následně jsou aktualizovány ještě z hlediska úpravy horní / dolní tečny (zachování konvexnosti obálky).

Pokud je vyžadována striktnost konvexní obálky, lze využít widget „Check-box“, který do výpočtu zahrne odpovídající funkci.

3.4 Algoritmus Graham Scan

3.4.1 Potřebné vzorce

Výpočet úhlu mezi 2 vektory u, v (viz. 3.1.1)
Výpočet pozice bodu vzhledem k linii (viz. 3.2.1)

3.4.2 Princip

Nejprve je nalezen pivot. Body v datasetu jsou setříděny na základě úhlu (bod, pivot, osa x), takže vznikne tzv. „star-sahped polygon“. U bodů se stejným úhlem se maže ten, který je bližší pivotu. Body se procházejí postupně v rámci orientace CVV a rozhoduje se, v jaké polorovině vůči poslední hraně obálky jsou (při první iteraci je jako hrana brána osa x). V případě, že analyzovaný bod leží v levé polorovině, přidá se do konvexní obálky. V opačném případě jsou hrany (poslední body) z obálky odebírány, dokud neplatí první případ.

3.4.3 Výpočet

1. „Pivot“ q = min(y_i), x = „bod na poloose procházející pivotem a rovnoběžné s -x“
2. Setřídění datasetu dle úhlu (p_i, q, x)
3. Když existují 2 body se stejným úhlem (p_i, q, x)
4. Vymaž ten, který je bližší ke q
5. Inicializace j = 2, H = ∅
6. Přidání q, p_1 do H (poslední 2 prvky označené p_t, p_{t-1})
7. Opakování pro j < počet bodů datasetu
8. {
9. Když je p_j vlevo od přímky p_{t-1}, p_t
10. {
11. Přidání p_j do H
12. j = j + 1
13. }
14. Jinak
15. Vymazání posledního bodu H
16. }

3.4.4 Řešení

Ve zdrojovém kódu je algoritmus zkonstruován tak, že konvexní obálka neobsahuje duplicitní body a je striktní.

3.5 Algoritmus Minimum Area Enclosing Box using Convex Hull

3.5.1 Potřebné vzorce

Výpočet úhlu mezi 2 vektory u, v (viz. 3.1.1)
Výpočet pozice bodu vzhledem k linii (viz. 3.2.1)

Rotace / transformace obdélníku (x,y → x‘,y‘) o úhel ω:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{bmatrix} cos\omega & sin\omega \\ -sin\omega & cos\omega \end{bmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix}$$

3.5.2 Princip

Tento algoritmus pracuje s již vytvořenou konvexní obálkou a znalostí faktu, že nejméně jedna strana minimálního ohraničujícího obdélníku je kolineární s hranou konvexní obálky. Daný obdélník je natáčen pomocí transformace ve směru hran konvexní obálky. Výsledný obdélník je ten s nejmenší plochou.

3.5.3 Výpočet

1. Inicializace „Minimální obdélník“ B = ∅, „Minimální plocha“ a = nejvyšší možné číslo pro daný typ proměnné
2. Pro všechny hrany e_i ∈ „Konvexní obálka“ H
3. {
4. Transformace souřadnic bodů tak, aby poloosa +x‘ ležela v e_i, e_{i_start} = (0,0), body datasetu leží v polorovině +y‘
5. Nalezení $p_1 = \min(x_i‘)$
6. Nalezení $p_2 = \max(x_i‘)$
7. Nalezení $p_3 = \min(y_i‘)$
8. Nalezení $p_4 = \max(y_i‘)$
9. „Plocha vypočteného polygonu“ a_act = (p_2_x – p_1_x) * (p_4_y – p_3_y)
10. Když a_act < a
11. a = a_act
12. }

3.5.4 Řešení

Jelikož minimální ohraničující obdélník nemá souřadnice vrcholů v celých číslech, jsou ve zdrojovém kódu využívány pro tento případ funkce předchozích algoritmů upravené pro reálná čísla.

4. Problematické situace (+ řešení doplňkových úloh)

4.1 Existence kolineárních bodů v datasetu

V algoritmu Jarvis Scan je tento problém řešen tak, že při porovnávání úhlu je přidán vzdálenější bod se stejným úhlem.

1. Když (úhly bodu s max. úhlem a testovaného bodu jsou stejné a zároveň vzdálenost od testovaného bodu je větší než vzdálenost od bodu s max. úhlem)
2. {
3. Aktualizuj hodnotu úhlu //Není tolik nutné, zde hraje roli pouze tolerance výpočtu.
4. Aktualizuj index bodu
5. }

4.2 Konstrukce striktních konvexních obálek

Do zdrojového kódu byla přidána funkce (strictCH, viz. 9), která odebere mezilehlé kolineární body z konvexní obálky. Funkce je vykonána po zaškrtnutí CheckBoxu „Strict convex hull“.

1. Pro(všechny body v konvexní obálce)
2. {
3. Když(3 po sobě jdoucí body jsou kolineární)
4. {
5. Vymaž prostřední z nich
6. Sniž index i o 1 //pro případ více než 3 kolineárních bodů
7. }
8. }
9. Vrať konvexní obálku

4.3 Generování bodů

V aplikaci je možné generovat různé typy množin bodů (random, grid, circle, ellipse, star-shaped)

4.3.1 Random

Vygeneruje zadaný počet náhodných bodů, jejichž souřadnice jsou v celých číslech, na ploše 400×400 se středem v bodě (210, 210).

$$x = modulo(\text{náhodné číslo}, 400) + 10$$
$$y = modulo(\text{náhodné číslo}, 400) + 10$$

4.3.2 Grid

Vygeneruje body uspořádané do čtvercové sítě, jejichž souřadnice jsou v celých číslech, na ploše 400×400 se středem v bodě (210, 210). Počet bodů v jednom řádku a v jednom sloupci je odmocnina zadaného čísla zaokrouhlená dolů.

$$x = i_r \left(400/zaokrouhleno\_d\left(\sqrt{n}\right)\right) + 10$$
$$y = i_s \left(400/zaokrouhleno\_d\left(\sqrt{n}\right)\right) + 10$$

i_r = pozice v řádku
i_s = pozice ve sloupci
n = zadaný počet bodů

4.3.3 Circe a ellipse

Vygeneruje zadaný počet bodů na kružnici či elipse se středem v bodě (210, 210). Poloměr kružnice je 200 a velikost os elipsy jsou a = 200, b = 100.

Pro kružnici:

$$\omega = 2\pi/n$$

$$x = x_S + 200 \cdot \cos(i \cdot \omega)$$
$$y = y_S + 200 \cdot \sin(i \cdot \omega)$$

Pro elipsu:

$$\omega = 2\pi/n$$

$$x = x_S + 200 \cdot \cos(i \cdot \omega)$$
$$y = y_S + 100 \cdot \sin(i \cdot \omega)$$

i = index bodu
S = střed kružnice/elipsy
n = zadaný počet bodů

4.3.4 Star shape

Vygeneruje zadaný počet bodů na dvou kružnicích (viz. 4.3.3) o různých poloměrech, jejichž hodnota se liší alespoň o 20. Středy kružnic jsou totožné (210, 210) a maximální poloměr je 200. Body se na první a druhou kružnici s přibývajícím úhlem umísťují střídavě.

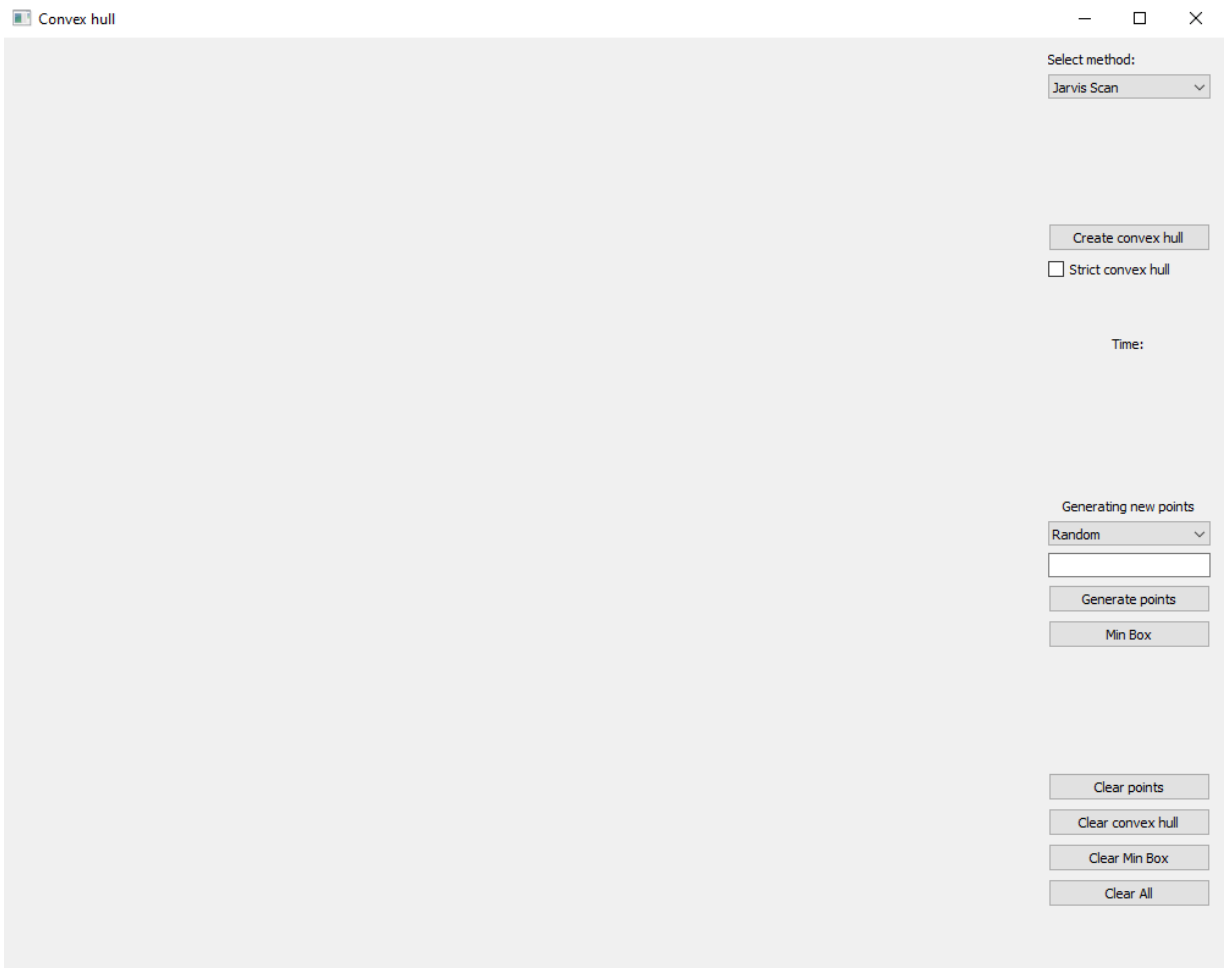
5. Vstupní data

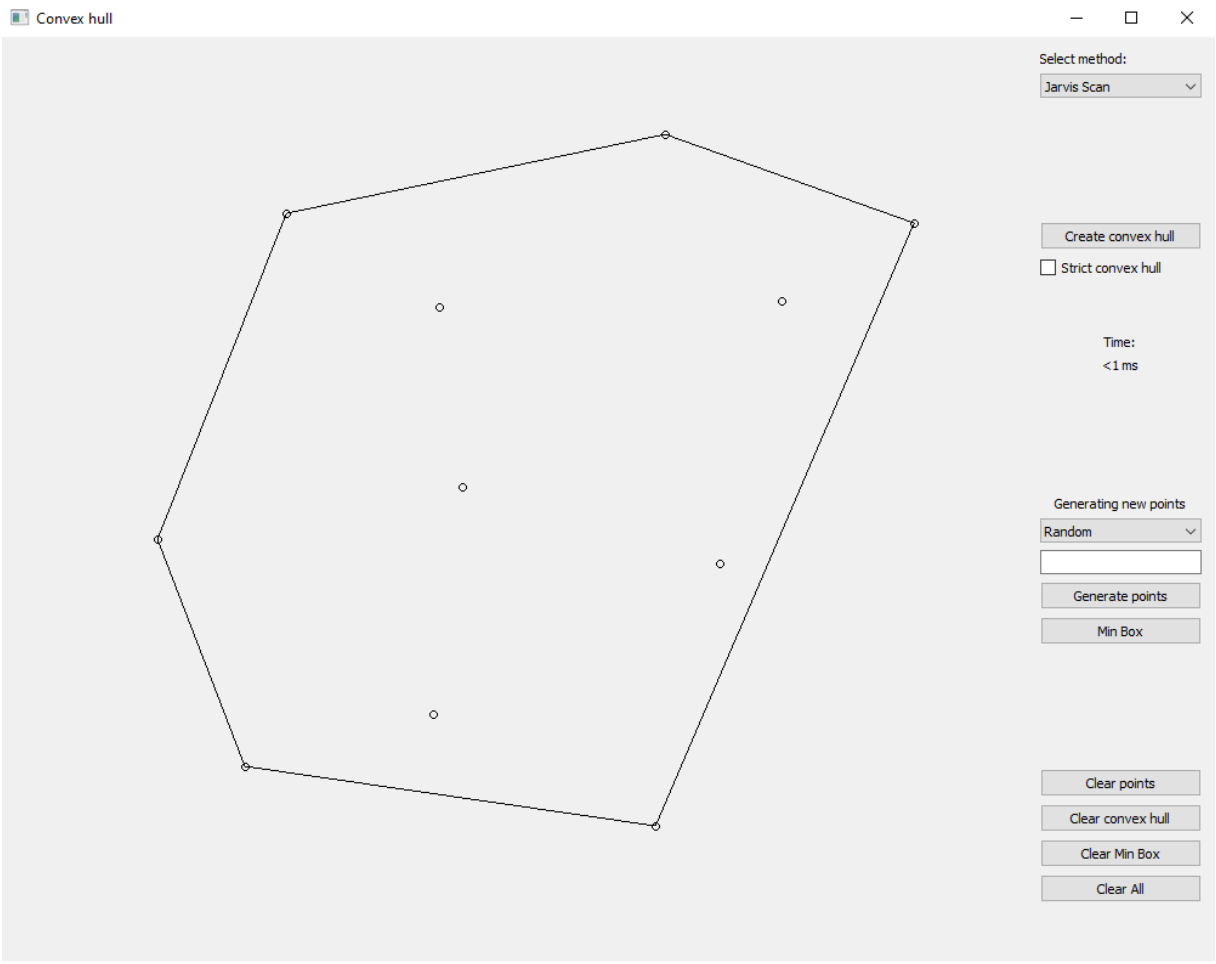
K aplikaci nejsou k dispozici žádná vstupní data ani možnost jejich načtení. K výpočtům se používá generování bodů dle zadaného tvaru, nebo lze body ručně zvolit klikáním levého tlačítka myši. Datový typ: std::vector<QPoint>

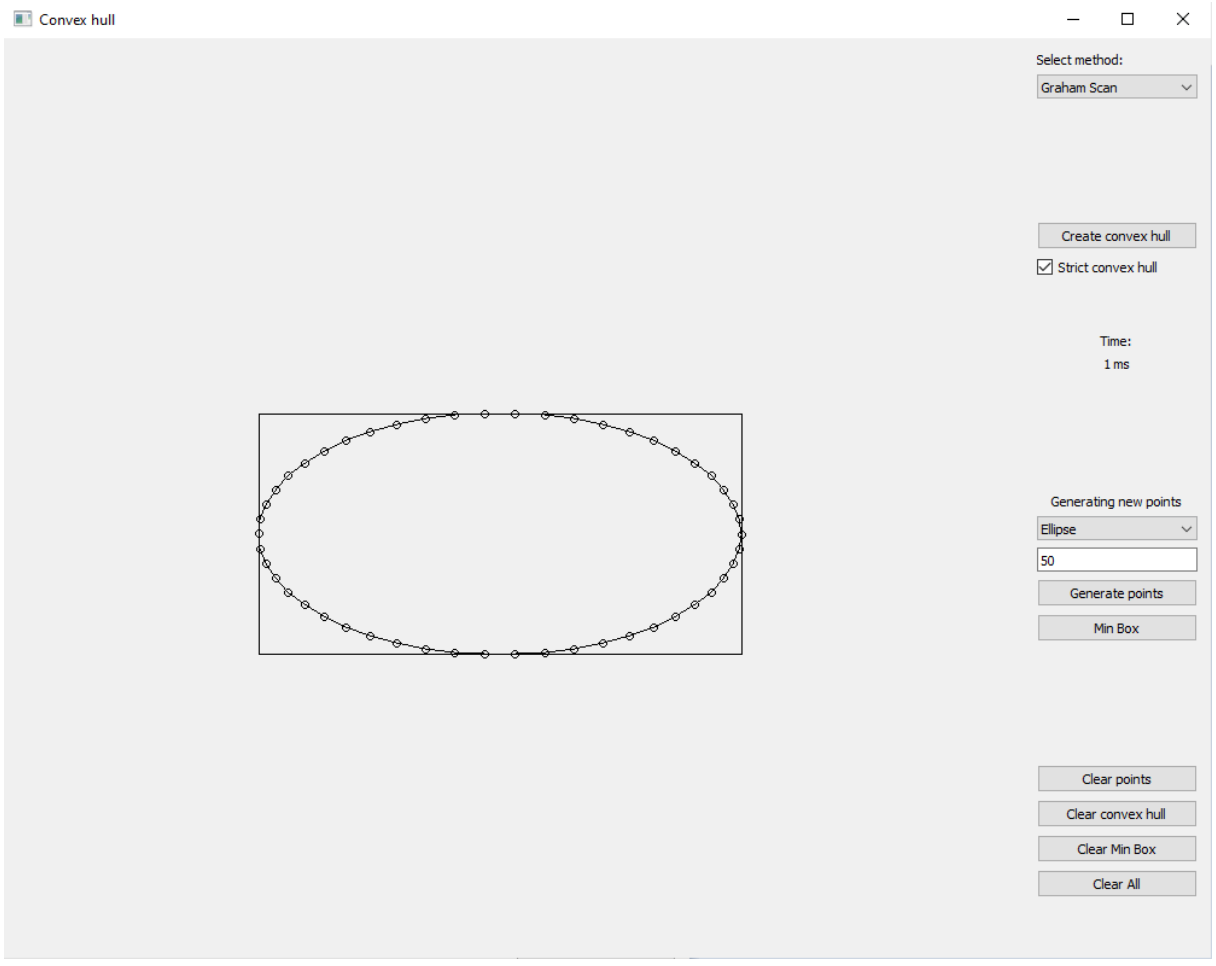
6. Výstupní data

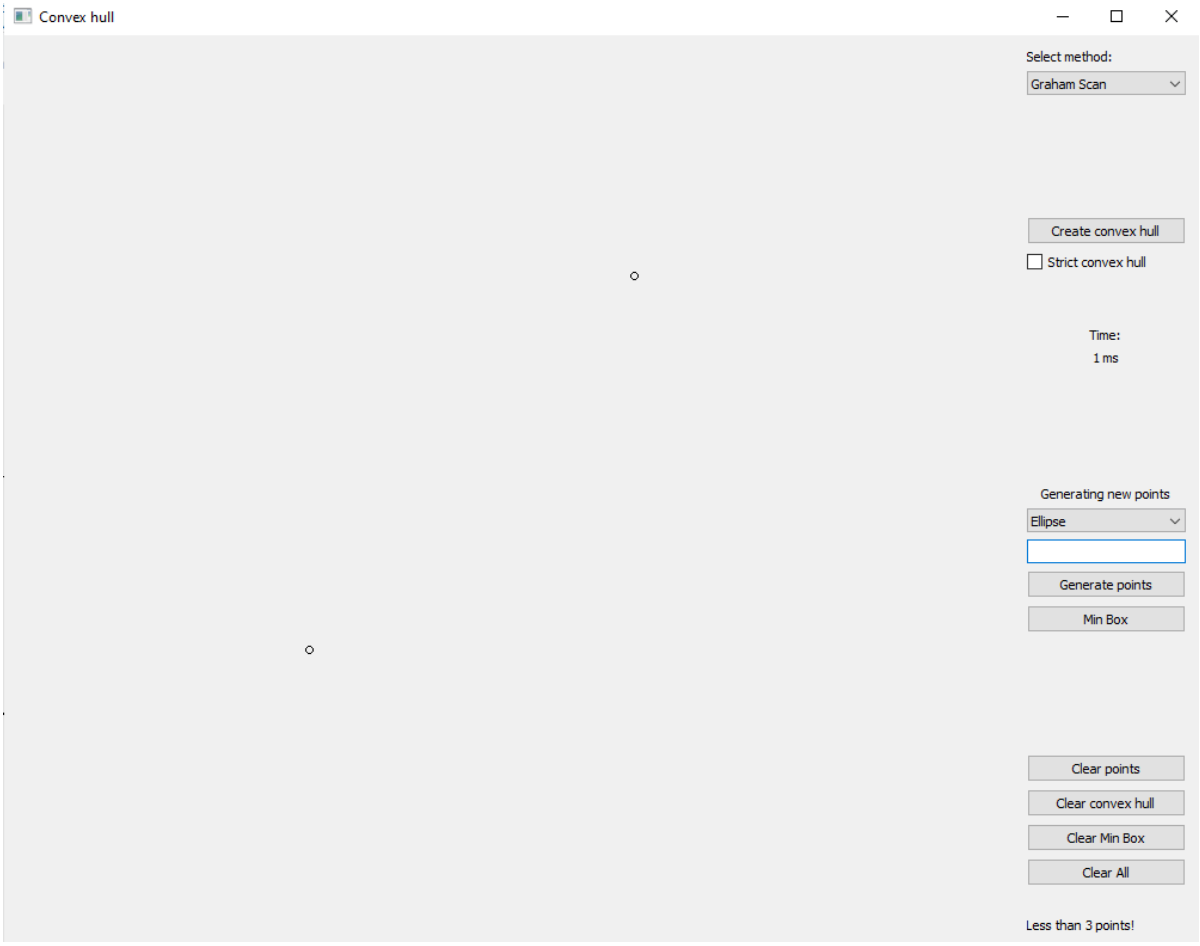
Aplikace neposkytuje žádná výstupní data. Výsledkem je vykreslení bodů, konvexní obálky a minimálního ohraničujícího obdélníku. Datový typ: QPoint, QPolygon, QPolygonF. Zároveň je zde uvedena doba trvání výpočtu.

7. Vzhled aplikace




Convex hull
Select method:
Jarvis Scan
Create convex hull
Strict convex hull
Time:
<1 ms
Generating new points
Random
Generate points
Min Box
Clear points
Clear convex hull
Clear Min Box
Clear All


Convex hull
Select method:
Graham Scan
Create convex hull
Strict convex hull
Time:
1 ms
Generating new points
Ellipse
50
Generate points
Min Box
Clear points
Clear convex hull
Clear Min Box
Clear All



8. Dokumentace

Třídy:

Algorithms:

Třída obsahuje 12 funkcí, které se používají pro analýzu zadaných množin bodů:

stati□int getPointLinePosition(QPoint &q, QPoint &p1, QPoint &p2);

Funkce zjišťuje, zda bod q leží vlevo či vpravo od vybrané úsečky spojující body p1 a p2. Také určuje, zda neleží přímo na úsečce. Funkce vrací hodnoty *int(1 pokud je bod vlevo, 0 pokud je bod vpravo, -1 pokud bod leží na přímce).*

static double getAngle2Ve□to□s(QPoint &p1, QPoint &p2, QPoint &p3, QPoint &p4);

Funkce vrací velikost úhlu mezi dvěma vektory, které jsou určeny body p1,p2 a p3,p4. Úhel je vracen v radiánech.

static double getPointLineDistance(QPoint &q, QPoint &p1, QPoint &p2);

Funkce vrací vzdálenost bodu q od linie tvořené body p1 a p2.

static double getDistance2Points(QPoint &p1, QPoint &p2);

Funkce vrací vzdálenost mezi body p1 a p2.

static void rotatePolygon(QPolygonF &pol, double angle);

Funkce transformuje polzgon pol o úhel angle.

stati□QPolygon ja□visS□an(std::ve□to□<QPoint> &points);
Funkce vrací konvexní obálku typu QPolygon, kterou počítá pomocí metody Jarvis Scan.

stati□QPolygon qHull(std::ve□to□<QPoint> &points);
Funkce vrací konvexní obálku typu QPolygon, kterou počítá pomocí metody Quick Hull

static QPolygon sweepLine(std::ve□to□<QPoint> &points);
Funkce vrací konvexní obálku typu QPolygon, kterou počítá pomocí metody Sweep Line.

stati□QPolygon g□ahamS□an(std::ve□to□<QPoint> &points);
Funkce vrací konvexní obálku typu QPolygon, kterou počítá pomocí metody Graham Scan.

stati□void qh(int s, int e, std::ve□to□<QPoint> &points, QPolygon &□h);
Rekurzivní funkce pro vyhledávání nejvzdálenějšího bodu v metodě Quick Hull.

static QPolygon st□□tCH(QPolygon &□h);
Funkce vrací konvexní obálku s odstraněnými body, které by jinak ležely na úsečce mezi jinými dvěma body.

static QPolygonF minimum□e□tanhle(QPolygon &□h);
Funkce vrací polygon opsaného obdélníka s nejmenší plochou. Používá přitom postupnou transformaci jednotlivých hran konvexní obálky a vyhledává nejmenší obsah. Jedná se tedy o metodu Brutal Force.

GeneratePoints:
Třída obsahuje 5 funkcí určené k automatickému generování množin bodů.

static std::vector<QPoint> gene□atingSqua□e(int &n);
Funkce generuje vektor bodů ve tvaru grid o velikosti n*n.

static std::vecto□<QPoint> gene□atingCi□□le(int &n);
Funkce generuje n-počet bodů na kružnici.

static std::ve□to□<QPoint> gene□atingEllipse(int &n);
Funkce generuje n-počet bodů na elipse.

stati□std::ve□to□<QPoint> gene□ating□andomField(int &n);
Funkce generuje n-počet náhodných bodů.

stati□std::ve□to□<QPoint> gene□atingSta□(int &n);
Funkce generuje n-počet bodů ve tvaru Star.

Draw:
Třída obsahuje 3 privátní proměnné a 10 funkcí. Třída se používá pro vykreslení bodů, konvexní obálky, obdélníka s minimální plochou a získání souřadnic určovaného bodu na obrazovce.

Proměnné v třídě draw:
std::ve□to□<QPoint> points;
Vektor bodů, se kterými bude program dále pracovat.

QPolygon ch;

Polygon obsahující konvexní obálku.

QPolygonF minBounda□y;

Polygon obsahující obdélník o minimální ploše.

Funkce v třídě draw:

*void mouseP□essEvent(QMouseEvent *e);*

Funkce snímá pozici myši při kliknutí.

*void paintEvent(QPaintEvent *e);*

Funkce vykresluje generované a vytvořené body, konvexní obálku a obdélník o minimální ploše.

void clearCH() {□h.□lea□(); □epaint();}

Funkce vymaže konvexní obálku a překreslí Canvas.

void clearPoints() {points.□lea□(); □epaint();}

Funkce vymaže body v *points* a překreslí Canvas.

void □lea□MinBounda□y() {minBounda□y.□lea□(); □epaint();}

Funkce vymaže polygon obdélníka s minimální plochou a překreslí Canvas.

void setCH(QPolygon &hull) {□h=hull;}

Funkce nastaví polygon konvexní obálky.

void setPoints(std::ve□to□<QPoint> &pts) {points=pts;}

Funkce nastaví vektor bodů.

void setMinBoundary(QPolygonF &bound) {minBounda□y=bound;}

Funkce nastaví polygon obdélníka s minimální plochou.

QPolygon getCH() {□etu□n □h;}

Funkce vrací konvexní obálku.

std::ve□to□<QPoint> getPoints() {□etu□n points;}

Funkce vrací vektor bodů.

SortbyY:

Operátor. Srovná QPointy podle velikosti Y. Pokud je Y stejné, řadí dle X.

SortbyYF:

Obdobný operátor jako SortbyY pro reálná čísla.

SortbyX:

Operátor. Srovná QPointy podle velikosti X. Pokud je X stejné, řadí dle Y.

SortbyXF:

Obdobný operátor jako SortbyX pro reálná čísla.

Widgets:

Třída obsahuje 7 funkcí, které rozhodují o následných procesech po stisknutí daného tlačítka.

void on_pushButton_□li□ked();

Funkce se spustí po stisknutí tlačítka Create convex hull a vytvoří konvexní obálku nad množinou bodů za pomocí metody, která se vybere v ComboBoxu nad tlačítkem.

void on_pushButton_2_□li□ked();

Funkce se spustí po stisknutí tlačítka Clear points a smaže vytvořené body.

void on_pushButton_3_□li□ked();

Funkce se spustí po stisknutí tlačítka Clear convex hull a smaže vytvořenou konvexní obálku.

void on_pushButton_4_□li□ked();

Funkce se spustí po stisknutí tlačítka Clear Min Box a smaže vytvořenou obdélník s minimální plochou.

void on_pushButton_4_□li□ked();

Funkce se spustí po stisknutí tlačítka Clear All a smaže vše, co bylo vytvořeno.

void on_pushButton_genPoints_□li□ked();

Funkce se spustí po stisknutí tlačítka Generate points a vygeneruje množinu bodů dle přednastavených parametrů. Ty jsou n: počet bodů a tvar množiny. Mřížka, kruh, elipsa, star shape, random.

void on_pushButton_minBox_clicked();

Funkce se spustí po stisknutí tlačítka Min Box a vytvoří obdélník s minimální plochou.

9. Závěr

Byla vytvořena aplikace podle zadání. Řešeny byly všechny doplňkové úlohy.

9.1 Naměřené hodnoty

Formát:	random [n]
	circle [n]
	ellipse [n]
	grid [n]

dle formátu:	Jarvis Scan t [ms]										Průměr [ms]	Rozptyl [ms]
1000	1	1	1	1	1	1	1	1	1	1	1,0	0,0
	4	4	4	4	3	3	3	4	5	4	3,8	0,6
	3	3	4	3	3	3	3	3	3	3	3,1	0,3
	1	1	1	1	1	1	1	1	1	1	1,0	0,0
2000	1	2	1	1	1	1	2	1	1	1	1,2	0,4
	7	7	7	8	8	7	7	7	6	7	7,1	0,6
	6	6	6	5	6	6	6	6	6	7	6,0	0,5
	1	1	1	1	1	1	1	1	1	1	1,0	0,0
5000	2	2	2	3	2	2	3	2	3	3	2,4	0,5
	18	19	19	19	18	18	18	19	18	18	18,4	0,5
	14	14	14	15	14	15	15	15	15	14	14,5	0,5
	1	1	1	1	1	1	1	1	1	1	1,0	0,0
10000	5	5	5	4	4	5	4	4	4	5	4,5	0,5
	40	40	40	41	41	41	40	40	39	41	40,3	0,7
	29	30	30	30	29	30	29	29	29	30	29,5	0,5
	2	1	1	2	1	1	1	2	1	2	1,4	0,5
50000	11	13	11	14	15	11	13	14	11	11	12,4	1,6
	184	184	184	184	184	184	183	183	185	183	183,8	0,6
	145	144	146	144	145	145	145	145	143	144	144,6	0,8
	9	9	9	9	10	9	10	9	9	9	9,2	0,4
100000	29	29	31	26	26	36	32	23	29	23	28,4	4,1
	341	341	341	341	340	341	343	340	345	341	341,4	1,5
	290	288	289	289	290	284	289	289	290	289	288,7	1,8
	19	18	18	19	19	19	19	19	19	19	18,8	0,4
250000	52	59	58	52	64	52	72	58	79	59	60,5	8,9
	851	845	851	851	852	851	850	851	849	849	850,0	2,0
	718	722	725	720	711	719	724	724	723	726	721,2	4,4
	47	47	47	47	48	47	48	47	47	47	47,2	0,4
500000	106	105	119	106	106	105	107	105	107	105	107,1	4,3
	1696	1691	1696	1709	1701	1717	1720	1716	1717	1775	1713,8	23,9
	1441	1443	1448	1445	1432	1445	1449	1448	1444	1441	1443,6	4,9
	95	94	95	94	95	95	95	95	94	95	94,7	0,5
750000	160	161	159	160	159	160	160	160	161	160	160,0	0,7
	2566	2569	2542	2551	2548	2543	2543	2527	2567	2530	2548,6	14,8
	2160	2175	2149	2170	2170	2169	2179	2168	2147	2169	2165,6	10,5
	140	141	142	142	141	141	141	140	140	141	140,9	0,7
1000000	213	215	213	215	215	215	213	213	211	214	213,7	1,3
	3427	3396	3394	3402	3379	3433	3388	3382	3381	3386	3396,8	19,0
	2874	2893	2904	2876	2904	2890	2893	2884	2890	2872	2888,0	11,5
	184	184	186	185	185	185	184	184	186	185	184,8	0,8

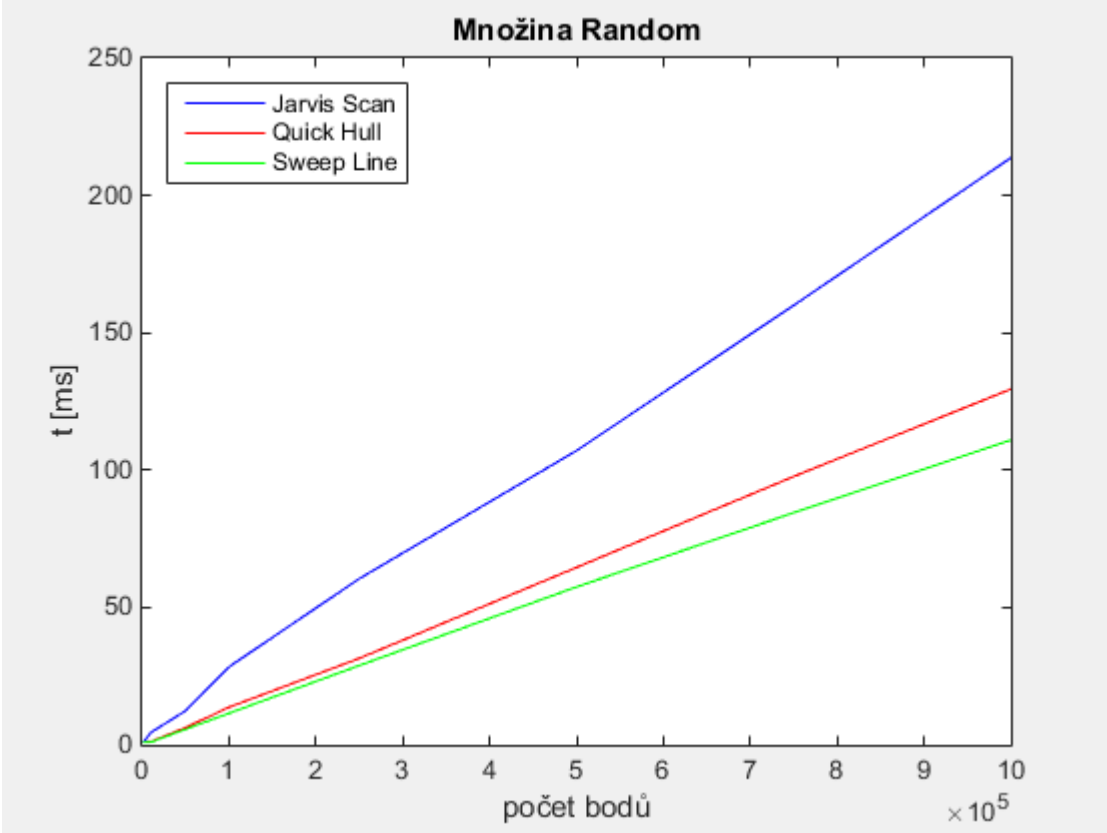
dle formátu:	Quick Hull t [ms]										Průměr [ms]	Rozptyl [ms]
1000	1	1	1	1	1	1	1	1	1	1	1,0	0,0
	1	1	1	1	1	1	1	1	1	1	1,0	0,0
	1	1	1	1	1	1	1	1	1	1	1,0	0,0
	1	1	1	1	1	1	1	1	1	1	1,0	0,0
2000	1	1	1	1	1	1	1	1	1	1	1,0	0,0
	1	1	1	1	1	1	1	1	1	1	1,0	0,0
	1	1	1	1	1	1	1	1	1	1	1,0	0,0
	1	1	1	1	1	1	1	1	1	1	1,0	0,0
5000	1	1	1	1	1	1	1	1	1	1	1,0	0,0
	2	3	3	2	2	2	3	2	3	2	2,4	0,5
	1	2	1	1	2	1	2	2	1	1	1,4	0,5
	1	1	1	1	1	1	1	1	1	1	1,0	0,0
10000	1	1	1	1	1	1	1	2	1	2	1,2	0,4
	5	4	4	4	5	4	4	5	4	5	4,4	0,5
	3	3	3	4	3	3	3	3	3	4	3,2	0,4
	1	1	1	1	1	1	1	1	1	1	1,0	0,0
50000	6	6	6	7	7	6	6	7	6	6	6,3	0,5
	22	21	21	21	21	21	21	21	21	21	21,1	0,3
	18	17	18	18	18	18	18	17	18	17	17,7	0,5
	2	2	2	2	2	2	2	2	2	2	2,0	0,0
100000	13	13	14	14	14	14	14	13	14	14	13,7	0,5
	41	40	40	40	40	40	40	40	40	41	40,2	0,4
	35	37	35	35	35	35	35	35	35	35	35,2	0,6
	4	4	4	4	5	5	4	4	4	4	4,2	0,4
250000	30	31	31	32	32	32	32	32	32	31	31,5	0,7
	99	99	99	98	99	98	98	98	99	100	98,7	0,7
	87	87	87	88	88	89	88	87	88	87	87,6	0,7
	13	13	14	14	14	14	14	14	14	14	13,8	0,4
500000	65	65	65	65	64	64	64	64	65	65	64,6	0,5
	200	200	200	200	200	199	202	200	200	198	199,9	1,0
	175	175	175	175	175	174	176	175	176	176	175,2	0,6
	31	31	31	31	32	32	32	31	32	30	31,3	0,7
750000	98	98	98	98	98	98	98	97	97	97	97,7	0,5
	302	301	301	301	301	302	301	300	301	301	301,1	0,6
	265	265	265	266	266	265	265	265	265	265	265,2	0,4
	49	50	49	48	50	49	49	50	50	49	49,3	0,7
1000000	130	129	129	129	131	130	130	129	129	129	129,5	0,7
	403	402	403	402	403	402	402	403	402	402	402,4	0,5
	354	353	354	354	353	353	353	352	352	353	353,1	0,7
	64	64	63	63	63	64	64	63	62	62	63,2	0,8

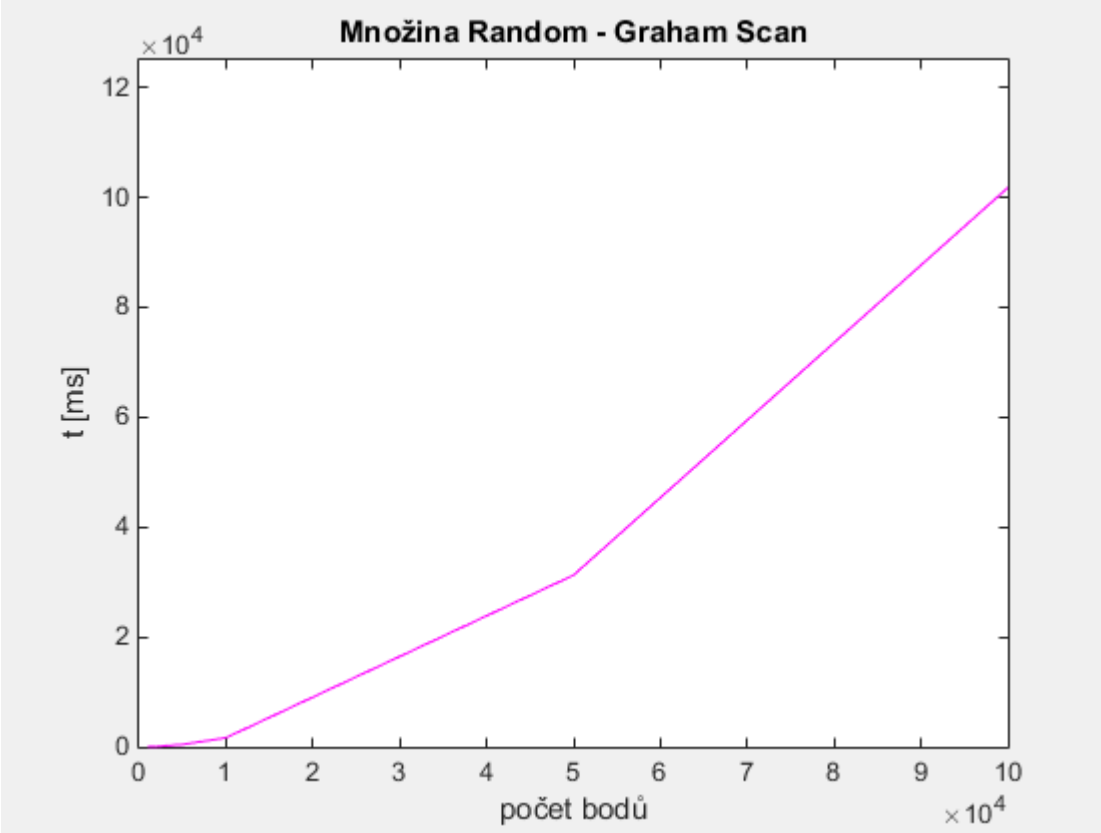
dle formátu:	Sweep Line t [ms]										Průměr [ms]	Rozptyl [ms]
1000	1	1	1	1	1	1	1	1	1	1	1,0	0,0
	1	1	1	1	1	1	1	1	1	1	1,0	0,0
	1	1	1	1	1	1	1	1	1	1	1,0	0,0
	1	1	1	1	1	1	1	1	1	1	1,0	0,0
2000	1	1	1	1	1	1	1	1	1	1	1,0	0,0
	1	1	1	1	1	1	1	1	1	1	1,0	0,0
	1	1	1	1	1	1	1	1	1	1	1,0	0,0
	1	1	1	1	1	1	1	1	1	1	1,0	0,0
5000	1	1	1	1	1	1	1	1	1	1	1,0	0,0
	1	1	1	1	1	1	1	1	1	1	1,0	0,0
	1	1	1	1	1	1	1	1	1	1	1,0	0,0
	1	1	1	1	1	1	1	1	1	1	1,0	0,0
10000	1	1	1	1	1	1	1	1	1	1	1,0	0,0
	1	1	1	1	1	1	1	1	1	1	1,0	0,0
	1	1	1	1	1	1	1	1	1	1	1,0	0,0
	1	1	1	1	1	1	1	1	1	1	1,0	0,0
50000	6	5	6	5	6	5	6	6	6	6	5,7	0,5
	1	1	1	2	1	2	2	1	1	1	1,3	0,5
	1	2	1	1	1	1	1	2	2	1	1,3	0,5
	2	2	2	2	2	2	2	2	2	2	2,0	0,0
100000	11	11	12	11	11	12	12	12	12	11	11,5	0,5
	3	3	3	4	3	3	3	3	3	3	3,1	0,3
	3	3	3	3	3	3	3	3	3	3	3,0	0,0
	5	4	5	5	5	5	5	5	4	5	4,8	0,4
250000	29	29	28	29	29	30	29	28	29	29	28,9	0,6
	7	8	8	7	8	7	7	7	8	7	7,4	0,5
	8	8	8	8	8	8	8	8	8	8	8,0	0,0
	12	12	12	12	12	12	12	12	12	13	12,1	0,3
500000	57	57	58	58	57	58	58	57	58	57	57,5	0,5
	16	16	16	16	16	16	16	17	16	17	16,2	0,4
	16	16	17	15	16	16	16	17	17	16	16,2	0,6
	24	24	23	23	23	23	24	23	23	23	23,3	0,5
750000	85	84	84	85	85	84	84	85	84	85	84,5	0,5
	25	25	25	24	25	25	25	25	25	24	24,8	0,4
	24	25	25	24	25	24	25	24	25	25	24,6	0,5
	34	34	34	34	34	34	35	34	34	36	34,3	0,7
1000000	111	111	111	112	110	110	112	112	111	110	111,0	0,8
	33	34	33	33	34	33	33	33	34	34	33,4	0,5
	33	34	34	34	34	33	32	34	33	33	33,4	0,7
	43	43	42	43	43	43	43	43	43	42	42,8	0,4

dle formátu:	Graham Scan t [ms]										Průměr [ms]	Rozptyl [ms]
1000	16	9	20	12	19	19	17	9	8	21	15,0	5,0
	13	13	21	14	14	13	14	16	18	13	14,9	2,7
	14	12	11	13	11	14	11	14	12	12	12,4	1,3
	13	13	14	14	13	14	14	13	14	14	13,6	0,5
2000	71	69	64	66	64	68	76	56	73	63	67,0	5,7
	35	36	39	35	37	39	39	38	36	35	36,9	1,7
	26	27	30	27	28	26	26	29	29	30	27,8	1,6
	57	57	58	58	58	58	59	59	59	58	58,1	0,7
5000	456	446	443	418	425	435	402	333	439	455	425,2	36,5
	93	99	95	100	95	93	94	102	95	94	96,0	3,2
	74	69	70	70	71	71	73	68	74	68	70,8	2,3
	398	401	394	401	400	401	420	397	421	399	403,2	9,4
10000	1640	1665	1635	1662	1659	1655	1661	1669	1647	1649	1654,2	11,1
	194	203	193	191	201	195	200	194	190	201	196,2	4,6
	141	132	144	142	139	142	135	136	134	140	138,5	4,0
	1614	1553	1567	1564	1543	1554	1569	1539	1572	1588	1566,3	22,1
50000	31190	31316	31266	31278	31309	31208	31258	31194	31245	31291	31255,5	45,8
	960	1010	962	991	976	984	989	970	1004	1003	984,9	17,7
	673	670	692	711	682	697	705	672	686	712	690,0	16,0
	33800	33652	34686	33504	33644	33971	33950	33561	33598	33783	33814,9	344,3
100000	101845	101730	101691	101783	101801	101920	101776	101721	101753	101795	101781,5	65,9
	2074	1937	1973	2037	2021	1956	1984	2077	1995	1969	2002,3	48,3
	1364	1342	1349	1380	1366	1371	1360	1349	1355	1382	1361,8	13,4
	123358	123461	123522	123201	123487	123339	123640	123564	123545	123372	123448,9	130,7
250000											Příliš dlouhý čas	
	5117	4935	4846	4995	5025	4921	4984	5003	5087	4871	4978,4	87,1
	3516	3441	3354	3398	3410	3495	3547	3456	3473	3485	3457,5	58,3
											Příliš dlouhý čas	
500000											Příliš dlouhý čas	
	10053	9716	10264	9900	9983	9997	10105	9951	10047	9894	9991,0	145,1
	6932	7039	6955	7135	6982	7063	7049	7084	6991	7032	7026,2	61,8
											Příliš dlouhý čas	
750000											Příliš dlouhý čas	
	14914	15020	15462	15332	15293	15051	15368	14975	15472	15530	15241,7	229,8
	10340	10124	10271	10212	10356	10176	10077	10234	10263	10158	10221,1	90,1
											Příliš dlouhý čas	
1000000											Příliš dlouhý čas	
	20099	19995	19966	19987	20090	20103	20015	20039	20021	19981	20029,6	51,3
	13508	13734	13530	13657	13606	13525	13697	13644	13619	13740	13626,0	84,8
											Příliš dlouhý čas	

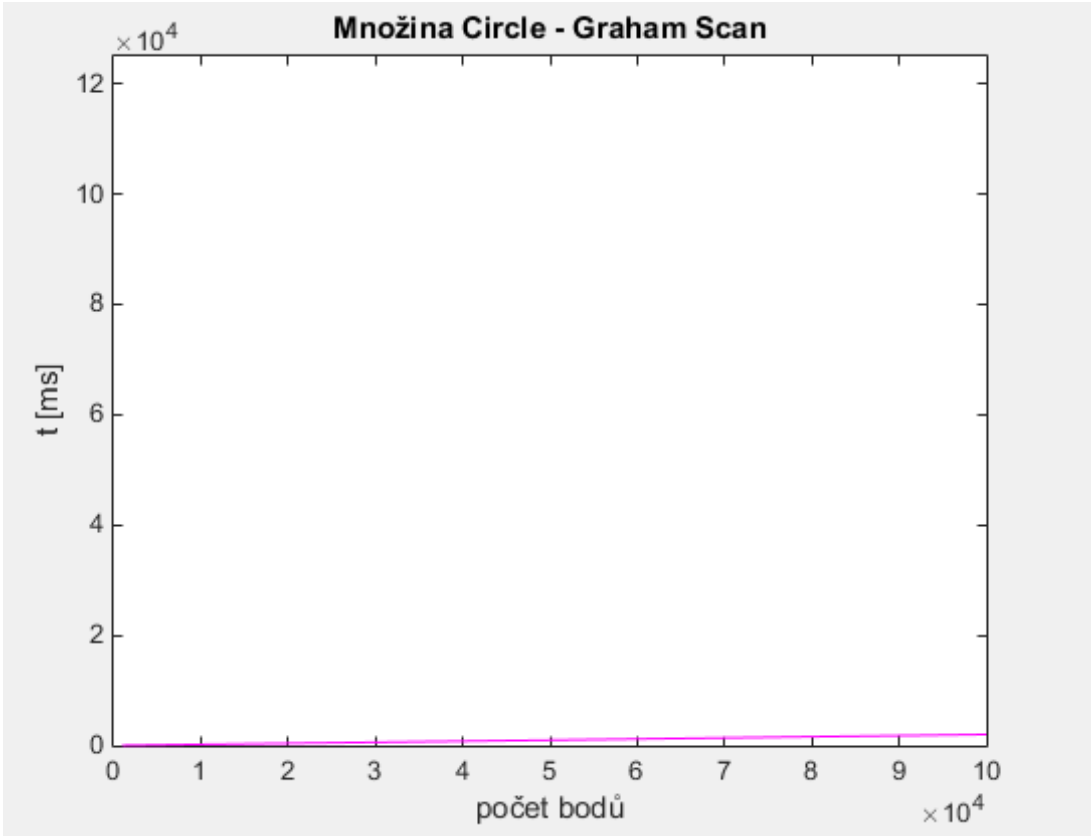
9.2 Grafy

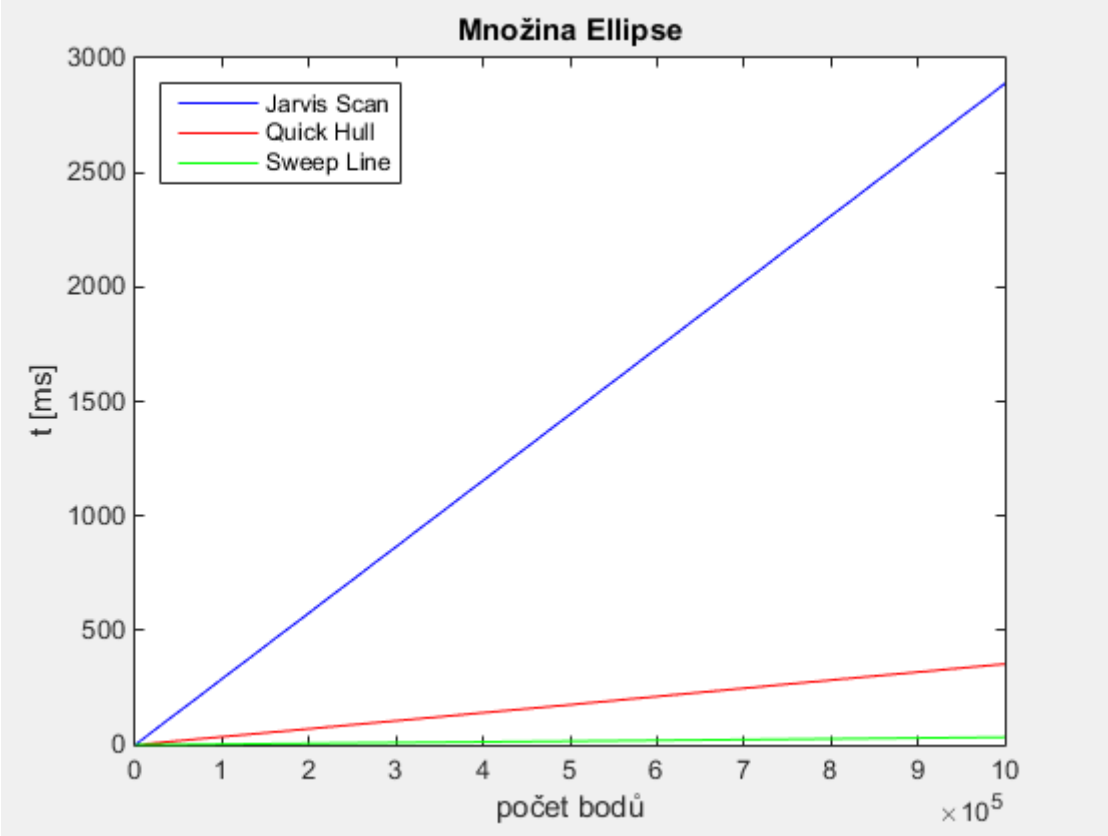
Grafy byly vytvořeny zvlášť pro každou vstupní množinu bodů. Jelikož je třídění bodů v algoritmu Graham Scan neefektivně napsáno, jsou doby běhu tohoto algoritmu několikanásobně vyšší než u ostatních algoritmů, tudíž byly pro Graham Scan vytvořeny samostatné grafy s pevnou osou y (doba běhu) pro porovnání.

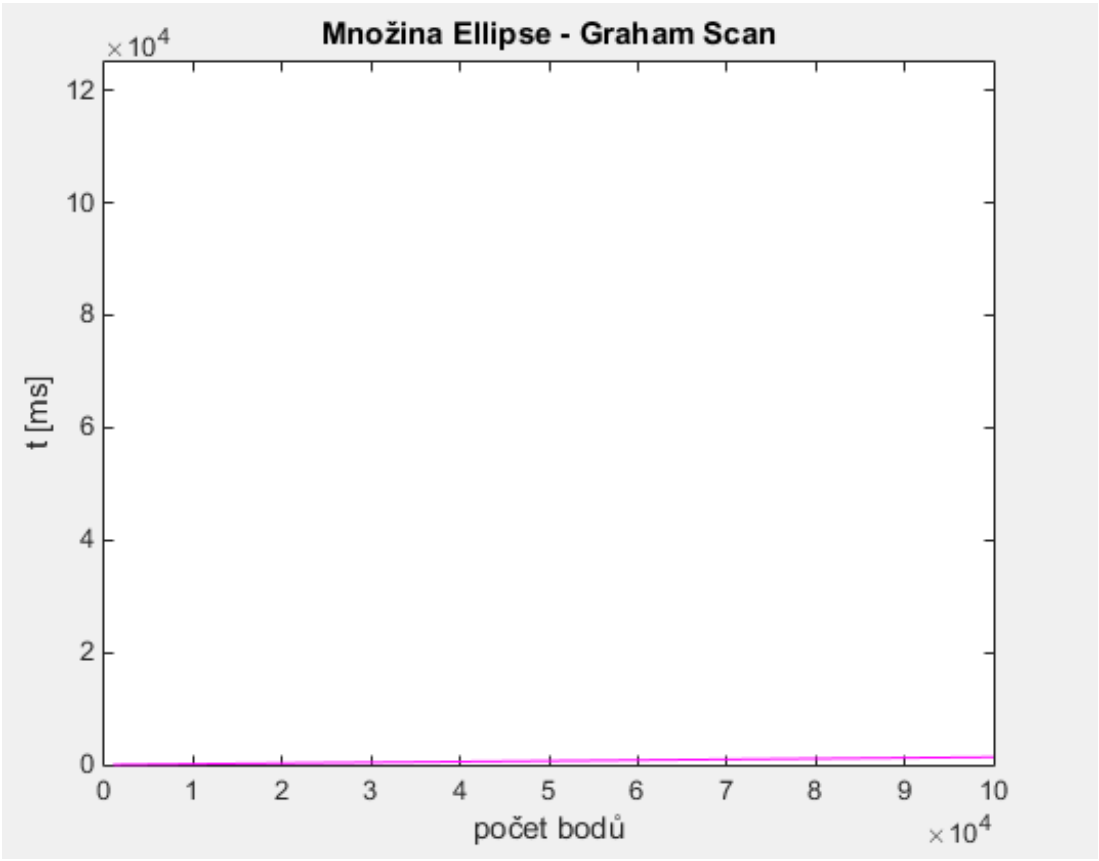


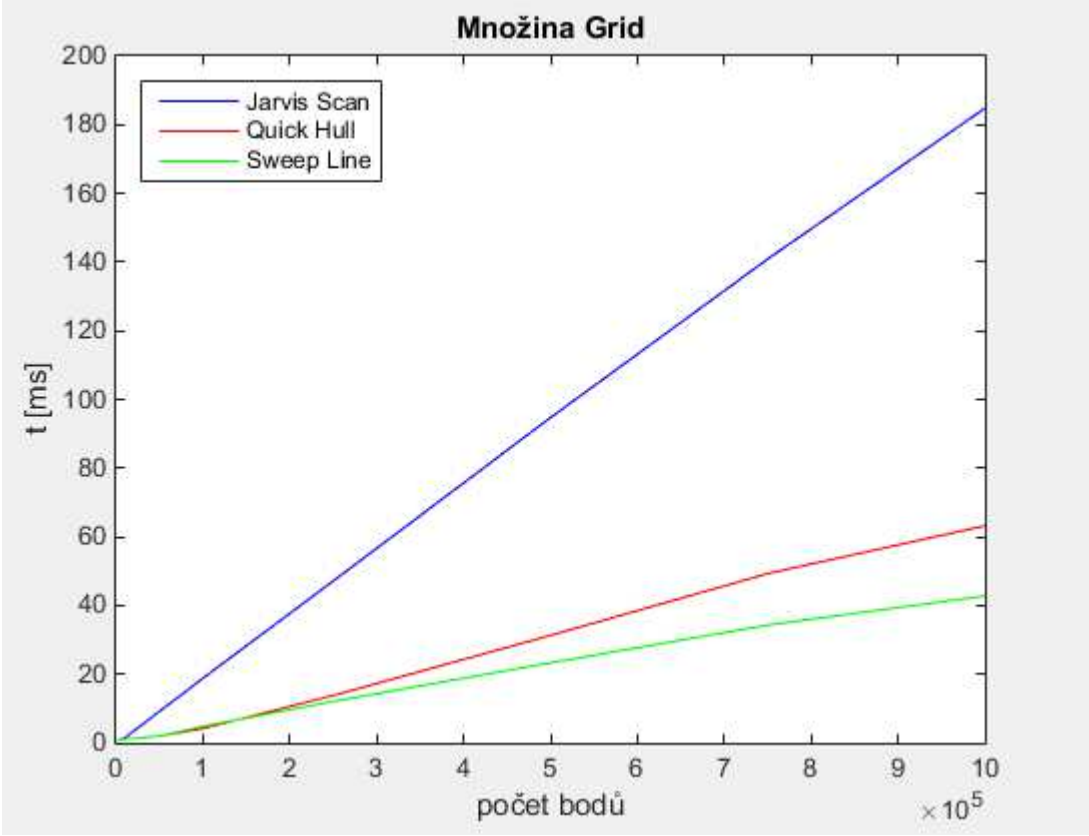


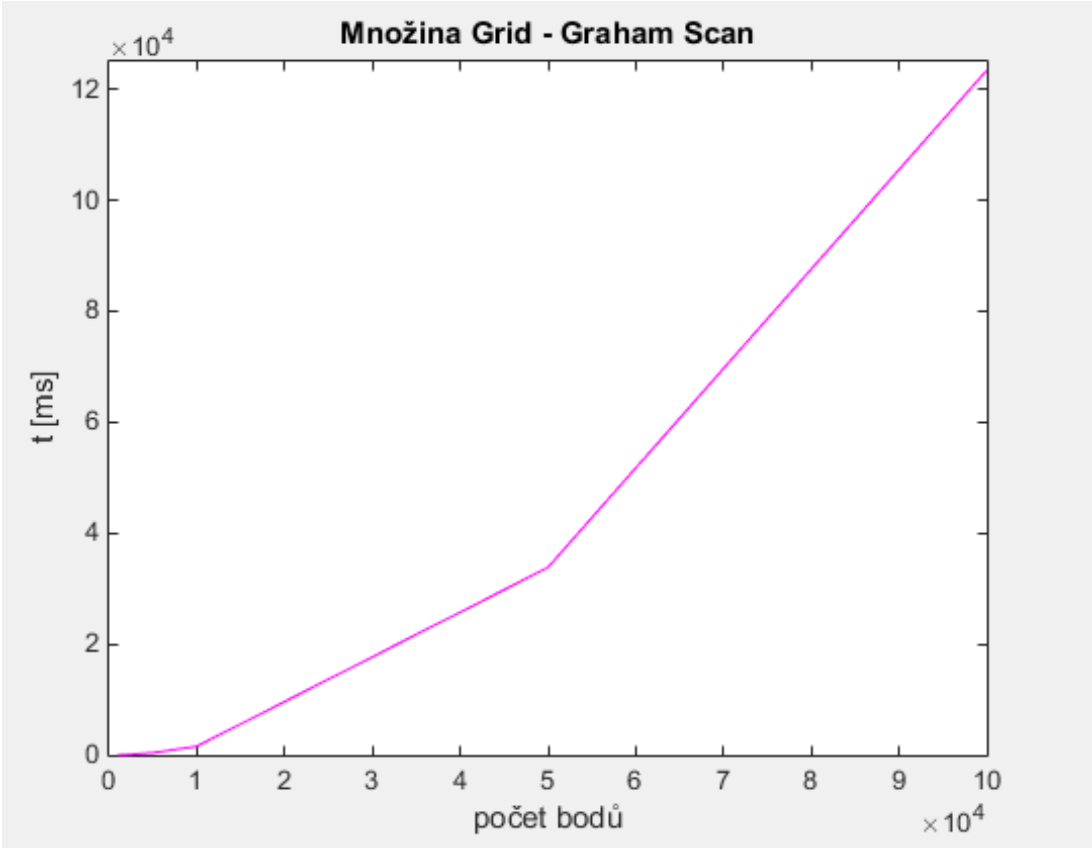

Množina Circle
Jarvis Scan
Quick Hull
Sweep Line
t [ms]
počet bodů
×10^5


Množina Circle - Graham Scan
×10^4
t [ms]
počet bodů
×10^4


Množina Ellipse
Jarvis Scan
Quick Hull
Sweep Line
t [ms]
počet bodů
×10^5


Množina Ellipse - Graham Scan
×10^4
t [ms]
počet bodů
×10^4


Množina Grid
Jarvis Scan
Quick Hull
Sweep Line
t [ms]
počet bodů
×10^5


Množina Grid - Graham Scan
×10^4
t [ms]
počet bodů
×10^4

9.3 Zhodnocení měřených dat

Nejlépe z testovaných algoritmů dopadl algoritmus Sweep Line pro všechny generované množiny bodů.

Algoritmus Jarvis Scan není příliš efektivní, nicméně je dobře využitelný pro náhodně množiny nebo čtvercovou síť.

Algoritmus Quick Hull poskytuje podobné výsledky pro všechny generované množiny (cca 100 ms – 400 ms pro 1000000 bodů). Disponuje vysokou efektivitou, nicméně je vhodný spíše pro náhodné množiny nebo čtvercovou síť.

Algoritmus Sweep Line řeší rychleji kruhové a eliptické množiny bodů než body náhodné či uspořádané v mřížce.

Algoritmus Graham Scan brzdí neefektivní řazení bodů podle úhlů. Z výsledků je ale patrné, že je velmi dobře využitelný pro kruhové a eliptické množiny. V případě rychlejšího řazení bodů ve zdrojovém kódu předpokládáme, že by mohl některým algoritmům, které nejsou vhodné pro tyto množiny, konkurovat (např. Jarvis Scan).

9.4 Další poznámky

Zdrojový kód algoritmu Graham Scan byl testován ve více verzích Qt Creatoru a není se staršími verzemi kompatibilní. Byl shledán kompatibilním s verzí 4.10.0 založené na 5.13.1.

Porovnání Graham Scanu a ostatních algoritmů není přesné. Graham Scan pravděpodobně nebyl napsán efektivně.

Do Aplikace byly přidány dodatečné funkce na mazání bodů, konvexní obálky a minimálního ohraničujícího obdélníku. Je zde také widget pro chybové hlášení, který je využit v případě nedostatečného počtu bodů pro tvorbu obálky nebo generování počtu bodů menšího než 1.

9.5 Náměty na zlepšení

Zdrojový kód Graham Scanu, především třídění dle úhlů, by se dal napsat efektivněji.
Bylo by možné ošetřit, aby aplikace hlásila zadání bodů pouze v přímce. Je pravděpodobné, že pro tuto konstelaci program havaruje.

10. Přílohy

1) Zdrojový kód aplikace (algorithms.h / .cpp; draw.h / .cpp; generatepoints.h / .cpp; CH.pro; main.cpp; sortbyx.h/ .cpp; sortbyxf.h/ .cpp; sortbyy.h/ .cpp; sortbyyf.h/ .cpp; widgets.h / .cpp / .ui)

5. 11. 2019
David Němec, Jan Šartner

Oprava TZ a zdrojového kódu: 1. 12. 2019