České vysoké učení technické v Praze
Fakulta stavební

Algoritmy v digitální kartografii

Konvexní obálky

Bc. Petra Pasovská
Bc. David Zahradník

Obsah

1 Zadání

Níže uvedené zadání je kopie ze stránek předmětu.

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**

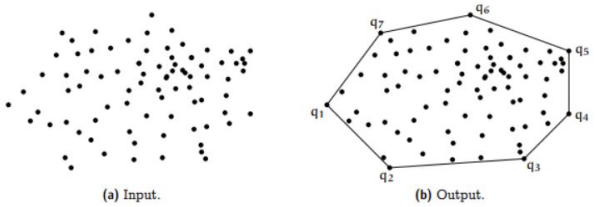
1.1 Údaje o bonusových úlohách

Z bonusových úloh byl vytvořen algoritmus pro automatické generování množin bodů různých tvarů. Uživatel si může v aplikaci sám zvolit, zda chce vygenerovat kruh, elipsu, čtverec, star-shaped či grid. Byla vložena i možnost náhodného rozmístění bodů v daném zobrazovacím okně. Dále byl vytvořen algoritmus pro výpočet konvexní obálky metodou Graham Scan. V rámci aplikace je i fungující výpočet konstrukce Minimum Area Enclosing Box. V neposlední řadě byly všechny vytvořené konvexní obálky zredukovány tak, aby vytvářely striktně konvexní obálky.

2 Popis a rozbor problému

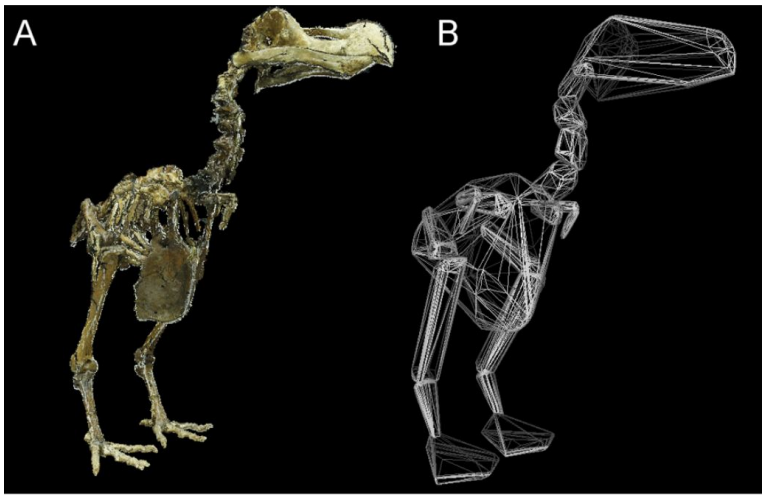
Hlavním cílem této úlohy je tvorba aplikace, která pro vygenerované množství bodů vytvoří konvexní obálku za pomoci různých algoritmů. Pro jednotlivé metody byla zapisována i doba trvání algoritmu. Výsledné časy jsou v závěru následně porovnány.

Lze říci, že konvexní obálka množiny M je nejmenší konvexní množina, která množinu M obsahuje. V současné době mají konvexní obálky, v některých literaturách označovány jako konvexní obaly, mnoho využití. Často se využívají jako první odhad tvaru nějakého prostorového jevu, např. detekce kolizí, detekce natočení budov a jejich tvaru v kartografii, analýza shluků atd. [Zdroj: 1]



Obrázek 1: Ukázka vstupních bodů, kolem nichž je vytvořena konvexní obálka. [zdroj: 2]

Konvexní obálky využívá celá řada vědních oborů. Zajímavé bylo využití konvexních obálek v paleontologii, kde za pomoci konvexních obálek jsou vědci schopni určit přibližně tvar těla vyhynulých živočichů, jejichž kosti byly nalezeny.



Obrázek 2: Využití konvexních obálek v paleontologii [zdroj: 3]

3 Popis použitých algoritmů

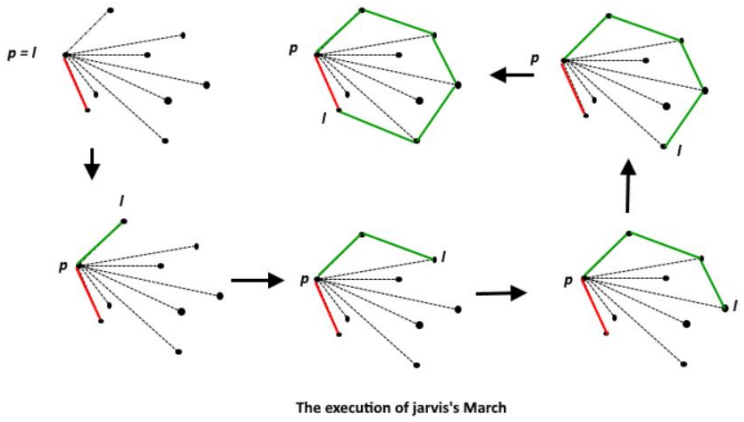
Existuje několik způsobů jak vytvořit konvexní obálku. V této úloze byly použity pro tovrbu 4 metody - Jarvis Scan, Quick Hull, Sweep Line a Graham Scan.

3.1 Jarvis Scan

Tato metoda bývá přirovnávána ke způsobu balení dárků (alternativní název Gift Wrapping Algorithm). Předpokladem pro algoritmus Jarvis Scan je, že 3 body nesmí ležet na jedné přímce. Metoda je poměrně snadná pro zápis, velkou nevýhodou je však časová náročnost O(n^2), které lze dosáhnout, pokud body z množiny S leží na kružnici. Běžný čas výpočtu bývá O(n*h), kde n je počet vstupních bodů a h je počet bodů, které tvoří obálku. [zdroj: 1]

Metoda je pojmenována po R. A. Jarvisu, který ji publikoval v roce 1973.

Abychom byli schopni algoritmus sestavit, je potřeba nalézt pivot, označme jej q. Nalezení pivota má časovou náročnost O(n). Pivota nalezneme jako bod s minimální hodnotou souřadnice Y. Následně porovnáváme úhel, který svírá pivot a bod následující a předcházející pivotu, dokud nenalezneme maximální úhel. Když takovýto úhel nalezeneme, je přidán mezi body konvexní obálky. V algoritmu dojde k přeindexování bodů a jsou porovnávány následující body, dokud nově vložený bod není pivot.



Obrázek 3: Princip Jarvis Scan algoritmu [zdroj: 4]

3.1.1 Problematické situace

K chybě v algoritmu může dojít v případě, že v množině bodů bude existovat více bodů s minimální Y souřadnicí. V tom případě se jedná o kolineární body. Tento problém lze ošetřit výběrem pivota s minimální Y souřadnicí a maximální X souřadnicí.

Ošetření problematické situace:

1. Nalezení pivota q: $q = min(y_i)$
2. Při nalezení více bodů s minimální souřadnicí: $maxX_{\forall q \in P}$

3.1.2 Implementace metody

1. Nalezení pivota q: $q = min(y_i)$
2. Přidej bod q do konvexní obálky: $q \rightarrow H$
3. Inicializuj: $p_j = q; p_{j+1} = p_{j-1}$
4. Opakuj, dokud: $p_{j+1} \neq q$

   Nalezni p_{j+1}: $p_{j+1} = argmax_{\forall p_i \in P} \angle(p_{j-1}, p_j, p_i)$

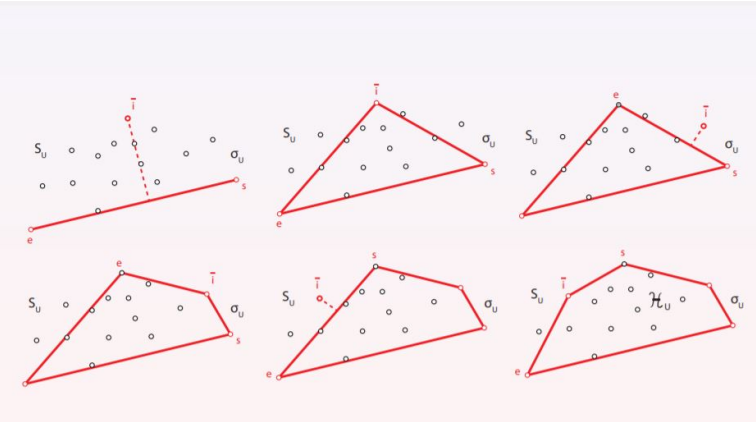
   Přidej p_{j+1}: $p_{j+1} \rightarrow H$

   Přeindexování bodů: $p_{j-1} = p_j; p_j = p_{j+1}$

3.2 Quick Hull

Metoda Quick Hull slouží k vytvoření konvexní obálky nad konečným počtem bodů. Využívá techniku "Rozděl a panuj", označovanou anglicky "Divide and Conquer". V této metodě lze najít analogii s QuickSortem, odkud také pochází označení algoritmu. Jedná se o poměrně rychlý algoritmus, časová náročnost je O(n*log(n)), v nejhorším případě je však časová náročnost kvadratická O(n^2).

Pro výpočet metodou Quick Hull je nejprve zapotřebí nalézt extrémní body, v aplikaci byly souřadnice setříděny podle x-ové souřadnice. Bod s nejnižší a nejvyšší hodnotou x-ové souřadnice je vložen do množiny, do které uchováváme body konvexní obálky. Těmito body je vedena pomyslná přímka, která množinu bodů rozdělí na dvě množiny - horní a dolní. V každé polorovině nalezneme nejvzdálenější bod od přímky, tento bod přidáme do množiny bodů patřících do konvexní obálky a vytvoříme přímky tohoto bodu a krajních bodů předešlé přímky. Následně pokračujeme analogicky a nad každou nově vzniklou přímkou nalezneme nejvzdálenější bod.



Obrázek 4: Princip Quick Hull algoritmu [zdroj: 5]

3.2.1 Implementace globální metody

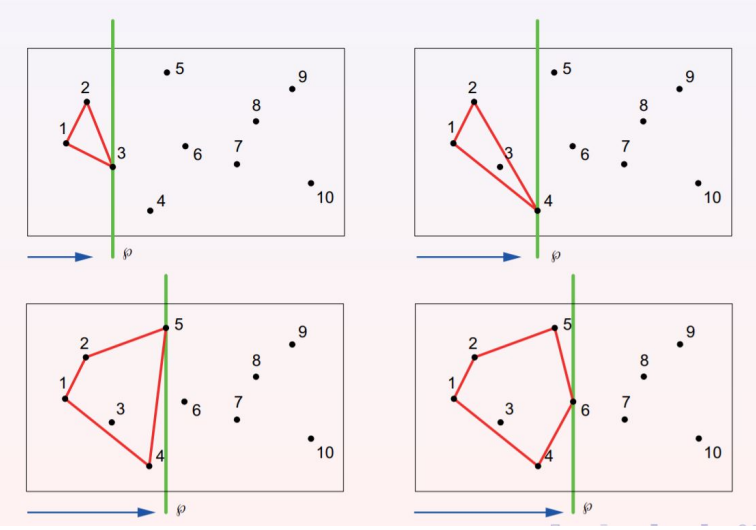
1. Vytvoření množiny konvexní obálky, horní a dolní množiny: $H = 0; S_U = 0; S_L = 0$
2. Nalezení extrémních hodnot: $q_1 = min_{\forall p_i \in S}(x_i); q_3 = max_{\forall p_i \in S}(x_i)$
3. Přidání extrémních bodů do horní a dolní množiny: $S_U \leftarrow q_1; S_U \leftarrow q_3; S_L \leftarrow q_1; S_L \leftarrow q_3$
4. Pro všechny body množiny: $\forall p_i \in S$

   Rozhodnutí, zda bod patří do horní množiny: $if(p_i \in \sigma_l(q_1, q_3))S_U \leftarrow p_i$

   V opačném případě: $S_L \leftarrow p_i$
5. Přidání krajního bodu do konvexní obálky: $H \leftarrow q_3$
6. Nalezení nejvzdálenějšího bodu c v horní části od přímky, přidání do množiny konvexní obálky a opakování vůči nově vzniklé přímce.
7. Přidání krajního bodu do konvexní obálky: $H \leftarrow q_1$
8. Opakování hledání nejvzdálenějšího bodu v dolní části.

3.3 Sweep line

Metoda Sweep line, v češtině označovaná také jako metoda zametací přímky, využívá strategii inkrementální konstrukce. Množinu bodů si v dané dimenzi rozdělíme na zpracovanou a nezpracovanou část. Rozdělovací kritérium je ve většině případů jedna ze souřadnicových os. Je zde tedy nutné body podle této osy seřadit a následně vyhodnocovat každý následující nezpracovaný bod. Body se vyhodnocují v závislosti na jejich poloze vůči tečně. Časová složitost této metody je O(n log(n)).



Obrázek 5: Princip algoritmu zametací přímky [zdroj: 5]

3.3.1 Problematické situace

Problémy v této metodě mohou nastat v případě, že na vstupu budou duplicitní body. Tyto duplicitní body vytvoří singularity, které znemožní správný výpočet algoritmu. Tuto situaci je nutné ošetřit na začátku algoritmu tím, že budou z množiny všech bodů odstraněny duplicitní body.

Ošetření problematické situace:

1. Nalezení duplicitního bodu: $if((points[j].x == points[i].x)\&\&(points[j].y == points[i].y))$
2. Odstranění bodu z množiny: $Delete\ points[i]$

3.3.2 Implementace metody

1. Seřazení bodů množiny podle osy x: $SortP_s = sort(P)\ by\ x$
2. Vyhodnocení v případě, že bod p_3 leží v levé polorovině od přímky p_1, p_2: $if(p_3 \in \sigma_L(p_1, p_2))$

   Změna indexů následovníků: $n[1] = 2; n[2] = 3; n[3] = 1$

   Změna indexů předchůdců: $p[1] = 3; p[2] = 1; p[3] = 2$
3. Vyhodnocení v případě, že bodp_3 leží v levé polorovině od přímky p_1, p_2: $if(p_3 \in \sigma_P(p_1, p_2))$

   Změna indexů následovníků: $n[1] = 3; n[3] = 2; n[2] = 1$

   Změna indexů předchůdců: $p[1] = 2; p[3] = 1; p[2] = 3$
4. Vyhodnocování následujících bodů: $for\ p_i \in P_S, i$>3

   Porovnání hodnoty souřadnice y: $if(y_i > y_{i-1})$

   Změna indexů při splnění podmínky: $p[i] = i - 1; n[i] = n[i - 1]$

   V opačném případě: $p[i] = p[i - 1]; n[i] = i - 1$

   Přeindexování následníka předchůdce a předchůdce následníka: $n[p[i]] = i; p[n[i]] = i$

   Zhodnocení polohy následníka následníka vůči přímce bodu a následníka: $while(n[n[i]]) \in \sigma_R(i, n[i])$

   Změna indexů: $p[n[n[i]]] = i; n[i] = n[n[i]]$

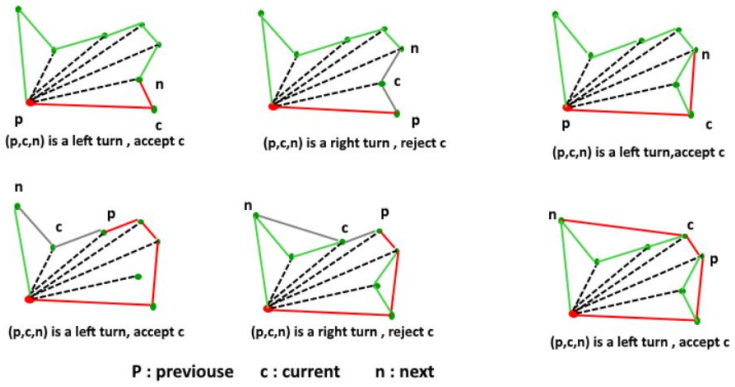
   Zhodnocení polohy předchůdce předchůdce vůči přímce bodu a předchůdce: $while(p[p[i]]) \in \sigma_L(i, p[i])$

   Změna indexů: $n[p[p[i]]] = i; p[i] = p[p[i]]$

3.4 Graham Scan

Algoritmus Graham Scan slouží k vytvoření konvexní obálky nad konečným množstvím bodů. Metoda je pojmenována po Ronaldu Grahamovi, který ji publikoval v roce 1972. Časová náročnost metody je O(n log(n)). Hlavní myšlenka algoritmu je taková, že každá uspořádaná trojice bodů musí splňovat kritérium levotočivosti (uvažujeme uspořádání hran v CCW orientaci - proti směru hodinových ručiček).

Pro výpočet je nejprve nutné body seřadit dle námi zvoleného kritéria, v tomto případě podle souřadnice Y. Bod s nejnižší souřadnicí Y označíme jako pivot. Následně je vypočtena směrnice s osou X vůči pivotu všech ostatních bodů. Body jsou následně podle tohoto úhlu setříděny. Po setřídění je možné přejít k vyhodnocování polohy bodů, kde jsou vždy testovány 2 poslední přidané body v polygonu konvexních obálek a následující setříděný bod.



Obrázek 6: Princip vyhodnocení polohy bodu v metodě Graham Scan [zdroj: 6]

3.4.1 Problematické situace

Problém může nastat, pokud nalezneme dva body, které svírají s pivotem a osou X stejný úhel. Tato situace je ošetřena tím způsobem, že při nalezení stejného úhlu je vypočtena vzdálenost pivota a obou bodů. Následně je vzdálenost porovnána a bod s kratší vzdáleností je z množiny bodů odstraněn a do dalšího výpočtu již není uvažován.

Tuto problematickou situaci ošetřujeme uvnitř kódu (viz Implementace metody - 3. řádek).

3.4.2 Implementace metody

1. Nalezení pivota q: $q = min_{\forall p_i \in S}(y_i), q \in H$
2. Setřídění bodů dle úhlu s osou x: $\forall p_i \in S$ sort by $\omega_i = \angle(p_i, q, x)$
3. Při nalezení stejného úhlu: $\omega_k = \omega_l \rightarrow$ delete the closer point
4. Vložení pivota a prvního bodu do množiny: $H \leftarrow q; H \leftarrow p_1$

5. Opakuj pro všechna: $forj < n$
6. Vyhodnocení polohy bodu: if p_j vpravo od předešlých bodů $\rightarrow popS$
7. V opačném případě přidej bod do konvexní obálky: $p_j \rightarrow H$

4 Informace o bonusových úlohách

V podkapitolách jsou blíže popsány bonusové úlohy kromě algoritmu Graham Scan, ten byl již vysvětlen a popsán v rámci použitých algoritmů. Zároveň není blíže popsán stav kolineárních bodů při metodě Jarvis Scan. Tento stav je specifikován v rámci kapitoly zabývající se algoritmem Jarvis Scan v podkapitole Problematické situace.

4.1 Automatické generování množin bodů

Pro tvorbu automatického generování množin bodů je možné zvolit útvar, v němž budou generované body vykresleny. V rámci aplikace si může uživatel vybrat mezi kruhem, čtvercem, elipsou, gridem, náhodném rozmístění či star shaped útvarem. Veškeré body jsou odsazeny, aby elipsy, které bod reprezentují, nebyly nějakým způsobem oříznuté.

4.1.1 Circle

Pro tvorbu kruhu byl nejprve náhodně vygenerován střed kružnice a její poloměr. V závislosti na počtu bodů byl určen středový úhel po sobě jdoucích bodů. Po znalosti těchto hodnot bylo snadné vygenerovat body na kružnici:
$X = X_0 + r * cos(\phi)$
$Y = Y_0 + r * sin(\phi)$

4.1.2 Ellipse

Elipsa byla generována obdobně jako kruh s tím rozdílem, že nebyl generován poloměr, ale hodnota hlavní poloosy (a) a vedlejší poloosy (b).
$X = X_0 + a * cos(\phi)$
$Y = Y_0 + b * sin(\phi)$

4.1.3 Square

Při tvorbě čtverce je nejprve vygenerováno náhodné umístění levého horního rohu. Poté je náhodně vypočtena délka hrany. Se znalostí délky hrany jsme schopni určit zbylé vrcholy čtverce a zároveň zobrazit body na hranách. Při generování čtverce je vstupní množství zredukováno tak, aby počet bodů odpovídal počtu nutnému pro zobrazení čtverce. Pro představu - z pěti bodů čtverec nesestavíte, tudíž pokud to je nutné, tak dochází k přepočítání počtu bodů. Na tento fakt je uživatel aplikací upozorněn.

4.1.4 Star Shaped

Při tvorbě Star Shaped polygonu je automaticky vygenerován střed a poloosy a, b. Pro polygony typu star shaped je typické, že z jeho středu lze vidět do všech vrcholů. Dva po

sobě jdoucí body budou vytvořeny s různou vzdáleností od středu, čímž této podmínky docílíme. Středový úhel mezi dvěma následujícími body bude opět určen v závislosti na množství vstupních dat.
$X_i = X_0 + a * cos(\phi)$
$Y_i = Y_0 + a * sin(\phi)$
$X_{i+1} = X_0 + b * cos(\phi)$
$Y_{i+1} = Y_0 + b * sin(\phi)$

4.1.5 Grid

Pro tvorbu pravidelné mřížky je zapotřebí, aby vstupní počet bodů byl druhou mocninou nějakého přirozeného čísla. Pokud tato podmínka není splněna, je počet redukován a uživatel je o změně informován. Takovýto případ může nastat například při zadání 18 bodů. Počet je v tomto případě zredukován na 16 a vytvořený grid má velikost 4 x 4. Podobně jako při tvorbě čtverce je vygenerován první bod, od nějž jsou následně zbylé body vypočteny.

4.1.6 Random

Body byly náhodně rozmístěny v rámci zobrazovacího okna.

4.2 Konstrukce striktně konvexních obálek

Striktně konvexní obálky jsou takové, že po sobě následující 3 body neleží na stejné přímce. V případě vygenerovaného čtverce o sto bodech bude striktně konvexní obálka obsahovat pouze 4 body, a to vrcholy. Pro tvorbu striktně konvexních obálek byla po vygenerování klasických obálek volána funkce na určení pozice bodu vůči linii. Pokud byl následující bod vyhodnocen tak, že leží na orientované úsečce $\overrightarrow{AB}$, bod B byl z množiny odebrán a následně byla testována další trojice bodů. Zároveň bylo nutné odebrat duplicitní body.

4.3 Konstrukce Minimum Area Enclosing Box

V rámci bonusových úloh bylo i sestrojení obdélníku s minimální plochou s využitím konvexní obálky. Tvorba takovýchto obdélníků se využívá v kartografii pro detekci tvaru a natočení budov. Případně by se mohla metoda používat v rámci generalizace, zde by však bylo zapotřebí ošetřit, aby výsledná data byla topologicky čistá. Při tvorbě minimálního obdélníku s využitím konvexní obálky se vychází z předpokladu, že nejméně jedna strana obdélníka je kolineární se stranou konvexní obálky. Složitost takovéto konstrukce má časovou náročnost O(n).

4.3.1 Implementace metody

1. Inicializuj minimální plochu: $A_{min} = \infty$
2. Inicializace místního souřadnicového systému a výpočet úhlu následujících bodů: $p = [0, 0]; k = [0, \overrightarrow{PK}]$
3. Pro všechny body převod do místního systému: $points_i = p; points_{i+1} = k$

4. Nalezení Minimum Bounding Rectangle, který se dotýká konvexní obálky v extrémních bodech (min X, max X, min Y, max Y)

5. Porovnání s minimální plochou. $A < A_{min}$

   Nahraď:$A_{min}; Save(minX, maxX, minY, maxY); SaveAngle$

6. Vytvoř obdélník z (min X, max X, min Y, max Y).

7. Nalezení největší osy obdélníku v místní soustavě.

8. Transformace do globálního systému.

5 Vstupní data

Vstupními daty je množina bodů, která je vygenerovaná uživatelem. Uživatel může v tomto případě generovat body sám za pomoci kurzoru. Případně je v aplikaci implementována funkce pro automatické generování bodů. Uživatel má navíc možnost vlastní volby, jakým způsobem budou body generovány, zda budou body v kruhu, ve čtverci, elipse či v gridu.

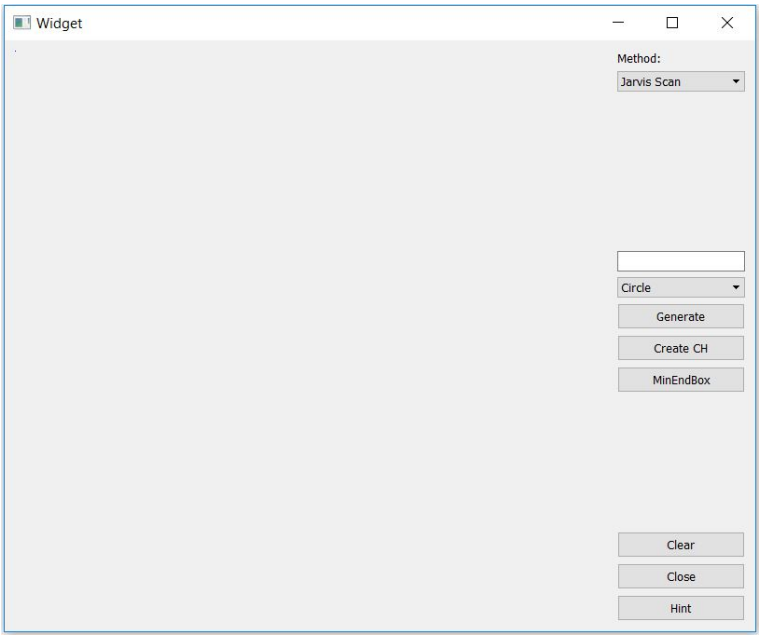
Při automatickém generování bodů je nutné nastavit i počet bodů, ze kterých bude daný útvar tvořen. Ne všechny tvary však lze vytvořit z vloženého množství bodů. V případě, že vstup nebude zcela odpovídat zvolenému útvaru, zobrazí se uživateli vyskakovací okno, které upozorní na nevhodný vstup a případnou modifikaci vloženého počtu. Tento případ nastane například při vložení tří bodů, ze kterých by uživatel chtěl vytvořit čtverec.

6 Výstupní data

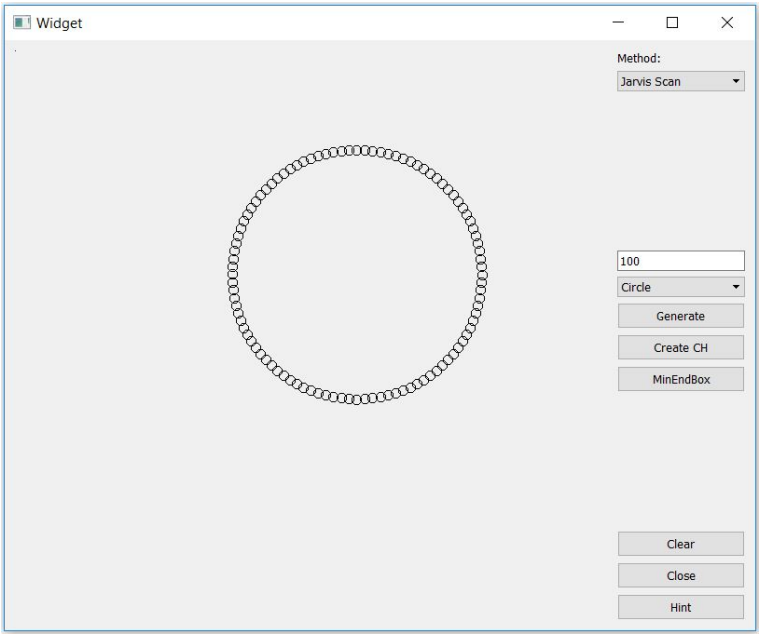
Výstupem této úlohy je grafická aplikace, ve které jsou vstupní data analyzována a následně je v závislosti na jejich vzájemně poloze vytvořena konvexní obálka. Konvexní obálku je možné vytvořit čtyřmi způsoby za pomoci nejznámějších algoritmů - Jarvis Scan, Quick Hull, Sweep Line a Graham Scan.

Zároveň jsou výstupem grafy a tabulky s porovnáním jednotlivých dob výpočtů na různém množství dat.

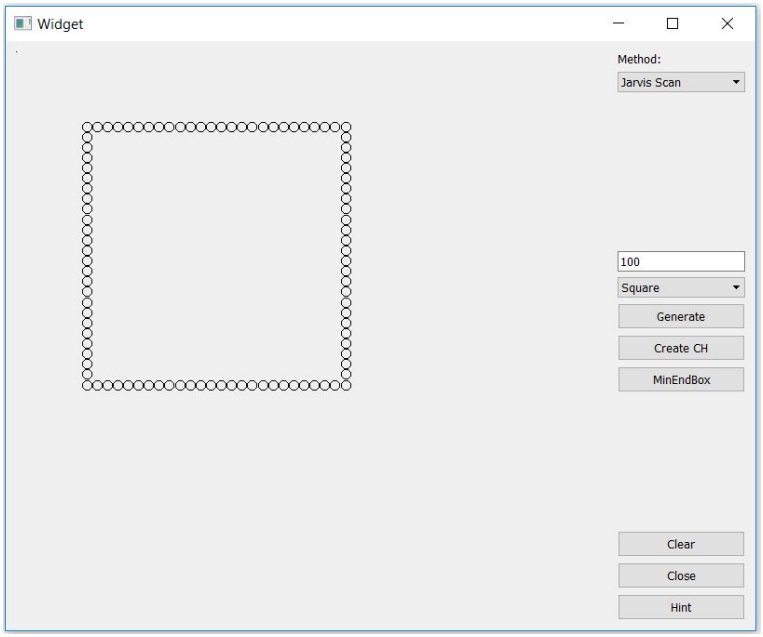
7 Aplikace



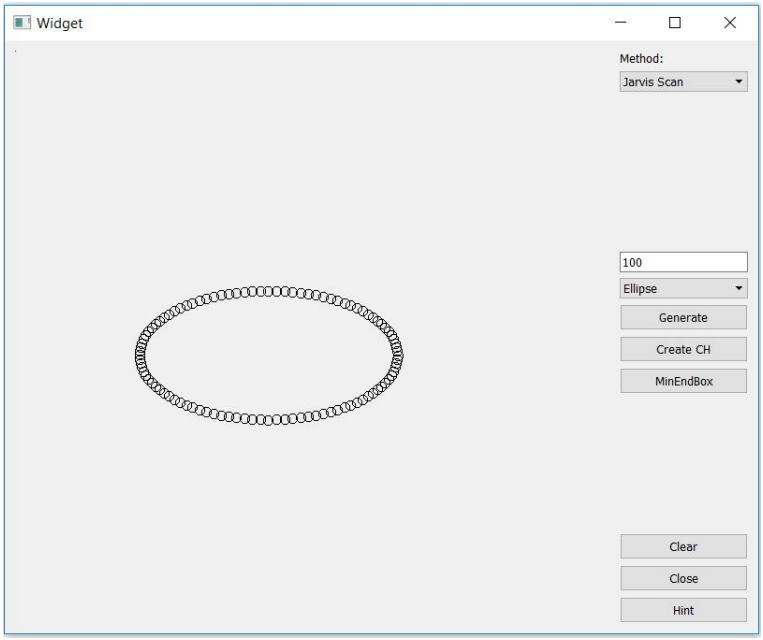
Obrázek 7: Zobrazené okno po spuštění aplikace



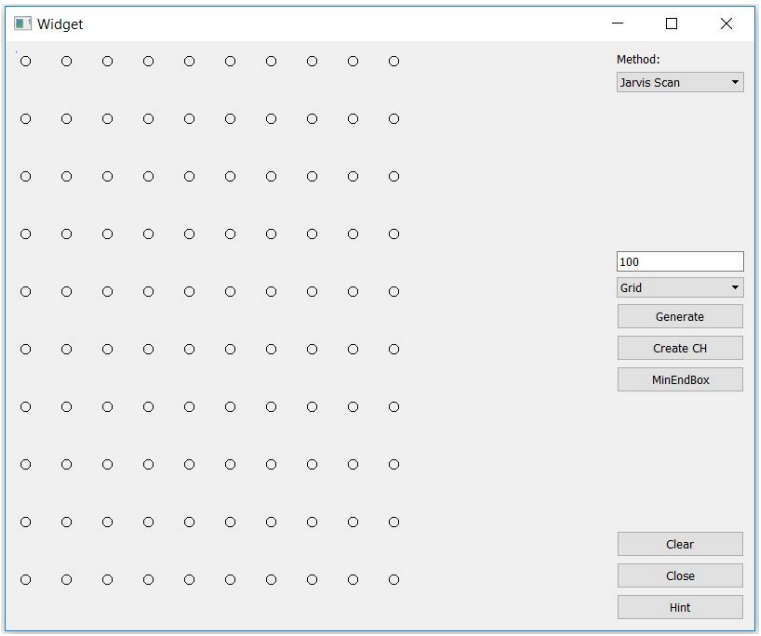
Obrázek 8: Vygenerované body v kruhu



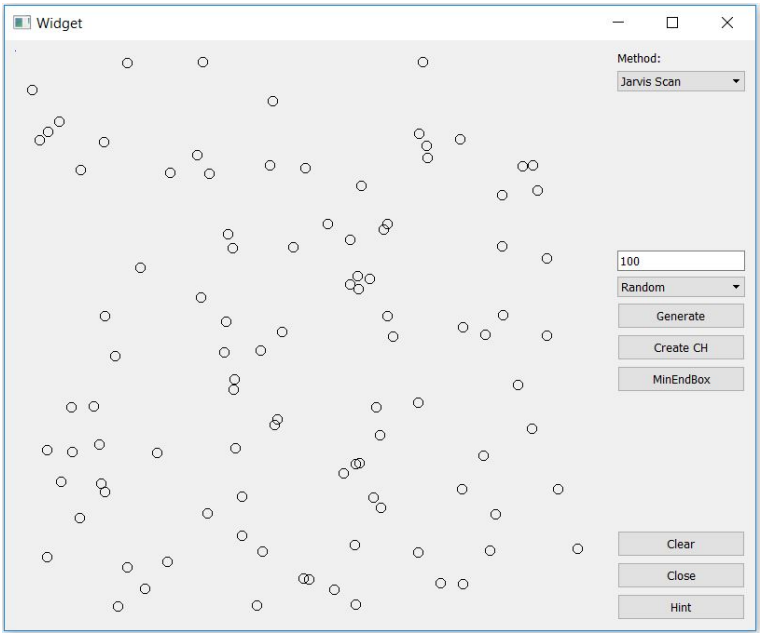
Obrázek 9: Vygenerované body ve čtverci



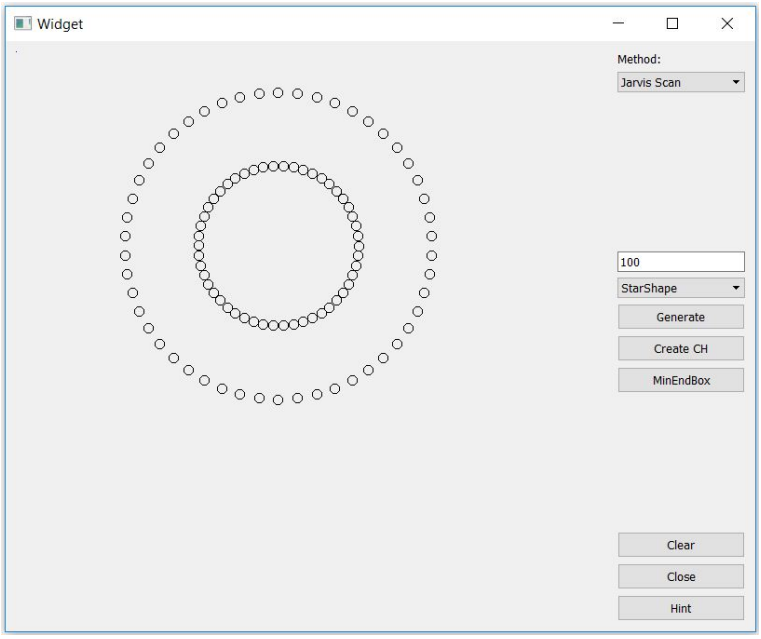
Obrázek 10: Vygenerované body v elipse



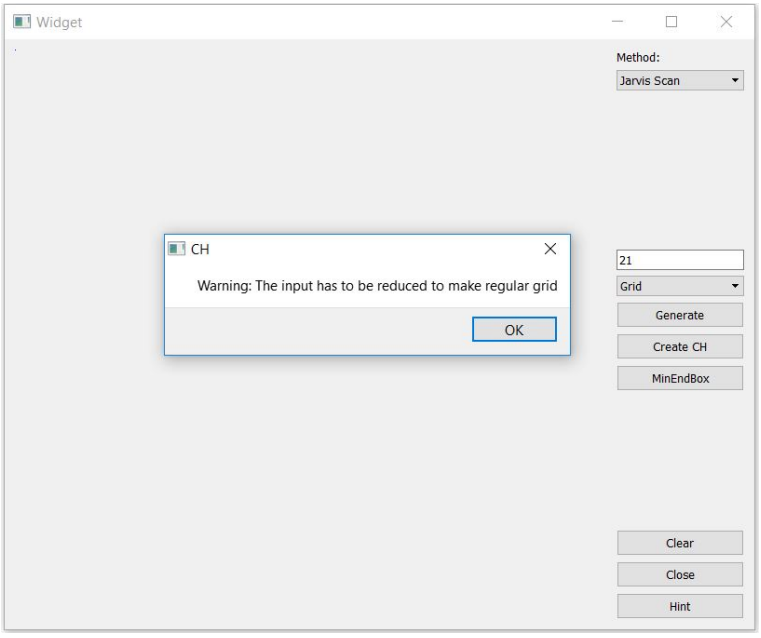
Obrázek 11: Vygenerované body v pravidelné mřížce



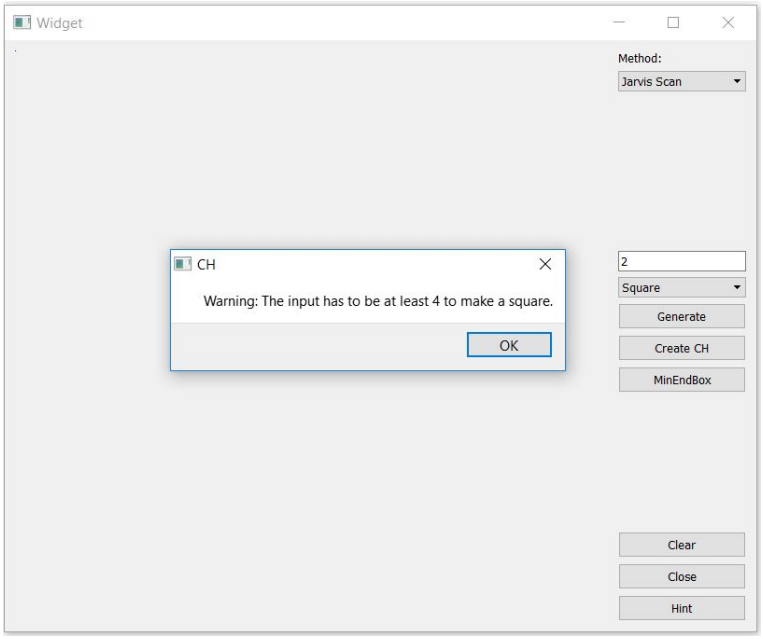
Obrázek 12: Náhodně vygenerované body



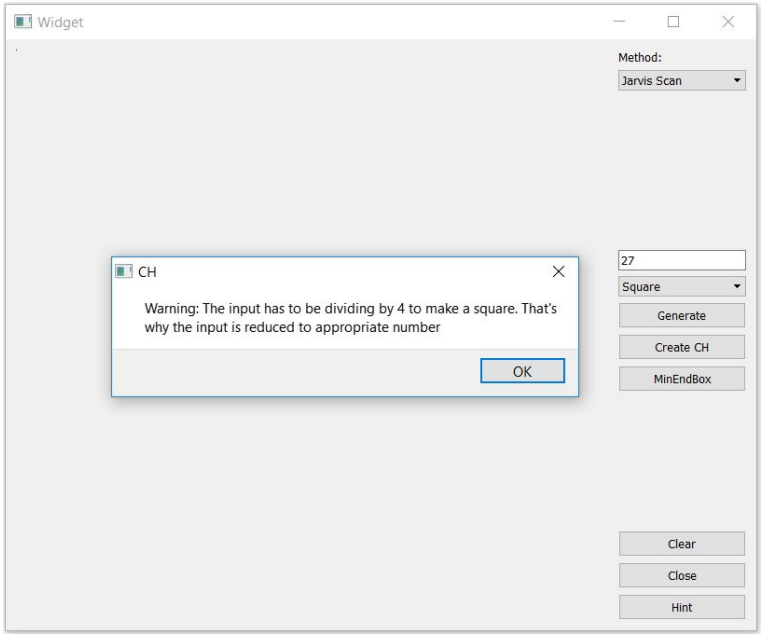
Obrázek 13: Vygenerované body ve tvaru Star Shaped



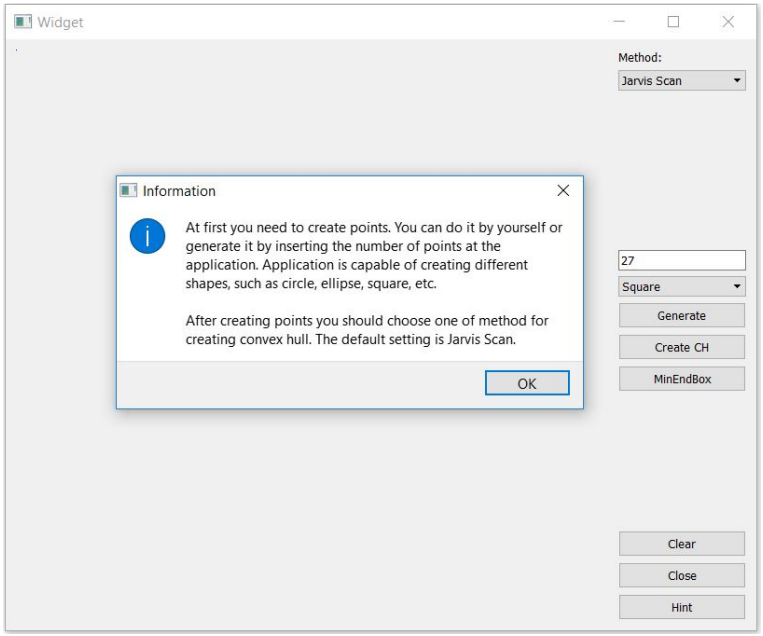
Obrázek 14: Varování na redukci vloženého počtu bodů při tvorbě gridu



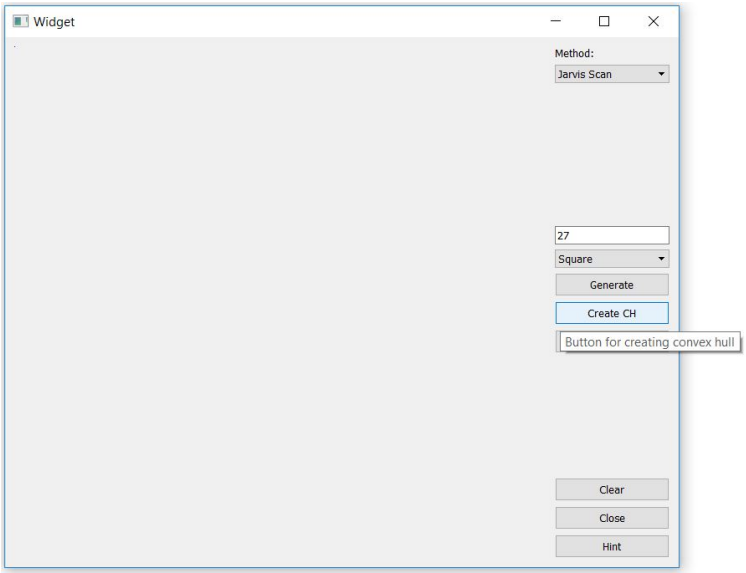
Obrázek 15: Varování při neplatném vstupu při tvorbě čtverce



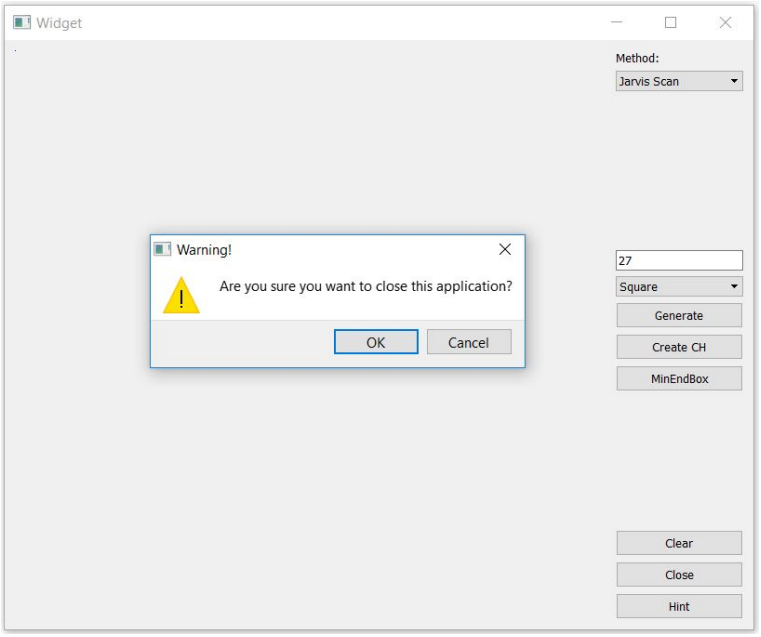
Obrázek 16: Varování na redukci vloženého počtu bodů při tvorbě čtverce



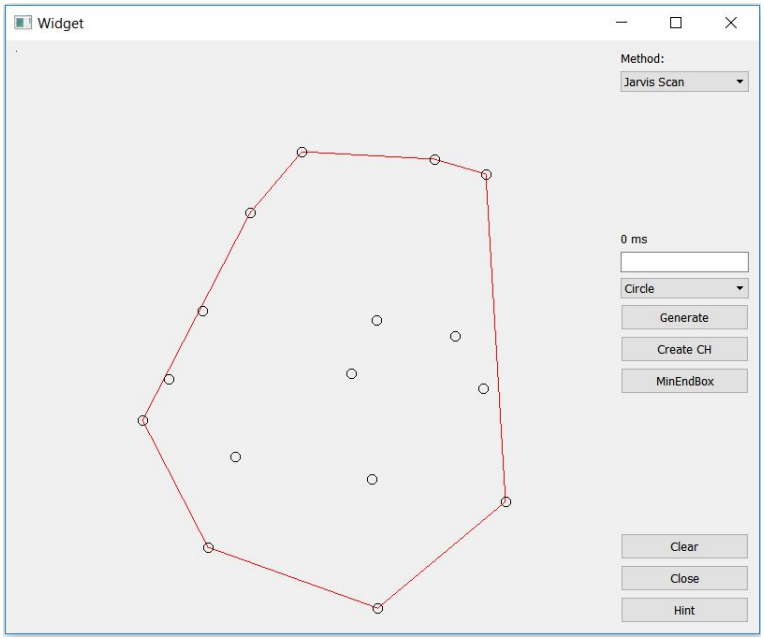
Obrázek 17: Nápověda, pokud si uživatel nebude jistý, jak v aplikaci postupovat



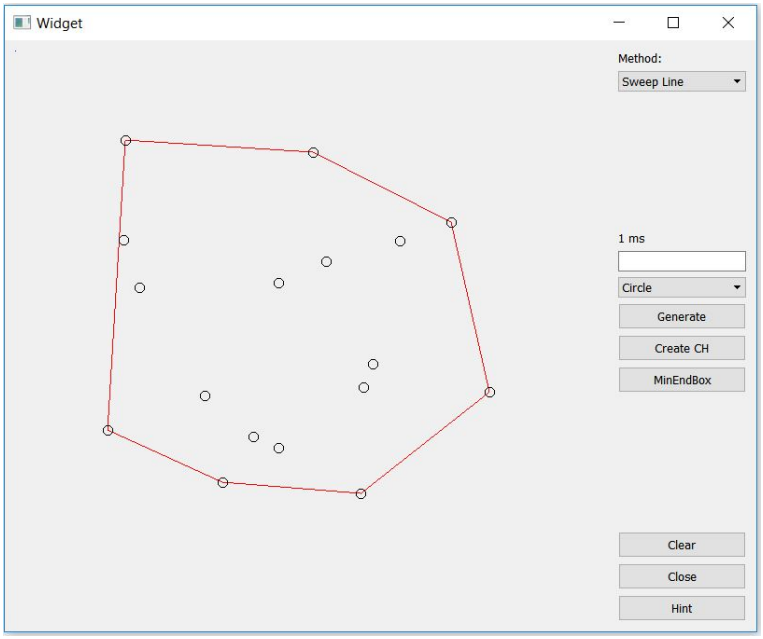
Obrázek 18: Po přiložení kurzoru se zobrazí informace o objektu



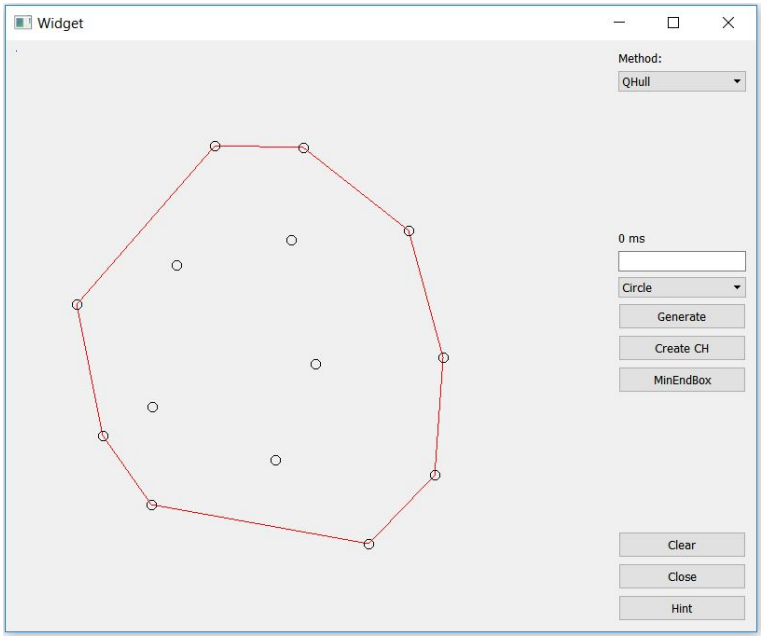
Obrázek 19: Vyskakovací okno při ukončování aplikace



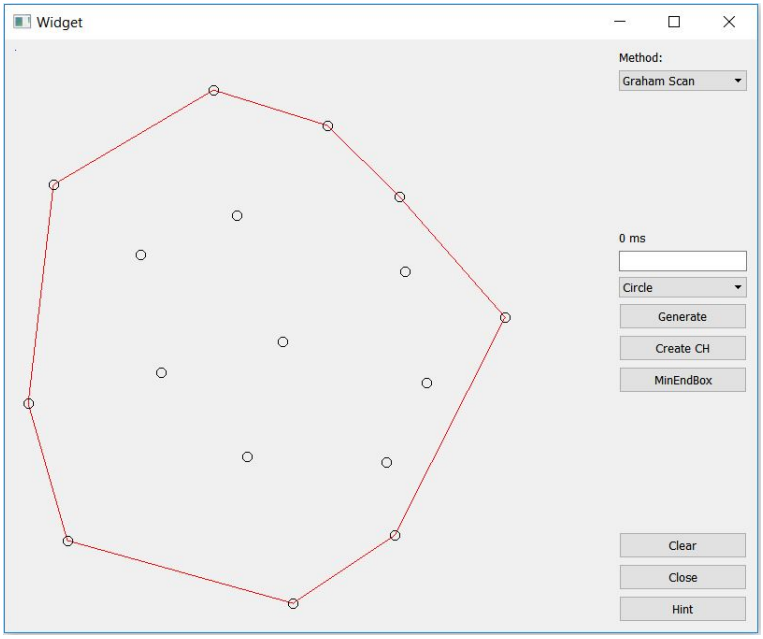
Obrázek 20: Tvorba konvexní obálky metodou Jarvis Scan



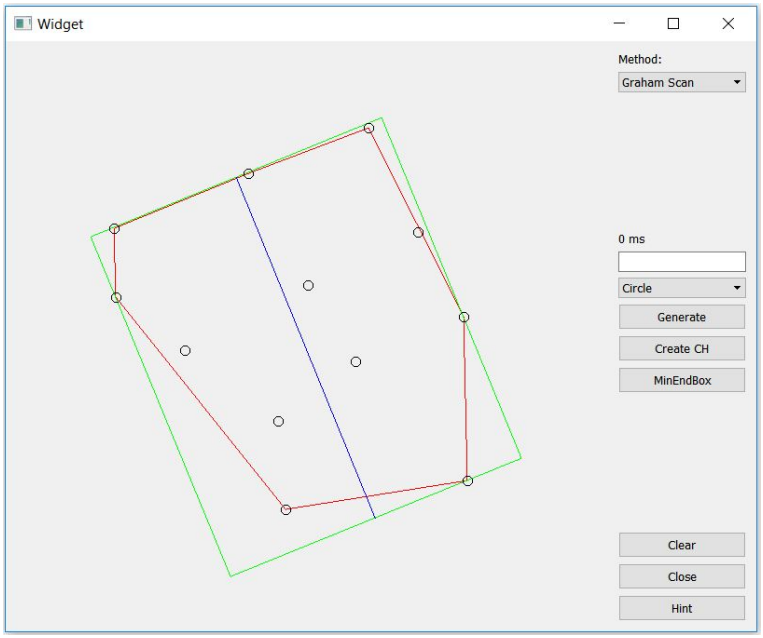
Obrázek 21: Tvorba konvexní obálky metodou Sweep Line



Obrázek 22: Tvorba konvexní obálky metodou Quick Hull



Obrázek 23: Tvorba konvexní obálky metodou Graham Scan



Obrázek 24: Tvorba minimálního ohraničujícího obdélníku

8 Dokumentace

8.1 Třídy

8.1.1 Algorithms

Třída Algorithms obsahuje několik metod. Metody jsou určeny pro výpočty použitých algoritmů.

double length2Points(QPoint q, QPoint p)
Metoda, jejíž návratová hodnota je typu double, vrací velikost spojnice mezi dvěma body.

void polygonTransform(QPoint p, QPoint k, QPoint p1, QPoint k1, QPolygon &pol)
Metoda je přetížená pro podobnostní transformaci QPolygon a QLine. Transformační klíč je dán dvěma body se souřadnicemi v obou souřadných soustavách. Body p a k jsou v souřadné soustavě, ze které pochází pol. Body p1 a k1 jsou v souřadné soustavě, do které chceme transformovat. Výsledek transformace se přeuloží do proměnné pol.

rotateByAngle(QPolygon &points, double angle)
Metoda je přetížená pro rotaci QPolygon a QLine. Prvky proměnné pol se pouze pootočí podle úhlu angle. Proměnná úhel je v radiánech.

TPosition getPointLinePosition(QPoint &q, QPoint &a, QPoint &b)
Tato metoda slouží k určení pozice bodu q vůči linii tvořené body a, b. Výstupem metody je LEFT, RIGHT nebo ON.

double get2LinesAngle(QPoint &p1,QPoint &p2,QPoint &p3, QPoint &p4)
Pomocí této funkce je vypočítán úhel mezi dvěma přímkami, kde první přímka je tvořena body p_1 a p_2, druhá přímka je tvořena body p_3 a p_4.

QPolygon CHJarvis (vector<QPoint >&points)
Metoda pro výpočet konvexní obálky nad vektorem bodů metodou Jarvis Scan. Metoda vrácí konvexní obálku s typem QPolygon.

QPolygon QHull (vector<QPoint >&points)
Metoda pro výpočet konvexní obálky nad vektorem bodů metodou Quick Hull. Metoda vrácí konvexní obálku s typem QPolygon.

void qh (int s, int e, vector<QPoint >&p, QPolygon &h)
Pomocná metoda pro rekurzi v metodě Qhull. Na vstupu jsou indexy bodů s a e, které určují přímku, podle níž se určí, zda bod p patří do konvexní obálky H. Metoda nic nevrací, pouze ukládá body, které patří do konvexní obálky.

QPolygon GrahamScan (vector<QPoint >&points)
Metoda pro výpočet konvexní obálky nad vektorem bodů metodou Graham Scan. Metoda vrácí konvexní obálku s typem QPolygon.

QPolygon CHSweep (vector<QPoint >&points)
Metoda pro výpočet konvexní obálky nad vektorem bodů metodou Sweep Line. Metoda vrácí konvexní obálku s typem QPolygon.

void minimumAreaEnclosingBox (QPolygon &ch, QPolygon &rectangle, QLine &direction)
Metoda pro výpočet hlavních směrů budovy. Na vstupu je konvexní obálka ch a prázdné proměnné rectangle a direction. Do rectangle se uloží minimální ohraničující obdélník a do direction hlavní směr budovy.

QPolygon GrahamScanNew (vector<QPoint >&points)
Opravený algoritmus Graham Scan, který funguje i na větší množství bodů.

QPolygon deleteDuplicityCH (QPolygon ch)
Metoda pro odstranění duplicitních bodů.

QPolygon exactlyCH (QPolygon ch)
Metoda pro výpočet striktně konvexní obálky.

8.1.2 Draw

Třída Draw obsahuje několik metod. Metody jsou určeny pro generování a vykreslování proměných.

void paintEvent(QPaintEvent *e)
Metoda slouží k vykreslení vytvořených, generovaných bodů a zobrazení výsledků použitých algoritmů.

void mousePressEvent(QMouseEvent *e)
Metoda uloží bod se souřadnicemi místa kliknutí v zobrazovacím okně.

void clearCanvas()
Metoda slouží k vymazání proměnných a k překreslení

std::vector<QPoint >generateGrid(int n)
Metoda generuje pravidelnou mřížku. Na vstupu je počet generovaných bodů. Návratová hodnota je vektor bodů.

std::vector<QPoint >generateRandomPoints(int n)
Metoda generuje náhodné body. Na vstupu je počet generovaných bodů. Návratová hodnota je vektor bodů.

std::vector<QPoint >generateStarShape(int n)
Metoda generuje body do tvaru hvězdy. Na vstupu je počet generovaných bodů. Návratová hodnota je vektor bodů.

std::vector<QPoint >generateSquare(int n)
Metoda generuje body do tvaru čtverce. Na vstupu je počet generovaných bodů, změna v počtu bodů se projení s násobkem čtyř. Návratová hodnota je vektor bodů.

std::vector<QPoint >generateEclipse(int n)
Metoda generuje body do tvaru elipsy. Na vstupu je počet generovaných bodů. Návratová hodnota je vektor bodů.

std::vector<QPoint >generateCircle(int n)
Metoda generuje body do tvaru kruhu. Na vstupu je počet generovaných bodů. Návratová hodnota je vektor bodů.

void setCH(QPolygon ch_)
Metoda slouží pro převod konvexní obálky do vykreslovacího okna.

setRectangle(QPolygon rectangle_)
Metoda slouží pro převod minimální ohraničujícího obdélníku do vykreslovacího okna.

setDirection(QLine direction_)
Metoda slouží pro převod hlavního směru budovy do vykreslovacího okna.

setPoints(std::vector<QPoint >points_)
Metoda slouží pro převod vektoru bodů do vykreslovacího okna.

std::vector<QPoint >getPoints()
Metoda slouží pro převod vektoru bodů z vykreslovacího okna.

QPolygon getConvexHull()
Metoda slouží pro převod konvexní obálky z vykreslovacího okna.

8.1.3 SortByXAsc

Třída SortByXAsc slouží k porovnání souřadnic v ose x.

bool operator()(QPoint &p1, QPoint &p2)
Přetížený operátor () vrátí bod s větší souřadnicí x z dvojice bodů.

8.1.4 sortByYAsc

Třída sortByYAsc slouží k porovnání souřadnic v ose y.

bool operator()(QPoint &p1, QPoint &p2)
Přetížený operátor () vrátí bod s větší souřadnicí y z dvojice bodů.

8.1.5 Widget

void on_pushButton_clicked()
Při stisknutí tlačítka Create CH se volají metody třídy Algorithms dle metody vybrané v comboboxu.

void on_generate_clicked()
Při stisknutí tlačítka Generate se volají metody třídy Draw dle metody vybrané v comboboxu.

void on_pushButton_2_clicked()
Při stisknutí tlačítka Clear se zavolá metoda třídy Draw clearCanvas.

void on_minimumAreaEnclosingBox_clicked()
Při stisknutí tlačítka MinEndBox se nad konvexní obálkou zavolá metoda třídy Algorithms minimumAreaEnclosingBox.

void on_pushButton_3_clicked()
Při stisknutí tlačítka Close je uživatel vyzván, zda chce skutečně aplikaci ukončit.

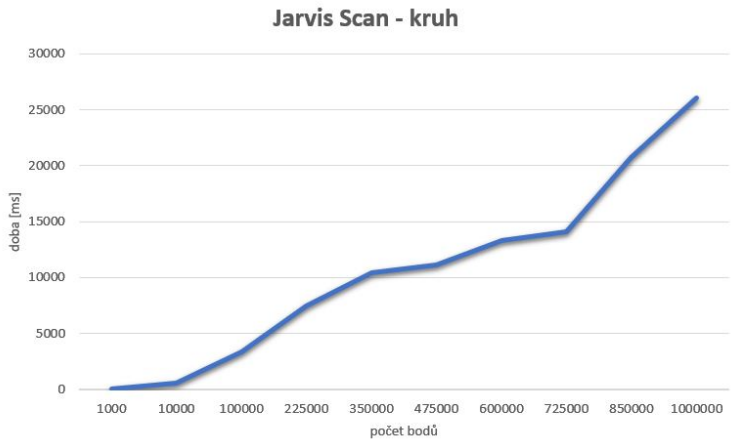
void on_pushButton_4_clicked()
Při stisknutí tlačítka Hint je uživateli zobrazena nápověda, jakým způsobem postupovat, neboť při stisknutí tlačítka pro tvorbu konvexní obálky nad prázdnou množinou je aplikace ukončena.

9 Testování

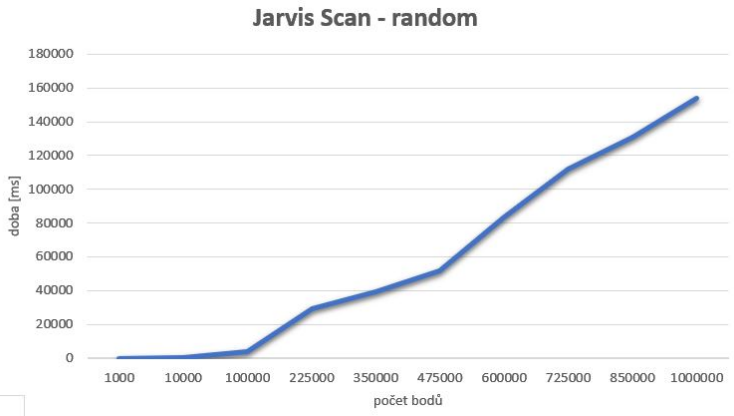
Po vytvoření funkční aplikace byly jednotlivé algoritmy otestovány na různém množství dat a na různých typech množin. Testování bylo prováděno nad výpočtem striktně konvexních obálek, které jsou výstupem aplikace. Samotné testování bylo prováděno v režimu Release. Testované typy množin byly voleny kruh, grid a náhodné rozmístění. Jednotlivé doby byly uloženy do tabulek a následně zobrazeny v grafu.

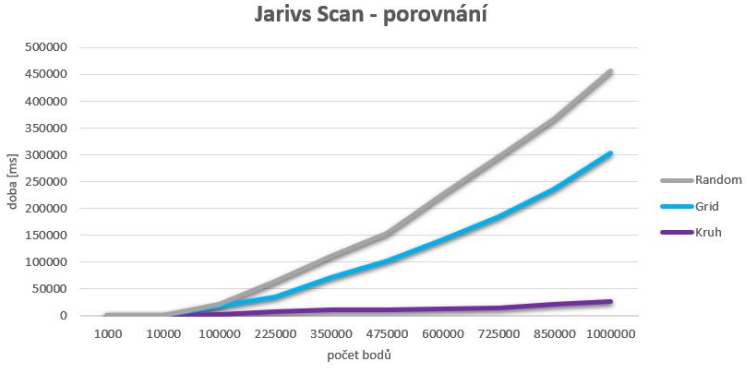
9.1 Jarvis Scan

Tato metoda dosahovala nejhorších výsledků. Pro milion bodů byla doba výpočtu kolem 5 minut. Přestože by tato metoda měla pro body na kružnici dosahovat nejhorších výsledků, překvapivě je doba pro kružnici nejlepší. Nejhorší doby dosahuje pro náhodné rozmístění bodů.


Jarvis Scan - kruh
30000
25000
20000
15000
10000
5000
0
doba [ms]
1000
10000
100000
225000
350000
475000
600000
725000
850000
1000000
počet bodů

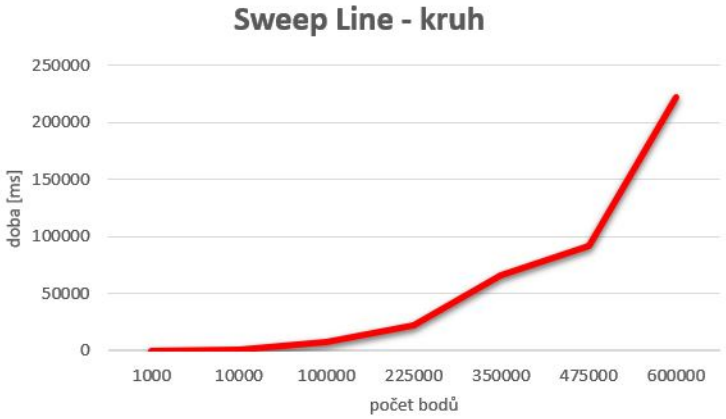

Jarvis Scan - grid
300000
250000
200000
150000
100000
50000
0
doba [ms]
1000
10000
100000
225000
350000
475000
600000
725000
850000
1000000
počet bodů

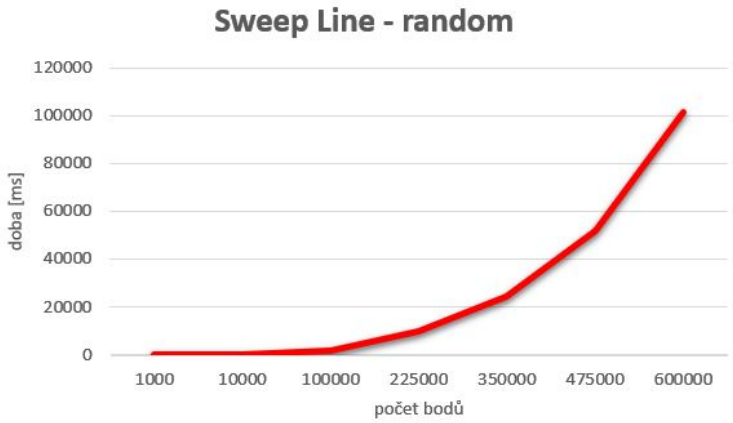

Jarvis Scan - random
180000
160000
140000
120000
100000
80000
60000
40000
20000
0
doba [ms]
1000
10000
100000
225000
350000
475000
600000
725000
850000
1000000
počet bodů


Jarivs Scan - porovnání
500000
450000
400000
350000
300000
250000
200000
150000
100000
50000
0
doba [ms]
1000
10000
100000
225000
350000
475000
600000
725000
850000
1000000
počet bodů
Random
Grid
Kruh

9.2 Sweep Line

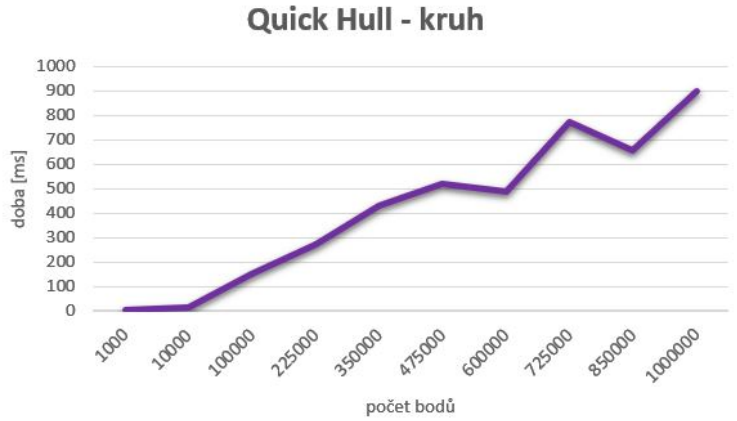
Z dosavadních znalostí by tato metoda měla dosahovat velmi dobrých výsledků. Přesto v naší aplikaci dochází k chybě, při testování množiny bodů o 725.000 bodech začne aplikace pro náhodné rozmístění a kruh padat a je nefunkční. Na chybu se bohužel nepodařilo přijít.

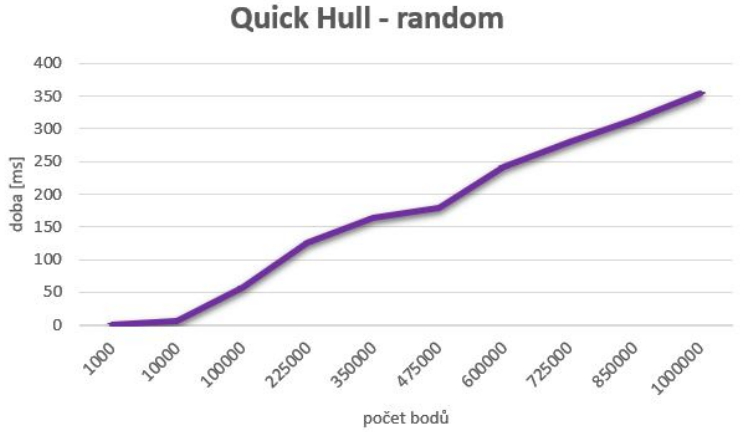


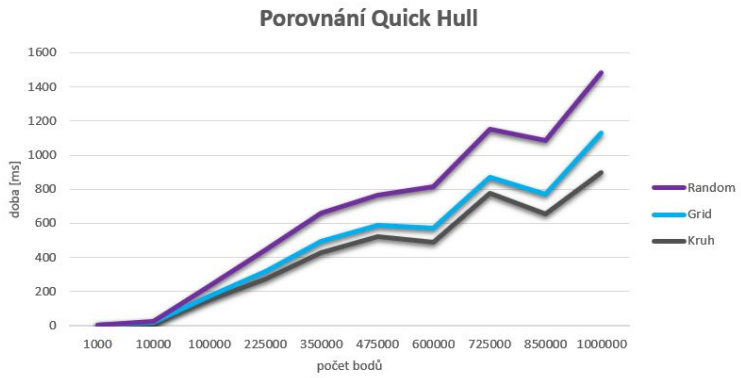


9.3 Quick Hull

Metoda Quick Hull dosahovala poměrně přiznivých výsledků. Velmi zajímavý byl graf porovnání výpočtu pro testované množiny, kde lze vidět, že funkce mají zhruba stejný průběh. Opět vychází nejlepší časy pro kruh a nejhorší pro náhodné rozmístění bodů.

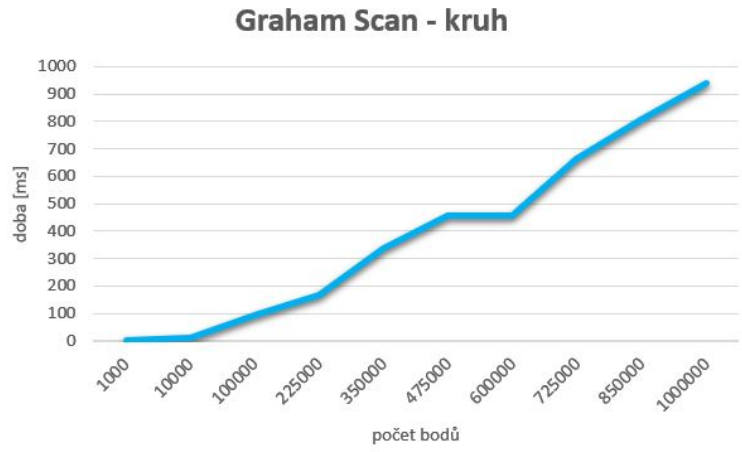


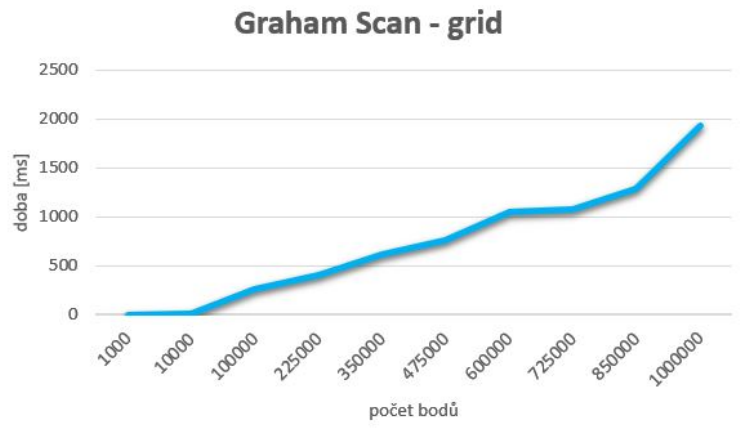


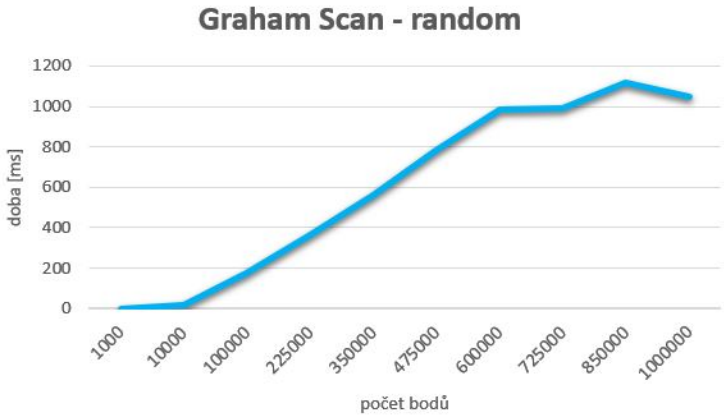


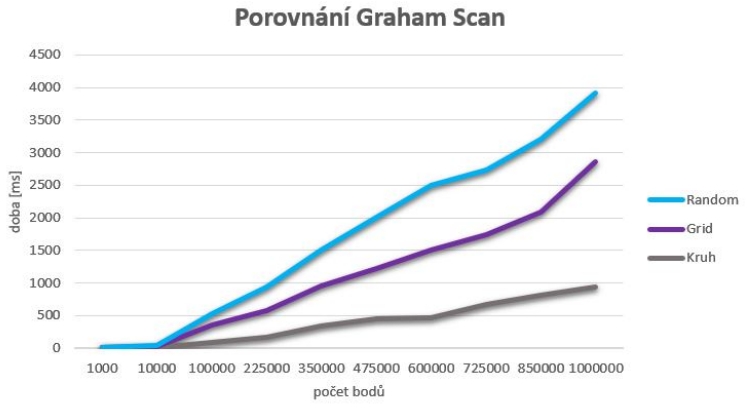
9.4 Graham Scan

Tento algoritmus dosahuje nejlepších výsledků. Metoda je rychlá a funkční. Pro kruh opět dosahuje nejlepších výsledků a nejhorších výsledků pro náhodné rozmístění bodů.






Graham Scan - random
1200
1000
800
600
400
200
0
doba [ms]
1000
10000
100000
225000
350000
475000
600000
725000
850000
1000000
počet bodů


Porovnání Graham Scan
4500
4000
3500
3000
2500
2000
1500
1000
500
0
doba [ms]
1000
10000
100000
225000
350000
475000
600000
725000
850000
1000000
počet bodů
Random
Grid
Kruh

9.5 Tabulky výsledných průměrných dob pro jednotlivé algoritmy

Níže lze vidět tabulky s průměrnýmu naměřenými dobami jednotlivých algoritmů. Čas je uváděn v msec. První řádek vždy označuje počet bodů, druhý řádek množinu generovnou v kruhu, třetí řádek množinu v gridu a poslední řádek náhodné rozmístění bodů.

1000	**10000**	**100000**	**225000**	**350000**	**475000**	**600000**	**725000**	**850000**	**1000000**
76.2	603	3323	7460	10409	11155	13280	14127	20775	26073
32	893	14664	27210	60533	91470	128758	170449	216169	276887
11	191	3562	29533	39542	51781	83605	112170	130838	153814

Tabulka 1: Jarvis Scan

1000	**10000**	**100000**	**225000**	**350000**	**475000**	**600000**	**725000**	**850000**	**1000000**
1.8	15	151	273	430	520	487	774	657	898
1	3	23	43	65	68	86	96	112	231
1	7	58	126	164	180	242	280	314	355

Tabulka 2: Quick Hull

1000	**10000**	**100000**	**225000**	**350000**	**475000**	**600000**	**725000**	**850000**	**1000000**
1	115	7581	22199	65410	91985	222160	-	-	-
1	4	23	52	67	83	87	110	108	230
2	35	1475	9938	24547	52011	101730	-	-	-

Tabulka 3: Sweep Line

1000	**10000**	**100000**	**225000**	**350000**	**475000**	**600000**	**725000**	**850000**	**1000000**
1.6	11	91	169	339	457	458	662	805	938
1	16	256	406	611	763	1052	1081	1289	1926
1	17	174	364	562	782	984	993	1117	1048

Tabulka 4: Graham Scan

10 Závěr

V rámci úlohy byla vytvořena aplikace, která je schopna na vygenerovaných bodech zkonstruovat konvexní obálku. Zároveň je v rámci aplikace možné náhodně vygenerovat téměř libovolné množství bodů do určitého tvaru. Na výběr je poměrně široké spektrum běžných tvarů, což jistě uživatel ocení.

Součástí úlohy bylo i testování doby výpočtu jednotlivých konvexních obálek. Testování bylo prováděno nad striktně konvexními obálkami, samotný výpočet je tedy zatížen ještě o vyhodnocení, zda následující tři body neleží na stejné přímce. Výsledné doby dopadly dle očekávání. Algoritmus Jarvis Scan měl znatelně nejhorší čas. Naopak metoda Graham Scan se jeví jako nejlepší řešení.

11 Náměty na vylepšení

11.1 Graham Scan

Pro algoritmus Graham Scan byl nejprve napsán kod, který byl velmi náročný a aplikace nebyla funkční na větším množství dat. Z toho důvodu byla napsána nová metoda, která je již funkční a při testování dosahovala nejlepších možných výsledků. Z toho důvodu je v kódu pojmenována jako GrahamScanNew. Původní funkce byla v kódu ponechána pro případnou doeditaci a její zprovoznění.

11.2 Generování množin bodů

Při tvorbě generování gridu napadla autory myšlenka, že by se body generovaly v některém ze známých kartografických zobrazení. Z bakalářského studia znají velké množství zobrazovacích rovnic, pomocí nichž by se body zobrazovaly. Zajímavé rozmístění by měla zejména kuželová zobrazení. Při zobrazení bodů v rámci některého kartografického zobrazení by výsledná konvexní obálka představovala celý svět, případně polokouli, záleželo by na typu zvoleného zobrazení.

11.3 Testování algoritmů

V rámci této úlohy bylo provedeno několik testování nad odlišným množstvím bodů a na jejich odlišném zobrazení. Časově bylo testování poměrně náročné, za úvahu jistě stojí popřemýšlení, zda by nebylo výhodnější vytvořit takovou funkci, která by pro předem definované tvary a množství bodů sama provedla testování. Zároveň obsahuje platforma Qt i knihovnu grafů, časová úspora by tedy byla velmi výrazná.

11.4 Tvorba tabulek v LaTeXu

V rámci dokumentace bylo i prezentování výsledných měření prostřednictvým tabulek. V původních tabulkách byl ještě sloupec, který označoval typy množin (kruh, grid, random). Bohužel s tímto sloupcem byla tabulka oříznutá a přes veškerou snahu se autorům nepodařilo nalézt řešení, jakým způsobem tabulku posunout vlevo. Z toho důvodu byl

tento sloupec odstraněn a tabulky byly popsány v záhlaví podkapitoly. S tímto řešením nejsou autoři zcela spokojeni, tabulky nejsou dostatečně přehledné.

11.5 Minimum Area Enclosing Box

Při výpočtu minimálního ohraničujícího obdélníku autorům z neznámé příčiny nefungovala transformace, ač se to snažili vyřešit. Po konzultaci byla část pro transformaci vypůjčena od kolegy, kterému fungovala bez problémů. Vypůjčená část je v kódu vyznačena.

12 Reference

1. MARTÍNEK, Petr. Konvexní obálka rozsáhlé množiny bodů v Eˆd [online][cit. 31.10.2018]. Dostupné z: http://graphics.zcu.cz/files/86_BP_2010_Martinek_Petr.pdf

2. Convex Hulls: Explained. [online][cit. 31.10.2018] Dostupné z: https://medium.com/@harshitsikchi/convex-hulls-explained-baab662c4e94

3. Convex-hull mass estimates of the dodo (Raphus cucullatus). [online][cit. 31.10.2018] Dostupné z: https://www.semanticscholar.org/paper/Convex-hull-mass-estimates-of-the-dodo-(Raphus-of-a-Brassey-OMahoney/12e07d3b712561cad16501ac8096120e14901eb8

4. GeeksforGeeks: Convex Hull - Set 1 (Jarvis's Algorithm or Wrapping. [online][cit. 5.11.2018] Dostupné z: https://www.geeksforgeeks.org/convex-hull-set-1-jarviss-algorithm-or-wrapping/

5. BAYER, Tomáš. Geometrické vyhledávání [online][cit. 5.11.2018]. Dostupné z: https://web.natur.cuni.cz/ bayertom/images/courses/Adk/adk4.pdf

6. GeeksforGeeks: Convex Hull - Set 2 (Graham Scan). [online][cit. 12.11.2018] Dostupné z: https://www.geeksforgeeks.org/convex-hull-set-2-graham-scan/