ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE FAKULTA STAVEBNÍ, OBOR GEODÉZIE A KARTOGRAFIE KATEDRA GEOMATIKY					
název předmětu ALGORITMY V DIGITÁLNÍ KARTOGRAFII					
číslo úlohy 2.1	název úlohy Konvexní obálky a jejich konstrukce.1 (první verze opravy)				
školní rok 2020/2021	studijní skup. 60	číslo zadání U2	Zpracoval Usik Svetlana,Vaňková Zuzana	datum 4. 12. 2020	klasifikace

Obsah

1 Zadání

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.

2 Popis a rozbor problému

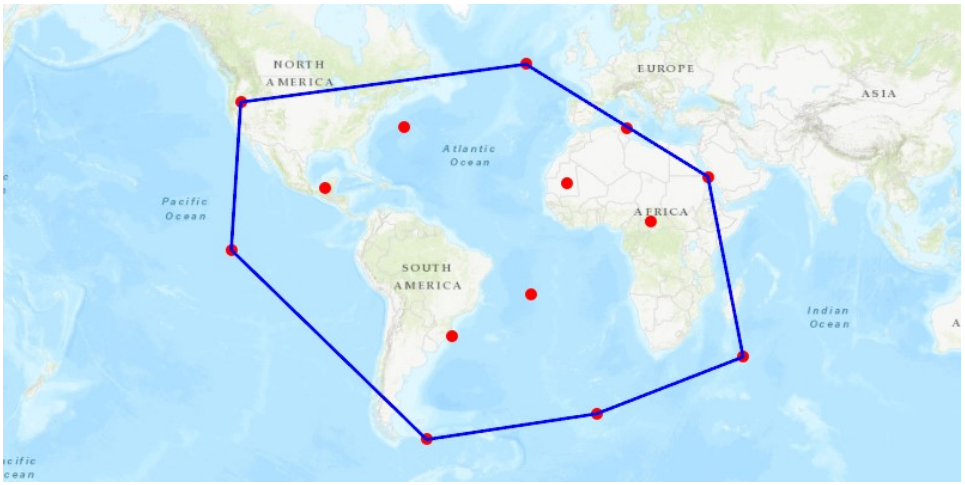
Cílem úlohy bylo vytvořit konvexní obálku nad množinou vygenerovaných bodů (a i jiných tvarů). Konvexní obálka je často využívána jako pomocná geometrická struktura v mnoha algoritmech. V kartografii se využívá například pro detekci tvaru a natočení budov.

„Konvexní obálka ϰ konečné množiny S v dvojrozměrném euklidovském prostoru představuje:

1. *nejmenší konvexní mnohoúhelník P obsahující S (tj. neexistuje P‘ ⊂ P).*
2. *konvexní mnohoúhelník P s nejmenší plochou.*
3. *průsečnici všech polorovin obsahujících S.*
4. *sjednocení všech trojúhelníků, jejichž vrcholy tvoří body v S.*

Množinu S označíme jako konvexní, pokud spojnice libovolných dvou prvků leží zcela uvnitř této množiny.“ [1]

Tedy jednoduše řečeno jde o to, vytvořit kolem množiny takový polygon, který bude obsahovat všechny body množiny, ale nebude možné vytvořit menší polygon, který by všechny body též obsahoval. Polygon také musí být konvexní (tedy když spojíme libovolné dva body/prvky množiny tak bude celá spojnice ležet v polygonu).



Obr. 1 Ukázka konvexní obálky [2]

Existuje množství metod, které se používají pro konstrukci konvexní obálky. V této úloze byly využity jedny z nejčastěji používaných metod, a to: Jarvis Scan, Quick Hull a Sweep Line. Některé metody jsou využitelné i v jiné než dvojrozměrné dimenzi, ale pro tuto úlohu byl uvažován pouze dvojrozměrný prostor.

[1]

2.1 Údaje o bonusových úlohách

Byly vypracovány bonusové úlohy kromě Minimum Area Enclosing Box (nebylo vyřešeno z časových důvodů).

Pro tvorbu konvexní obálky využita ještě jedna z často využívaných metod: Graham Scan. Algoritmus byl v rámci bonusových úloh přidán do kódu a je popsán v dalších kapitolách.

Konstrukce striktně konvexních obálek pro všechny uvedené algoritmy byla vytvořena závěrečnou kontrolou v každém algoritmu a odstraněním nežádoucích bodů z konvexní obálky.

Byla ošetřena singularita u algoritmu Jarvis Scan týkající se existence kolineárních bodů v datasetu. Dle obdržených informací se má jednat v zadání úlohy o Graham Scan, ale singulární případ u Jarvis Scan byl již ošetřen a popsán před obdržením informace. Graham Scan byl poté též ošetřen ohledně singularit a postup byl popsán.

Dále byl naprogramován algoritmus pro automatické generování konvexních / nekonvexních množin bodů různých tvarů. Konkrétními použitými tvary jsou: kruh, elipsa, čtvercová mřížka, náhodná množina bodů a čtverec.

3 Popisy algoritmů

V rámci úlohy měla být vytvořena aplikace, kde nad různou vstupní množinou je pomocí vybraného algoritmu zkonstruována konvexní obálka. Pro tvorbu konvexní obálky mělo být naprogramováno několik níže popsaných algoritmů. Dále měl být vytvořen algoritmus pro generování vstupní množiny různých tvarů (body, kruh, elipsa, čtverec, …). Také měl být naprogramován algoritmus pro určování hlavního směru budov (Minimum Area Enclosing Box). V této kapitole jsou popsány všechny potřebné algoritmy.

3.1 Jarvis Scan

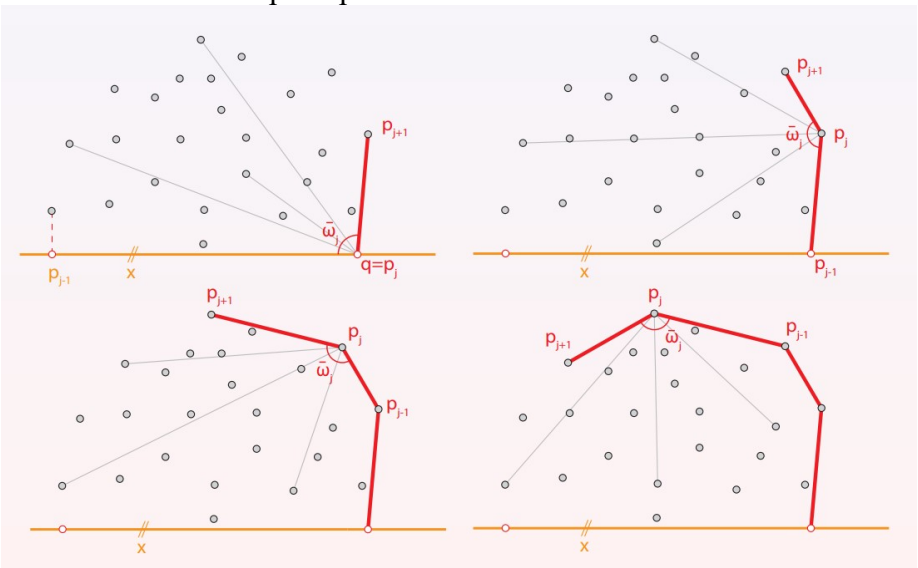
Tento algoritmus připomíná Gift Wrapping Algorithm. Vyhledává body konvexní obálky na základě maximálního úhlu. Předpokladem ovšem je, že množina vstupních bodů neobsahuje tři kolineární body. Tento algoritmus lze rozšířit pro prostorové body. Jarvis Scan má jednoduchou implementaci, jeho nevýhodou je pak exponenciální časový odhad, tedy je nevhodný pro velké množiny bodů.

Algoritmus vybírá první výchozí bod podle nejmenší y souřadnice. O tomto bodu víme, že bude ležet v konvexní obálce. Bodem vedeme rovnoběžku s osou x, procházíme všechny body množiny a počítáme úhly mezi rovnoběžkou a spojnicí pivotu s body množiny. Poté úhly porovnáme a bod s největším úhlem přidáme do konvexní obálky.

V dalším kroku by bylo třeba ošetřit singulární situaci, kdy ke dvěma bodům je téměř stejný úhel, tj. 3 body jsou kolineární. V takovém případě by měl být přidán bod vzdálenější od pivotu. O singulárním případu a jeho ošetření v algoritmu je více popsáno v kapitole 4.1.

Dále se nově přidaný bod stane pivotem, přímka od které se měří úhly je určena tímto bodem a bodem předchozím a postup se opakuje dokud nedojdeme opět k výchozímu bodu.

Grafické znázornění postupu:



Obr. 2 Ukázka Jarvis Scan [1]

Zápis algoritmu (s neošetřeným singulárním případem):

1. Nalezení pivota q, $q = min(y_i)$.
2. Přidej $q \rightarrow \varkappa$.
3. Inicializuj $p_{j-1} \in X, p_j = q, p_{j+1} = p_{j-1}$.
4. Opakuj, dokud $p_{j+1} \neq q$:
5. nalezni $p_{j+1} = arg\ max\ \forall p_i \in P < (p_{j-1}, p_j, p_i)$,
6. přidej $p_{j+1} \rightarrow \varkappa$,
7. $p_{j-1} = p_j$; $p_j = p_{j+1}$.

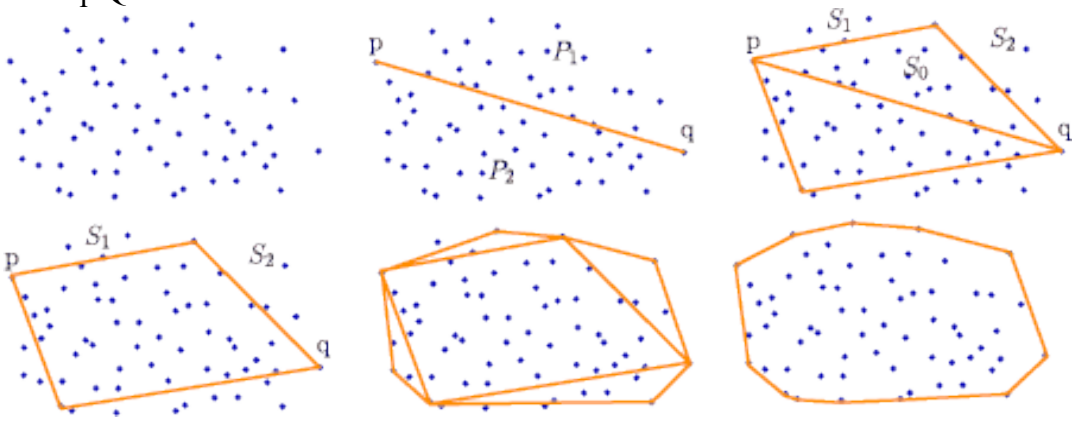
[1]

3.2 Quick Hull

Jde o rychlejší metodu tvorby konvexní obálky, časová náročnost je ve většině případů O(n.log(n)), ale v ojedinělých nejhorších případech může dosáhnout až exponenciálního růstu jako třeba Jarvis Scan, tedy $O(n^2)$. Princip algoritmu spočívá ve vyhledávání nejvzdálenějšího bodu od přímky určené dvěma body množiny. Obálka tedy expanduje všemi směry. Quick Hull používá princip Divide and Conquer, tj. rozděl a panuj, jak bude patrné z dalšího popisu.

Při tvorbě konvexní obálky vycházíme z přímky tvořené dvěma body (jeden s nejmenší souřadnicí x a druhý naopak s největší). Množina bodů je tím rozdělena na dvě poloviny, přičemž body prvotní přímky patří do obou polovin a jsou přidány i do konvexní obálky. V horní polovině najdeme bod nevzdálenější od přímky a takový bod přidáme do konvexní obálky. Tím vznikají dvě přímky a od každé opět hledáme nejvzdálenější bod v horní polovině bodů, který dosud neleží uvnitř konvexní obálky, jak je patrné z obrázku 3. Takto nalezené body přidáme do konvexní obálky a postup opakujeme pro všechny další přímky, dokud nejsou všechny body v horní polovině zařazeny do konvexní obálky. Totéž opakujeme pro dolní polovinu bodů.

Princip Quick Hull:



Obr. 3 Ukázka Quick Hull [3]

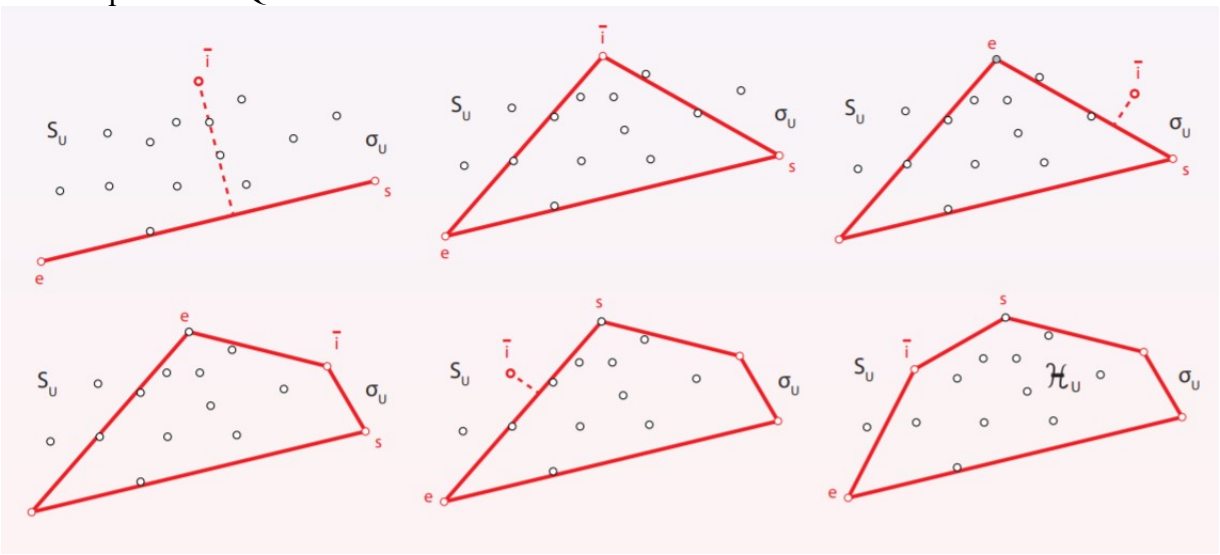
Zápis algoritmu:

1. Inicializace konvexní obálky, horní a dolní množiny bodů: $\varkappa = \emptyset$, $S_U = \emptyset$, $S_L = \emptyset$.
2. Nalezení extrémních bodů - bodů dělící přímky:
   $q_1 = min\, \forall p_i \in S(x_i)$, $q_3 = max\, \forall p_i \in S(x_i)$.
3. Přidání bodů přímky do horní množiny: $S_U \leftarrow q_1$, $S_U \leftarrow q_3$.
4. Přidání bodů přímky do dolní množiny: $S_L \leftarrow q_1$, $S_L \leftarrow q_3$.
5. Cyklus pro všechny body množiny: *for* $\forall p_i \in S$,
6. zařaď bod do horní množiny, pokud je v horní polovině:
   if $(p_i \in \sigma_l\, (q_1, q_3))$ $S_U \leftarrow p_i$
7. jinak zařaď bod do dolní množiny: *else* $S_L \leftarrow p_i$.
8. Přidání dolního extrémního bodu do konvexní obálky: $\varkappa \leftarrow q3$.
9. Vyhledání nejvzdálenějšího bodu v horní množině, přidání do konvexní obálky, opakování pro všechny nové přímky (rekurzivní procedura nad horní množinou): *Quick Hull*$(1, 0, S_U, \varkappa)$.
10. Přidání horního extrémního bodu do konvexní obálky: $\varkappa \leftarrow q_1$.
11. Opakování bodu 9. pro dolní množinu: *Quick Hull*$(0, 1, S_L, \varkappa)$.

Výše popsaný algoritmus je globální procedurou, body 9. a 11. jsou však sami lokální procedurou, proto je ještě třeba popsat je zvlášť. Algoritmus lokální procedury bychom zapsali takto:

1. Nalezení bodu: $\underline{p} = arg\, max\, \forall p_i \in S \,||\, p_i - (p_s, p_e)||$, $\underline{p} \in \sigma_r(p_s, p_e)$.
2. Pokud exituje bod vpravo od hrany: *If* $\underline{p} \neq \emptyset$
3. Rekurzivní volání nad prvním segmentem $(p_s, \underline{p})$, $\underline{i}$ je index $\underline{p}$:
   Quick Hull$(s, \underline{i}, S, \varkappa)$.
4. Přidání bodu: $\varkappa \leftarrow \underline{p}$.
5. Rekurzivní volání nad druhým segmentem $(\underline{p}, p_e)$, $\underline{i}$ je index $\underline{p}$:
   Quick Hull$(\underline{i}, e, S, \varkappa)$.

Lokální procedura Quick Hull:



Obr. 4 Ukázka Quick Hull – Lokální procedura (v horní polovině bodů) [1]

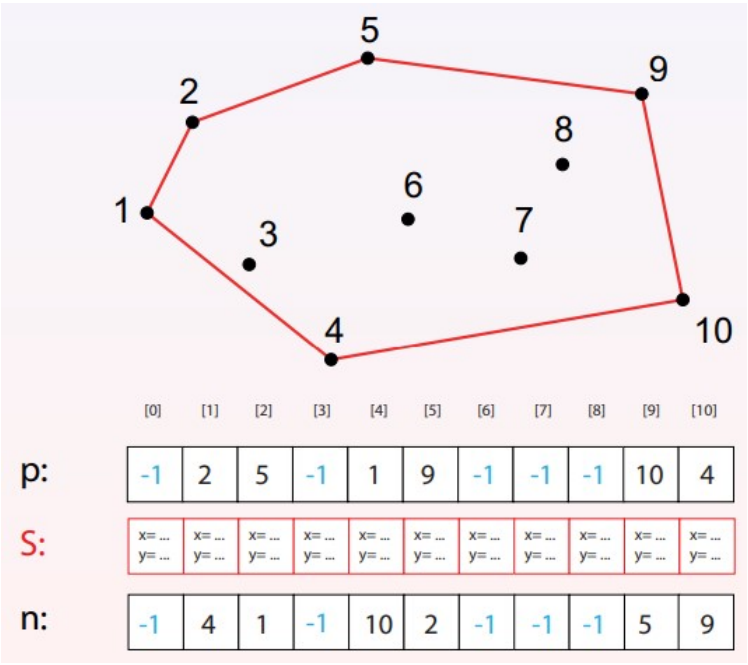
p_s, p_e – jsou body vstupní hrany, které jsou jakožto koncové body segmentu předávány v této lokální proceduře. Lokální procedura je patrná na obr. 4.

[1]

3.3 Sweep Line

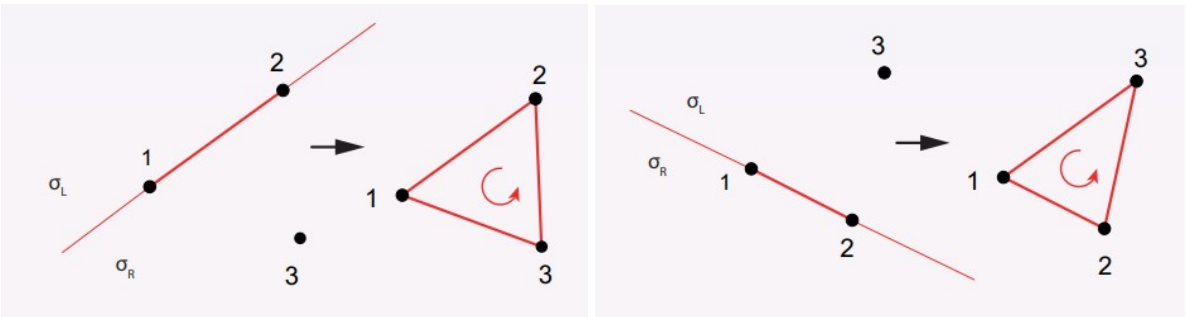
Sweep Line (nebo též Metoda zametací přímky) je algoritmus, který funguje na principu inkrementální (přírustkové) konstrukce. Množina bodů je v tomto algoritmu rozdělena na zpracovanou/nezpracovanou část pomocí nadroviny. Složitost tohoto algoritmu je logaritmická (O(n.log(n))) – podobná jako Quick Hull ve většině případů. Metodu lze převést do vyšší dimenze. Algoritmus je citlivý na singularity.

Na počátku algoritmu stojí předzpracování, ve kterém setřídíme body podle souřadnice x. Pro algoritmus použijeme celkem tři seznamy. První seznam obsahuje všechny vrcholy, druhý seznam předchůdců a třetí seznam následníků. Vrchol bez předchůdce/následníka má hodnotu v seznamu předchůdců/následníků -1. Pokud vrchol nemá ani předchůdce, ani následníka, leží uvnitř konvexní obálky a netvoří její vrchol, viz obr. 5.



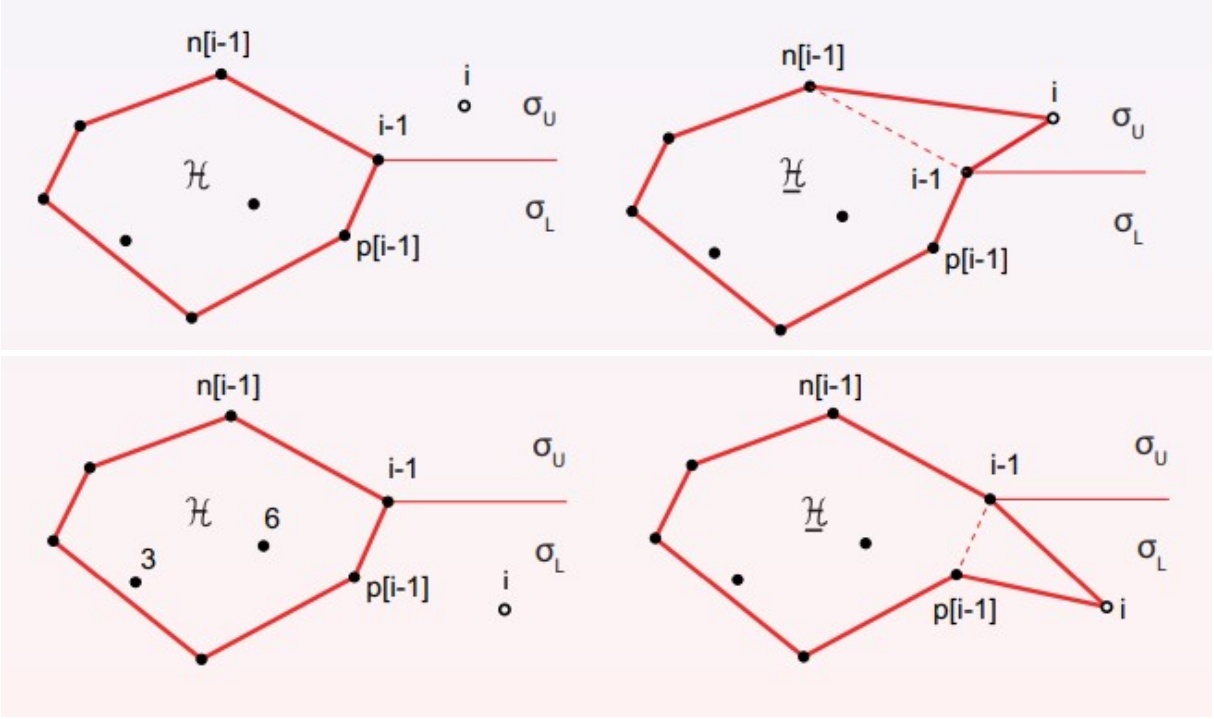
Obr. 5 Ukázka reprezentace konvexní obálky – Sweep Line [1]

Na počátku spojíme první dva body (dostaneme hranu tvořenou body s dvěma nejnižšími souřadnicemi x). Poté přidáme další bod, který leží v levé nebo pravé polorovině vzhledem k přímce – to je důležité proto, že musíme určit předchůdce a následníky a je nutné zachovat CCW orientaci. To je iniciální fáze algoritmu.



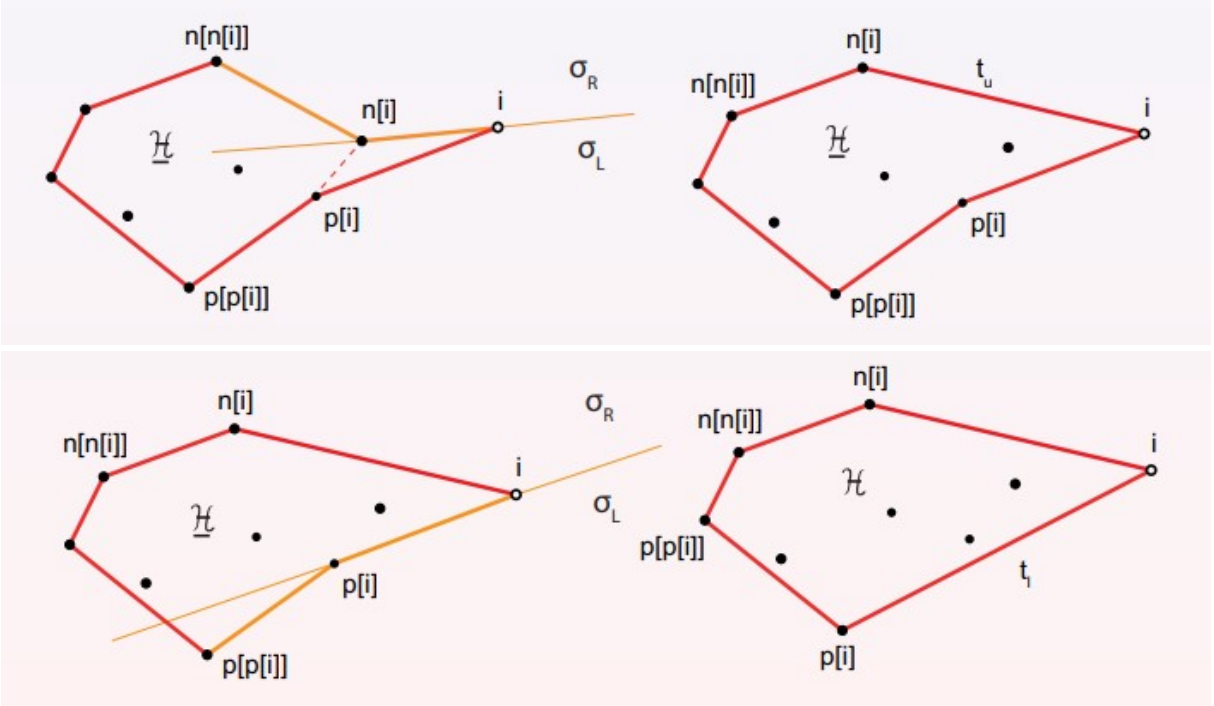
Obr. 6 Iniciální fáze algoritmu, zachování CCW orientace – Sweep Line [1]

Dále přichází na řadu iterativní fáze I., ve které je vytvořena aproximace konvexní obálky se zachovanou CCW orientací jež nemusí být konvexní. Podle souřadnice y dojde k rozpojení obálky mezi danými body a nový vrchol je spojen s předchůdcem a následovníkem (je nahrazena sekvence vrcholů). Taktéž předchůdce a následovník je propojen s novým bodem.



Obr. 7 Iterativní fáze I., zachování CCW orientace, aproximace konvexní obálky – Sweep Line [1]

A nakonec dojdeme k iterativní fázi II., kdy musíme nekonvexní aproximaci převést na konvexní obálku. Nekonvexní vrcholy jsou v tomto kroku vynechány. Nekonvexní vrcholy jsou nahrazeny horní a dolní tečnou.



Obr. 8 Iterativní fáze II., náhrada nekonvexních vrcholů horní a dolní tečnou – Sweep Line [1]

Zápis algoritmu:

1. Sort $P_s = sort(P)$ *by x*.
2. If $(p_3 \in \sigma L(p_1, p_2))$
3. $n[1] = 2; n[2] = 3; n[3] = 1$
4. $p[1] = 3; p[2] = 1; p[3] = 2$
5. else
6. $n[1] = 3; n[3] = 2; n[2] = 1$
7. $p[1] = 2; p[3] = 1; p[2] = 3.$
8. For $p_i \in P_s$, $i > 3$
9. if $(y_i > y_i - 1)$
10. $p[i] = i\text{-}1; n[i] = n[i\text{-}1];$
11. else
12. $n[i] = i\text{-}1; p[i] = p[i\text{-}1];$
13. $n[p[i]] = i; p[n[i]] = i;$
14. while $(n[n[i]]) \in \sigma R(i, n[i])$
15. $p[n[n[i]]] = i; n[i] = n[n[i]];$ //Poradi!
16. while $(p[p[i]]) \in \sigma L(i, p[i])$
17. $n[p[p[i]]] = i; p[i] = p[p[i]];$ //Poradi!

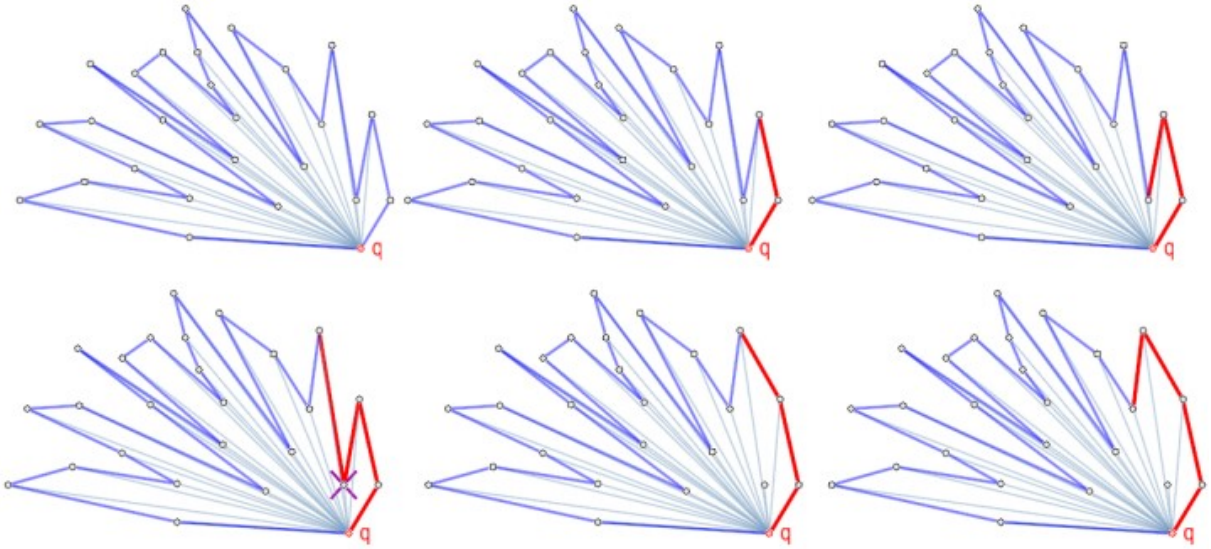
[1]

3.4 Popisy algoritmů – bonusové úlohy

3.4.1 Graham Scan

Tento algoritmus není možné převést do vyšší dimenze (nelze rozšířit do R^3). Jeho časová náročnost je logaritmická O(n.log(n))), tedy patří k těm rychlejším. Je možné jej využít i na velké množiny. Principem je převod star-shaped polygonu na konvexní obálku.

Nejprve nalezneme pivot a to tak, že vybereme bod s nejmenší souřadnicí y. Z vrcholů setříděných podle úhlu vytvoříme star-shaped polygon. Na počátku zařadíme pivot a následující bod s nejmenším úhlem do konvexní obálky. Přidávání dalších bodů probíhá tak, že vezmeme poslední body zařazené do konvexní obálky a další bod v pořadí podle úhlů. Potom testujeme levotočivost trojúhelníku, který tvoří tyto body. Pokud má trojúhelník CCW orientaci, zařadíme testovaný bod do konvexní obálky a posuneme se na další bod. Pokud ne, odebereme poslední bod konvexní obálky.



Obr. 9 Ukázka konstrukce Graham Scan [1]

Zápis algoritmu:

1. Nalezení pivota $q = min\, \forall p_i \in S(y_i),\ q \in H.$
2. Setřídění $\forall p_i \in S$ dle $\omega_i = \angle(p_i, q, x)$, index j odpovídá setříděnému pořadí.
3. Pokud $\omega_k = \omega_l$, vymaž bod p_k , p_l bližší ke q.
4. Inicializuj $j = 2;\ S = \emptyset$
5. $S \leftarrow q,\ S \leftarrow p_l$ (indexy posledních dvou prvků p_t , p_{t-1})
6. Opakuj pro $j < n$:
7. if p_j vlevo od p_{t-1}, p_t :
8. $S \leftarrow p_j$
9. $j = j + 1$
10. else pop S.

[1]

3.4.3 Generování konvexních / nekonvexních množin bodů různých tvarů

Generování tvarů je prováděno uvnitř třídy Draw, v metodě generatePoints, kde podle zvoleného typu jsou vygenerovány různé množiny. Počet bodů je volen vepsáním do příslušného pole. Pokud je počet pro daný obrazec nedostatečný, objeví se chybová hláška (např. pro kružnici jsou potřeba minimálně 3 body). Pokud je číslo menší než nula, nebo není celé, objeví se opět chybové hlášení. V těchto případech nejsou vykresleny žádné body.

Náhodné body

Náhodné body se generuji pomocí funkce rand v závislosti na šířce w a výšce h. Aby bylo dosaženo vizuálně korektního generování bodů uvnitř okna, hodnoty šířky a výšky byly odsazeny o 10 bodů od okrajů rámečku.

Kružnice

Pokud je vstupní počet bodů menší než tři, je uživateli zobrazena chybová hláška. Pro tvorbu kruhu byl nejdříve zvolen jeho minimální poloměr tak, že se posoudilo, který ze dvou rozměru okna je menší. Dále byl nastaven střed kružnice jako polovina velikosti výšky a šířky okna. Středový úhel pro generaci křižnice se získává tak, že délka kružnice rovnoměrně se vydělí počtem zadaných bodu, dále kružnice se vygeneruje dle následujících rovnic:

$X = X_0 + r.\cos(fi)$
$Y = Y_0 + r.\sin(fi)$

Čtvercová síť
Čtvercová síť se generuje pomocí matematické operace odmocniny (sqrt) z uživatelem zadaného počtu bodů.

Protože je čtvercová síť sítí o rozměrech $\sqrt{n} * \sqrt{n}$, tak pokud je vstupem číslo, které má neceločíselnou odmocninu, objeví se upozornění na tuto skutečnost a na to, že bude použito nejbližší nižší číslo s celočíselnou odmocninou.

Elipsa
Představme si, že kreslící pole rozdělíme na 3 x 3 stejných polí. V prostředním poli se vygeneruje v náhodné pozici střed. Poté se náhodně vygenerují poloosy a a b o velikosti maximálně jako hrana jednoho pole v daném směru. Omezení generování třetinou pole je použito kvůli tomu, aby elipsa nebyla vykreslena mimo viditelnou oblast. Na základě zadaných parametrů jsou poté vygenerovány body na obvodu elipsy, dle požadovaného počtu.

Čtverec
Nejprve vygenerujeme levý horní bod. Zvolíme náhodně souřadnici v oříznutém rámečku tak, aby zbyl vpravo a dole prostor pro další body a aby se nahoře a vlevo nevykresloval příliš na hraně (například souřadnice x se vygeneruje jako: *rand()%(w-70)+10,* tj. náhodné číslo z rozměru kreslící plochy menší o 70 kvůli prostoru napravo, který bude výsledně 60 bodů, a o 10 bude zvětšené kvůli odsazení od levého kraje). Poté se ze zbývajícího rozměru dolů a vpravo vybere menší, aby byl celý čtverec vidět. Z tohoto čísla se opět vygeneruje náhodné číslo představující délku hrany. Ovšem opět bude generování posunuté a to tak, aby napravo i dole zbývaly vždy body od kraje a aby byl nějaký nejmenší rozměr hrany čtverce. Dále se požadované body rozdělí rovnoměrně mezi 4 strany a zbylé souřadnice se již snadno dopočítají.

4 Problematické situace a jejich rozbor

4.1 Jarvis Scan – ošetření singularity

Singulární případ u tohoto algoritmu nastává, pokud tři a více bodů je kolineárních, tj. tři a více bodů leží na přímce. V algoritmu hledáme bod, k němuž je od výchozí přímky největší úhel. Leží-li body na přímce, která je totožná s konvexní obálkou, je výsledek nejednoznačný. K více bodům máme stejný (maximální) úhel. Abychom tomuto zabránili, musíme upravit algoritmus.

Aktuálně největší nalezený úhel porovnáváme s dalším vypočteným úhlem. Pokud jsou úhly totožné (v rámci nastavené tolerance) je vypočtena vzdálenost od aktuálního bodu konvexní obálky k bodům, k nimž je totožný úhel. Do konvexní obálky je následně zařazen bod s největší vzdáleností. Bod s menší vzdáleností není do konvexní obálky zařazen, dojde k jeho přeskočení.

Do algoritmu (popsaný v kapitole 3.1 Jarvis Scan) přidáme mezi body 5. a 6. následující:

pokud $|angle_{MAX} - angle_i| < tolerance$,
vypočti vzdálenost d_i, d_{iMAX}.
Pokud $d_i > d_{iMAX}$, tak $angle_{MAX} = angle_i$.

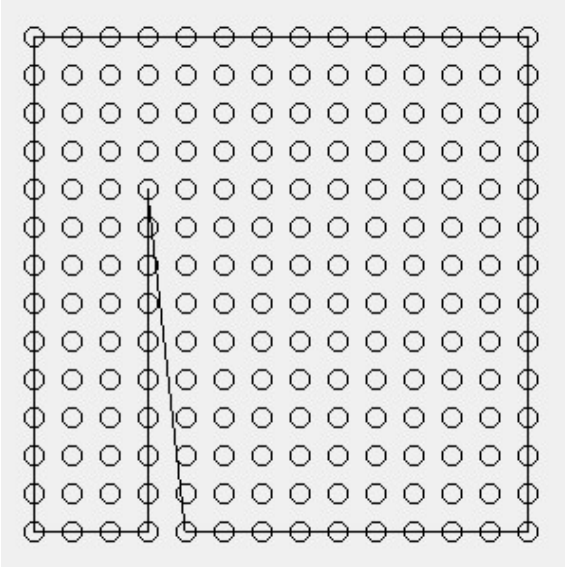
4.2 Konstrukce striktně konvexních obálek pro všechny algoritmy

U konvexní obálky může nastat, že bod ležící na úsečce mezi body konvexní obálky je též zařazen do konvexní obálky, avšak zařazení tohoto bodu je nežádoucí, proto jej musíme odebrat z konvexní obálky. Dále může nastat, že dva identické body jsou zařazeny do konvexní obálky, což je opět nežádoucí a jeden z nich musíme odstranit. U některých algoritmů je toto ošetřeno již v kódu kvůli špatnému výsledku při práci s takovými singularitami.

Celkové řešení je pak takové, že na závěr všech algoritmů ještě body projdeme a nežádoucí z konvexní obálky vyřadíme. K tomu existuje ve třídě Algorithms metoda fixPolygon, která upraví vracený výstup.

4.3 Sweep line – ošetření singularity

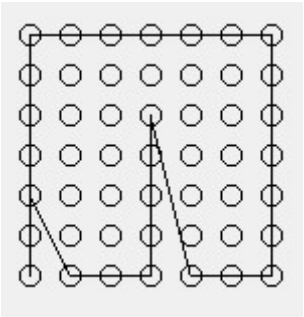
Po naprogramování algoritmu pro generování tvarů byla otestována metoda Sweep line na čtvercové mřížce a byl nalezen problém, patrně vzniklý kvůli citlivosti algoritmu na singularitu. Vytvořená obálka nebyla konvexní, jak je možno vidět na obrázku níže.



Obr. 10 Problém u Sweep line

Nejprve tedy byly odstraněny duplicitní body a to následujícím způsobem. Body byly setříděny podle souřadnice x. Poté proběhl for cyklus, který zapisoval neduplicitní body do pomocné proměnné. Tedy pokud bod neměl shodnou souřadnici x nebo y (stačila 1 z nich) s bodem následujícím, byl do proměnné uložen. Pokud by se vyskytovaly duplicitní body na vstupu, byl by uložen až poslední z nich. Po konci cyklu byl ještě uložen bod poslední.

Toto ovšem výše zmíněný problém nevyřešilo, a i nadále v některých případech vznikali problémy u mřížky.



Obr. 11 Trvající problém u Sweep line

U mřížky vzniká velké množství duplicit ve stejné jedné souřadnici (mnoho kolineárních bodů). Proto byl problém hledán v třídění bodů podle souřadnic. Kód pro třídění podle x byl upraven takto:

Původní kód: *return p1.x() < p2.x();* // Nepřipouští možnost rovnosti souřadnice x
Nový kód: *return p1.x() < p2.x()||((p1.x() == p2.x()) && (p1.y() < p2.y()));*
// Pokud jsou souřadnice x rovné, tak třídění probíhá podle y

Touto úpravou byla problémová situace vyřešena.

4.1.4 Graham Scan – ošetření singularity

Problém nastává, pokud jsou další dva body na přímce s pivotem, ke kterému jsou vztaženy všechny vypočtené úhly. Je to podobné problému u algoritmu Jarvis Scan. Pro odstranění problému úhly nejprve seřadíme podle velikosti. Poté porovnáváme vždy úhel a následující úhel v seřazeném pořadí. Pokud je rozdíl v rámci nastavené tolerance, jsou úhly shodné. Pak musí být vypočtena vzdálenost od pivotu. S bodem s menší vzdáleností není ve výpočtu uvažováno, dále počítáme pouze se vzdálenějším bodem.

5 Data

5.1 Vstupní data

Vstup tvoří buď ručně vkládaná množina bodů, nebo automaticky generovaná množina bodů o zvoleném počtu bodů a zvoleném tvaru.

5.2 Výstupní data

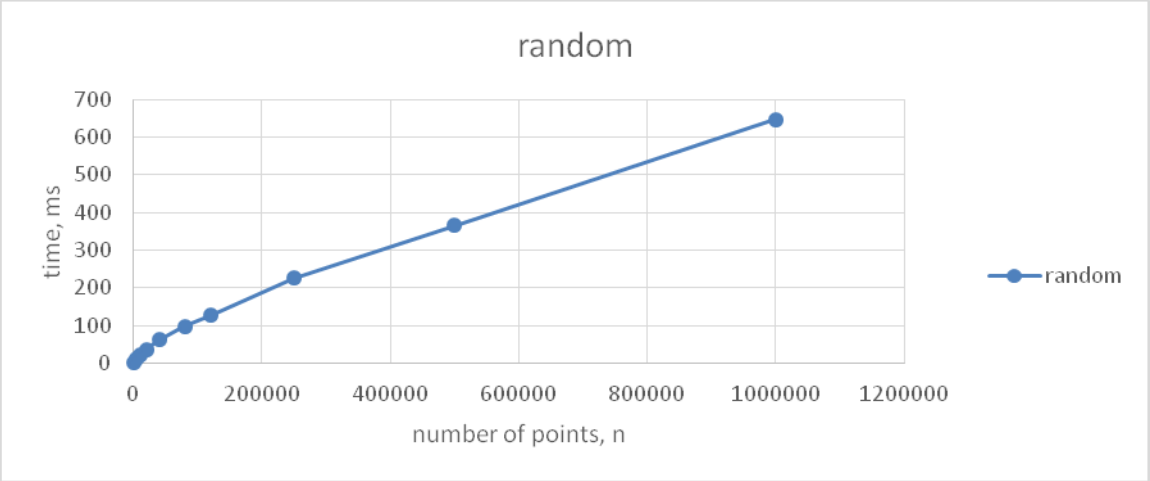
Výstupem grafické aplikace je vygenerovaná konvexní obálka podle zvoleného algoritmu, která je též graficky zobrazena. Dále pak doba běhu zvoleného algoritmu, která je též zobrazena v grafické aplikaci.

6 Testování doby běhu algoritmů

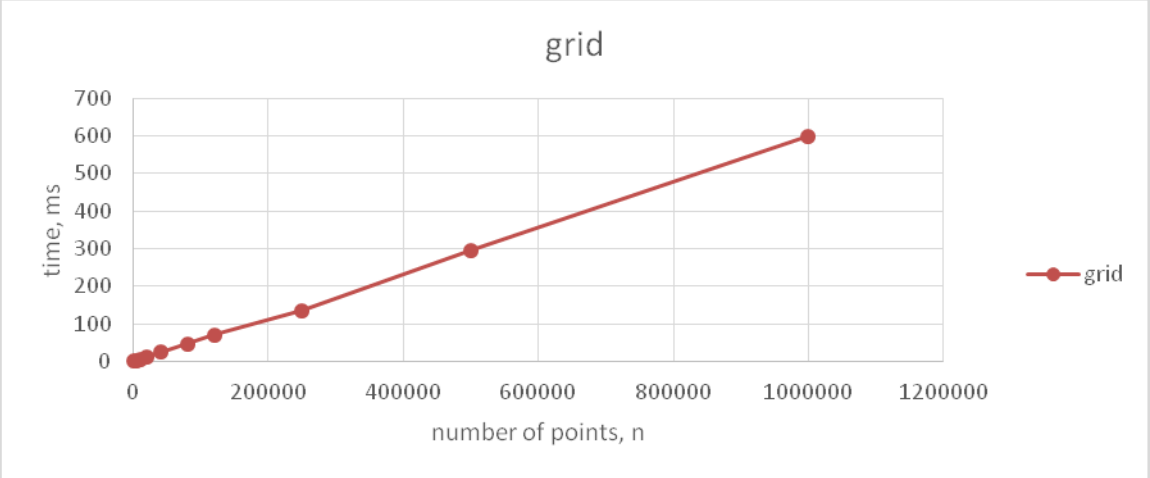
V rámci úlohy byla dle zadání otestována doba běhu algoritmů Jarvis Scan, Quick Hull a Graham Scan a to na těchto množinách bodů: náhodné body, rastr, body na kružnici. Měření bylo provedeno 10x, a na 10 různě velkých množinách (daný rozsah byl od 1 000 bodů do 1 000 000 bodů).

Měření bylo provedeno a výsledky byly zaznamenány. Originální soubor .XLS je přiložen. V přiloženém souboru jsou záložky pro algoritmy, tabulky s výslednými hodnotami a grafy pro dané druhy množin bodů a také graf zobrazující všechny druhy množin.

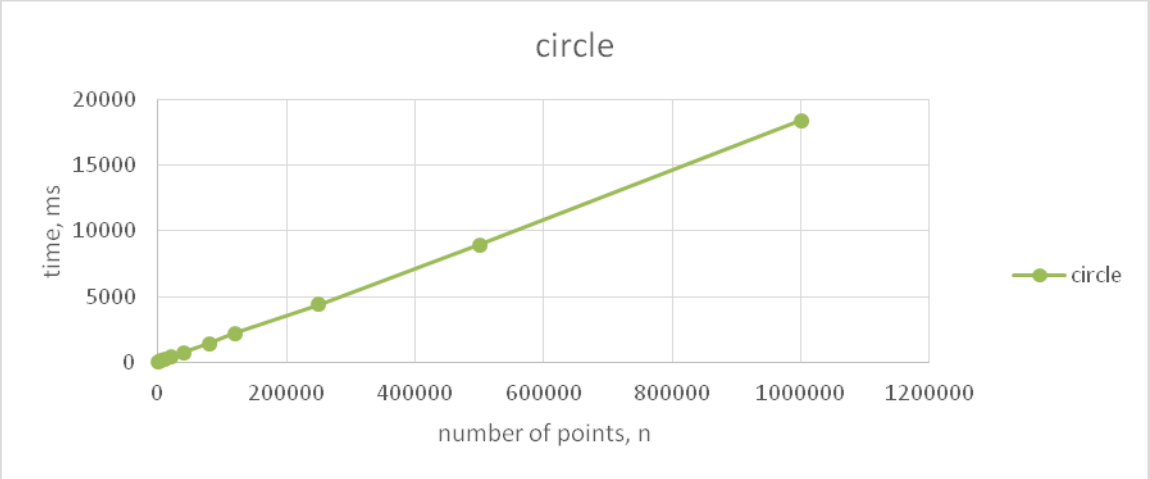
6.1 Grafy Jarvis Scan



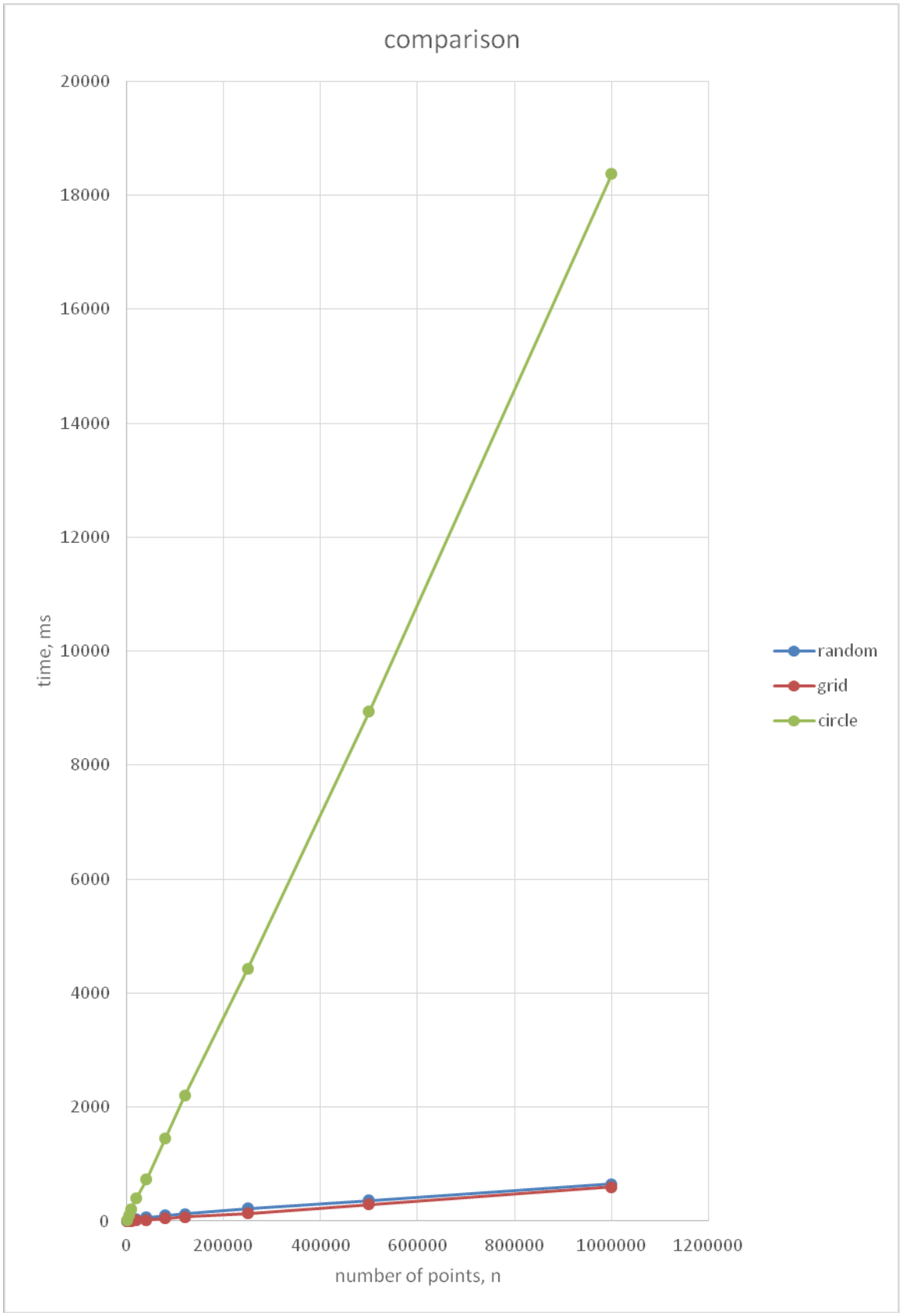
Graf. 1 Jarvis Scan na náhodné množině bodů



Graf. 2 Jarvis Scan na mřížce bodů

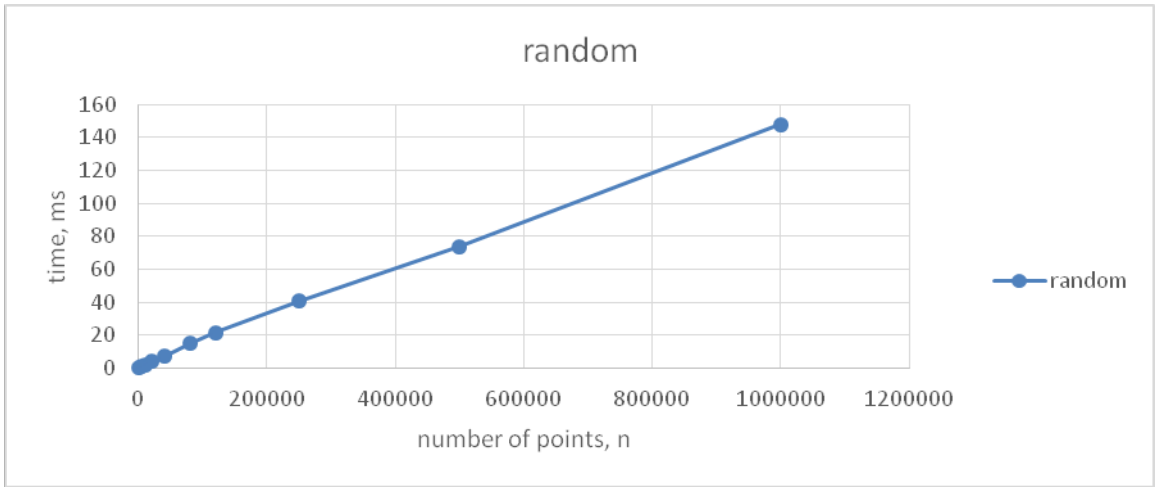


Graf. 3 Jarvis Scan na kruhové množině bodů

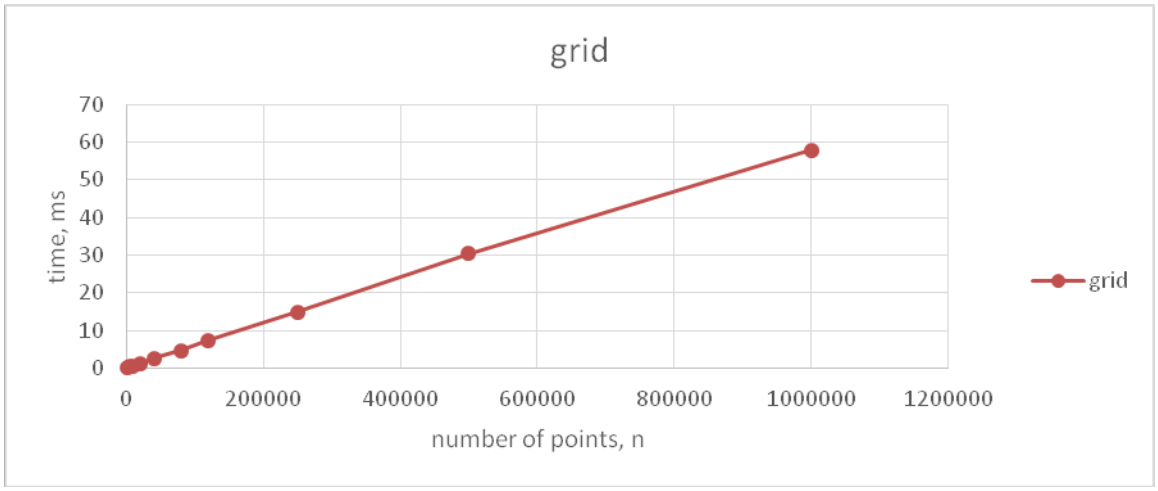


Graf. 4 Jarvis Scan – porovnání množin

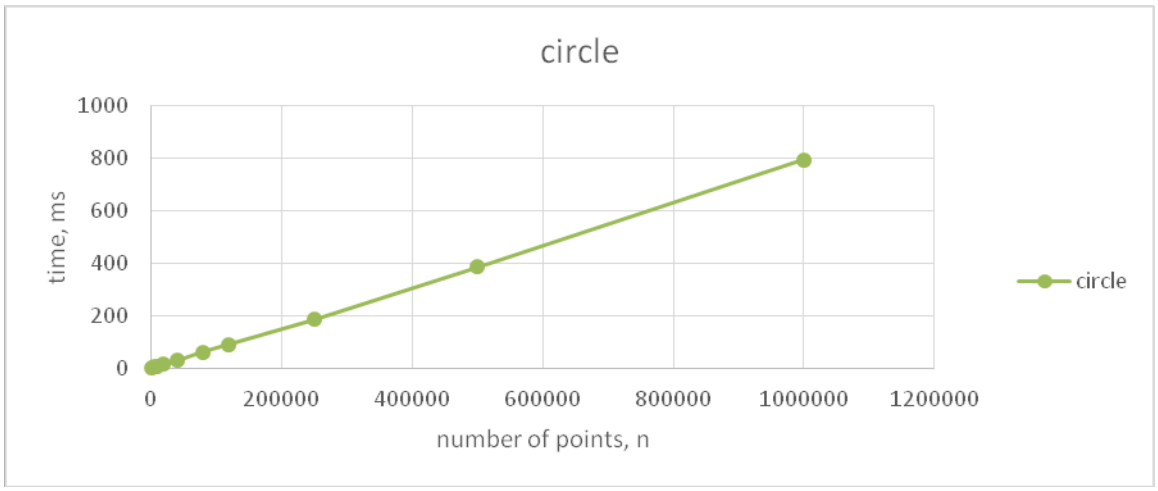
6.2 Grafy Quick Hull



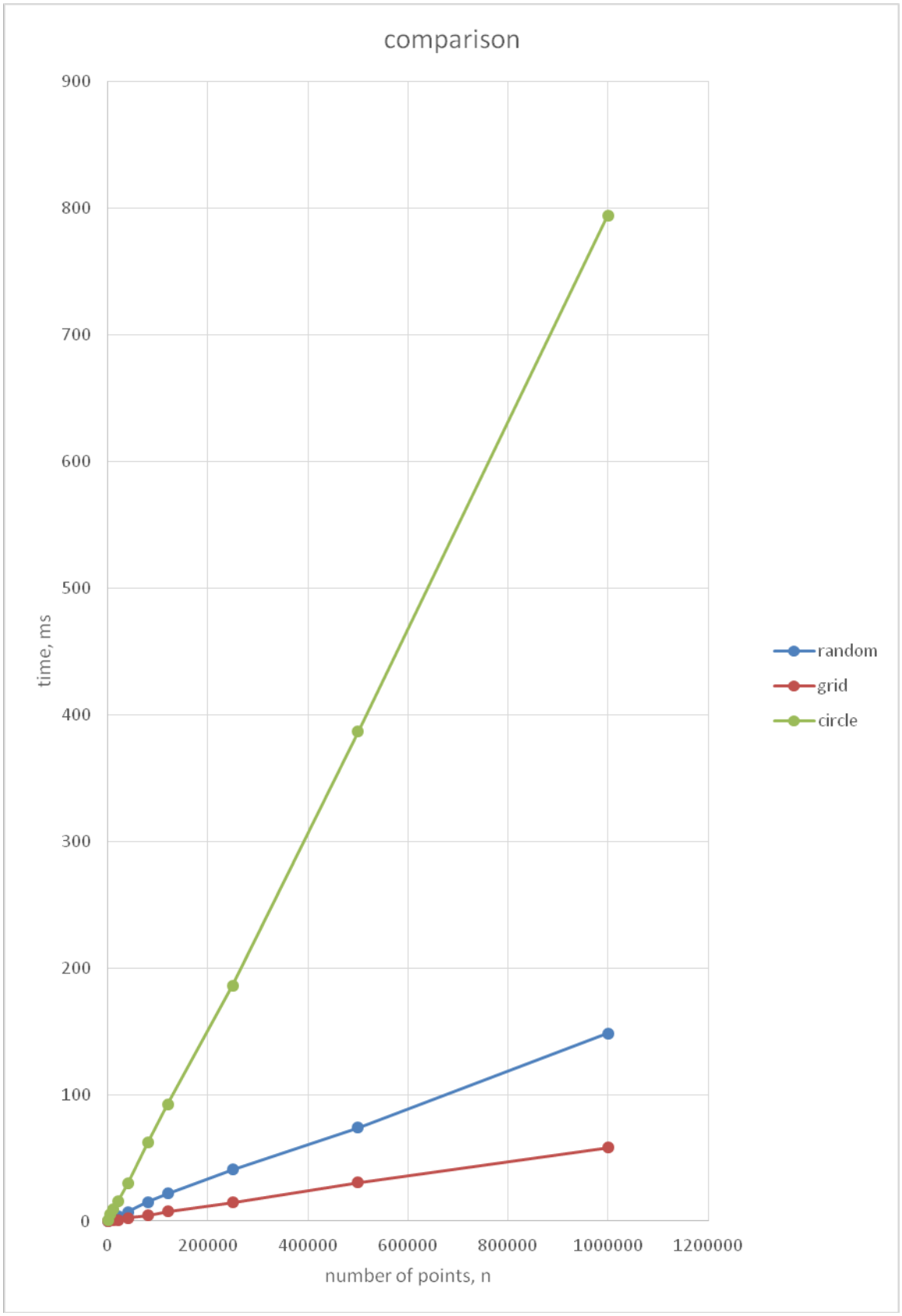
Graf. 5 Quick Hull na náhodné množině bodů



Graf. 6 Quick Hull na mřížce bodů

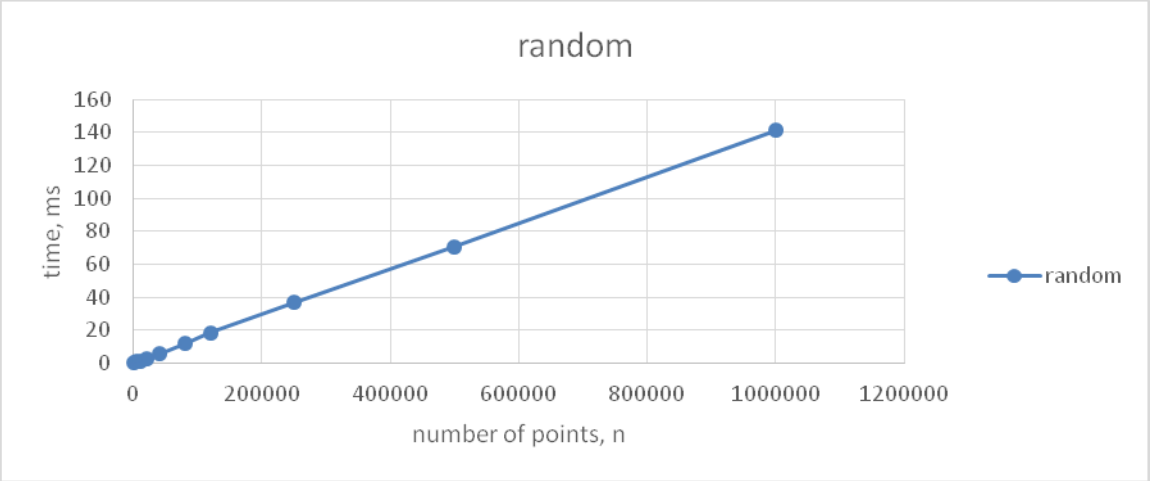


Graf. 7 Quick Hull na kruhové množině bodů

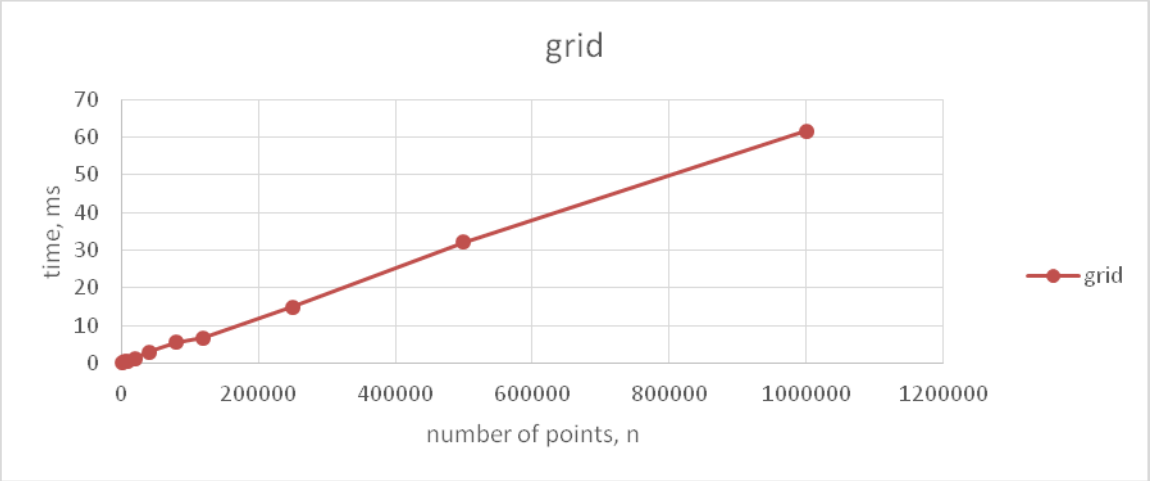


Graf. 8 Quick Hull – porovnání množin

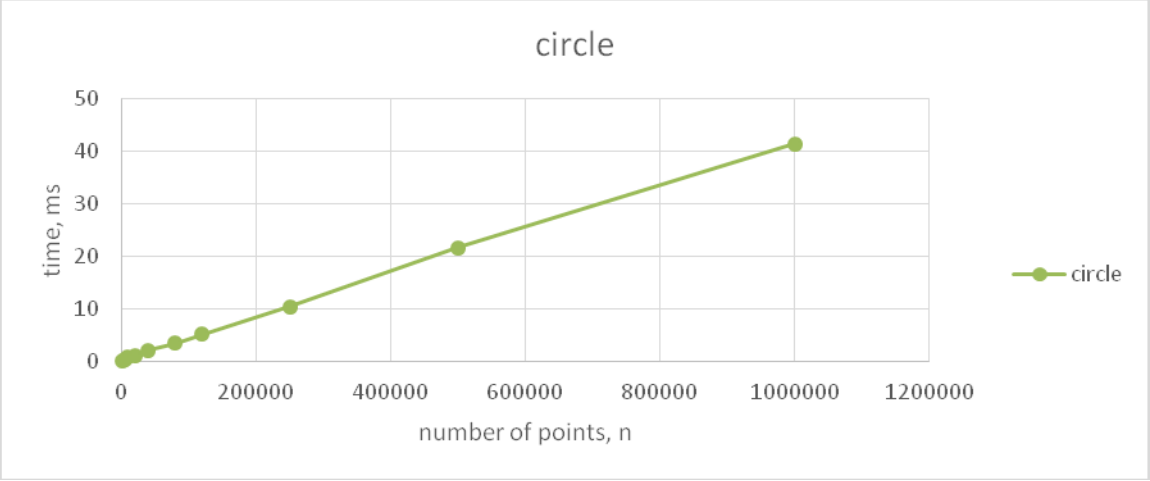
6.3 Grafy Sweep Line



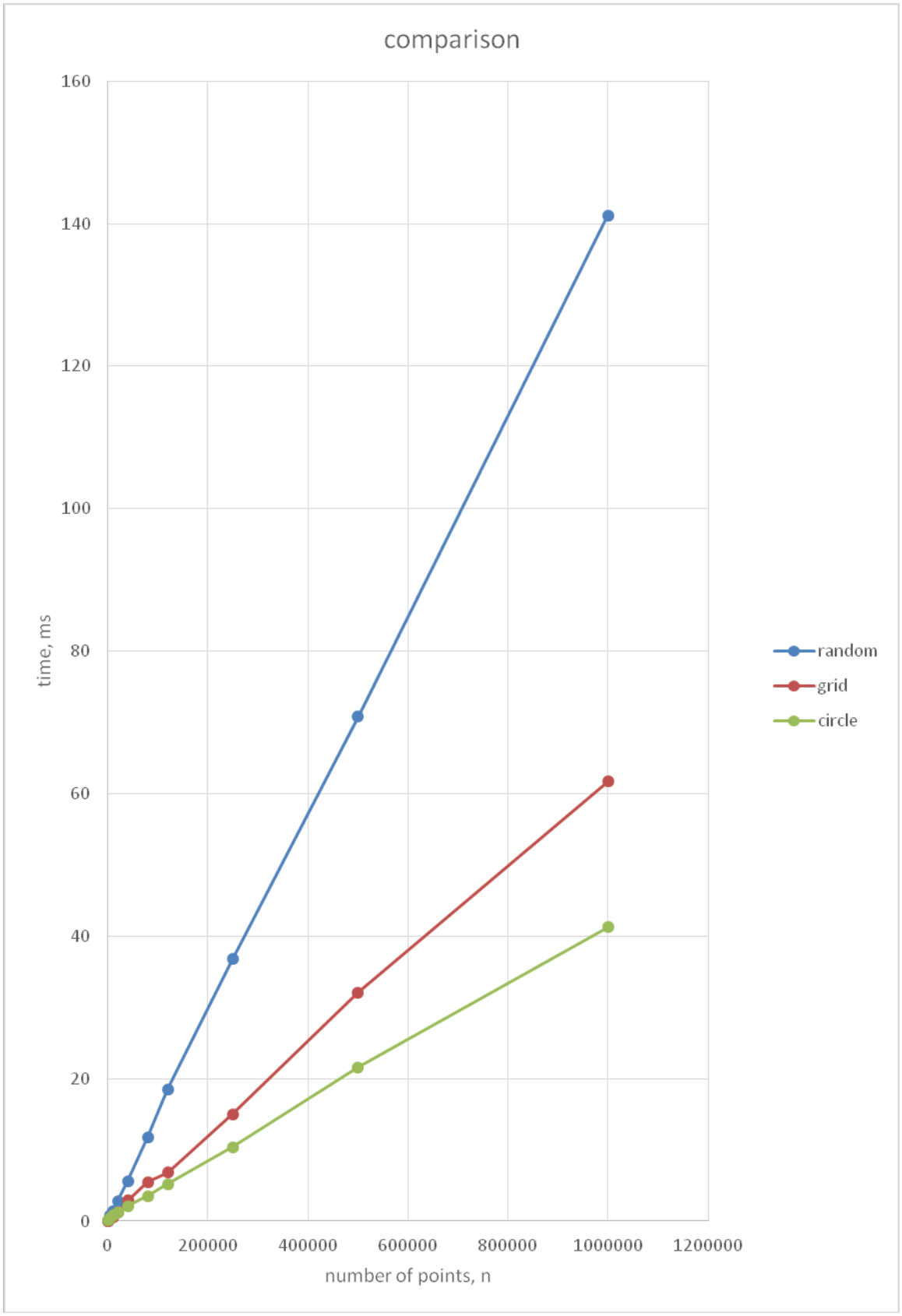
Graf. 9 Sweep Line na náhodné množině bodů



Graf. 10 Sweep Line na mřížce bodů



Graf. 11 Sweep Line na kruhové množině bodů



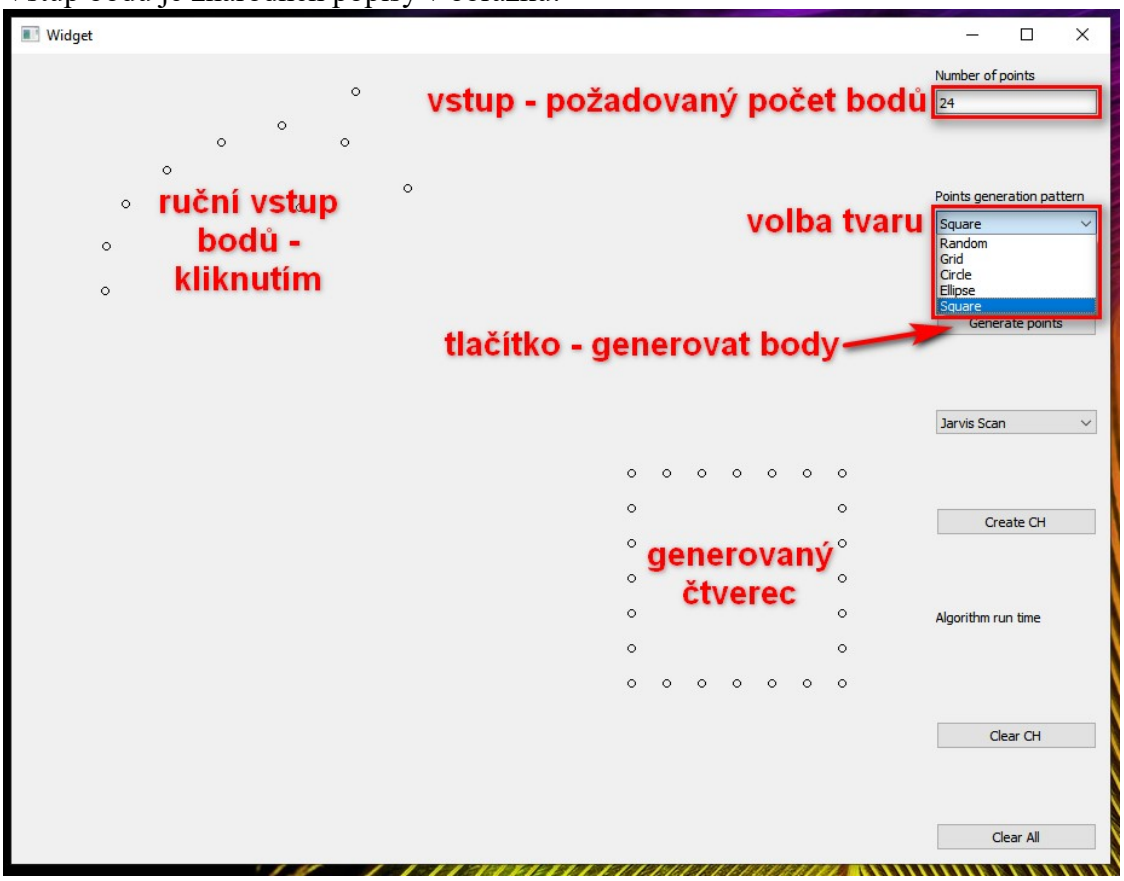
Graf. 12 Sweep Line – porovnání množin

6.4 Zhodnocení

Doba běhu algoritmů byla otestována. U algoritmů Jarvis Scan a Quick Hull se ukázalo, že nejpomalejší je výpočet na kružnici a také s rostoucím počtem bodů nejvíce roste jeho doba běhu. Jarvis Scan byl pak na mřížce i na náhodné množině téměř stejně rychlý a ve všech případech pomalejší než zbylé algoritmy. Quick Hull byl rychlejší než Jarvis Scan, ale o poznání se lišila doba běhu na mřížce a na náhodné množině, kde na mřížce byl o poznání rychlejší a měl na ní i pomalejší růst časů vzhledem k množství bodů. Sweep line byl nejrychlejší na kružnici, v čemž výrazně předčil ostatní algoritmy. U mřížky a náhodných bodů byla doba běhu podobná jako u Quick Hull. Pro výpočet bodů na kružnici je jednoznačně nejlepší metoda Sweep line. Pro body na mřížce se pro menší hodnoty zdá trochu lepší Sweep line, ale pro větší pak zase Quick Hull. Pro náhodnou množinou bodů se zdá být Sweep line o trochu lepší nežli Quick Hull. Jarvis scan je dobrý pouze pro malé hodnoty bodů, pro větší počet dochází k prudkému nárůstu času.

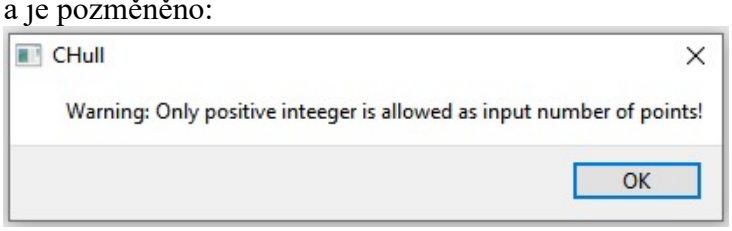
7 Printscreen vytvořené aplikace

Vstup bodů je znárodněn popisy v obrázku:

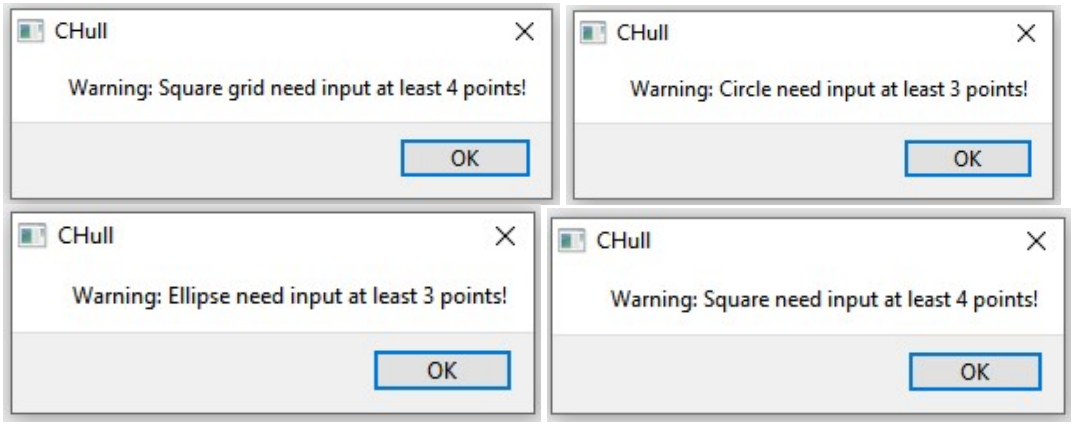


Obr. 12 Vstup bodů

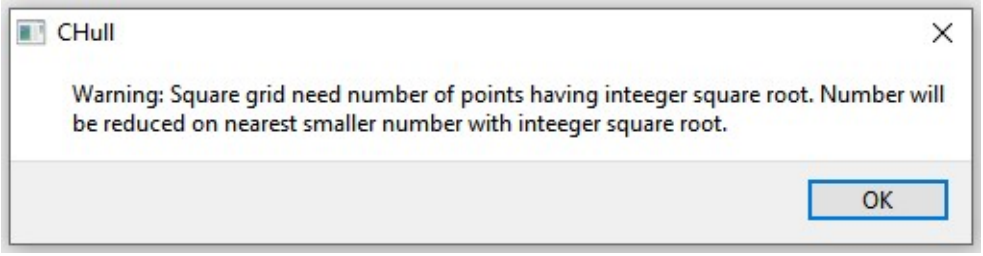
Chybný vstup při generování bodů ohlásí chybové hlášky. Taktéž, není-li číslo úplně korektní a je pozměněno:



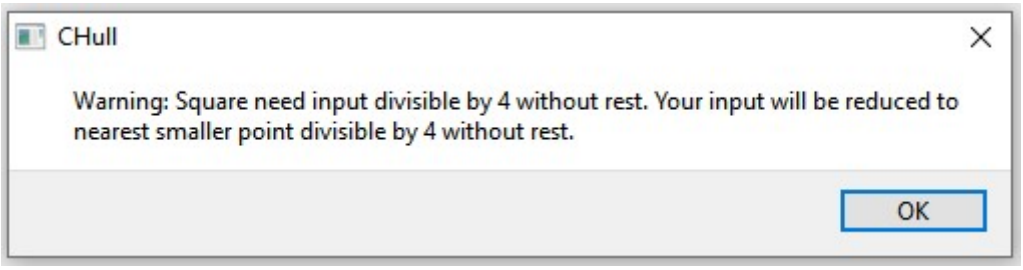
Obr. 13 Vstup je 0 nebo záporné či desetinné číslo



Obr. 14 Nedostatek bodů pro daný obrazec

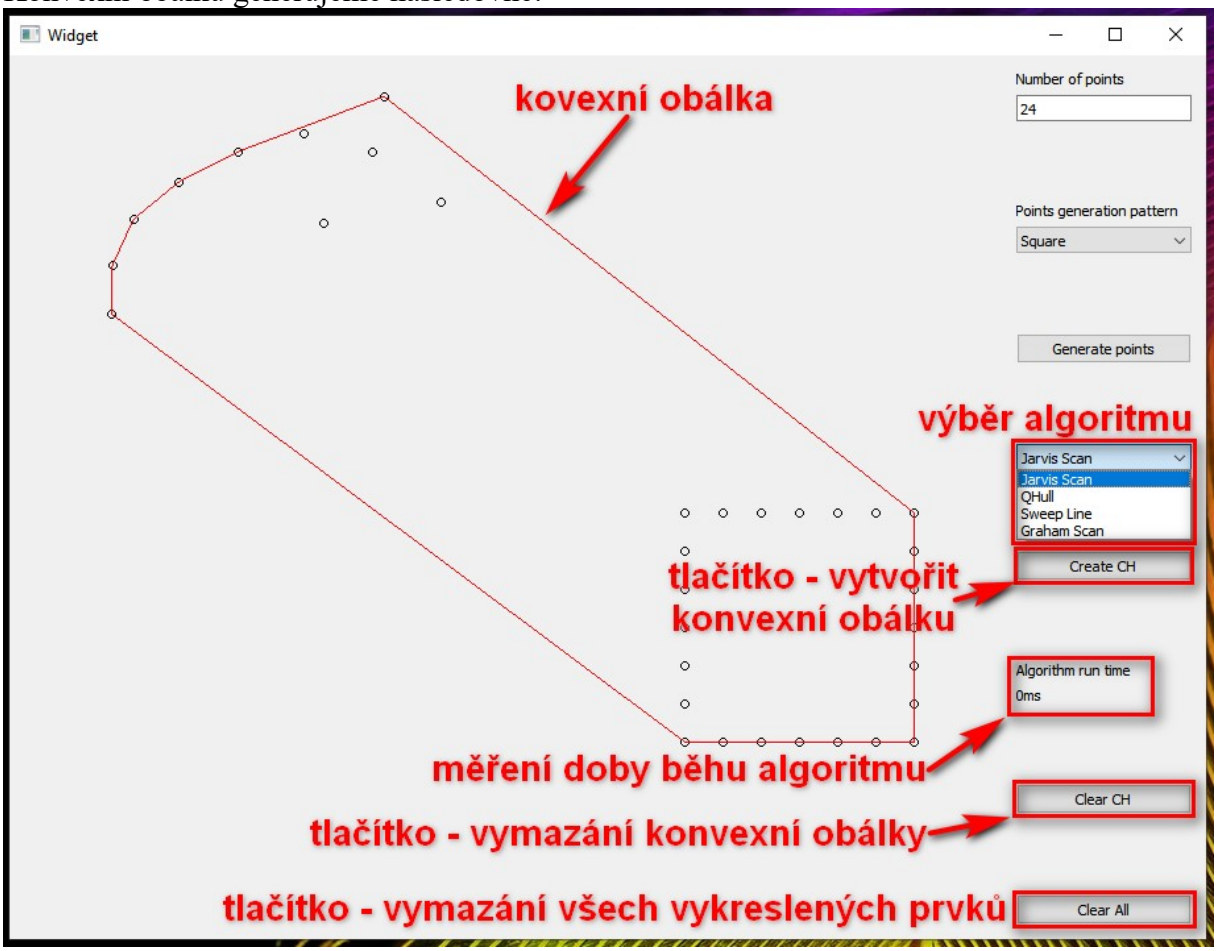


Obr. 15 Z počtu bodů nelze udělat celá odmocnina, číslo bude sníženo pro tvorbu čtvercové mřížky



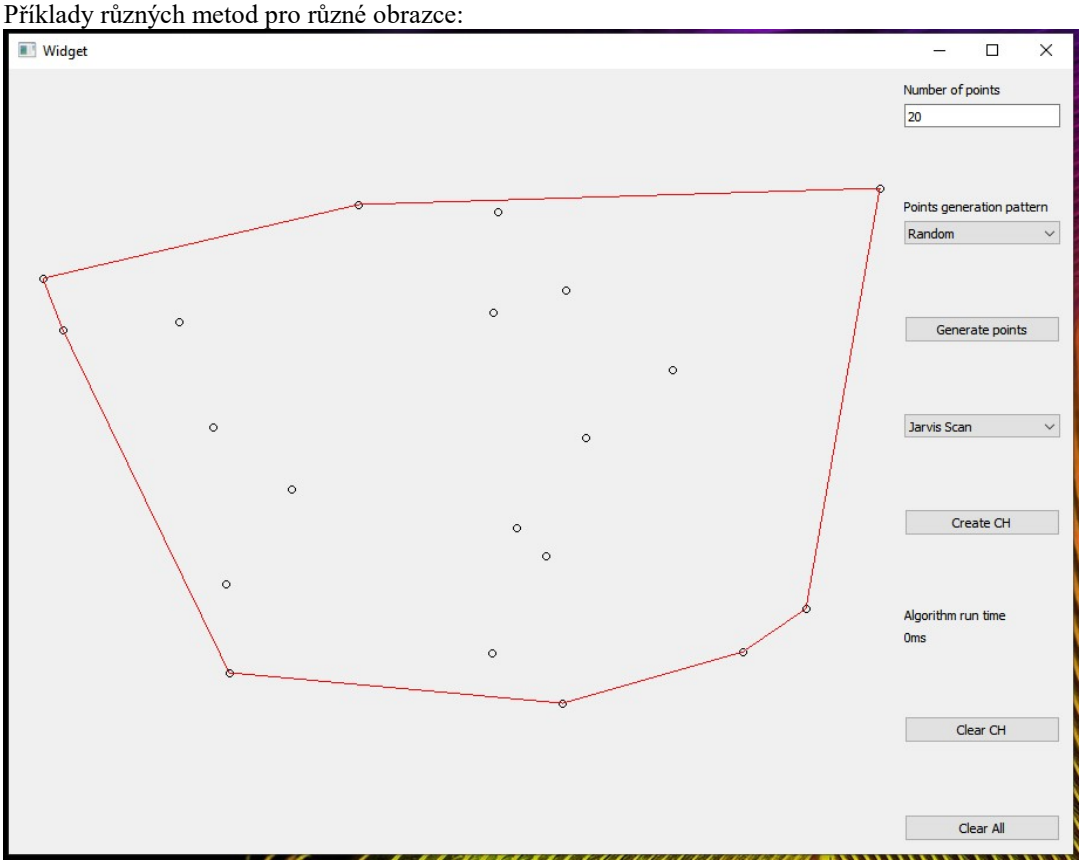
Obr. 16 Počet bodů pro čtverec musí být dělitelný 4 beze zbytku

Konvexní obálku generujeme následovně:

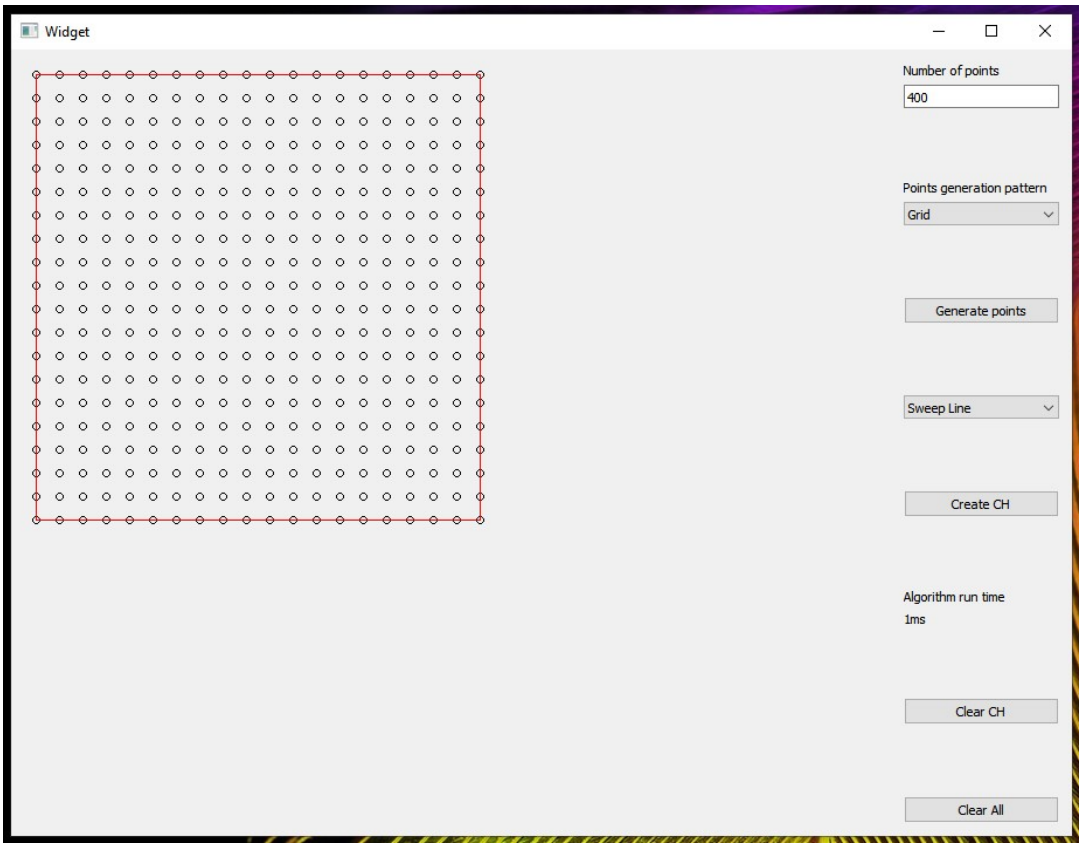


Obr. 17 Nedostatek bodů pro daný obrazec

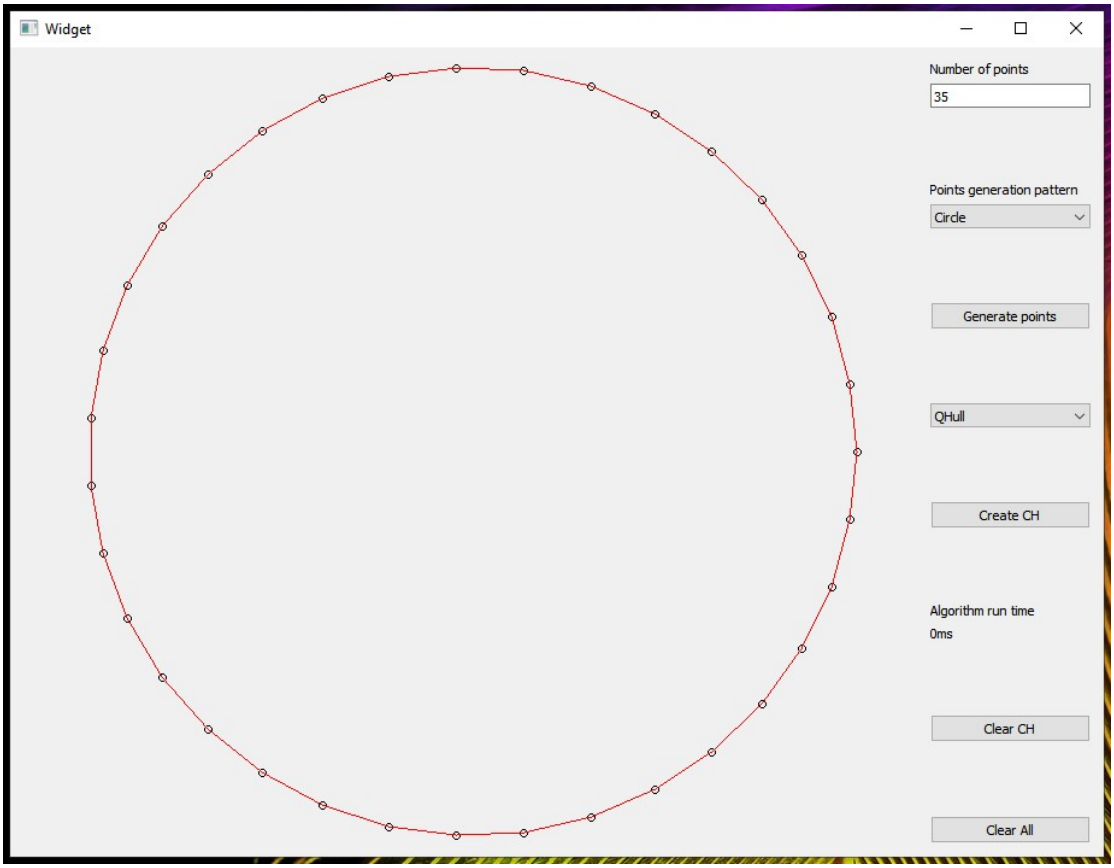
Příklady různých metod pro různé obrazce:



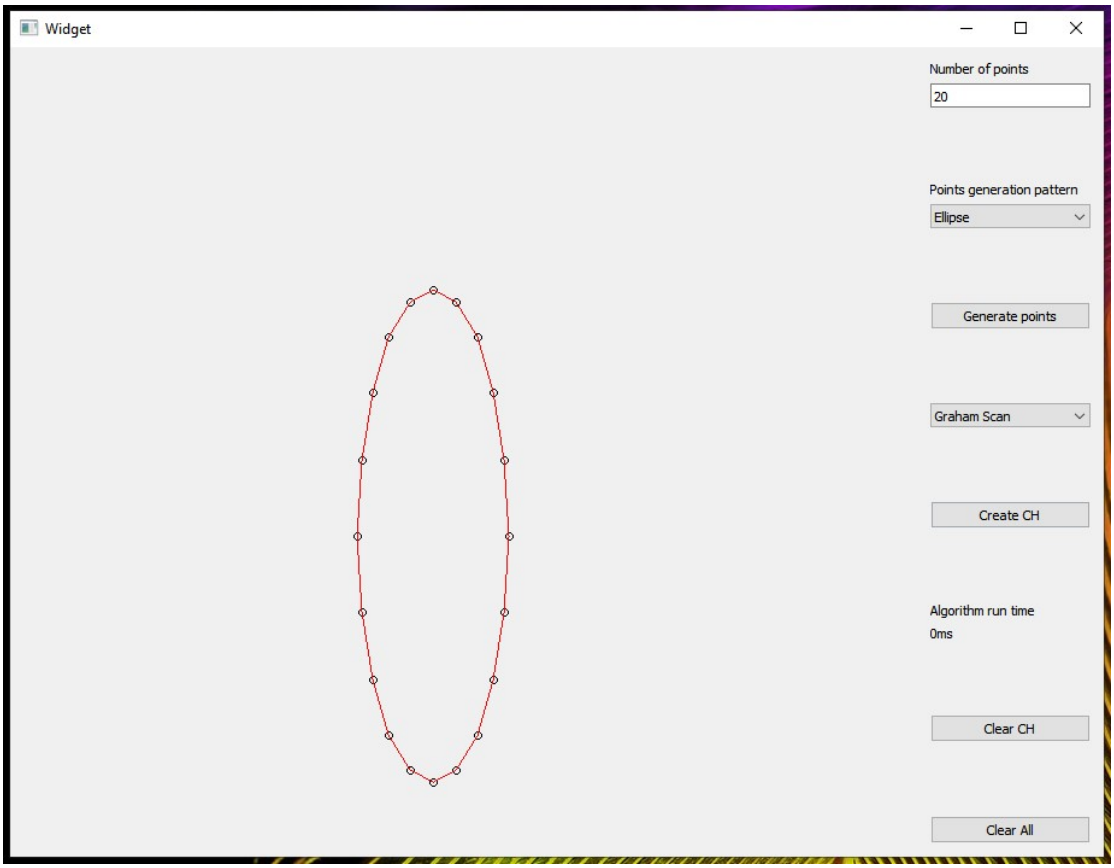
Obr. 18 Jarvis Scan, náhodné body



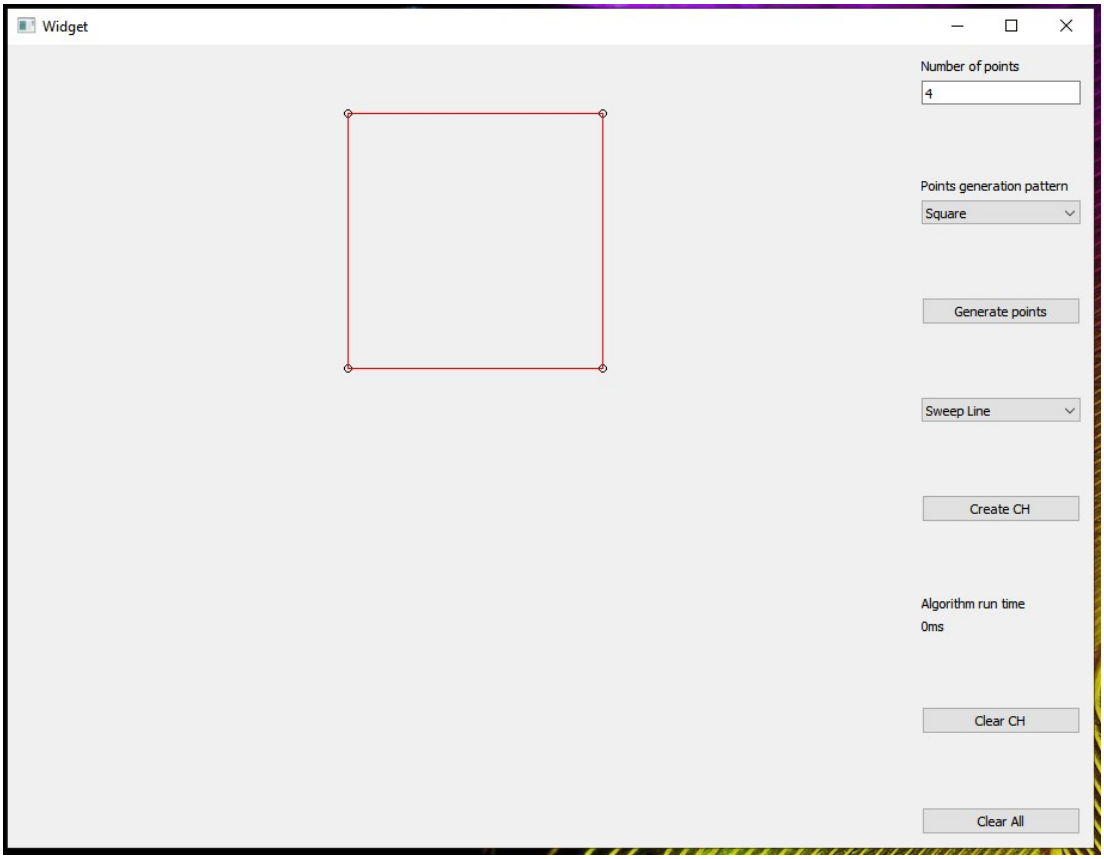
Obr. 19 Sweep line na mřížce



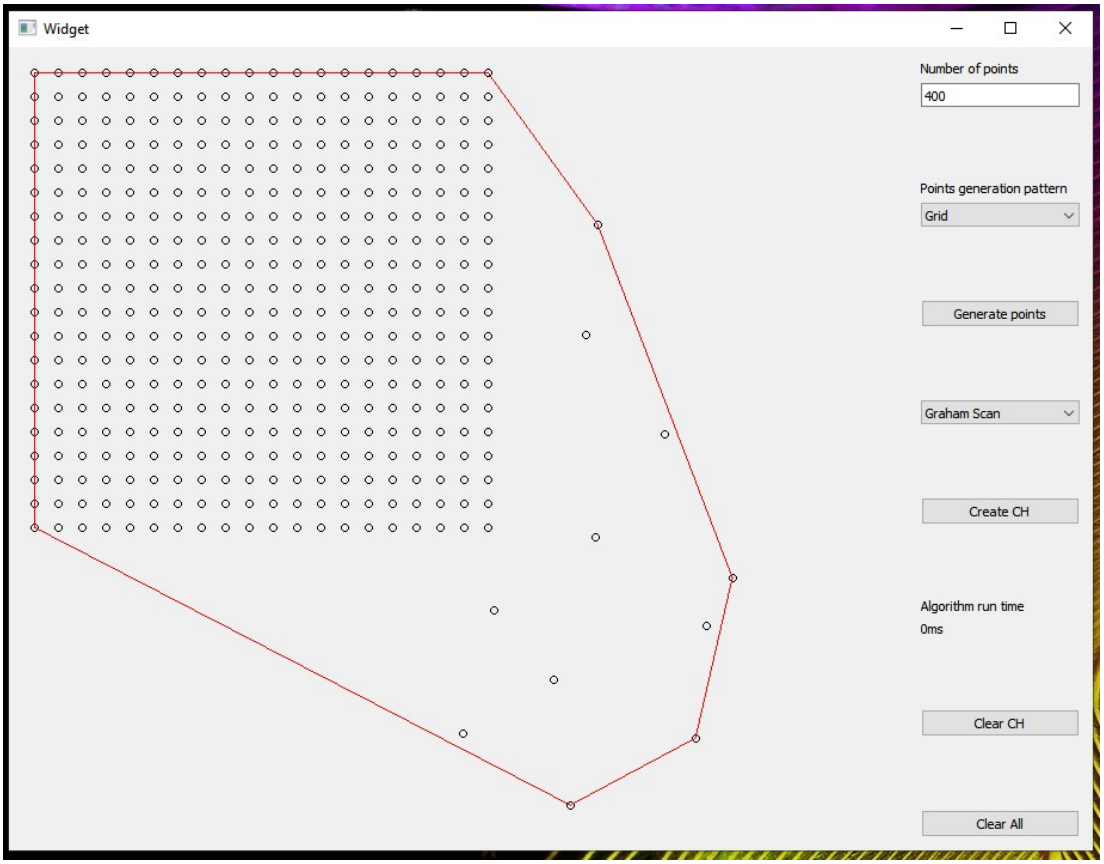
Obr. 20 QHull na kruhu



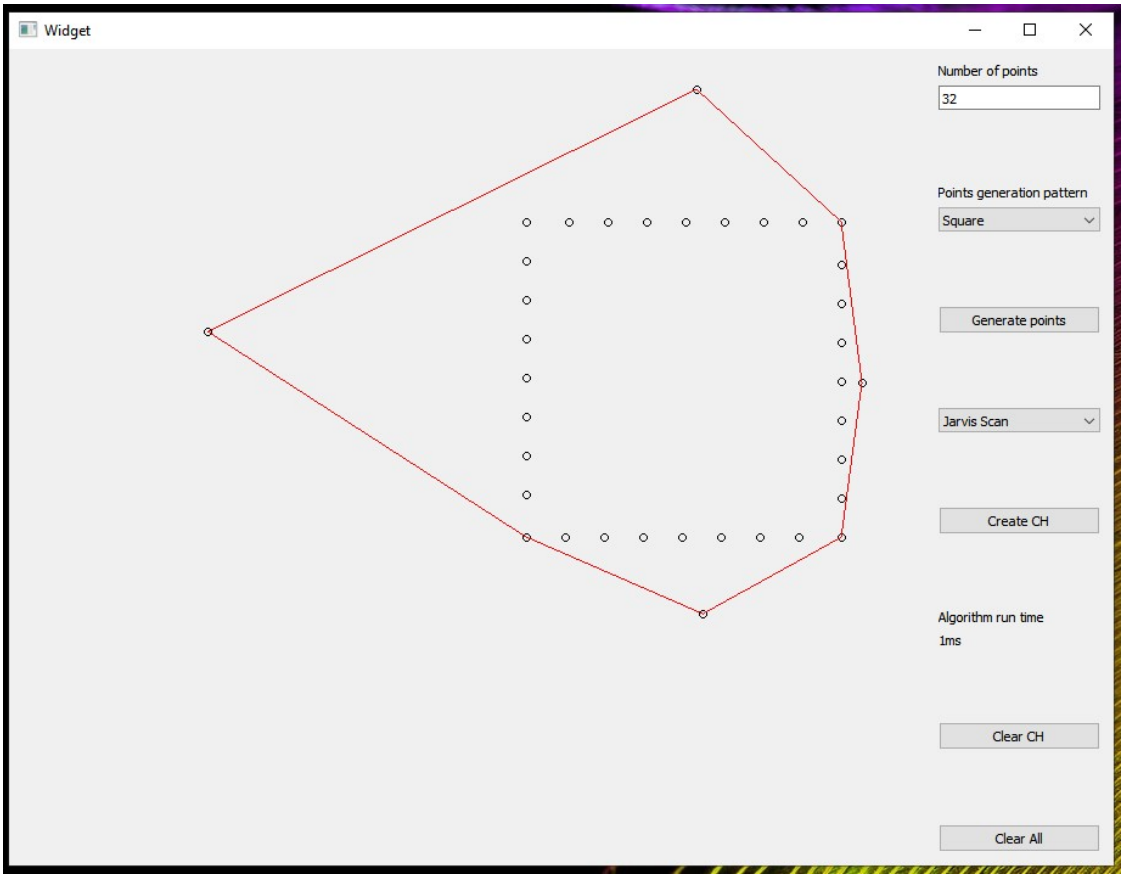
Obr. 21 Graham Scan na elipse



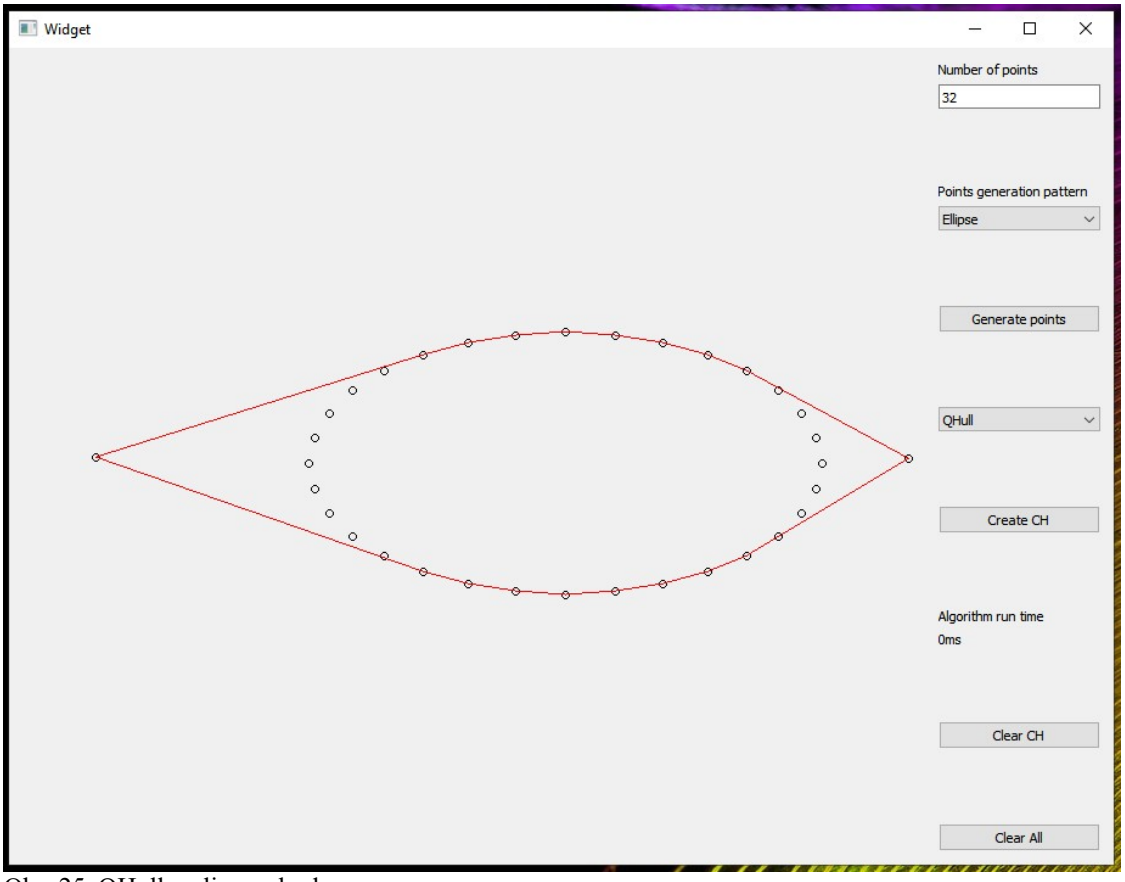
Obr. 22 Sweep line na čtverci



Obr. 23 Graham Scan - mřížka a body



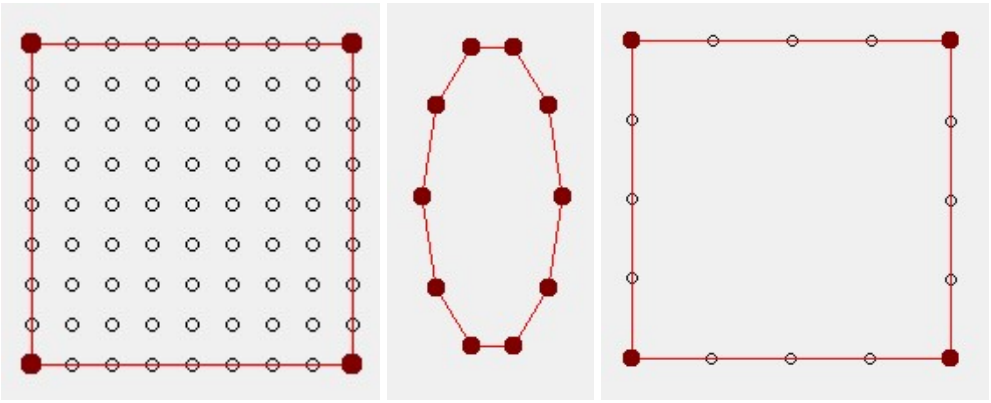
Obr. 24 Jarvis Scan - čtverec a body



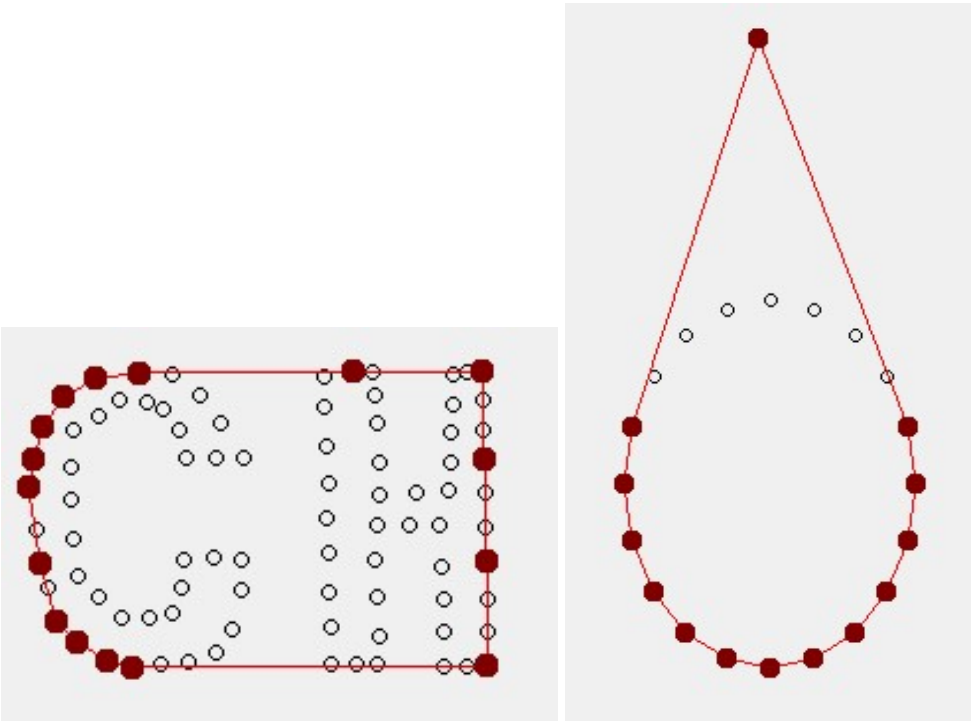
Obr. 25 QHull – elipsa a body

7.1 Vylepšení aplikace – Printscreen

Aplikace ve své druhé verzi byla doplněna o zvýraznění bodů zařazených do konvexní obálky. Realizace je znázorněna na následujících obrázcích, kde červené body jsou body zařazené do konvexní obálky a ostatní body patří pouze do množiny bodů.



Obr. 26 Ukázka zvýraznění bodů konvexní obálky



Obr. 27 Ukázka zvýraznění bodů konvexní obálky

8 Dokumentace

Dokumentace obsahuje mnoho tříd, které obsahují metody a jsou popsány níže. Navíc třída Algorithms obsahuje ještě strukturu Angle.

8.1 Třída Algorithms

Třída obsahuje jednu strukturu a množství metod. Algoritmy, které metody využívají již jsou popsány v předchozích kapitolách.

Struktura Angle
Struktura definuje datový tip využitý pro Graham Scan. Je složen z Qpoint point (bodu o souřadnicích x a y), double angle (úhlu tvořeném rovnoběžkou s osou x a daným bodem s vrcholem v pivotu) a double dist (vzdáleností daného bodu od pivotu).

double getAngle(QPoint &q1, QPoint &q2, QPoint &q3, QPoint &q4)
Tato funkce vrací hodnotu úhlu, který svírají dvě hany polygonu. Každá hrana je reprezentovaná dvěma body. Pro výpočet bylo využito skalárního součinu.

int getPointLinePosition(QPoint &q, QPoint &p1, QPoint &p2)
Tato funkce určuje polohu bodu q vůči přímce zadané dvěma body p1 a p2. Přímka reprezentuje jednu hranu polygonu.

Ve funkci jsou ošetřené singulární případy, kdy bod leží na hraně a na vrcholu polygonu. Pomocná funkce distancePoints slouží k zjištění vzdálenosti mezi trojicí bodů q, p1 a p2. Pokud absolutní hodnota rozdílu vzdálenosti mezi body p1, p2 a součtu vzdáleností mezi body p1, q a p2, q je menší než stanovená hodnota tolerance, pak se bod nachází blízko hrany a je vrácena hodnota 2. Podobně se kontroluje totožnost bodu q a jednoho z bodů p1 nebo p2. V případě, že jsou body totožné v rámci hodnoty tolerance, se vrací hodnota 2. Tolerance byla nastavena na hodnotu 0.4, aby se uživatel dokázal jednoduše dostat do singulárních situací.

Dále se z vektorů vypočte determinant. Jestliže je hodnota determinantu větší než 0, bod se nachází v levé polorovině vzhledem ke zkoumané přímce a funkce vrací hodnotu 1. Pokud je hodnota determinantu menší než 0, pak dojde k opačnému případu a bod q leží v pravé polorovině. Pokud nenastane žádný z výše uvedených případů, funkce vrátí hodnotu 1, což bude znamenat, že se někde stala chyba.

double getPointLineDist(QPoint &a, QPoint &p1, QPoint &p2)
Metoda, která vrací vzdálenost bodu od přímky zadané dvěma body.

double distancePoints(QPoint &p1, QPoint &p2)
Toto je pomocná funkce, která počítá vzdálenost mezi dvěma body pomocí Pythagorovy věty.

QPolygon jarvis(std::vector<QPoint> &points)
Metoda, pro výpočet konvexní obálky nad množinou bodů s využitím algoritmu Jarvis Scan. Metoda vrací konvexní obálku.

QPolygon qhull(std::vector<QPoint> &points)
Metoda, pro výpočet konvexní obálky nad množinou bodů s využitím algoritmu Quick Hull. Metoda vrací konvexní obálku. Jde o globální proceduru algoritmu, metoda volá i proceduru lokální ve funkci qh.

void qh(int s, int e, std::vector<QPoint> &points, QPolygon &ch)
Metoda, která tvoří lokální rekurzivní proceduru algoritmu Quick Hull (metoda QHull). Přidávají se v ní body do konvexní obálky, ale nic nevrací.

QPolygon graham(std::vector<QPoint> &points)
Metoda, pro výpočet konvexní obálky nad množinou bodů s využitím algoritmu Graham Scan. Metoda vrací konvexní obálku.

QPolygon sweepLine(std::vector<QPoint> &points)
Metoda, pro výpočet konvexní obálky nad množinou bodů s využitím algoritmu Sweep Line. Metoda vrací konvexní obálku.

QPolygon fixPolygon(QPolygon &ch)
Metoda, určená k závěrečné kontrole a úpravě konvexní obálky tak, aby šlo o striktně konvexní obálku (rušení bodů na přímce, rušení duplicit).

8.2 Třída Draw

Jde o třídu která dědí od třídy Widget a její metody slouží k vykreslení požadovaného výstupu.

void *mousePressEvent* (QMouseEvent *e)
Tato metoda nastavuje pozici kurzoru přiřazením souřadnic *x* a *y*. Slouží pro grafický vstup bodů.

void *paintEvent*(QPaintEvent *e)
Metoda, použitá pro vykreslení bodů. Metoda též vykresluje konvexní obálku.

QPoint getPoints()
Metoda vracející body třídy Draw.

QPolygon getCH()
Metoda, která vrací privátní konvexní obálku třídy Draw. Slouží k předávání proměnných jiným objektům.

void setCH(QPolygon &ch_)
Metoda, pro nastavení konvexní obálky.

void generatePoints(int num_points, int w, int h, int type);
Tato metoda slouží k automatickému generování bodů různých tvarů. A to v nastaveném množství (num_points), v daných rozměrech plátna (w, h) a dle zvoleného tvaru (type).

8.3 Třída sortByAngle

Třída, sloužící k setřídění proměnných typu Angle definovaných ve struktuře. Třídění probíhá podle velikosti úhlu a pokud je stejný, rozhodneme podle vzdáleností.

8.4 Třída sortByX

Třída, která je metodami volána k setřídění bodů podle souřadnice x. Pokud je souřadnice shodná, rozhodne se dle souřadnice y.

8.5 Třída sortByY

Třída, která je metodami volána k setřídění bodů podle souřadnice y. Pokud je souřadnice shodná, rozhodne se dle souřadnice x.

8.6 Třída uniquePoints

Třída, která se používá k setřídění bodů dle jejich jedinečnosti. Tedy unikátní body budou na začátku seznamu a opakující se na jeho konci.

8.7 Třída Widget

void on_pushButton_clicked()
Podle vybrané varianty v combo boxu se po stisknutí tlačítka Create CH volá daná metoda, která realizuje výpočet a následnou tvorbu konvexní obálky prostřednictvím předání třídě Draw. Také je v této metodě realizováno měření doby běhu vybraného algoritmu.

void on_pushButton_2_clicked()
Tlačítko Clear CH slouží k vyčištění kreslícího okna od kresby konvexní obálky.

void on_generatePoints_clicked()
Kliknutím na toto tlačítko je zavolána metoda generatePoints třídy Draw, která vygeneruje body dle zadaných parametrů.

void on_pushButton_3_clicked()
Tlačítko Clear All slouží k vyčištění kreslícího okna od kresby konvexní obálky i od kresby bodů.

9 Závěr

Byla vytvořena aplikace umožňující vygenerovat množinu bodů a poté nad ní pomocí zvoleného algoritmu vytvořit konvexní obálku.

Poznámka k realizaci algoritmu Graham scan: do cyklu v bodě 6 v kapitole 3.4.1 se ukázalo jako nutné začlenit while cyklus, protože ve chvíli, kdy bod vyřadíme z konvexní obálky, tak se na j-té pořadí posouvá bod původně na pozici j+1. Pokud bychom algoritmus zapsali standardně (podle nápovědy: for (int i = 2; i < n; i++)), tak se vždy posuneme na další bod a tím bychom přeskočili body, které se takto posunuly. Případně musíme i++ posunout pouze do části, kde zařazujeme bod do konvexní obálky.

10 Přílohy

Elektonicky:

Test.XLSX – Výsledky měření doby běhu algoritmů a příslušné grafy
Soubory aplikace

11 Seznam literatury

[1] BAYER, Tomáš. *Konvexní obálka množiny bodů.: Graham Scan. Jarvis Scan. Quick Hull. Inkrementální metoda. Divide and Conquer. Rotating Calipers.* [online]. [cit. 2020-11-16]. Dostupné z: https://web.natur.cuni.cz/~bayertom/images/courses/Adk/adk4.pdf?fbclid=IwAR2pwyGS7l0ap9-ofYH4qLu_opyUgO5-YRrcPGOKwsP3Ht9dlWiFuwuAeoU

[2] *ArcGIS Runtime SDK for Java* [online]. [cit. 2020-11-16]. Dostupné z: https://developers.arcgis.com/java/latest/java/sample-code/convex-hull/?fbclid=IwAR285NqSrjYoJOTxERmYXh9QRGV5OmgWqawHQ-RaDTxjufCyp1aCgoaWfzI

[3] *Quicker than Quickhull* [online]. [cit. 2020-11-16]. Dostupné z: https://link.springer.com/article/10.1007/s10013-014-0067-1/figures/2?fbclid=IwAR1oYPkxyXm_MMp5b6F13Bh4RYvWuDvaDCmbL8fllCCEfrdAwUVCgVy-ZWc

V Praze dne 4. 12. 2020 Usik, Vaňková