Algoritmy v digitální kartografii

Digitální model terénu a jeho analýzy

Zimní semestr 2018/2019

Tereza Kulovaná
Markéta Pecenová

Obsah

1 Zadání

Zadání úlohy bylo staženo ze stránek předmětu 155ADKG.

Vstup: množina $P = \{p_1, ..., p_n\}$, $p_i = \{x_i, y_i, z_i\}$.

Výstup: polyedrický DMT nad množinou P *představovaný vrstevnicemi doplněný vizualizací sklonu trojúhelníků a jejich expozicí.*

Metodou inkrementální konstrukce vytvořte nad množinou P vstupních bodů 2D Delaunay triangulaci. Jako vstupní data použijte existující geodetická data (alespoň 300 bodů) popř. navrhněte algoritmus pro generování syntetických vstupních dat představujících významné terénní tvary (kupa, údolí, spočinek, hřbet, ...).

Vstupní množiny bodů včetně níže uvedených výstupů vhodně vizualizujte. Grafické rozhraní realizujte s využitím frameworku QT. Dynamické datové struktury implementujte s využitím STL.

Nad takto vzniklou triangulací vygenerujte polyedrický digitální model terénu. Dále proveďte tyto analýzy:

- S využitím lineární interpolace vygenerujte vrstevnice se *zadaným krokem* a v *zadaném intervalu*, proveďte jejich vizualizaci s rozlišením zvýrazněných vrstevnic.
- Analyzujte sklon digitálního modelu terénu, jednotlivé trojúhelníky vizualizujte v závislosti na jejich sklonu.
- Analyzujte expozici digitálního modelu terénu, jednotlivé trojúhelníky vizualizujte v závislosti na jejich expozici ke světové straně.

Zhodnoťte výsledný digitální model terénu z kartografického hlediska, zamyslete se nad slabinami algoritmu založeného na 2D Delaunay triangulaci. Ve kterých situacích (různé terénní tvary) nebude dávat vhodné výsledky? Tyto situace graficky znázorněte.

Zhodnocení činnosti algoritmu včetně ukázek proveďte alespoň na tři strany formátu A4.

Hodnocení:

Krok	**Hodnocení**
Delaunay triangulace, polyedrický model terénu.	10b
Konstrukce vrstevnic, analýza sklonu a expozice.	10b
Triangulace nekonvexní oblasti zadané polygonem.	*+5b*
Výběr barevných stupnic při vizualizaci sklonu a expozice.	*+3b*
Automatický popis vrstevnic.	*+3b*
Automatický popis vrstevnic respektující kartografické zásady (orientace, vhodné rozložení).	*+10b*
Algoritmus pro automatické generování terénních tvarů (kupa, údolí, spočinek, hřbet, ...).	*+10b*
3D vizualizace terénu s využitím promítání.	*+10b*
Barevná hypsometrie.	*+5b*
Max celkem:	**65b**

Čas zpracování: 3 týdny

V rámci této úlohy byly implementovány bonusové úlohy č.

2 Popis a rozbor problému

Úloha **Digitální model terénu a jeho analýzy** se zabývá vytvořením aplikace, která Delaunayho triangulací vytvoří nad vstupní množinou bodů P digitální model terénu. Dále pro model lineární interpolací vypočte vrstevnice a sklon a expozici trojúhelníků ke světovým stranám. Vše je v aplikaci graficky zobrazeno.

Své využití konvexní obálky nalézají v mnoha oborech. V kartografii se hojně využívají při detekci tvarů a natočení budov pro tvorbu minimálních ohraničujících obdélníků. Dále jsou vhodné pro analýzu tvarů či shluků. Konvexní obálky lze sestrojit v libovolném $\mathbb{R}^n$ prostoru, avšak pro účely této úlohy byl uvažován pouze prostor $\mathbb{R}^2$. V rámci úlohy se testuje výpočetní rychlost použitých algoritmů při konstrukci obálek nad danými vstupními množinami bodů.

Vzniklá aplikace k tvorbě konvexní obálek využívá tří výpočetních algoritmů: *Jarvis Scan*, *Quick Hull* a *Sweep Line*.

3 Algoritmy

Tato kapitola se zabývá popisem algoritmů, které byly v aplikaci implementovány.

3.1 Jarvis Scan

Prvním zvoleným algoritmem je *Jarvis Scan*. Způsob, jakým vytváří konvexní obálku, nápadně připomíná balení dárků (proto je též občas nazýván jako *Gift Wrapping Algorithm*). Mezi nevýhody tohoto algoritmu patří nutnost předzpracování dat a nalezení tzv. pivotu. Algoritmus dále není vhodný pro velké množiny bodů a ve vstupní množině S nesmí být žádné tři body kolineární. Časová náročnost algoritmu je až $O(n^2)$, jeho výhodou však je velmi snadná implementace.

Mějme množinu bodů S a pivota $q \in S$, jehož souřadnice Y je minimální ze všech bodů, a přidejme ho do konvexní obálky H. Následně do H přidejme takový bod, který s posledními dvěma body přidanými do konvexní obálky svírá maximální úhel. Na začátku výpočtu je nutno inicializovat pomocný bod s, jehož souřadnice X je minimální a Y shodná s pivotem q, který zajistí dostatečný počet bodů pro výpočet prvního úhlu. Algoritmus končí ve chvíli, kdy nově přidaným bodem do konvexní obálky H je opět pivot q.

Zjednodušený zápis algoritmu lze zapsat způsobem uvedeným níže:

1. Nalezení pivota q: $q = \min(y)$
2. Inicializace pomocného bodu s: $s = [\min(x), \min(y)]$
3. Proveď: $q \in H$
4. Inicializace: $p_{j-1} = s$, $p_j = q$
5. opakuj kroky I–III, dokud $p_{j+1} \neq q$:

I. Najdi p_{j+1}: $\sphericalangle p_{j-1}, p_j, p_{j+1} = \max$

II. Proveď: $p_{j+1} \in H$

III. Přeindexuj: $p_{j-1} = p_j$, $p_j = p_{j+1}$

3.2 Quick Hull

Druhý algoritmus použitý v aplikaci je *Quick Hull*, který k výpočtu konvexní obálky využívá strategii *Divide and Conquer*. Hlavní výhodou algoritmu je jeho rychlost, která není ovlivňována velkým počtem rekurzivních kroků, jak tomu bývá u jiných algoritmů. Časová náročnost výpočtu bývá v nejhorším případě $O(n^2)$ a nastává tehdy, pokud všechny body množiny S náleží konvexní obálce H.

Mějme body q_1, resp. q_3, jejichž souřadnice X je minimální, resp. maximální ze všech bodů z množiny S. Veďme těmito body pomyslnou přímku, která prostor rozdělí na horní (S_U) a dolní (S_L) polorovinu. Zbylé body množiny S roztřídíme do daných polorovin podle jejich pozice od přímky. V polorovině následně hledáme bod, který je od dané přímky nejvzdálenější, přidáme ho do konvexní obálky dané poloroviny a přímkami spojíme bod s krajními body přímky předchozí. Proces opakujeme, dokud od nově vzniklých přímek již neexistují vhodné body. Na závěr do konvexní obálky H přidáme bod q_3, body konvexní obálky H_U z poloroviny S_U, bod q_1 a nakonec body konvexní obálky H_L poloroviny S_L. Do konvexní obálky H je důležité přidávat body v tomto pořadí, jinak by došlo k nesprávnému vykreslení konvexní obálky H.

Algoritmus *Quick Hull* se skládá z globální a lokální procedury. Globální část zahrnuje rozdělení množiny na dvě poloroviny a spojení již nalezených bodů konvexních obálek polorovin do jediné H. V lokální části se rekurzivně volá metoda, která hledá nejvzdálenější body od přímky v dané polorovině a přidává je do konvexní obálky dané poloroviny H_i.

Globální procedura:

1. Inicializace: $H = \emptyset$, $S_U = \emptyset$, $S_L = \emptyset$
2. Nalezení $q_1 = \min(x)$, $q_3 = \max(x)$
3. Proveď: $q_1 \in S_U$, $q_3 \in S_U$, $q_1 \in S_L$, $q_3 \in S_L$
4. Postupně pro všechna $p_i \in S$:

   Podmínka (p_i je vlevo od q_1, q_3) $\rightarrow S_U$

   Jinak $p_i \rightarrow S_L$
5. Proveď: $q_3 \in H$
6. Lokální procedura pro S_U
7. Proveď: $q_1 \in H$
8. Lokální procedura pro S_L

Lokální procedura nad polorovinou S_i:

I. Pro všechny $p_i \in S_i$ kromě bodů přímky:

Podmínka (p_i vpravo) $\rightarrow$ vzdálenost d_i

Podmínka ($d_i > d_{max}$) $\rightarrow d_{max} = d_i$, $p_{max} = p_i$

II. Podmínka (bod p_{max} $\exists$)

opakuj krok I. nad první nově vzniklou přímku

$p_{max} \in H_i$

opakuj krok I. nad druhou nově vzniklou přímku

3.3 Sweep Line

Algoritmus *Sweep Line* neboli *Metoda zametací přímky* je dalším z algoritmů, které byly pro vytváření konvexních obálek implementovány. Jeho princip je založen na imaginární přímce, která se postupně přesouvá zleva doprava nad všemi body množiny S. Body, které „přejede“, přidá do dočasné konvexní obálky $\bar{\text{H}}$, která je následně upravena, aby byla konvexní. Pro tento algoritmus je opět nutné předzpracování vstupních dat (seřadit body $\in S$ podle souřadnice X) s náročností $O(n.\log(n))$. Další nevýhodou je citlivost algoritmu na singularity, konkrétně na duplicitní body. Ty je vhodné během předzpracování odstranit.

Algoritmus je postaven na znalosti pozice již vyhodnocených bodů vůči nově přidávanému bodu ukládáním jejich indexů do proměnných p (předchůdce) a n (následník). Zároveň je pro správné fungování algoritmu nutné dodržovat CCW orientaci (proti směru hodinových ručiček). Algoritmus má celkem tři fáze: iniciální a dvě iterativní.

V první fázi seřadíme body p_i z množiny S vzestupně podle souřadnice X. Následně z prvních dvou bodů vytvoříme přímku, jejíž koncové body umístíme do konvexní obálky H a indexy bodů umístíme do n a p. V první iterativní fázi vyhodnotíme, zda další přidávaný bod leží v horní či dolní polorovině v závislosti na jeho souřadnici Y vzhledem k předchozímu bodu, a opět obousměrně vyhodnotíme indexy n a p. Ve druhé iterativní fázi opravujeme dočasnou konvexní obálku $\bar{\text{H}}$ na konvexní H vložením horních a dolních tečen a vynecháním nekonvexních vrcholů.

Zjednodušený zápis algoritmu:

1. Seřazení p_i podle souřadnice X
2. Pro body p_0, p_1 proveď:

   $n[0] = 1$, $n[1] = 0$

   $p[0] = 1$, $p[1] = 0$
3. Pro všechna $p_i \in S$, $i > 1$ proveď:

   Podmínka ($y_i > y_{i-1}$) $\rightarrow S_U$: p[i] = i-1, n[i] = n[i-1]

   Jinak $\rightarrow S_L$: n[i] = i-1 p[i] = p[i-1]

Oprava indexů: $n[p[\text{i}]] = \text{i}$, $p[n[\text{i}]] = \text{i}$

Dokud $n[n[\text{i}]]$ je vpravo od přímky (i, $n[\text{i}]$):

$p[n[n[\text{i}]]] = \text{i}$, $n[\text{i}] = n[n[\text{i}]]$

Dokud $p[p[\text{i}]]$ je vlevo od přímky (i, $p[\text{i}]$):

$n[p[p[\text{i}]]] = \text{i}$, $p[\text{i}] = p[p[\text{i}]]$

4 Problematické situace

V algoritmu *Sweep Line* bylo nutné ošetřit singularitu, která způsobovala generování nekonvexní obálky. Konkrétně bylo nutné odstranit duplicitní body. V opravené verzi aplikace je situace ošetřena setříděním bodů podle souřadnice X a porovnáním vzdáleností mezi dvěma po sobě jdoucími body. Je-li vzdálenosti menší než stanovená mez ϵ, body jsou považovány za duplicitní a bod s větší souřadnicí X je ze vstupní množiny odstraněn.

Podmínka ($d_{p_i,p_j} < \epsilon$) $\rightarrow$ bod p_j nezahrnut (viz. Obrázek 2)

Algoritmus *Jarvis Scan* generuje špatné výsledky, není-li ošetřeno, že žádné tři body nejsou spolu kolineární. Tento problém se zejména projevoval při generaci setu *Grid*, který jich obsahuje spoustu. Singularita byla ošetřena vypočtením úhlu $\sphericalangle p_{jj}, p_j, p_i$, a pokud byl menší, než stanovená mez ϵ, byl pro další výpočty vybrán bližší z kolineárních bodů.

- Podmínka ($\sphericalangle p_{jj}, p_j, p_i) < \epsilon$

  Podmínka ($d_{p_j,p_i} < d_{min}$) $\rightarrow$ vyber bod p_i, $d_{min} = d_{p_j,p_i}$

Dalším problémem bylo, jak zajistit, aby byl generovaný rastr pravidelný. To bylo ošetřeno zaokrouhlením počtu vstupních bodů směrem nahoru tak, aby po odmocnění vznikl pravidelný rastr.

počet bodů v řádce/sloupci = $roundUp(\sqrt{num\_of\_points})$

5 Vstupní data

Aplikace si na základě ručního zadání vstupních parametrů uživatelem sama vygeneruje potřebná vstupní data. Z rozbalovací nabídky **Shape of set** uživatel volí prostorové uspořádání generované množiny bodů. Na výběr jsou možnosti *Random set*, *Grid* a *Circle*. V kolonce **Number of points** uživatel volí, kolik bodů bude generováno. Lze tak učinit buď přímým zadáním počtu bodů, nebo zvýšením/snížením počtu bodů o 1000 šipkami na boku. Aplikace omezuje minimální a maximální počet generovaných bodů na interval <1, 1000000>. Množina bodů se vygeneruje stisknutím tlačítka *Generate set*.

Uživatel má dále možnost volit, jaký výpočetní algoritmus bude použit pro tvorbu konvexní obálky. Rozbalovací nabídka **Method** nabízí celkem tři výpočetní algoritmy: *Jarvis Scan*, *Quick Hull* a *Sweep Line*. Konvexní obálka je generována stisknutím tlačítka *Create CH*. Pokud před spuštěním procesu nebyla vygenerována žádná vstupní množina bodů, uživatel je upozorněn chybovou hláškou.

6 Výstupní data

Vygenerovaná množina bodů a její konvexní obálka je vykreslena v grafickém okně aplikace. Aplikace dále vypisuje časy [ms], jak dlouho trvalo generování množiny a jak dlouho nad danou množinou běžel výpočetní algoritmus.

V rámci testování byly nad množinami bodů *Random set*, *Grid* a *Circle* postupně spuštěny všechny tři algoritmy. Pro každou množinu a algoritmus byla pro daný počet bodů $n = \{1000, 5000, 10000, 25000, 50000, 75000, 100000, 200000, 500000, 1000000\}$ aplikace spuštěna 10x, aby bylo získáno dostatečné množství testovacích dat. Z každé testované množiny bylo tedy získáno celkem 300 testovacích dat. Data byla ukládána do textového souboru a následně zpracována v *Excelu*.

7 Aplikace

V následují kapitole je představen vizuální vzhled vytvořené aplikace tak, jak ji vidí prostý uživatel.

8 Dokumentace

Tato kapitola obsahuje dokumentaci k jednotlivým třídám.

8.1 !Algorithms

Třída *Algorithms* obsahuje metody pro které nad vstupní množinou bodů vytváří konvexní obálky. Dále obsahuje pomocné metody k výpočtu úhlu mezi dvěma přímkami, metody k určování vztahu bodu a přímky a metodu pro vytvoření striktně konvexní obálky.

delaunayTriangulation

Metoda **delaunayTriangulation** vytváří nad množinou bodů Delaunayho triangulaci. Na vstupu je vektor bodů typu `QPoint3D`, metoda vrací uspořádaný vektor hran `Edge`, které tvoří jednotlivé trojúhelníky.

Input:

- *vector* <`QPoint3D`> *points*

Output:

- *vector* <`Edge`>

!createContours

Metoda **createContours** vytváří nad vstupní množinou hran vrstevnice na základě zadané minimální a maximální výšky a kroku, po kterém se vrstevnice budou vykreslovat. Metoda vrací vektor hran, které představují vrstevnice.

Input:

- *vector* <`Edge`> *dt*
- `double` z_min → minimální výška
- `double` z_max → maximální výška
- `double` dz → krok vrstevnic

Output:

- *vector* <`Edge`>

!getSlope

Metoda **getSlope** počítá sklon trojúhelníku, který je tvořen třemi body. Návratová hodnota typu `double` nabývá hodnot <0°;180°???> a vrací sklon trojúhelníku.

Input:

- `QPoint3D` p_1

- `QPoint3D` p_2
- `QPoint3D` p_3

!getAspect

Metoda **getSlope** počítá orientaci trojúhelníku, který je tvořen třemi body, ke světovým stranám. Návratová hodnota typu `double` vrací orientaci trojúhelníku ve stupních. Hodnota 0° je umístěna na xxx, orientace je pravotočiná/levotočivá.

Input:

- `QPoint3D` p_1
- `QPoint3D` p_2
- `QPoint3D` p_3

analyzeDTM

Metoda **analyzeDTM** vytváří z vektoru hran trojúhelníky a počítá pro ně sklon a orientaci. Vypočtené hodnoty ukládá do datového typu `Triangle`. Návratová hodnota metody je vektor trojúhelníků typu `Triangle`.

Input:

- *vector* <`Edge`> *dt*

Output:

- *vector* <`Triangle`>

getPointLinePosition

Metoda **getPointLinePosition** určuje polohu bodu q vzhledem k přímce tvořené dvěma body. Na vstupu jsou 3 body typu `QPoint3D`, návratová hodnota je nově definovaný typ `TPosition`.

Input:

- `QPoint3D` q
- `QPoint3D` a
- `QPoint3D` b

Output:

- `LEFT` → bod se nachází vlevo od přímky
- `RIGHT` → bod se nachází vpravo od přímky
- `ON` → bod se nachází na přímce

!getCircleRadius

Metoda **getCircleRadius** počítá poloměr kružnice, která je tvořena 3 body. Na vstupu jsou 3 body typu `QPoint3D`, návratová hodnota typu `double` vrací velikost poloměru kružnice.

Input:

- `QPoint3D` p_1
- `QPoint3D` p_2
- `QPoint3D` p_3
- !!!`QPoint3D` c

getDistance

Metoda **getDistance** počítá vzdálenost mezi dvěma body. Na vstupu jsou 2 body typu `QPoint3D`, návratová hodnota typu `double` vrací vzdálenost mezi dvěma body.

Input:

- `QPoint3D` p_1
- `QPoint3D` p_2

getNearestPoint

Metoda **getNearestPoint** slouží k nalezení nejbližšího bodu z množiny bodů vzhledem k danému bodu p. Na vstupu je daný bod p a vektor bodů typu `QPoint3D`. Návratová hodnota typu `int` vrací index nejbližšího bodu.

Input:

- `QPoint3D` p
- *vector* <`QPoint3D`> *points*

getDelaunayPoint

Metoda **getDelaunayPoint** slouží k nalezení třetího bodu trojúhelníku, který splňuje Delaunayho kritérium nejmenší opsané kružnice. Na vstupu jsou dva body typu `QPoint3D`, které představují orientovanou hranu, a vektor bodů typu `QPoint3D`. Návratová hodnota typu `int` vrací index hledaného bodu.

Input:

- `QPoint3D` $s \rightarrow$ počáteční bod hrany
- `QPoint3D` $e \rightarrow$ koncový bod hrany
- *vector* <`QPoint3D`> *points*

getContourPoint

Metoda **getContourPoint** počítá průsečík hrany trojúhelníku tvořené dvěma body typu `QPoint3D` s rovinou o dané výšce Z. Návratová hodnota je typu `QPoint3D`.

Input:

- `QPoint3D` p_1
- `QPoint3D` p_2
- `double` z

8.2 !Draw

Třída *Draw* obsahuje metody, které nahrávají a vykreslují vstupní množinu bodů. Dále zajišťuje vykreslení a smazání všech operací, kterou jsou nad množinou prováděny.

paintEvent

Metoda **paintEvent** vykresluje vstupní množinu bodů, Delaunayho triangulaci, vrstevnice a sklon a orientaci trojúhelníků.

clearDT

Metoda **clearDT** slouží k vymazání všech vykreslených dat.

getPoints

Metoda **getPoints** slouží k získání vektoru bodů z kreslící plochy. Metoda vrací vektor bodů typu `QPoint3D`.

getDT

Metoda **getPoints** slouží k získání vektoru hran z kreslící plochy. Metoda vrací vektor hran typu `Edge`.

setDT

Metoda **setDT** slouží k převedení Delaunayho triangulace do kreslícího okna.

setContours

Metoda **setContours** slouží k převedení vrstevnic do kreslícího okna.

setDTM

Metoda **setDTM** slouží k převedení digitálního modelů terénu do kreslícího okna.

8.3 Edge

Třída *Edge* slouží k manipulaci s orientovanými hranami. Definuje dva body typu `QPoint3D` jako počáteční a koncový bod hrany.

getS

Metoda **getS** slouží k získání počáteční body hrany.

getE

Metoda **getE** slouží k získání koncový body hrany.

switchOrientation

Metoda **switchOrientation** prohazuje orientaci hrany.

????operator

8.4 QPpoint3D

Třída **QPpoint3D** slouží k definování nového datového typu `QPoint3D`, který je odvozen od typu `QPointF` a který navíc obsahuje souřadnici Z.

getZ

Metoda **getZ** slouží k získání souřadnice Z daného bodu.

setZ

Metoda **setZ** slouží k nastavení souřadnice Z daného bodu.

8.5 SortByXAsc

Třída **SortByXAsc** má na vstupu dva body typu `QPoint3D`, návratová hodnota je typu `bool`. Metoda vrací bod s nižší souřadnicí X. Mají-li oba body shodnou souřadnici X, vrací bod s nižší souřadnicí Y.

Input:

- `QPoint3D` p_1
- `QPoint3D` p_2

Output:

- $0 \rightarrow$ bod p_2 má nižší x souřadnici
- $1 \rightarrow$ bod p_1 má nižší x souřadnici

8.6 Triangle

Třída **Triangle** slouží k definování nového datového typu `Triangle`, který v sobě uchovává informaci o třech bodech typu `QPoint3D`, které tvoří trojúhelník, a o sklonu a expozici trojúhelníku.

getPi

Metoda **getSlope** slouží k získání bodu P_i daného trojúhelníku.

getSlope

Metoda **getSlope** slouží k získání sklonu daného trojúhelníku.

getAspect

Metoda **getAspect** slouží k získání orientace daného trojúhelníku.

8.7 Widget

Metody třídy **Widget** slouží pro práci uživatele s aplikací. Metody na vstupu nemají žádné parametry a návratové hodnoty jsou typu `void`.

on_delaunay_button_clicked

Metoda **on_delaunay_button_clicked** nad vstupní množinou bodů zobrazí Delaunayho triangulaci.

on_clear_button_clicked

Metoda **on_clear_button_clicked** vrací aplikaci do výchozí polohy smazáním všeho, co bylo vykresleno.

on_contours_button_clicked

Metoda **on_contours_button_clicked** nad vygenerovanou trojúhelníkovou sítí z Delaunayho triangulace vykreslí vrstevnice.

on_dtm_button_clicked

Metoda **on_dtm_button_clicked** obarví trojúhelníky vygenerované Delaunayho triangulací v odstínech šedi podle hodnoty sklonu daného trojúhelníku.

!!!aspect

9 Závěr

V rámci úlohy *Konvexní obálky* byla vytvořena aplikace, která nad vstupní množinou bodů vytváří striktně konvexní obálky. V rámci testování, která trvalo dlouho do noci a použitým počítačům dala pořádně zabrat, byla shromážděna data průměrné doby výpočtu striktně konvexní obálky pro jednotlivé algoritmy. Z časových důvodů byly implementovány jen některé bonusové úlohy. Opravená verze aplikace lépe implementuje odstraňování duplicitních bodů v algoritmu *Sweep Line*, což výrazně přispělo ke snížení doby běhu algoritmu. V závislosti na tom byly upraveny hodnoty v tabulkách a grafy z nich vycházející.

Po nově provedeném opravném testování považujeme za nejvhodnější algoritmus pro výpočet konvexní obálky algoritmus *Sweep Line*, a to i přesto, že pro množinu *Grid* byl o něco málo rychlejší algoritmus *Quick Hull*. Jeho rychlost se projevila hlavně u množiny *Circle* (striktně konvexní obálku nad milionem bodů generoval průměrně pod desetinu sekundy). Pro množiny *Random* a *Grid* lze považovat algoritmus *Quick Hull* za srovnatelný s algoritmem *Sweep Line* co se týče výpočetní doby. Jeho slabina se však projevila u množiny *Circle*, jejíž prostorové uspořádání zaručuje, že všechny body náleží konvexní obálce. Tady *Quick Hull* trochu zaváhal a výpočtení doba se prodloužila. Jako nevyhovujícím pro tvorbu konvexních obálek byl shledán algoritmus *Jarvis Scan*. Výpočet obálky mu i na malých množinách trval o poznání déle než ostatním algoritmům, avšak překvapila nás rychlost, s jakou se vypořádal s kružnicí v porovnání s jinými množinami.

Závěrem by bylo vhodné podotknout, že data z testování nejsou 100% spolehlivá. Již v průběhu testování bylo zaznamenáno, že doba výpočtu algoritmu velmi závisí na výkonu použitého počítače (rozdíl v rychlostech byl až dvojnásobný) a také na tom, zda jsou v době testování na počítači spuštěny jiné aplikace (např. prohlížeč) nebo se provádí jiné úkony (např. psaní technické zprávy). To může být jednou z příčin vzniku odchylek a nepřesností, které se v datech občas vyskytují. Pro zachování přibližně konzistentních podmínek při testování byly použity dva notebooky s podobným výkonem.

Do budoucna by jistě šla rozšířit nabídka generovaných množin bodů a naprogramovat celková automatizace testování. Aktuální verze kódu pro testování obsahovala pouze cyklus na 10 opakování téhož výpočtu. Dále by mohl být naprogramován další výpočetní algoritmus, *Graham Scan*, na který již autorky neměly čas. Mezi pozitivní přínosy úlohy zajisté patří objevení způsobu hromadného exportu grafů z *Excelu* do formátu *.png.

10 Zdroje

1. *BAYER, Tomáš. 2D triangulace, DMT* [online][cit. 4. 12. 2018]. Dostupné z: https://web.natur.cuni.cz