

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE  
FAKULTA STAVEBNÍ

BAKALÁŘSKÁ PRÁCE

Praha 2015

Adam Laža

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE  
FAKULTA STAVEBNÍ  
STUDIJNÍ PROGRAM GEODEZIE A KARTOGRAFIE  
OBOR GEODEZIE, KARTOGRAFIE A GEOINFORMATIKA



BAKALÁŘSKÁ PRÁCE  
IMPLEMENTACE NÁSTROJE PRO INTERPOLACI  
METODOU PŘIROZENÉHO SOUSEDÁ DO GRASS GIS  
INTERPOLATION OF TOOL FOR A NATURAL NEIGHBOUR  
INTERPOLATION INTO GRASS GIS

Vedoucí práce: Ing. Martin Landa, Ph.D.

Katedra geomatiky

Praha 2015

Adam Laža



## ZADÁNÍ BAKALÁŘSKÉ PRÁCE

studijní program: Geodézie a kartografie

studijní obor: Geodézie, kartografie a geoinformatika

akademický rok: 2014/15

Jméno a příjmení studenta: Adam Laža

Zadávající katedra: K155

Vedoucí bakalářské práce: Ing. Martin Landa, Ph.D.

Název bakalářské práce: Implementace nástroje pro interpolaci metodou přirozeného souseda do systému GRASS GIS

Název bakalářské práce v anglickém jazyce: Implementation of a tool for natural neighbour interpolation in GRASS GIS

Rámcový obsah bakalářské práce: Cílem bakalářské práce je návrh a implementace modulu do open source projektu GRASS GIS. Tento modul bude nad vektorovými nebo rastrovými bodovými daty provádět interpolaci metodou přirozeného souseda. Součástí bakalářské práce bude i porovnání výstupů z navrženého nástroje pro GRASS GIS s jinými srovnatelnými GIS produkty.

Datum zadání bakalářské práce: 5.2.2015 Termín odevzdání: 15.5.2015

(vyplňte poslední den výuky  
příslušného semestru)

Pokud student neodevzdal bakalářskou práci v určeném termínu, tuto skutečnost předem písemně zdůvodnil a omluva byla děkanem uznána, stanoví děkan studentovi náhradní termín odevzdání bakalářské práce. Pokud se však student rádně neomluvil nebo omluva nebyla děkanem uznána, může si student zapsat bakalářskou práci podruhé. Studentovi, který při opakováném zápisu bakalářskou práci neodevzdal v určeném termínu a tuto skutečnost rádně neomluvil nebo omluva nebyla děkanem uznána, se ukončuje studium podle § 56 zákona o VŠ č. 111/1998. (SZŘ ČVUT čl. 21, odst. 4)

*Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.*

vedoucí bakalářské práce

vedoucí katedry

Zadání bakalářské práce převzal dne: 5.2.2015

student

Formulář nutno vyhotovit ve 3 výtiscích – 1x katedra, 1x student, 1x studijní odd. (zašle katedra)

Nejpozději do konce 2. týdne výuky v semestru odešle katedra 1 kopii zadání BP na studijní oddělení a provede zápis údajů týkajících se BP do databáze KOS.

BP zadává katedra nejpozději 1. týden semestru, v němž má student BP zapsanou.  
(Směrnice děkana pro realizaci studijních programů a SZZ na FSv ČVUT čl. 5, odst. 7)

## Abstract

Cílem této bakalářské práce je návrh a implementace nástroje pro interpolaci metodou přirozeného souseda pro GRASS GIS 7. Starší verze GRASS GIS 6 sice tuto metodu nabízí v rámci volitelně doinstalovatelných balíčků Add-Ons, ale jako modul napsaný v Bashu a s vnitřní závislostí na knihovně *nn-c*. Tato knihovna používá knihovnu *Triangle*, která nedovoluje zařazení tohoto nástroje do oficiální distribuce GRASS GISu.

V rámci bakalářské práce byl tento modul přepsán do jazyka Python tak, aby vyhovoval verzi 7. Práce se dále zabývá možnostmi budoucí implementace bez závislosti na knihovně *Triangle* tak, aby mohl být tento interpolační nástroj zařazen do oficiální distribuce GRASS GISu. Část textu této práce se dále zabývá porovnání rychlosti a kvality výstupu mezi GRASS GIS a Esri ArcGIS.

**Klíčová slova:** GIS, GRASS GIS, interpolace, přirozený soused, Delaunayho triangulace, Voronoiův diagram

## Abstract

The aim of this bachelor thesis is the design and the implementation of a tool for a natural neighbour interpolation in GRASS GIS 7. Previous version of GRASS GIS 6 contains such a tool as an Add-Ons package, but this module is written in Bash and with inner dependency on *nn-c* library. This library contains *Triangle* library which is not under the GNU GPL licence. Because of that it is not possible to add the interpolation tool into official GRASS GIS distribution.

**Keywords:** GIS, GRASS GIS, natural neighbour interpolation, Delaunay triangulation, Voronoi diagram

## **Prohlášení**

Prohlašuji, že bakalářskou práci na téma „Implementace nástroje pro interpolaci metodou přirozeného souseda do GRASS GIS“ jsem vypracoval samostatně. Použitou literaturu a podkladové materiály uvádím v seznamu zdrojů.

V Praze dne .....  
.....  
(podpis autora)

## **Poděkování**

Chtěl bych poděkovat mému vedoucímu bakalářské práce Ing. Martinovi Landovi, Ph.D. za cenné rady a čas, který mi věnoval při konzultacích. Dál bych chtěl poděkovat Ing. Tomášovi Bayerovi, Ph.D. za konzultace a za vhled, který mi do problematiky triangulací poskytl. Také bych rád poděkoval své rodině, přátelům a okolí za podporu po celou dobu studia.

# Obsah

<b>Úvod</b>	<b>10</b>
<b>I Teoretická část</b>	<b>12</b>
<b>1 GRASS GIS</b>	<b>13</b>
1.1 GRASS GIS Add-Ons . . . . .	13
<b>2 Delaunayova triangulace</b>	<b>14</b>
2.1 Triangulace obecně . . . . .	14
2.2 Validní a regulérní triangulace . . . . .	15
2.3 Optimální triangulace . . . . .	18
2.4 Voronoiův diagram . . . . .	21
2.5 Voroniovy diagramy a Delaunayova triangulace . . . . .	23
<b>3 Algoritmy pro tvorbu DT</b>	<b>24</b>
3.1 Lokální optimalizace . . . . .	25
3.2 Paprskovitý algoritmus . . . . .	26
3.3 Algoritmus inkrementálního vkládání . . . . .	27
3.4 Algoritmus inkrementální konstrukce (Step-by-Step) . . . . .	28
3.5 Algoritmus Rozděl a panuj . . . . .	30
<b>4 Datové struktury</b>	<b>31</b>
4.1 Jednoduchá trojúhelníková struktura . . . . .	32
4.2 Trojúhelníková struktura se sousedy . . . . .	32
4.3 Vertex-based struktura se sousedy . . . . .	33
4.4 Half-edge datová struktura . . . . .	34
4.5 Dart-based datová struktura . . . . .	36

<b>5 Interpolace metodou přirozeného souseda</b>	<b>38</b>
5.1 Lineární interpolace . . . . .	38
5.2 Sibsonova interpolace . . . . .	40
5.3 Farinova interpolace . . . . .	40
<b>II Praktická část</b>	<b>41</b>
<b>6 Postup řešení</b>	<b>42</b>
6.1 Bash . . . . .	42
6.1.1 v.surf.nnbathy . . . . .	42
6.2 Python . . . . .	43
6.2.1 v.surf.nnbathy.py . . . . .	43
6.3 OOP — Objektově orientované moduly . . . . .	49
6.3.1 v.surf.nnbathy . . . . .	49
6.3.2 r.surf.nnbathy . . . . .	49
6.3.3 nnbathy . . . . .	49
<b>7 Možnosti budoucí implementace</b>	<b>51</b>
7.1 Knihovna Triangle . . . . .	52
7.2 Modul <i>v.delaunay</i> . . . . .	53
7.3 Převod mezi strukturami . . . . .	54
7.4 Knihovna CGAL . . . . .	55
7.4.1 v.surf.nn . . . . .	55
<b>Závěr</b>	<b>57</b>
<b>Seznam použitých zkratek</b>	<b>59</b>
<b>Použité zdroje</b>	<b>60</b>
<b>A Srovnání GRASS GIS a Esri ArcGIS</b>	<b>62</b>

<b>B Delaunay struktura</b>	<b>67</b>
<b>C Winged-edge struktura</b>	<b>69</b>
<b>D Ukázka zdrojových kódů</b>	<b>70</b>
D.1 nnbathy.py . . . . .	70
D.2 v.surf.nnbathy.py . . . . .	74
D.3 r.surf.nnbathy.py . . . . .	76
D.4 v.surf.nn . . . . .	76
<b>E Seznam tabulek a obrázků</b>	<b>80</b>
<b>F Obsah CD</b>	<b>82</b>

## Úvod

V reálném životě se většinou nesetkáváme s případem, kdy pro naši oblast zájmu, ať už se jedná o interval, plochu nebo prostor, máme dostatek bodových dat o daném jevu. Mnohem častěji máme k dispozici pouze soubor bodových dat, která jsou buď náhodně nebo uspořádaně rozmištěna v naší oblasti zájmu. Fyzické zhuštění takovéto sítě a sběr dalších dat může být časově či finančně náročné, příliš obtížné nebo v rámci možností metod zcela nereálné.

Obvykle ovšem potřebujeme znát hodnotu daného jevu i mimo měřené body, nejčastěji pro celé zájmové území. V takovéto chvíli je nutné použít nějaký interpolační nástroj, který vypočte přibližnou hodnotu v území mezi měřenými body. Jako příklad může být uvedeno vytvoření výškopisu či DMT pro území, kde máme k dispozici data o výšce v pravidelné mřížce nebo teplotní mapy na základě údajů z nepravidelně rozmištěných meteostanic.

Tato práce se zabývá interpolací přirozeného souseda a její implementací do GRASS GISu 7 (<http://grass.osgeo.org>). Toto téma bakalářské práce jsem si vybral z několika důvodů. Za prvé mě zajímají geografické informační systémy, zejména pak open-source projekty. Dle mého názoru je to právě cesta open-source software, kterou se budou v budoucnosti soukromé firmy, vývoj a věda ubírat, a je proto dobré s nimi získávat zkušenost už během studia. Za druhé práce na tomto tématu mi umožnila proniknout do GISů nejen z pohledu uživatele, ale přiučit se alespoň elementární základy jako vývojář. A za třetí interpolace metodou přirozeného souseda je nástroj, který poskytuje kvalitní výstupy. Jeho začlenění do standardní distribuce GRASS GISu by mohlo být hojně využito.

Až doposud byla interpolace metodou přirozeného souseda prováděna pomocí modulů *v.surf.nn bathy* (kapitola 6.1.1) a *r.surf.nn bathy* (kapitola 6.3.2) z rozšíření Add-Ons (viz 1.1), které jsou navíc napsané jako shellový skript pro Bash<sup>1</sup> (kapitola 6.1). Tyto moduly jsou však podporované pouze pro GRASS GIS do verze 6. Nová verze GRASS GIS 7 již moduly psané jako shellový skript nepodporuje. Jako první krok je tedy nutné moduly přepsat do podporovaného programovacího jazyka - Pythonu (kapitola 6.2).

---

<sup>1</sup>Bash je unixový shell a v linuxových systémech slouží jako příkazový interpret.

Dále je třeba zjistit, jak postupovat, aby mohl být interpolační nástroj začleněn do oficiální distribuce GRASS GISu. Jednou z cest by mohlo být zbavení se závislosti knihovny *nn-c*, která provádí samotnou interpolaci, na knihovně *Triangle*, která aplikuje Delaunayovu triangulaci. Tuto knihovnu nelze vzhledem k nevyhovující licenci začlenit do standardní distribuce GRASS GISu. V tomto případě je nutné vyzkoumat, jaké nástroje by mohly knihovnu *Triangle* nahradit. Bude zjištěno, jaké prostředky má k dispozici samotný GRASS GIS nebo zda je možné použít nějaký jiný open source nástroj, který by triangulaci provedl.

Jako další cesta se jeví možnost využítí knihovny *CGAL*. Tato knihovna už přímo v sobě obsahuje jak Delaunayovu triangulaci, tak samotnou interpolaci. Knihovna *CGAL* je špička v oblasti výpočetní geometrie a o jejím začlenění do GRASS GISu se již uvažuje.

## Část I

# Teoretická část

# 1 GRASS GIS

GRASS (Geographic Resources Analysis Support System) GIS je geografický informační systém pro správu a analýzu prostorových dat, obrazových záznamů, produkci map a grafických výstupů, prostorové modelování a 3D vizualizaci. Na mnoha platformách (GNU/Linux, MS Windows, MAC OS) umožňuje práci s rastrovými i vektorovými daty, a to buď pomocí příkazové řádky nebo grafického uživatelského rozhraní. GRASS GIS je otevřený a volně šířitelný software pod licencí GNU GPL.

Historie[5] GRASS GISu začíná v roce 1982, kdy začal být vyvíjen U.S. Army Corps of Engineer/CERL (Construction Engineering Research Lab) pro vojenské účely. Nicméně koncem osmdesátých let byly veškeré zdrojové kódy dány k dispozici veřejnosti. Na začátku devadesátých let se začal pomocí internetu celosvětově rozširovat. V roce 1995 CERL odstoupil od projektu a vývoje se později ujal GRASS Development Team, který zahrnoval odborníky z celého světa.

GRASS je jeden z nejznámějších open-source GIS softwarů, jehož vývoj trvá déle než třicet let. Jádro softwaru je napsáno v jazyce C. Avšak snahou vývojářů je rozšíření GRASSu mezi širší odbornou veřejnost a proto v rámci snadnějšího použití jsou do programu začleněny moduly napsané v jazyce Python nebo C/C++. Aktuálně je k dispozici verze 7, na jejímž vývoji se podílí několik vývojářů z řad dobrovolníků po celém světě.

## 1.1 GRASS GIS Add-Ons

GRASS GIS (<http://grass.osgeo.org/>) je od roku 1982 v neustálém vývoji. Síla a úspěch GRASS GISu je založená hlavně na komunitě uživatelů. S ohledem na to, že filozofií vývojového týmu GRASS GISu vést uživatele k tvorbě jejich vlastních nástrojů a aplikací pro GRASS GIS. Pokud uživatel vyvine nějaký nástroj, který by mohl být užitečný ostatním uživatelům, má možnost svůj kód zveřejnit a zpřístupnit ostatním uživatelům pomocí Add-Ons (<http://grass.osgeo.org/download/addons/>).

## 2 Delaunayova triangulace

Interpolace metodou přirozeného souseda vychází z Delanayovi triangulace (DT) či Voronoiova diagramu (VD). Jak DT tak VD jsou důležité konstrukce ve výpočetní geometrii. Delaunayova triangulace lze odvodit z Voronoiova diagramu a stejně tak Voronoiův diagram se dá naopak odvodit z Dalunayovi triangulace.

### 2.1 Triangulace obecně

**Definice[6]:** Triangulace  $\Delta$  nad množinou bodů  $P$  představuje takové planární rozdělení, které vytvoří soubor  $m$  trojúhelníků  $t = \{t_1, t_2, \dots, t_m\}$  a hran tak, aby platilo:

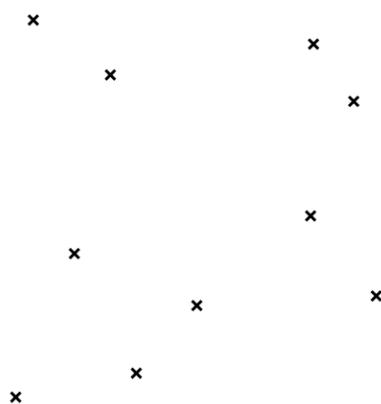
Libovolné dva trojúhelníky  $t_i, t_j \in \Delta$ ,  $(i \neq j)$ , mají společnou nejvýše hranu.

Sjednocení všech trojúhelníků  $t \in \Delta$  tvoří doménu  $\Omega$ .

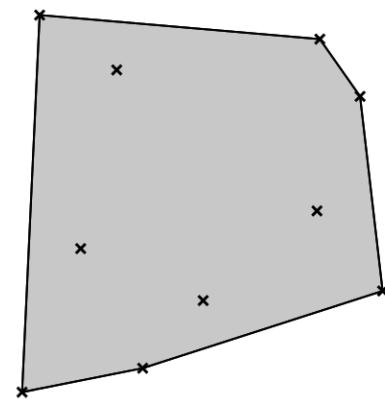
Uvnitř žádného trojúhelníku neleží žádný další bod z  $P$ .

Triangulace má širokou škálu aplikací. V kartografii a GISech se využívá pro tvorbu DMT. V DPZ ji lze využít pro tvorbu prostorových modelů, své uplatnění najde také v počítačové grafice a vizualizaci prostorových dat a mnoho dalších.

V našem případě budeme triangulaci provádět uvnitř takzvaného *konvexního obalu*. Konvexní obal množiny bodů  $P$  představuje takový konvexní mnohoúhelník, který obsahuje všechny body množiny bodů  $P$ , má co nejmenší plochu a ve kterém zároveň spojnice mezi kterýmkoliv body množiny bodů  $P$  leží uvnitř obalu.



Obrázek 1: Soubor bodů

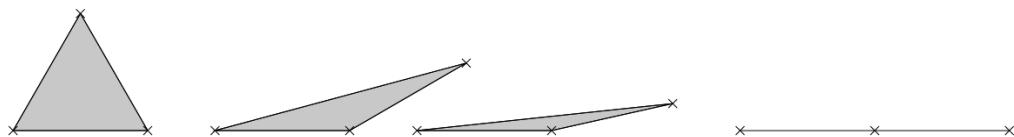


Obrázek 2: Konvexní obal nad doménou  $\Omega$

## 2.2 Validní a regulérní triangulace

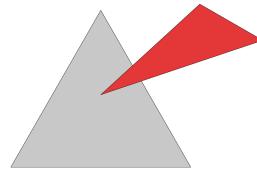
Triangulace je proces vytváření trojúhelníkové sítě mezi body v prostoru. Ne všechny trojúhelníkové sítě jsou pro nás ale zajímavé. Proto se tato práce bude zabývat triangulační sítí se specifickými vlastnostmi.

- Žádný z trojúhelníků  $t_{ijk}$  není zdegenerovaný, tzn. že vrcholy  $i, j, k$  neleží na přímce.



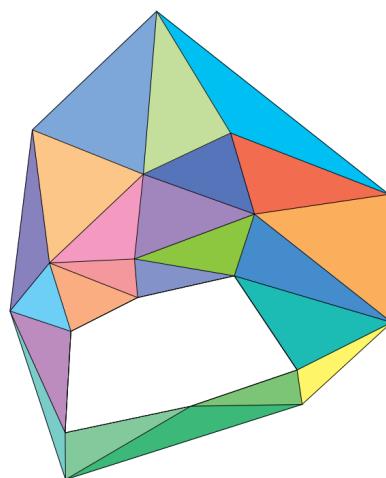
Obrázek 3: Degenerace trojúhelníku

- Žádná dvojice trojúhelníků se nepřekrývá, tzn.  $Int(t_{ijk}) \cap Int(t_{lmn}) = \emptyset$ .

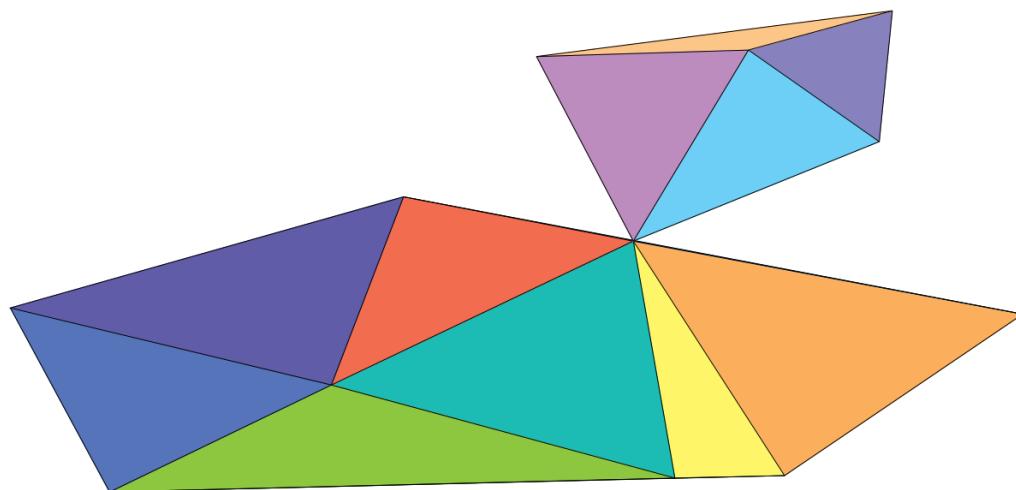


Obrázek 4: Překrývající se trojúhelníky

- Hranice dvou trojúhelníků se setkávají pouze na jejich hranách nebo v jejich vrcholech.
- Sjednocení všech trojúhelníků v celé triangulační síti se rovná celé doméně  $\Omega$ , nad kterou triangulaci provádíme.
- Doména  $\Omega$  musí být spojitá.
- Triangulační síť nesmí obsahovat díry.
- Jestliže vrchol  $v_i$  leží na hranici konvexního obalu, pak musí existovat právě dvě hraniční hrany, které mají vrchol  $v_i$  jako společný vrchol. To mimo jiné znamená, že počet hraničních vrcholů je roven počtu hraničních hran.



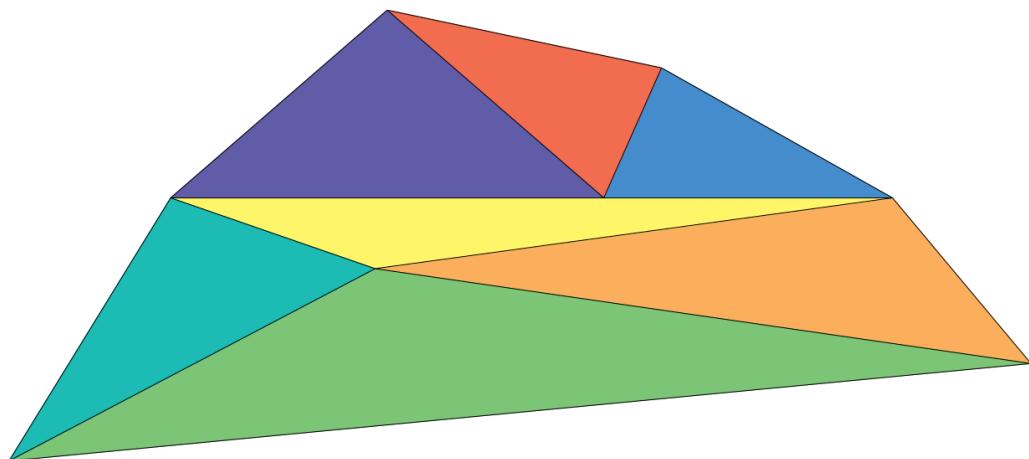
Obrázek 5: Nespojité doména, triangulace obsahující díry



Obrázek 6: Triangulace validní, ale neregulérní

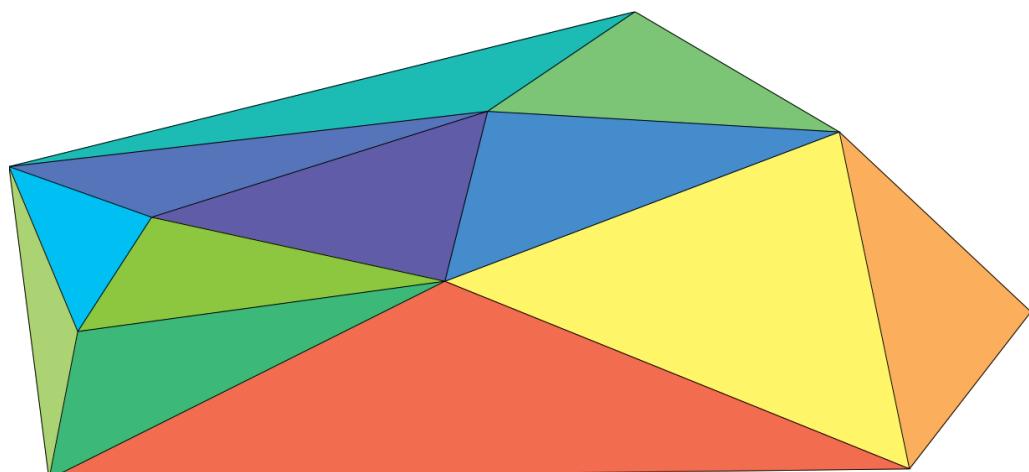
Pokud jsou naplněny první čtyři podmínky, může se triangulace nazvat *validní*. V případě, že naplníme všech sedm podmínek, dá se mluvit o *regulérní* triangulaci.

V případě triangulace na obrázku 6 se jedná o triangulaci validní, neboť splňuje všechny z prvních čtyř podmínek. Nicméně nesplňuje poslední podmínu, neboť existují více jak právě dvě hraniční hrany, které vstupují do jednoho vrcholu na konvexním obalu, a triangulaci tedy nelze nazvat i regulérní.



Obrázek 7: Triangulace nevalidní

Triangulace na obrázku 7 není validní, protože nesplňuje podmínu č. 3. Tím pádem se nejedná ani o triangulaci regulérní.



Obrázek 8: Regulérní triangulace

Na obrázku 8 už můžeme vidět validní a regulérní triangulaci. Jedná se též o triangulaci *optimální*, což je termín, kterým se budeme zabývat v kapitole 2.3.

## 2.3 Optimální triangulace

Nad doménou  $\Omega$  neexistuje pouze jediná trojúhelníková síť, v závislosti na počtu bodů v množině a jejich konfiguraci existuje poměrně velké množství možností, jak může trojúhelníková síť vypadat. Ne všechny sítě jsou však vhodné k dalšímu zpracovávání, a právě proto je snaha najít *optimální* triangulaci.

Při řešení otázky, jak vypadá optimální triangulace, je zásadní zamyslet se nad tvarem trojúhelníků. V ideálním případě by byly všechny trojúhelníky rovnostranné, leč tento případ se v případě náhodně roztroušených dat nevyskytuje.

Při tvorbě optimální triangulace se tedy problém řeší z opačného konce. Zásadní snahou při tvorbě sítě by mělo být vyhýbání se podlouhlým, štíhlým nebo téměř degenerovaným trojúhelníkům, tedy trojúhelníkům s příliš ostrými nebo s příliš tupými úhly.

### Kruhová podmínka

Definice: Nechť hrana  $\overline{p_i p_j}$  inciduje s trojúhelníkem  $t_1$  tvořeným vrcholy  $p_i, p_j, p_k$  a trojúhelníkem  $t_2$  tvořeným vrcholy  $p_i, p_j, p_l$  a kružnice  $k$  procházející body  $p_i, p_j, p_k$ . Hrana  $\overline{p_i p_j}$  je nelegální tehdy a právě jen tehdy, jestliže bod  $p_l$  leží uvnitř  $k$ .

Pokud tedy bod  $p_l$  leží uvnitř kružnice  $k$ , je hrana  $\overline{p_i p_j}$  a tedy diagonála čtyřúhelníku nelegální, stejně tak jako oba trojúhelníky  $t_1$  a  $t_2$ . K jejich legalizaci je provést tzv. *Edge swaping*.

**MaxMin a MinMax úhlová podmínka** Pokud nad množinou bodů  $P$  provedeme všechny možné triangulace, můžeme nejoptimálnější triangulaci určit pomocí MaxMin popř. MinMax podmínky. V případě MaxMin podmínky je pro každou možnou triangulaci nalezen největší minimální vnitřní úhel trojúhelníku a ten je porovnán s největším minimálním úhlem ostatních triangulací, což vede k eliminaci trojúhelníků s velmi tupými úhly. U MinMax podmínky se postupuje obdobně, pouze se porovnávají nejmenší maximální vnitřní úhly trojúhelníků a eliminují se tak trojúhelníky s velmi ostrými úhly.

**MinMax podmínka**[6] Eliminace trojúhelníků s příliš tupými úhly. Triangulace  $\Delta(P)$  je vzhledem k tomuto kritériu na rozdíl od triangulace  $\Delta'(P)$  optimální, je-li největší úhel  $\alpha$  generovaný triangulací  $\Delta(P)$  je menší než největší úhel  $\alpha'$  generovaný triangulací  $\Delta'(P)$ .

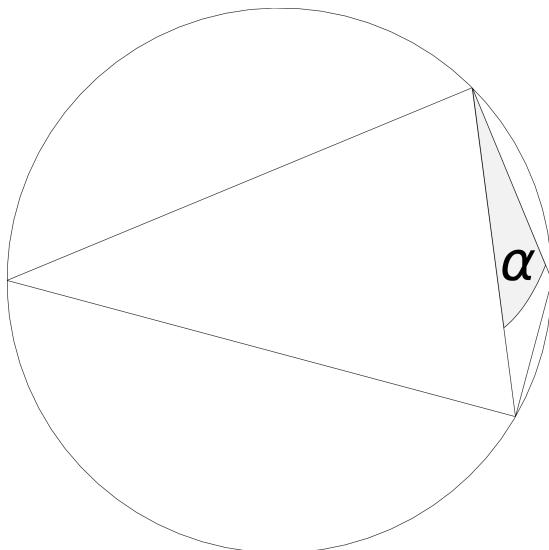
$$\begin{aligned}\alpha_{max} &= \max(\alpha_i(\Delta)) \\ \alpha'_{max} &= \max(\alpha'_i(\Delta)) \\ \alpha_{max} &< \alpha'_{max}\end{aligned}$$

**MaxMin podmínka** Eliminace trojúhelníků s příliš ostrými úhly. Triangulace  $\Delta(P)$  je vzhledem k tomuto kritériu na rozdíl od triangulace  $\Delta'(P)$  optimální, je-li nejmenší úhel  $\alpha$  generovaný triangulací  $\Delta(P)$  je větší než nejmenší úhel  $\alpha'$  generovaný triangulací  $\Delta'(P)$ .

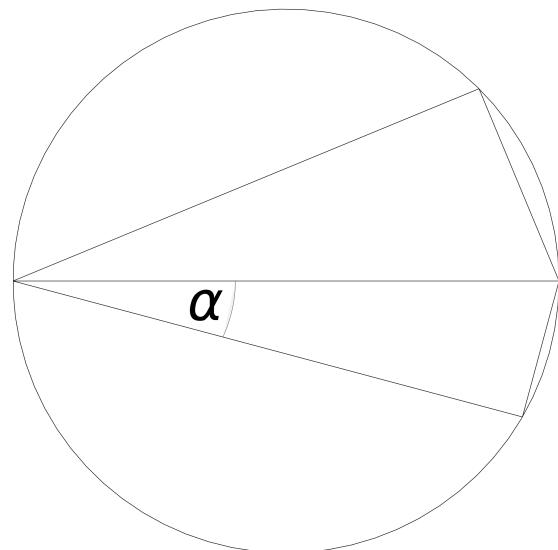
$$\begin{aligned}\alpha_{min} &= \min(\alpha_i(\Delta)) \\ \alpha'_{min} &= \min(\alpha'_i(\Delta)) \\ \alpha_{min} &> \alpha'_{min}\end{aligned}$$

**Neutrální případ pro MaxMin podmíncu** Problém nastává ve chvíli, kdy některé body, mezi kterými chceme triangulaci provést, leží na kružnici nebo se tomu limitně blíží. V takových chvílích nastává takzvaný neutrální případ pro MaxMin podmíncu. V některých případech to může vést k nejednoznačnému určení, která z triangulací je optimální a musíme si pomocí hodnocením na základě nejen MaxMin ale i MinMax podmíncu.

Nejčastěji tento jev nastává u čtyřúhelníku, jehož všechny vrcholy leží na kružnici. Jak je vidět na obrázcích 9 a 10, maximální minimální úhel triangulace je v obou případech úhel  $\alpha$ , což je obvodový úhel nejkratší strany čtyřúhelníku. V tomto případě se jedná o neutrální případ pro MaxMin podmíncu. Pokud by tedy triangulace byla hodnocena pouze na základě MaxMin podmíncu, mohly by být obě triangulace prohlášeny za optimální. Už od pohledu je ale zřejmé, že lepší tvar trojúhelníků poskytuje triangulace na obrázku 9. Pokud je ale triangulace hodnocena i s ohledem na MinMax podmíncu, je prokázáno, že triangulace na obrázku 10 je optimální.



Obrázek 9: Případ 1



Obrázek 10: Případ 2

**Edge swaping, legalizace** Na obrázcích 9 a 10 jsme si ukázali případ neutrálního případu pro MinMax podmínku. Proces, při kterém jsme zaměnili diagonálu čtyřúhelníku tak, aby byli oba trojúhelníky lokálně optimální, se nazývá *Edge swapping*. Pokud budeme tento proces aplikovat pro celou triangulaci dojde k takzvané *legalizaci* triangulace.

**Delaunayova triangulace** Triangulace, která je optimální dle kruhové nebo MaxMin úhlové podmínky a která je definována nad konvexním obalem množiny bodů, se nazývá *Delaunayova triangulace*<sup>2</sup>.

---

<sup>2</sup>Delaunayova triangulace je pojmenována podle ruského matematika Borise Nikolaeviče Delaunayho[13], který ji definoval v roce 1934.

## 2.4 Voronoiův diagram

Voronoiův diagram souvisí s Delaunayovou triangulací. Představme si opět množinu bodů  $P = \{p_1, \dots, p_N\}$  v rovině  $E^2$  a nechť je  $d(p_i, p_j)$  Eukleidovská vzdálenost mezi body  $p_i$  a  $p_j$ . Rovinu poté rozdělíme na oblasti  $V(p_i, \dots, p_N)$ , kde každému bodu  $p_i$  z množiny  $P$  přiřadíme takovou oblast, která splňuje následující podmínu:

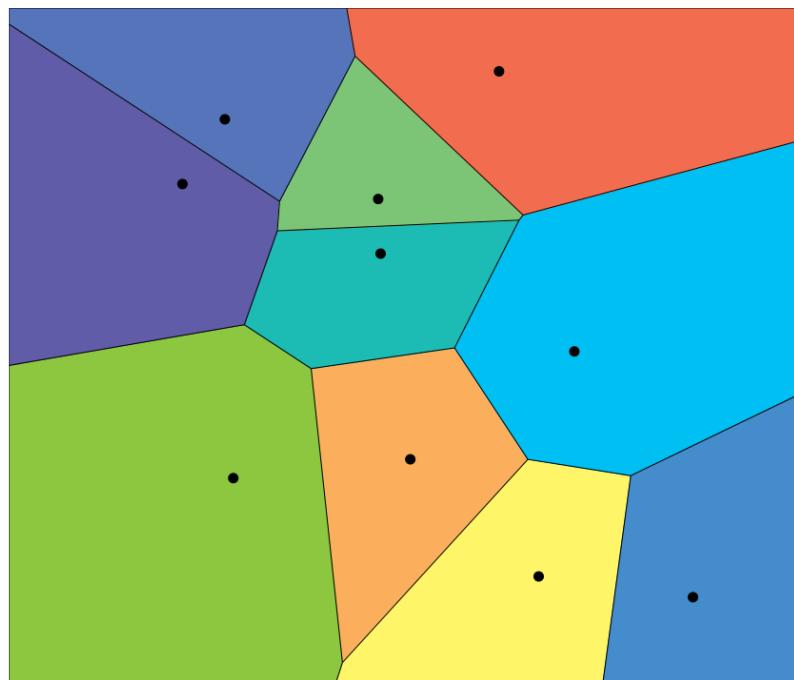
$$V(p_i) = \{x : d(x, p_i) < d(x, p_j), j = 1, \dots, N\}.$$

Pro každou takovou oblast přiřazenou bodu  $p_i$  platí, že všechny body uvnitř této oblasti jsou k bodu  $p_i$  blíž než ke kterémukoliv jinému bodu z množiny  $P$ . Toto rozdělení roviny se nazývá *Voronoiův diagram (VD)* množiny bodů a každá uzavřená buňka se nazývá *Voronoiova buňka*.

Voronoiův diagram má následující vlastnosti:

- Voronoiův diagram pro množinu bodů je pouze jeden.
- Všechny oblasti jsou konvexní.
- Oblast  $V(p_i)$  obsahuje jediný bod  $p_i \in P$ .
- Oblasti na okraji jsou neohraničené.
- Počet hran a uzel je přímo úměrný počtu bodů v množině  $P$ .
- Pokud žádné 4 body neleží na kružnici, uzly mají stupeň 3.
- Uzel leží ve středu kružnice určené 3 body z  $P$ , které leží v přilehlých oblastech a neleží na přímce.

Nechť  $H(p_i, P_j)$  je polovina obsahující všechny body v rovině, jejichž vzdálenost od bodu  $p_i$  je menší než od bodu  $p_j$ . Potom Voronoiova oblast  $V(p_i)$  je průnikem  $N - 1$  polovin,  $V(p_i) = \bigcap_{\substack{j=1, \dots, N \\ i \neq j}} H(p_i, p_j)$ , kde každá oblast má maximálně  $N - 1$  stran.



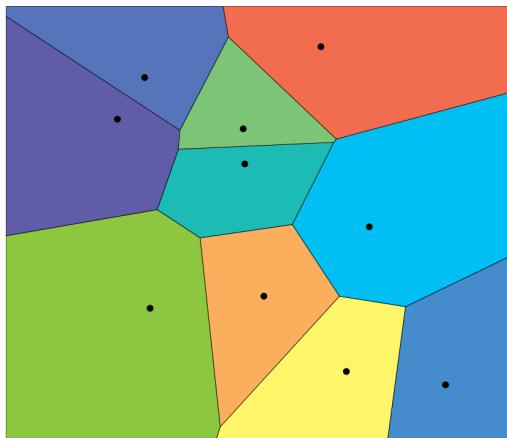
Obrázek 11: Voronoivy buňky

Z obrázku 11 je vidět, že oblasti přiřazené pro body na okraji konvexního obalu nejsou zcela ohraničené a uzavřené. Jednotlivé hranice oblastí nazýváme *Voronoiovy polygony*, které jsou složeny z takzvaných *Voronoiových hran a bodů*. O dvou bodech  $p_j$  a  $p_k$  můžeme tvrdit, že jsou *Voronoiovi sousedé*, pokud oblasti, kterým náleží, sdílí společnou *Voronoiovu hranu*.

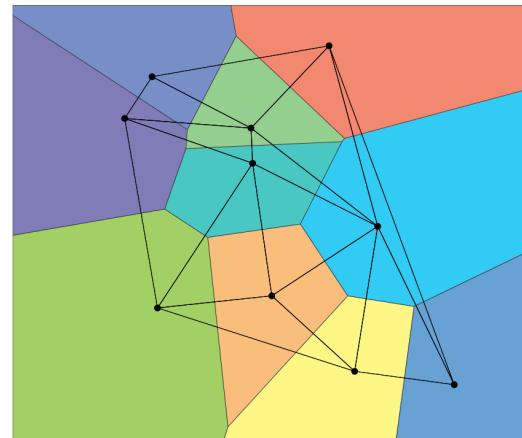
Občas se můžeme setkat i s označením *Thiessenovy polygony* podle klimatologa Thiessena, který Voronoi diagramy používal k interpolaci klimatických dat z náhodně distribuovaných meteorologických stanic.

## 2.5 Voronoiový diagramy a Delaunayova triangulace

DT je planární graf<sup>3</sup>, který je duální k VD. Jedna konstrukce tedy může být odvozena od druhé a naopak. Pokud pro množinu bodů v rovině provedeme rozdělení do Voronoiových diagramů a následně spojíme úsečkami všechny Voronoioví sousedy získáme Delaunayovu triangulaci.



Obrázek 12: Voronoiův diagram souboru bodů



Obrázek 13: Voronoiův diagram a Delaunayova triangulace

Na obrázku 12 můžeme vidět Voronoiův diagram pro množinu bodů. Pokud každou dvojici Voronoiových sousedů spojíme spojnicí, jak je vidět na obrázku 13, získáme několik trojúhelníků, které se nepřekrývají a tvoří trojúhelníkovou síť. Spojnice mezi body, které jsou přiřazeny k neuzavřeným oblastím, tvoří konvexní obal. Pro trojúhelníkovou síť uvnitř tohoto konvexního obalu se jedná o Delaunayovu triangulaci.

Trojúhelníky této síť se nazývají *Delaunayovi trojúhelníky*, které jsou tvořeny spojnicemi tzv. *Delaunayovými hranami*. Dále je také zřejmé, že Voronoiové hrany leží na ose Delaunayových hran.

---

<sup>3</sup>Planární graf je takový graf, který lze v rovině nakreslit bez křížení hran.

### 3 Algoritmy pro tvorbu DT

Triangulační algoritmy jsou poměrně dobře matematicky prozkoumaná oblast, ke kterým je k dispozici široký teoretický základ. Při hledání vhodného algoritmu je třeba zohlednit následující požadavky:

- Jednoduchost algoritmu a jeho snadná implementace.
- Dostatečná rychlosť i pro velké množiny bodů ( $n > 1E6$ ), ideálně aby se výpočetní čas blížil  $O(n \log(n))$ .
- Malá citlivost a vysoká stabilita pro případy nejednoznačné triangulace (body na kružnici).
- Převod do vyšších dimenzí.
- Možnost paralelizace algoritmu.

Vhodný algoritmus je vhodné vybírat na základě datových struktur a konkrétního případu. Ne vždy lze dokonale splnit všechny požadavky, např. jednoduché implementace mají delší výpočetní čas. Naopak algoritmy s kratším výpočetním časem jsou dost náročné na implementaci.

### 3.1 Lokální optimalizace

*Lokální optimalizace (LO)* je proces přetvoření libovolné triangulace na DT. Proces je prováděn pomocí prohazování nelegálních hran v dvojicích trojúhelníků tvořících konvexní čtyřúhelník na základě kruhové nebo MaxMin podmínky.

Pro množinu bodu  $P$  nechť  $e_i$  je vnitřní hrana triangulace a  $Q$  je čtyřúhelník tvořený dvěma trojúhelníky se společnou hranou  $e_i$ . Možnost, že čtyřúhelník je nekonvexní nebo že všechny jeho body leží na kružnici, nebude nyní brán v úvahu. Poté je za použití MaxMin nebo kruhové podmínky rozhodnuto, zda je třeba prohodit hranu  $e_i$ . V případě, že po použití Lokální optimalizace není potřeba prohodit hranu, můžeme dvojici trojúhelníků a jejich společnou hranu prohlásit za lokálně optimální. V případě, že budeme lokální optimalizaci používat opakovaně pro všechny hrany v triangulaci, dokud nebudou všechny hrany lokálně optimální, získáme optimální triangulaci.

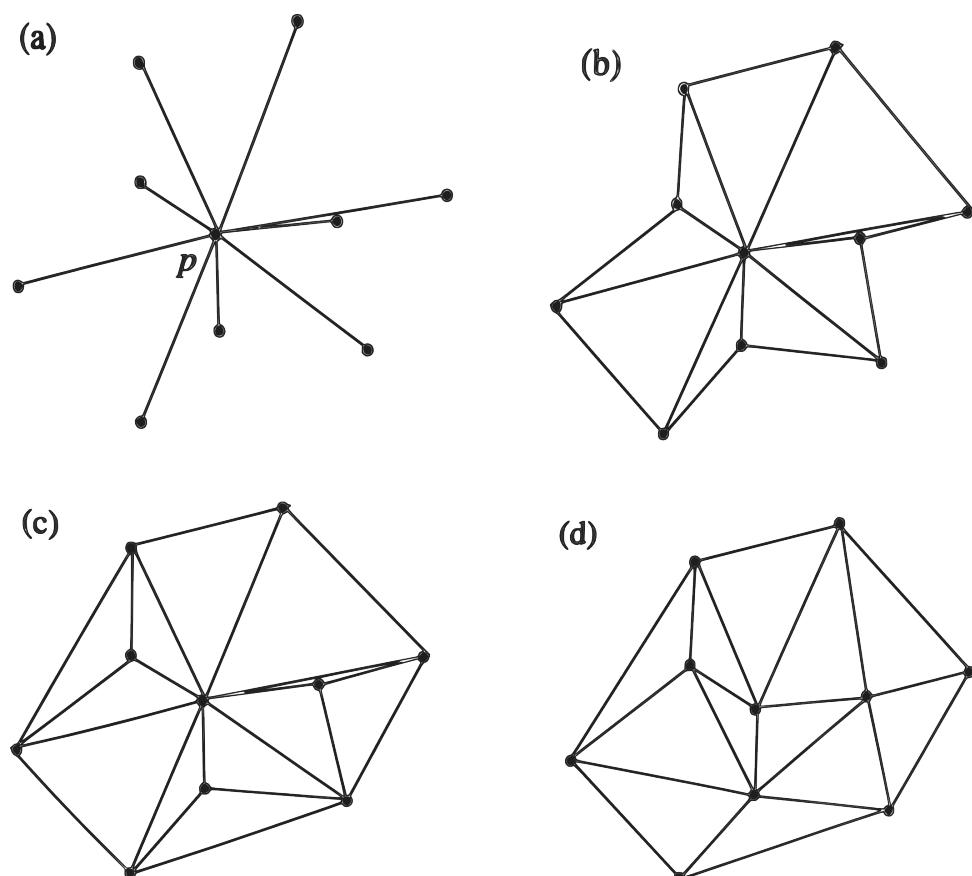
Algorithm 1: Lokální optimalizace

```

1: Vytvoř pomocnou triangulaci  $\Delta(P)$ 
2: legal=false
3: while  $\Delta(P) \neq$  legal do
4:   legal=true
5:   for  $\forall e_i \in \Delta(P)$  do
6:     Vezmi hranu  $e_i \in \Delta(P)$ 
7:     Nalezni trojúhelníky  $t_1, t_2$  incidující s  $e_i$ .
8:     if  $(t_1 \cup t_2)$  je konvexní a nelegální then
9:       Legalize  $(t_1, t_2)$ 
10:      legal=false
11:    end if
12:  end for
13: end while
```

### 3.2 Paprskovitý algoritmus

Tento algoritmus je podobný tomu předchozímu, pouze využívá jiný postup, jak nalézt počáteční triangulaci  $\Delta$ . Na začátku se nalezne bod  $p$  z množiny  $P$  takový, který je nejbližší jejímu středu. Poté je paprskovitě spojen se všemi zbývajícími body množiny  $P$ . Tyto body se následně seřadí podle orientace a vzdálenosti od bodu  $p$  a v tomto pořadí se spojí hranami. Potom se vytvoří hrany na hranici triangulace. Vzniklá triangulace se dále postupně po jednotlivých hranách legalizuje stejně jako v předchozím případě.



Obrázek 14: Paprskovitý (Radial sweep) algoritmus, zdroj: [1]

### 3.3 Algoritmus inkrementálního vkládání

Tento algoritmus je poměrně jednoduchý a snadný na implementaci, obzvlášť pokud je zvolena vhodná datová struktura. Jeho složitost je  $O(n^2)$ , kterou lze úpravami zlepšit až na  $O(n \log(n))$ . Tato metoda je tvořena třemi kroky. Na začátku je vytvořen konvexní obal množiny bodů  $P$  a pro jeho lomové body se provede triangulace. Do vzniklé triangulace se postupně vkládají body z množiny  $P$ . Tato nově vzniklá triangulace nemusí být nutně DT, a proto se provede ještě její legalizace.

---

Algorithm 2: Algoritmus inkrementálního vkládání

---

```

1: Vytvoření konvexního obalu  $\Omega$  nad množinou bodů  $P$ 
2: Vytvoř DT pro lomové body konvexního obalu
3: for  $i \in 1, \dots, n$  do
4:   Přidej  $p$  do DT
5:   Najdi trojúhelník  $t$  s vrcholy  $p_i, p_j, p_k$  takový, že  $p \in t$ 
6:   if  $p$  leží uvnitř  $t$  then
7:     Přidání hrany  $\overline{p, p_i}$ 
8:     Přidání hrany  $\overline{p, p_j}$ 
9:     Přidání hrany  $\overline{p, p_k}$ 
10:    Legalizace hrany  $p, \overline{p_i, p_j}, t$ 
11:    Legalizace hrany  $p, \overline{p_j, p_k}, t$ 
12:    Legalizace hrany  $p, \overline{p_k, p_i}, t$ 
13:   else if  $p$  leží na hraně  $p_i, p_j$  trojúhelníků  $t_1, t_2$  then
14:     Přidání hrany  $\overline{p, p_i}$ 
15:     Přidání hrany  $\overline{p, p_j}$ 
16:     Přidání hrany  $\overline{p, p_k}$ 
17:     Přidání hrany  $\overline{p, p_l}$ 
18:     Legalizace hrany  $p, \overline{p_j, p_k}, t$ 
19:     Legalizace hrany  $p, \overline{p_k, p_i}, t$ 
20:     Legalizace hrany  $p, \overline{p_i, p_l}, t$ 
21:     Legalizace hrany  $p, \overline{p_l, p_j}, t$ 
22:   end if
23: end for

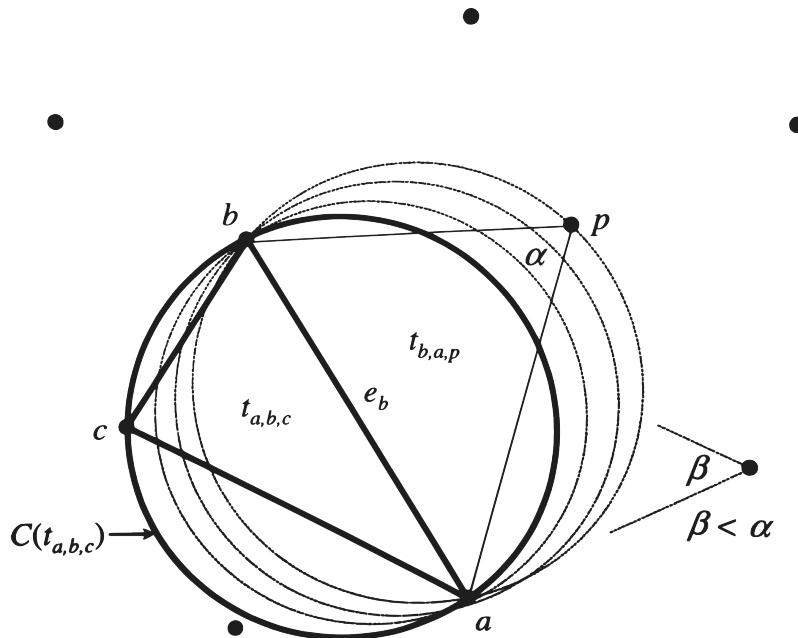
```

---

### 3.4 Algoritmus inkrementální konstrukce (Step-by-Step)

Tento postup, který je založen na postupném přidávání bodů do již vytvořené DT, tvoří triangulaci postupně, trojúhelník po trojúhelníku. Na začátku jsou vybrány dva body  $p_1, p_2$ , které jsou samy sobě Voronoiovými sousedy, a mezi nimi je vytvořena základní Delaunayova hrana  $e$ . K této hraně  $e$  se hledá další bod  $p$  tak, aby vyhovoval kruhové podmínce a který společně s hranou  $e$  vytvoří Delaunayův trojúhelník s vrcholy  $p_1, p_2$  a  $p$ , který se zapíše do DT. Každá Delaunayovská hrana je orientována, bod  $p$  hledáme pouze vlevo od ní. Pro konstrukci se používá modifikovaná datová struktura AEL (Active Edge List). Obsahuje hrany  $e$ , ke kterým hledáme body  $p$ , do struktury se neukládá topologický model, viz kapitola 4.

Základní vlastnost toho postupu je, že se v každém kroku ke stávající triangulaci připojí další bod a triangulaci tak rozšíří. Vznikající triangulace je už v procesu tvoření Delaunayova a není tedy třeba žádné následující optimalizace.



Obrázek 15: Algoritmus inkrementální konstrukce (Step-by-Step), zdroj: [1]

---

 Algorithm 3: Step-by-step algoritmus
 

---

```

1:  $p_1$ =náhodný bod z  $P$ ,  $p_2$ =nejbližší bod k  $p_1$ 
2: Vytvoř hrancu  $\bar{e} = \overline{p_1 p_2}$ 
3:  $p = d_D(e)$ , bod s nejmenší Delaunay vzdáleností vlevo od  $e$ 
4: if  $p = NULL$  then
5:     prohod' orientaci  $\bar{e} = \overline{p_1 p_2} \Rightarrow \bar{e} = \overline{p_2 p_1}$ 
6: end if
7:  $e_2 = \overline{p_2 p}, e_3 = \overline{p p_1}$ 
8: Přidej  $e, e_2, e_3$  do AEL
9: while AEL není prázdný do
10:     $e = \overline{p_1 p_2}$  první hrana AEL
11:    Změna orientace hrany  $e = \overline{p_1 p_2} \Rightarrow e = \overline{p_2 p_1}$ 
12:    Bod  $p$  s nejmenší Delaunay vzdáleností  $d_D(e)$  vlevo od  $e$ 
13:    if  $p \neq NULL$  then
14:         $e_2 = \overline{p_2 p}, e_3 = \overline{p p_1}$ 
15:        Přidej  $e, e_2, e_3$  do AEL
16:        Přidej  $e$  do DT
17:    end if
18:    pop( $e$ )
19: end while

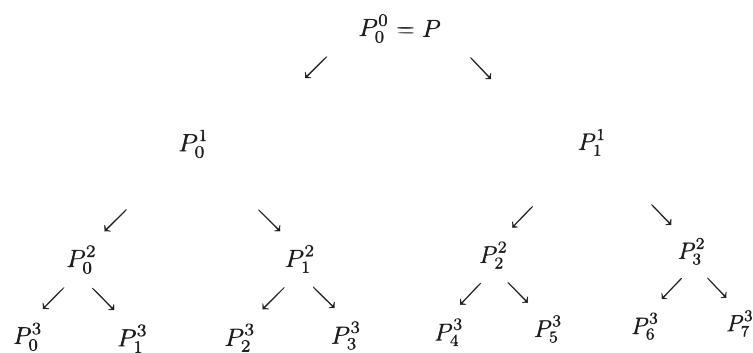
```

---

### 3.5 Algoritmus Rozděl a panuj

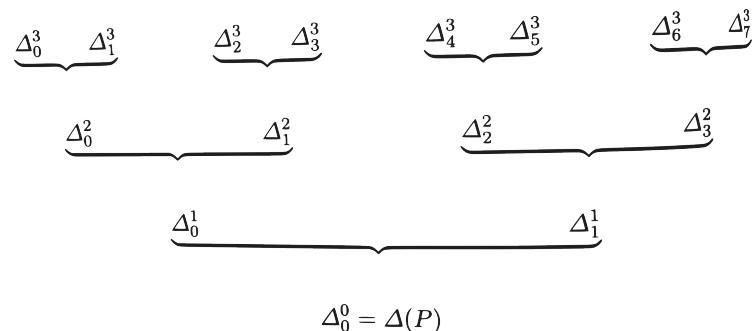
Rozděl a panuj (Divide and Conquer) je přístup, který vede k paralelizaci. Pro tvorbu Delaunayovské triangulaci nabízí nejlepší výsledky, co se výkonnosti týče. Tento přístup je založený na jednoduchých krocích:

1. Rekurzivní dělení množiny bodů  $P$  až do stavu, kdy se pro podmnožinu nabízí jednoduché geometrické řešení. Při postupném dělení množiny  $P$  se nakonec dostaneme do stavu, kdy nám zbydou buď tři body, v tom případě vytvoříme trojúhelník, nebo dva body, kdy vytvoříme hranu.



Obrázek 16: Rekurzivní dělení na dílčí podmnožiny, zdroj: [1]

2. Vytvoření prozatimní triangulace, která nemusí být Delaunayovská.
3. Připojení ke stávající triangulaci a její legalizace na Delaunayovskou. Právě propojování podmnožin na jednotlivých vrstvách je nejnáročnější část tohoto algoritmu.

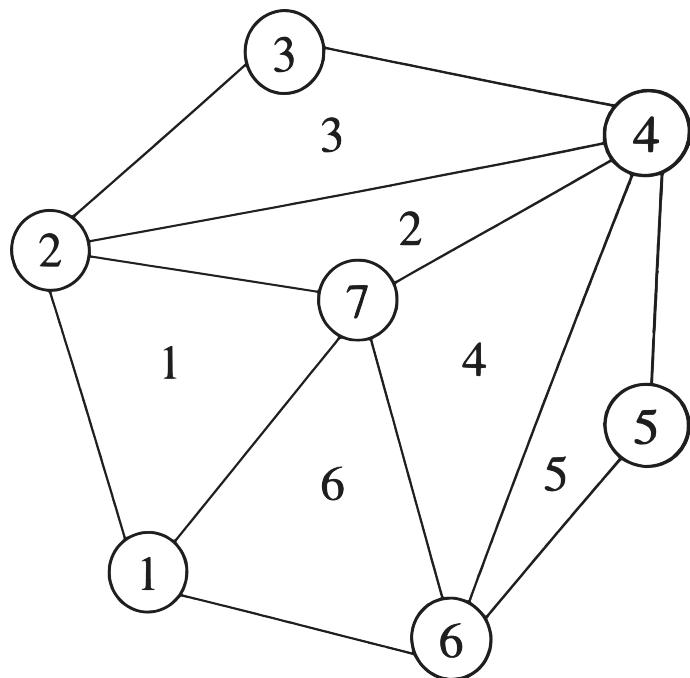


Obrázek 17: Propojování dílčích triangulací, zdroj: [1]

Algoritmus Rozděl a panuj využívá GRASS modul *v.delaunay*, o kterém budeme mluvit později, viz kapitola 7.2.

## 4 Datové struktury

Existuje mnoho různých datových struktur pro uložení topologických informací o triangulační síti. Každá nabízí nějaké výhody a uživatel většinou musí řešit dva problémy. Jednak požadavky na dostatečnou kapacitu pro uložení dat, a pak dostatečnou efektivitu při získávání informací ze struktury.



Obrázek 18: Příklad jednoduché triangulace

## 4.1 Jednoduchá trojúhelníková struktura

Tato struktura ukládá informace v nejednodušší podobě. Pracuje pouze s trojicemi id jednotlivých vrcholů trojúhelníků v seznamu nebo v poli. Trojúhelníky jsou seřazeny vzestupně podle id, zatímco na pořadí uložených vrcholů nezáleží. Struktura je velmi nenáročná, co se požadavků na kapacitu týče, nicméně toto je vykoupené tím, že nemáme k dispozici žádnou informaci, který trojúhelník sousedí s kterým.

Trojúhelník	Vrcholy		
	i	j	k
1	1	7	2
2	2	7	4
3	2	4	3
4	7	6	4
5	4	6	5
6	7	1	6

Tabulka 1: Jednoduchá trojúhelníková struktura

## 4.2 Trojúhelníková struktura se sousedy

Tato struktura přináší rozšíření o informaci, s kterými trojúhelníky daný trojúhelník sousedí. K zapotřebí je tedy další seznam, který obsahuje id trojúhelníku.

Trojúhelník	Vrcholy			Sousedé		
	i	j	k	$t_{j,k}$	$t_{k,i}$	$t_{i,j}$
1	1	7	2	2	-	6
2	2	7	4	4	3	1
3	2	4	3	-	-	2
4	7	6	4	5	2	6
5	4	6	5	-	-	4
6	7	1	6	-	4	1

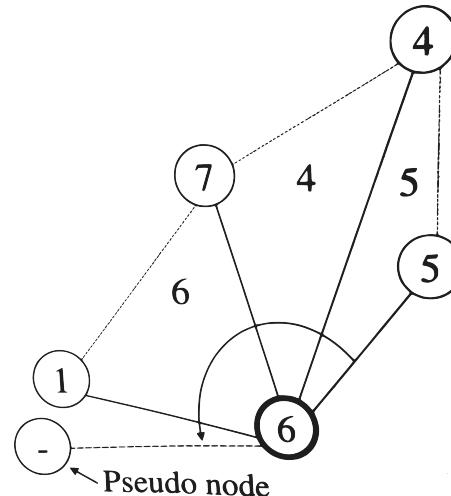
Tabulka 2: Trojúhelníková struktura se sousedy

### 4.3 Vertex-based struktura se sousedy

Vertex-based struktura se sousedy je poměrně úsporná struktura, co se objemu dat týče. Ke každému vrcholu  $v_i$  v síti jsou uloženy do seznamu sousedů vrcholy, které jsou s ním spojené. V případě, že  $v_i$  leží na konvexním obalu je seznam sousedů ukončen 'pseudo-vrcholem'.

Vrchol	Sousední vrcholy						Suma vrcholů
1	6	7	2	0			4
2	1	7	4	3	0		9
3	2	4	0				12
4	3	2	7	6	5	0	18
5	4	6	0				21
6	5	4	7	1	0		26
7	6	4	2	1			30

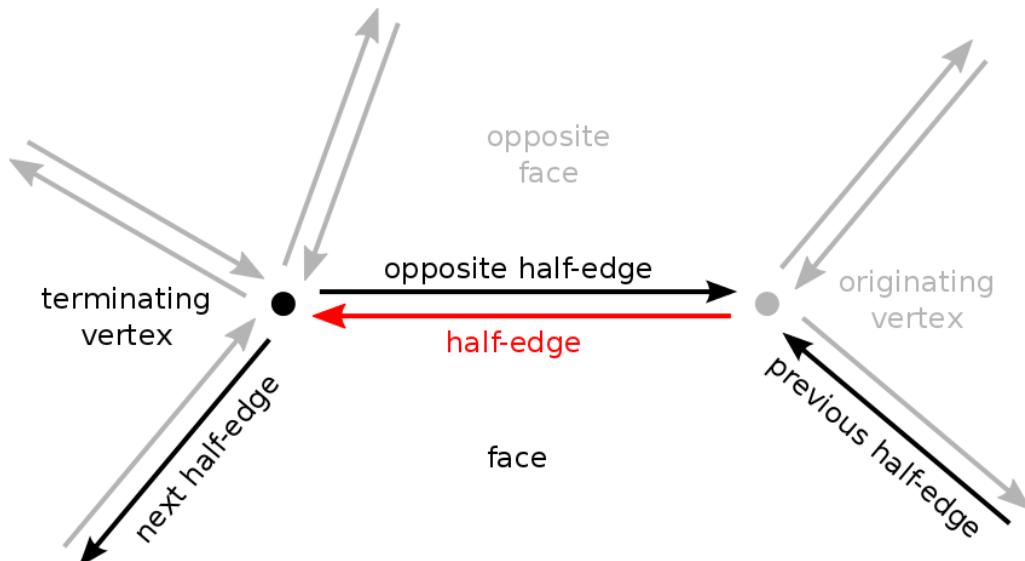
Tabulka 3: Vertex based struktura



Obrázek 19: Datová struktura s použitím pseudo-uzlu, zdroj: [1]

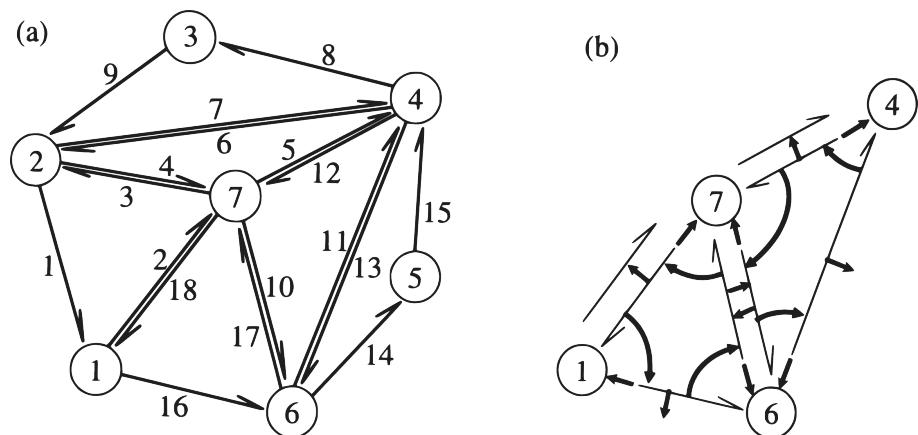
## 4.4 Half-edge datová struktura

Tato struktura uchovává topologický model na základě orientovaných polovičních hran. Princip je rozdělit každou hranu na dvě orientované *půlhrany*, z nichž každá směruje opačným směrem. Každému trojúhelníku můžeme tak přiřadit tři půlhrany, které jsou orientované v protisměru hodinových ručiček. Každá půlhrana začíná v *počátečním vrcholu* a směruje do *cílového vrcholu*. V half-edge struktuře si každá půlhrana uchovává ukazatel na svůj počáteční bod, na svůj koncový bod, na předchozí i následující půl-hranu, patřící stejnému trojúhelníku a nakonec pointer na opačně orientovanou hranu, tzv. *twin-edge*. Pointer na twin-edge hranu nemají pouze půlhrany ležící na konvexním obalu triangulace.



Obrázek 20: Half-edge struktura (Zdroj: <http://pointclouds.org/blog/nvcs/martin/index.php>)

Z half-edge struktury je možné odvodit mnohé jiné příbuzné struktury obsahující další informace o topologii, které ulehčují procházení celou sítí při vyhledávání, ale zároveň zvyšují nároky na kapacitu. Mezi tyto příbuzné struktury patří například *vertex-edge*, *face-edge* nebo *winged edge* datová struktura, kterou používá například modul *v.delaunay* v GRASS GISu, kapitola 7.2.



Obrázek 21: Half-edge struktura, zdroj: [1]

Půl-hrana	Startovní vrchol	Half-edge pointry	
		Další hrana v trojúhelníku	Twin-edge
1	2	2	-
2	1	3	18
3	7	1	4
4	2	5	3
5	7	6	12
6	4	4	7
7	2	8	6
8	4	9	-
9	3	7	-
10	7	11	17
11	6	12	13
12	4	10	5
13	4	14	11
14	6	15	-
15	5	13	-
16	1	17	-
17	6	18	10
18	7	16	2

Tabulka 4: Half-edge struktura

Listing 1: Definice datové struktury

---

```

class Half-edge
{
    Point *originating_vertex, *terminating_vertex;
    Half-edge *edge, *prev_edge, *next_edge, *
    twin_edge;

};

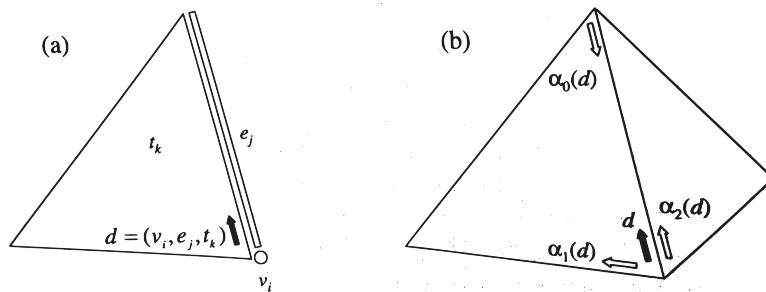
```

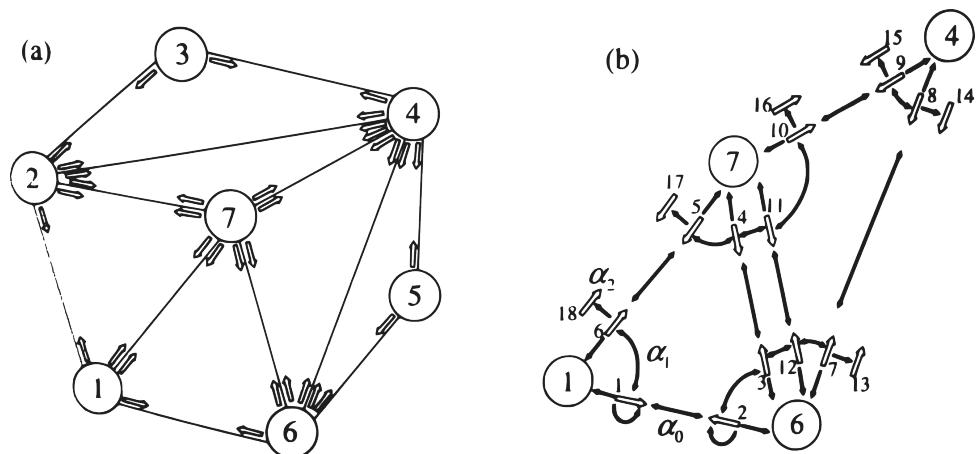
---

## 4.5 Dart-based datová struktura

Dart-based datová struktura nabízí extrémně rychlé procházení topologie napříč datovou strukturou. Ta může být representována jako množina ukazatelů, tzv. *dartů*  $D$ . Dart  $d \in D$  představuje trojici  $(v_i, e_j, t_k)$ , kde  $v_i$  je vrchol hrany  $e_j$  v trojúhelníku  $t_k$ . V jednom trojúhelníky tedy můžeme definovat celkem šest dartů. Každý dart  $d \in D$  odkazuje k jednomu vrcholu a třem dalším dartům z  $D$  pomocí takzvaných  $\alpha$ -*iterátorů*. Tyto iterátory jsou definovány takto (obrázek 22(b)):

- $\alpha_0(d)$  odkazuje k trojici s rozdílným vrcholem  $v$ , ale stejnou hranou  $e$  a trojúhelníkem  $t$ ,
- $\alpha_1(d)$  odkazuje k trojici s rozdílnou hranou  $e$ , ale stejným vrcholem  $v$  a trojúhelníkem  $t$ ,
- $\alpha_2(d)$  odkazuje k trojici s rozdílným trojúhelníkem  $t$ , ale stejným vrcholem  $v$  a hranou  $e$ .

Obrázek 22:  $\alpha$ -iterátory, zdroj: [1]



Obrázek 23: Dart-based datová struktura, zdroj: [1]

Dart d	Dartové pointery			
	Vrchol	$\alpha_0$	$\alpha_1$	$\alpha_2$
1	1	2	6	1
2	6	1	3	2
3	6	4	2	12
4	7	3	5	11
5	7	6	4	17
6	1	5	1	18
7	6	8	12	13
8	4	7	9	14
9	4	10	8	15
10	7	9	11	16
11	7	12	10	4
12	6	11	7	3

Tabulka 5: Dart based struktura

## 5 Interpolace metodou přirozeného souseda

Interpolace metodou přirozeného souseda je deterministická interpolační metoda pro prostorová data. Poskytuje spojitý, vyhlazený výstup, bez extrapolovaných hodnot.

Pro výpočet vah využívá Voronoiových diagramů. Do VD pro měřené body vkládá body určené k interpolaci. Vložením nového bodu dojde v jeho okolí k přetvoření VD. K vypočtení váhy se používá jak VD před vložením bodu, tak VD po vložení bodu. Voronoiova buňka nově vloženého bodu překrývá několik buněk z původního VD. Právě tento překryv, který nově vložený bod „ukradne“ z plochy původních buněk, slouží k výpočtu váhy pro interpolaci.

### 5.1 Lineární interpolace

Matematicky tedy můžeme zapsat:

$$V(p) = \sum_{i=1}^n V_i$$

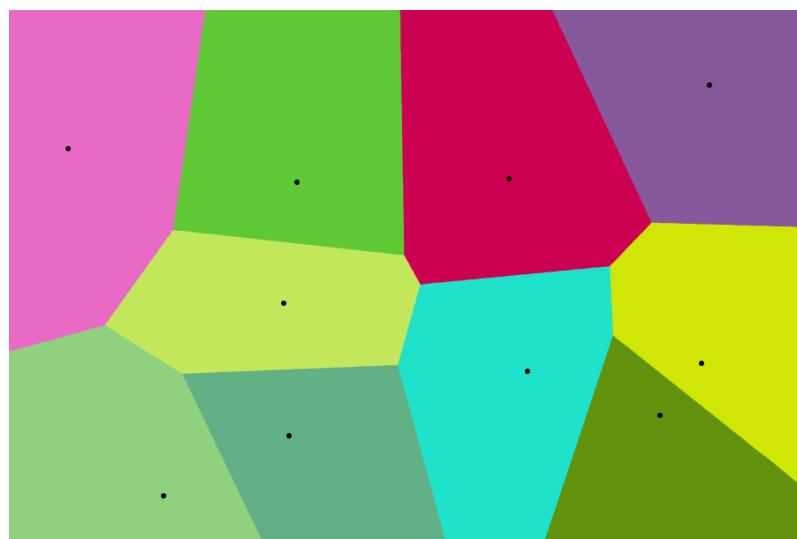
kde  $V(p)$  je plocha nově vloženého bodu a  $V_i$  je část plochy původních buněk, o kterou byly vložením buňky „okradeny“.

Váha pro jednotlivé sousedy se vypočte:

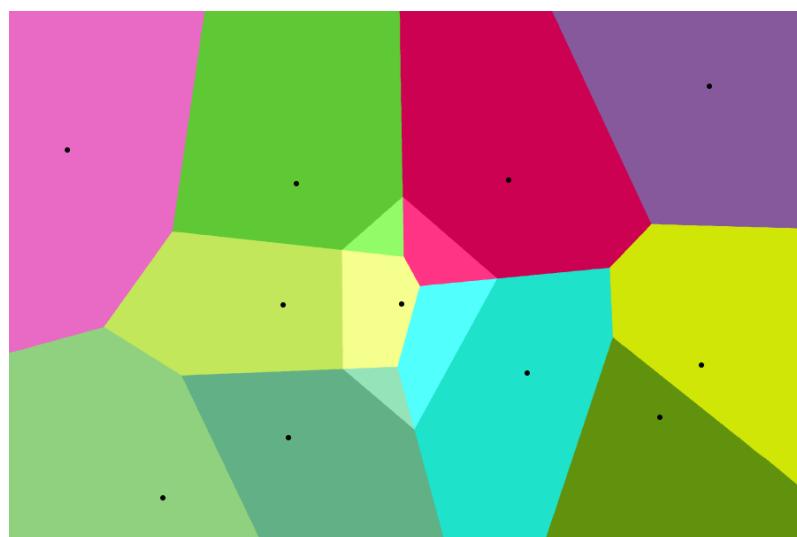
$$\lambda_i = \frac{V_i}{\sum_{i=1}^n V_i}$$

A konečně samotná hodnota pro interpolovaný bod není nic jiného než vážený průměr:

$$G(x, y) = \sum_{i=1}^n \lambda_i f(x_i, y_i)$$



Obrázek 24: Původní VD

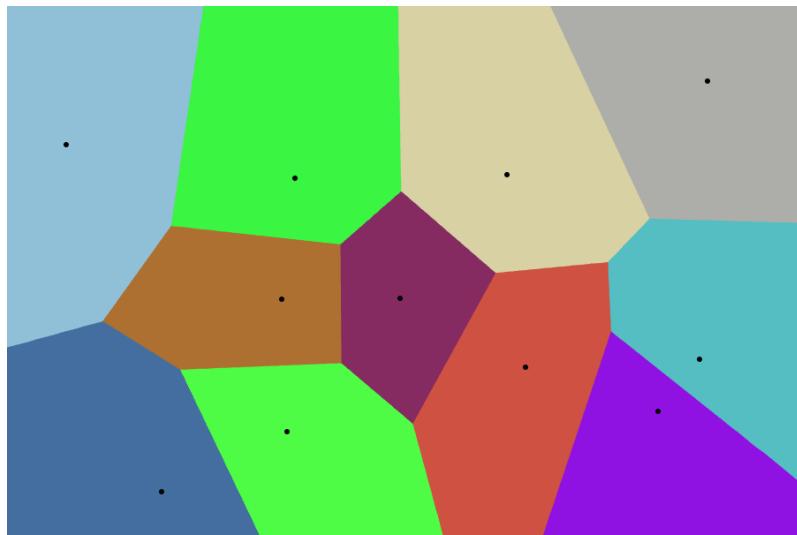


Obrázek 25: Ukradené plochy

Na obrázku 25 vidíme, že nově vzniklá buňka „ukradla“ nejvíce plochy ze žlutého polygonu, nejméně ze zeleného. Při výpočtu funkční hodnoty vloženého bodu bude mít tedy hodnota žlutého polygonu největší váhu, zatímco hodnota zeleného bodu váhu nejmenší.<sup>4</sup>

---

<sup>4</sup>Vytvořeno pomocí <http://alexbeutel.com/webgl/voronoi.html>



Obrázek 26: Nově vzniklý VD

## 5.2 Sibsonova interpolace

Sibsonova<sup>5</sup> interpolace je další metoda interpolace, která kromě vah přirozených sousedů počítá s funkcí gradientu  $\nabla_i$  pro všechny  $p_i \in P$ . Metoda počítá gradient na základě vážených nejmenších čtverců roviny procházející sousedními body. Více o tomto tématu zde: [9] [12].

## 5.3 Farinova interpolace

Profesor Gerald Farin přišel s mnohem obecnějším přístupem, který není omezen na souřadnice přirozených sousedů, ale může být použit pro všechny lokální souřadnice. Váha  $\lambda$  může být vnímána jako barycentrická souřadnice v Bezierovu simplexu. Více zde: [9] [12].

---

<sup>5</sup>Robin Sibson (naroden 1944), britský matematik. Jako první přišel s myšlenkou interpolace metodou přirozeného souseda.

## Část II

# Praktická část

## 6 Postup řešení

### 6.1 Bash

Při řešení otázky, jak implementovat metodu přirozeného souseda pro GRASS 7, se vycházelo z modulu napsaného pro GRASS GIS 6. Jednalo se o modul *v.surf.nnbathy* pro vektorová data. Tento modul byl napsán jako shellový skript. Pro novou verzi GRASS GISu 7, ve které si vývojáři kladou za cíl zpřístupnit tento software širší veřejnosti, ovšem tento shellový skript nebylo možno použít, neboť do nové verze se počítá pouze s moduly v jazyce Python a C/C++.

#### 6.1.1 *v.surf.nnbathy*

*v.surf.nnbathy*<sup>6</sup> je shellový skript. Slouží jako interface mezi příkazem *nnbathy* z externí knihovny *nn-c* a GRASS GISem. *v.surf.nnbathy* nabízí celkem tři algoritmy interpolace. Defaultně je nastaven *Watson's algorithm for Sibson natural neighbor interpolation*, viz kapitola 5.2. Další možností je *linear Delaunay interpolation*, kapitola 5.1 a poslední *Belikov and Semenov's algorithm for non-Sibsonian natural neighbor interpolation*. Pro Delaunayvou triangulaci, která je základem pro všechny tři algoritmy, se využívá knihovny *Triangle* napsanou Jonathanem Richardem Schewchukem. Parametry pro spuštění modulu jsou tyto (nepovinné v hranatých závorkách):

<b>output</b>	Proměnná typu <i>string</i> , název výstupní rastrové mapy.
<b>input</b>	Proměnná typu <i>string</i> , název vstupní vektorové mapy.
<b>[file]</b>	Proměnná typu <i>string</i> , název vstupního souboru.
<b>[column]</b>	Proměnná typu <i>string</i> , název sloupce z atributové tabulky, jehož data budou použity pro interpolaci.
<b>[layer]</b>	Proměnná typu <i>integer</i> , nastavení, zda se jedná od 2D nebo 3D vektorová data.
<b>[where]</b>	Proměnná typu <i>string</i> , SQL where podmínka.
<b>[alg]</b>	Proměnná typu <i>string</i> , název použitého algoritmu.

---

<sup>6</sup><http://grass.osgeo.org/grass64/manuals/addons/v.surf.nnbathy.html>

Volání v příkazové řádce pak může vypadat například takto:

Listing 2: bash version

---

```
user@my_comp:~$ v.surf.nnbathy input=input_vector_map  
output=output_raster_map zcolumn=value alg=nn
```

---

## 6.2 Python

Jako první krok pro implementaci interpolace přirozeného souseda pro GRASS GIS 7 bylo potřeba stávající shellový skript přepsat do podporovaného programovacího jazyka. Pro verzi 7 bylo možné přepsat modul buďto do jazyka C/C++ nebo Python. Z důvodu nepříliš velké zkušenosti s programováním byl pro začátek zvolen jazyk Python.

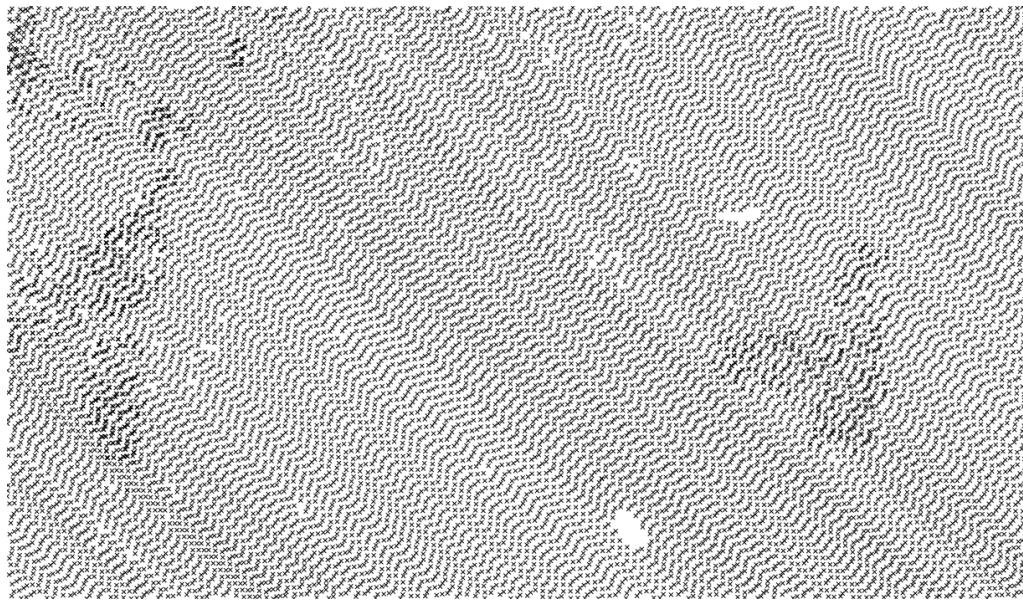
Nicméně ačkoliv byl modul přepsán do Pythonu, takže už mohl být použit i pro GRASS GIS 7, stále přetrvávala závislost na knihovně *nn-c*.

### 6.2.1 v.surf.nnbathy.py

V následující části této práce bude popsáno, jak Python modul funguje, jaká jsou vstupní a výstupní data, jaké vytváří dočasné soubory.

**Vstupní data** Stejně jako původní shellový skript, i tento modul pracuje se vstupními daty buď v podobě textového ASCII souboru nebo vektorové mapy.

V případě použití vstupní vektorové mapy s body je pak při volání modulu použit parametr *column*, který určuje z jakého sloupce atributové tabulky se budou brát hodnoty k interpolaci.



Obrázek 27: Vektorová mapa na vstupu

Druhou možností je použít textový ASCII soubor, který musí obsahovat  $n$  bodů na  $n$  řádcích ve čtyřech sloupcích. V prvních dvou sloupcích je uložen údaj o poloze v podobě x a y souřadnice. Ve třetím sloupci jsou pak uloženy hodnoty veličiny, kterou chceme interpolovat a v posledním sloupci nalezneme id bodu.

Listing 3: Příklad vstupního souboru

---

```
639524.309 221227.596 112.191 101960
639529.088 221223.045 112.225 101961
639529.152 221219.826 112.029 102115
639524.303 221224.435 112.212 102116
639519.444 221229.043 111.752 102117
639513.546 221229.699 111.472 102440
639518.307 221225.172 111.874 102441
639523.114 221220.597 112.243 102442
```

---

**Funkce `region()`** Každá operace prováděná v GRASS GISu je prováděna pouze na určitém rozsahu území, tzv. *výpočetním regionu*. *Výpočetní region* je určen jako obdélník daný mezními zeměpisnými souřadnicemi a počtem řádků a sloupců. Funkce `region()` všechna nastavení uloží do proměnných. Dále vypočte plochu výpočetního

regionu. Na rozdíl od GRASS GISu, který jako mezní kartografické souřadnice bere vnější rohy rohových buněk obdélníku, knihovna *nn-c* používá středy rohových buněk, a proto je třeba nastavení výpočetního regionu opravit o rozlišení buněk.

**Funkce `initials_controls()`** Ve chvíli, kdy je nastavený výpočetní region, můžeme provést úvodní kontroly a přípravy před samotným výpočtem. Zejména zda plocha výpočetního regionu není nulová a je kde provádět interpolaci. Dále je třeba zajistit jednoznačné určení vstupních dat, tedy zda se bude pracovat s ASCII souborem, nebo s vektorovou mapou, a jejich kontrolu, popřípadě SQL podmínu. Také kontrolujeme, zda z knihovny *nn-c* máme nainstalovaný program *nnbathy*, který interpolaci provádí. Též je třeba vytvořit dočasné pomocné soubory, které využijeme při práci s daty.

V případě, že pracujeme s vektorovou mapu, uložíme informace o bodových datech do dočasného proměnné *TMPcat* pomocí modulu *v.out.ascii*. Výstupem toho modulu je ASCII soubor o *n* řádcích a čtyřech sloupcích. V prvních dvou sloupcích je uložená poloha bodu, ve třetím jeho id a ve čtvrtém hodnota k interpolaci.

Listing 4: Pomocný soubor TMPcat

---

638234.122902785427868	221198.4894384436775	1	62.8177820000000001
638755.665974545176141	220976.783764891704777	2	9.1904880000000000
638729.530741120455787	219988.669646041787928	3	91.7999520000000005
638088.941303733270615	220228.186909802345326	4	76.839046999999994
638158.578729554312304	220794.514421981060877	5	2.0370010000000000
637781.724264170858078	219988.193243810994318	6	8.2989770000000001
638359.847223712014966	220692.375897407706361	7	15.5503260000000000
639137.670258715632372	221096.622500944242347	8	16.6130549999999999

---

Jelikož id bodu k dalším výpočtům nepotřebujeme do dočasné proměnné *TMPXYZ* si uložíme pouze informace o poloze a hodnotu k interpolaci. V případě, že nepracujeme s vektorovou mapou, ale ASCII souborem, tak tento soubor rovnou uložíme do proměnné *TMPXYZ*.

Listing 5: Pomocný soubor TMPXYZ

---

638234.122902785427868	221198.489438443677543	62.8177820000000001
638755.665974545176141	220976.783764891704777	9.1904880000000000
638729.530741120455787	219988.669646041787928	91.799952000000005
638088.941303733270615	220228.186909802345326	76.839046999999994
638158.578729554312304	220794.514421981060877	2.0370010000000000
637781.724264170858078	219988.193243810994318	8.298977000000001
638359.847223712014966	220692.375897407706361	15.5503260000000000
639137.670258715632372	221096.622500944242347	16.6130549999999999

---

**Funkce compute()** V této části kódu je volán program *nnbathy* s následujícími vstupními parametry:

- w proměnná typu *double*, omezuje extrapolaci přiřezením minimální váhy pro vrchol Delaunayovi sítě. Defaultně je nastavena nula, což zamezuje extrapolaci.
- i proměnná typu *string*, název vstupního souboru o *n* řádcích, se třemi sloupcí, x a y souřadnicí a hodnotou k interpolaci
- x dvojice *x<sub>min</sub>*, *x<sub>max</sub>* typu *double*, mezní hodnoty výstupní mřížky
- y dvojice *y<sub>min</sub>*, *y<sub>max</sub>* typu *double*, mezní hodnoty výstupní mřížky
- P proměnná typu *string*, použitá metoda interpolace
- n dvojice *double* x *double*, rozlišení výstupní mřížky

Výstupem z *nnbathy* je soubor *XYZout*. Obsahuje data o výstupní mřížce buňku po buňce ve třech sloupcích. V prvních dvou jsou x a y souřadnice, ve třetím vyinterpolovaná hodnota. V případě buňek mimo oblast, kde probíhala interpolace, je ve třetím sloupci uložena hodnota NaN.

Listing 6: XYZout

---

```
637725 221045 NaN
637735 221045 NaN
637745 221045 NaN
637755 221045 NaN
637765 221045 NaN
637775 221045 23.2274425578696
637785 221045 20.3234644594092
637795 221045 23.6841650075168
```

---

**Funkce convert()** Výstupní textový soubor z *nnbathy* je třeba upravit, aby s ním bylo možné dále pracovat v GRASS GISu. Pro další práci slouží dočasný soubor *TMP*. Při vytváření na začátku tohoto souboru vznikne hlavička, která obsahuje data o hraničních souřadnicích, prostorovém rozlišení, datovém typu a hodnotě žádná data (null či NaN).

Listing 7: Hlavička souboru TMP

---

```
north: 228495.0
south: 215005.0
east: 644995.0
west: 630005.0
rows: 1350
cols: 1500
type: double
null: NaN
```

---

Dále je potřeba vybrat vyinterpolované hodnoty jednotlivých buněk ze souboru *XYZout*, kde jsou uloženy ve třetím sloupci na samostatných řádcích, a vložit je do souboru *TMP* v pravidelné mřížce.

---

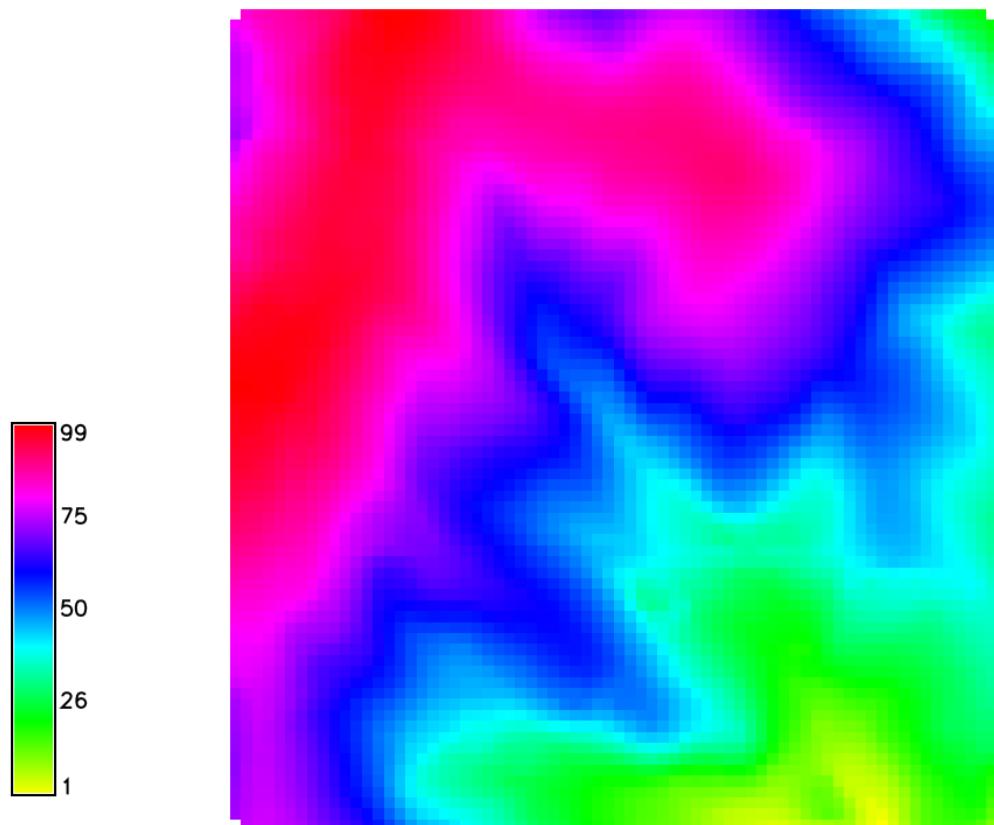
```
NaN NaN NaN NaN NaN NaN 20.7543487721 21.6412738464 21.0697741691
NaN NaN NaN 22.0420316367 21.2339890563 22.1237145752 20.8684520153
NaN NaN NaN 21.1184005087 21.2163721800 22.5528243734 20.6780204446
NaN NaN NaN 20.1954939692 22.8280745872 21.2507738978 21.6682239762
NaN NaN NaN 20.8488929136 21.3643657768 21.6917150115 20.9860960516
NaN NaN 20.3606443297 21.3323845603 22.8051229299 21.8176089189
```

```
NaN NaN 20.3078295678 20.6627164743 21.8458254363 21.2712885642  
21.6370709432 21.1827185754 21.6464451799 21.2912057725 NaN NaN NaN NaN  
22.6809840563 21.6959007615 20.1153796499 20.1067779317 NaN NaN NaN NaN  
22.3064205177 20.8191889879 20.8209937444 22.1952518030 NaN NaN NaN NaN  
NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN
```

---

**Funkce import\_to\_raster()** Soubor *TMP*, ve kterém máme uložené vyinterpolované hodnoty v pravidelné mřížce, slouží jako vstupní soubor pro GRASS modul *r.in.ascii*, který tyto hodnoty převede do rastru. Dále je použit modul *r.support*, který uloží příkazy do rastrových metadat. Na závěr je vytisknuta zpráva, že byla vytvořena rastrová mapa.

**Výstupní data** Výstupem modulu *v.surf.nnbathy* je rastrová mapa.



Obrázek 28: Výstupní rastrová mapa

### 6.3 OOP — Objektově orientované moduly

V průběhu práce na Python modulu *v.surf.nn bathy* bylo zjištěno, že v GRASS GISu verze 6 je k dispozici také shellový skript *r.surf.nn bathy*, který pracuje s rastrovými daty. Jelikož větší část kódu byla pro moduly *v.surf.nn bathy* a *r.surf.nn bathy* společná, byla hlavní výpočetní část spojena. Na místo dvou procedurálních modulů byla objektově vytvořena knihovna *nn bathy.py* (příloha D.1) a dva moduly *v.surf.nnabthy.py* (příloha D.2) pro vektorová data a *r.surf.nnabthy.py* (příloha D.3) pro data rastrová, které knihovnu *nn bathy* volají.

#### 6.3.1 v.surf.nn bathy

Z nového objektově orientovaného modulu pro vektorová data *v.surf.nn bathy* tak zmizela výpočetní část kódu. Zůstala úvodní část, která automaticky generuje GUI, úvodní vstupní kontroly a dále podmínka, která vyhodnocovala, zda vstupují data v podobě vektorové mapy nebo ASCII souboru.

#### 6.3.2 r.surf.nn bathy

Modul *r.surf.nn bathy* pracuje na podobném principu jako modul *v.surf.nn bathy*, jen pro rastrová data. Při volání je možnost použít méně parametrů.

**output** Proměnná typu *string*, název výstupní rastrové mapy.

**input** Proměnná typu *string*, název vstupní vektorové mapy.

**[alg]** Proměnná typu *string*, název použitého algoritmu.

I v tomto modulu zůstala část, která vytváří GUI. Protože ale modul pracuje s daty pouze v podobě rastrové mapy, nebyly potřeba žádně vstupní kontroly.

#### 6.3.3 nn bathy

V knihovně *nn bathy*, (příloha D.1) kterou oba moduly volají, tedy zůstala hlavní výpočetní část kódu.

**Rodičovská třída Nnbathy** V rodičovské třídě Nnbathy bylo definováno několik metod, které přebral funkce jednotlivých procedur z předchozího kódu. Navíc byly vytvořeny tyto metody:

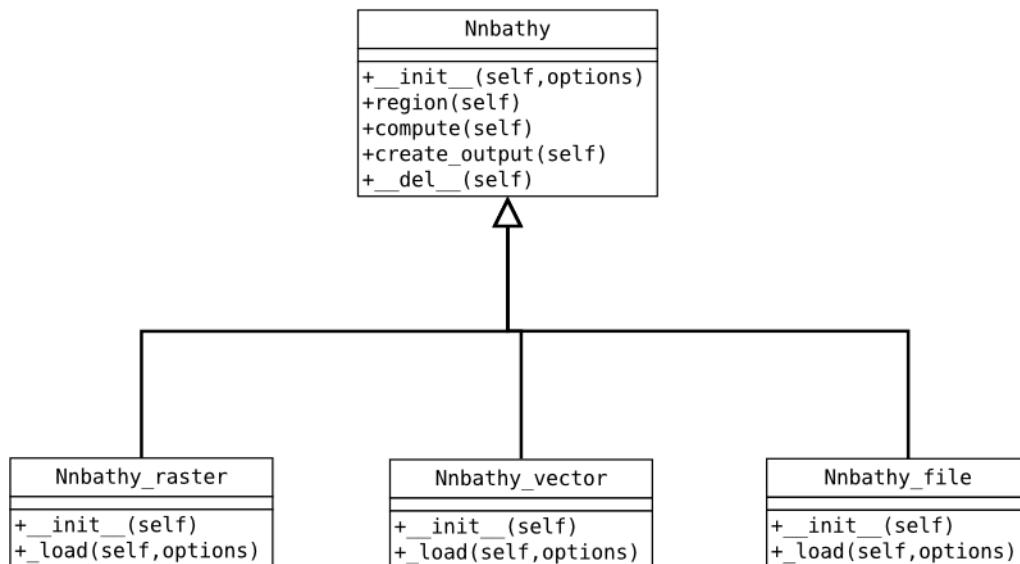
**Metoda `__init__`** Tato metoda (konstruktor) inicializuje dočasné soubory a volá metodu `region()`.

**Metoda `__del__`** Tato metoda (destruktor) slouží k odstranění dočasných souborů.

**Třída `Nnbathy_raster`** Tato třída slouží pro rastrová vstupní data, její rodičovská třída je `Nnbathy`.

**Třída `Nnbathy_vector`** Tato třída slouží pro vektorová vstupní data, její rodičovská třída je `Nnbathy`.

**Třída `Nnbathy_file`** Tato třída slouží pro vstupní data v podobě ASCII souboru, její rodičovská třída je `Nnbathy`.



Obrázek 29: UML diagram

## 7 Možnosti budoucí implementace

V současné době jsou oba moduly k dispozici v rámci balíčků Add-Ons<sup>7</sup>, viz kapitola 1.1. V adresáři *grass7/vector/v.surf.nnbathy* můžeme naleznout hlavní knihovnu *nnbathy.py*, modul *v.surf.nnbathy.py* a dokumentaci *v.surf.nnbathy.html*.

V adresáři *grass7/raster/r.surf.nnbathy* naopak nalezneme modul *r.surf.nnbathy.py* a jeho dokumentaci *v.surf.nnbathy.html*. Oba moduly je možné instalovat do GRASS GISu 7, pomocí modulu *g.extension* nebo pomocí GUI.

---

Listing 8: Stáhnutí modulu v.surf.nnbathy pomocí g.extension

---

```
g.extension extension=v.surf.nnbathy
```

---

Jelikož interpolace metodou přirozeného souseda poskytuje spojité, hladké výstupy bez extrapolovaných hodnot může být její použití v praxi poměrně využívané. Právě proto je snaha dostat tento interpolační nástroj do základní distribuce GRASS GISu. Tomu však brání vnitřní závislost na knihovně *Triangle*, která nespadá pod volně šířitelnou licenci GNU GPL. V následujících kapitolách rozebereme možnosti vyhnutí se této knihovně a následné implementace.

---

<sup>7</sup><http://grass.osgeo.org/download/addons/>

## 7.1 Knihovna Triangle

Knihovna Triangle verze 1.4<sup>8</sup> byla napsaná Jonathanem Richardem Shewchuk v roce 2002. Bohužel není distribuována pod volně šířitelnou licencí, která by dovolila její začlenění do oficiální distribuce GRASS GISu. V programu *nnbathy* má na starosti Delaunayovu triangulaci. Je tedy třeba najít vhodný způsob, jak tuto knihovnu nahradit.

Triangle pro tvorbu triangulace využívá Divide and Conquer algoritmus, viz kapitola 3.5, v kombinaci s Inkrementálním algoritmem, kapitola 3.3. Pro ukládání dat využívá svoji vlastní, poměrně obsáhlou datovou strukturu *delaunay* (Příloha B), do které si ukládá hraniční body regionu, všechny body triangulace, všechny trojúhelníky s hranami a jejich opsanými kružnicemi, včetně jejich sousedů.

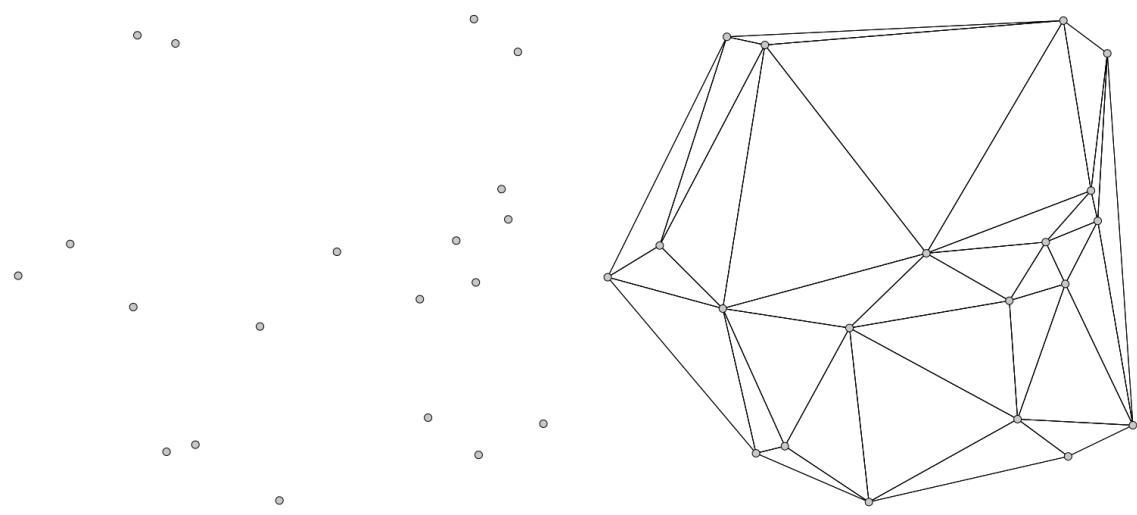
Tuto datovou strukturu *delaunay* dále využívá i samotný program *nnbathy*, který provádí interpolaci metodou přirozeného souseda. Právě převod na tuto datovou strukturu bude v budoucnosti klíčový v případě, že se podaří najít vhodný triangulační nástroj, který nahradí knihovnu Triangle.

---

<sup>8</sup><http://www.cs.cmu.edu/~quake/triangle.html>

## 7.2 Modul *v.delaunay*

Modul *v.delaunay*<sup>9</sup> je součástí oficiální distribuce GRASS GISu. Tento modul poskytuje Delaunayovu triangulaci nad vektorovou mapou s body nebo centroidy. Pro vytvoření triangulace používá modul algoritmus Rozděl a panuj, viz kapitola 3.5. Data ukládá do datové struktury winged-edge, kapitola 4.4, též příloha C.



Obrázek 30: Množina bodů pro triangulaci

Obrázek 31: Triangulace pomocí *v.delaunay*

Ovládání modulu je velmi jednoduché na výběr máme pouze z několika přepínačů a několika parametrů:

Přepínače: **-r** Počítej pouze s body v aktuálním regionu.

**-l** Vytvoř výstupní triangulaci pomocí linií, ne ploch.

**input** Povinný parametr. Název vstupní vektorové mapy.

Parametry: **[layer]** Název nebo číslo vrstvy. Slouží k připojení do databáze.

**output** Povinný parametr. Název výstupní vektorové mapy.

Listing 9: Volání modulu *v.delaunay* z příkazové řádky

---

```
v.delaunay input=input_map output=output_map
```

---

<sup>9</sup><http://grass.osgeo.org/grass70/manuals/v.delaunay.html>

### 7.3 Převod mezi strukturami

Z hlediska rychlosti se modul *v.delaunay* jeví jako adekvátní náhrada za knihovnu *Triangle*. Při testech bylo dokázáno, že pro náhodné konfigurace miliónu bodů dokáže triangulaci provést za méně než 15 s, což se dá považovat za dostatečně efektivní.

Jako problémová se jeví část převodu dat mezi strukturami. *v.delaunay* používá strukturu *edge* (příloha C), což je běžná datová struktura typu winged-edge, viz kapitola 4.4. Naproti tomu *Nnbathy* počítá interpolaci nad daty uloženými ve struktuře *delaunay* (Příloha B), která je poměrně netypická, protože data ukládá do polí za pomocí pointerů a indexů zároveň.

Nicméně naplnění struktury *delaunay* možné je, byť vyžaduje několikerý průchod přes všechny hrany triangulace. Na jeden průchod je možné získat celkový počet bodů *n*, získat *id* jednotlivých trojúhelníků, *id* jejich vrcholů a vypočítat středy opsaných kružnic. Dále je nutné označit pro každý trojúhelník jeho sousedy, tedy trojúhelníky, které s ním sdílí hranu. Opět je třeba projít celou triangulaci. Během tohoto průchodu je pravděpodobné, že se dostaneme k již zpracovanému trojúhelníku. Proto bude třeba vytvořit např. pomocný vektor typu boolean, do kterého se bude ukládat, zda trojúhelník už byl zpracován či nikoliv. Nejsložitější je naplnění struktur *n\_point\_triangles* a *point\_triangles* (viz Příloha B). Zde je ke každému bodu přiřazen počet trojúhelníků, do kterých náleží a jejich *id*. Opět je zapotřebí několikerý průchod strukturou winged-edge a pomocná struktura, do které budeme ukládat seznam *previous edge*.

Po delším zkoumání tedy bylo zjištěno, že převod mezi strukturami možný je. Nicméně časová i výpočetní náročnost společně s faktem, že se jedná o dost „kostrbaté“ řešení, ukázaly, že samotná náhrada knihovny *Triangle* jiným nástrojem, který bude zajišťovat DT, není správnou cestou. Největším problémem tohoto řešení je, že *Nnbathy* využívá strukturu *delaunay*, která je poměrně netypická, takže je hůře kompatibilní s jakýmkoliv jinými triangulačními nástroji, které používají běžné datové struktury.

Pro další řešení se tedy hledal takový nástroj, který by se nestaral pouze o DT, ale i o samotnou interpolaci. Tak by se bylo možné vyhnout závislosti nejen na triangulačním nástroji, ale i na knihovně *nn-c*.

## 7.4 Knihovna CGAL

Knihovna CGAL (The Computational Geometry Algorithms Library)<sup>10</sup> je open source projekt, který poskytuje snadný přístup k efektivním a spolehlivým geometrickým algoritmům ve formě C++ knihovny. CGAL se používá v různých oblastech, které potřebují geometrické výpočty, jako např. v GISech, CADu, molekulární biologii, počítačové grafice a robotice. Knihovna CGAL je distribuována pod licencí GNU GPL, což perfektně vyhovuje pro jeho použití pro GRASS GIS.

Jeden z balíčků této knihovny se přímo zabývá interpolací obecně ve 2D i 3D, implementací funkcí na výpočet souřadnic přirozeného souseda a různými metodami interpolace pro roztroušená prostorová data, z nichž většina je založena právě na metodě přirozeného souseda. K dispozici jsou celkem 4 interpolační metody a sice tyto:

- Linear Precision Interpolation (kapitola 5.1)
- Sibson's Continuous Interpolant (kapitola 5.2)
- Farin's Continuous Interpolant (kapitola 5.3)
- Quadratic Precision Interpolants

### 7.4.1 v.surf.nn

V současné chvíli probíhá vývoj nového modulu *v.surf.nn* (příloha D.4), který využívá knihovny *CGAL*. Modul zatím používá metodu *Linear Precision Interpolation*, tedy pouze lineární interpolaci bez ohledu na gradient, viz kapitola 5.1. Do budoucna se však uvažuje o přidání i ostatních metod a možnosti nechat uživateli volbu, kterou metodu chce použít.

Momentálně je modul *v.surf.nn* funkční a provádí jak DT (kapitola 2.3), tak lineární interpolaci metodou přirozeného souseda (kapitola 5.1). Modul je tedy zbavený závislosti na knihovně *nn-c* a hlavně na knihovně *Triangle*. Jedná se však pouze o experimentální modul, neboť možnosti knihovny *CGAL* musí být ještě hlouběji prozkoumány a na vývoji modulu je třeba strávit ještě mnoho času.

---

<sup>10</sup><http://www.cgal.org>

Tento experimentální modul *v.surf.nn* má zatím dva zásadní problémy, kterými se bude zabývat další vývoj. Za prvé je to jeho rychlosť. Jak je vidět v příloze A, rychlosť modulu *v.surf.nn* je několikanásobně pomalejší než modulu *v.surf.nnbathy* nebo nástroje *Natural neighbor interpolation* v ArcGISu. Při řešení tohoto problému se hledalo, která část kódu nejvíce brzdí celý modul. Po analýze kódu bylo objeveno, že kritická část je volání funkcí *CGAL::natural\_neighbor\_coordinates\_2()* a *CGAL::linear\_interpolation()*. Problém je, že vstupním parametrem po obě tyto funkce je pouze jediný bod, v našem případě jediná buňka rastru. To znamená, že třeba v případě uvedeném v příloze A, kde je modul volán nad územím deset × deset kilometrů s rozlišením buňky deset metrů, je každá funkce volána celkem milionkrát. Snahou dalšího vývoje tedy bude nalézt možnost, jak zpracovávat data dávkově a zrychlit tak celý proces.

Druhým problémem je rastrový výstup, který modul *v.surf.nn* tvoří. Jak je zřejmé z přílohy A, výstupní rastrová mapa modulu *v.surf.nn* je naprosto odlišná od výstupních map jak modulu *v.surf.nnbathy*, tak i ArcGIS nástroje *Natural Neighbor Interpolation*. Tento problém je naprosto zásadní, a pokud nebude vyřešen, tak se nedá uvažovat o zařazení modulu *v.surf.nn* mezi ostatní moduly GRASS GISu. V takovém případě by se ukázalo použití knihovny *CGAL* jako slepá ulička.

V době odevzdání této práce ještě nebyly odhaleny příčiny, proč modul interpoluje chybně. Na oba výše zmíněné problémy byla upozorněna i komunita vývojářů CGALu a to jednak pomocí mailing listu<sup>11</sup> *cgal-discuss@inria.fr*, dále pak pomocí webové stránky *StackOverflow*<sup>12, 13</sup>.

---

<sup>11</sup><http://cgal-discuss.949826.n4.nabble.com/>

<sup>12</sup><http://stackoverflow.com/questions/30354284/cgal-natural-neighbor-interpolation>

<sup>13</sup><http://stackoverflow.com/questions/30379735/cgal-linear-interpolation-different-output>

## Závěr

Cílem této bakalářské práce byla implementace nástroje pro interpolaci metodou přirozeného souseda do GRASS GISu. Na začátku byly tedy v Pythonu napsány dva procedurální moduly *v.surf.nn bathy* pro vektorová data a *r.surf.nn bathy* pro rastrová data.

Následně byly oba moduly přepsány jako objektově orientované a hlavní výpočetní část byla přesunuta do knihovny *nnbathy*, kterou oba moduly volají. Tyto moduly jsou v současnosti k dispozici v GRASS GISu v rámci balíčků Add-Ons a uživatel může po instalaci těchto rozšíření bez problému využívat interpolaci metodou přirozeného souseda. Nicméně tyto moduly jsou stále závislé na knihovně *nn-c*, která provádí samotnou interpolaci a hlavně na knihovně *Triangle*, která se stará o provedení DT.

Dále se tato práce zabývala možností využít modulu *v.delaunay*, který je už v oficiální distribuci GRASS GISu, jako náhradu za knihovnu *Triangle*. V rámci zjišťování, zda je možné modul *v.delaunay* použít, byly detailně prostudovány jednotlivé algoritmy, jak DT provést. Dále byly podrobně zkoumány datové struktury, které se pro ukládání triangulací běžně využívají. Všechny tyto poznatky byly zpracovány v teoretické části v kapitolách 3 a 4. Nicméně se ukázalo, že ačkoliv náhrada knihovny *Triangle* za modul *v.delaunay* je možná, tak převod mezi jednotlivými strukturami je příliš náročný a neefektivní, zejména kvůli netypické struktuře *delaunay*, kterou knihovna *nn-c* využívá.

Jako další řešení se tedy nabízelo použití knihovny *CGAL*. Tato knihovna dokáže nejen vytvořit Delaunayho triangulaci, ale nabízí i několik metod pro interpolaci metodou přirozeného souseda. Díky použití knihovny *CGAL* dojde k zbavení se závislosti na knihovnách *nn-c* a *Triangle*.

V současné chvíli je hotová experimentální verze modulu *v.surf.nn*, který sice funguje, nicméně je velmi pomalý a zdá se, že jeho výstupy nejsou přesné. Řešení těchto problémů je otázkou budoucího vývoje, na problémy byla upozorněna i komunita vývojářů CGALu.

V rámci této bakalářské práce bylo též provedeno porovnání výkonu a výstupu mezi GRASS GISem, zástupcem open source softwaru, a Esri ArcGIS, proprietárním softwarem. Za GRASS GIS byl testován modul *v.surf.nn bathy* se závislostí na

knihovně *nn-c* a dále modul *v.surf.nn*, který je postavený na knihovně *CGAL*. Za Esri ArcGIS byl testován nástroj *Natural Neighbour*, který se nachází v toolboxu Spatial Analyst. Výsledky jsou uvedeny v Příloze A.

## Seznam použitých zkraték

<b>CGAL</b>	The Computational Geometry Algorithms Library
<b>DMT</b>	Digitální model terénu
<b>DT</b>	Delaunayova triangulace
<b>DPZ</b>	Dálkový průzkum Země
<b>GIS</b>	Geographic Information System (Geografický informační systém)
<b>GNU GPL</b>	GNU General Public License
<b>GUI</b>	Graphical User Interface (Grafické uživatelské rozhraní)
<b>GRASS</b>	Geographical Resources Analysis Support System
<b>HW</b>	Hardware
<b>OS</b>	Operační systém
<b>QGIS</b>	Quantum GIS
<b>UML</b>	Unified Modeling Language
<b>USA-CERL</b>	US Army Construction Engineering Research Laboratories
<b>VD</b>	Voronoiův diagram

## Použité zdroje

- [1] HJELLE, Øyvind; DÆHLEN, Morten. *Triangulations and applications*. Berlin: Springer, 2010. 234 s. ISBN 978-3-642-06988-8.
- [2] SOCHOR, Jiří; ŽÁRA, Jiří. *Algoritmy počítačové grafiky*. Praha: České vysoké učení technické, 1994. 258 s. ISBN 978-8-001-00949-9.
- [3] GRASS Development Team. *GRASS 7 Programmer's Manual* [online]. URL: <<http://grass.osgeo.org/programming7>>
- [4] *GRASS GIS Tracker and Wiki* [online]. URL: <<http://trac.osgeo.org/grass>>
- [5] NETELER, Markus. *Praktická rukověť ke geografickému informačnímu systému GRASS* [online]. URL: <[http://geo.fsv.cvut.cz/data/grasswikicz/grass\\_prirucka/grass\\_prirucka\\_0.4.pdf](http://geo.fsv.cvut.cz/data/grasswikicz/grass_prirucka/grass_prirucka_0.4.pdf)>
- [6] BAYER, Tomáš. *Rovinné triangulace a jejich využití* [online]. URL: <<https://web.natur.cuni.cz/~bayertom/Adk/adk5.pdf>>
- [7] BAYER, Tomáš. *Voronoi diagram* [online]. URL: <<https://web.natur.cuni.cz/~bayertom/Adk/adk5.pdf>>
- [8] KOHOUT, Josef. *Paralelní Delaunayova triangulace ve 2D a 3D* [online]. URL: <[http://graphics.zcu.cz/files/DP\\_2002\\_Kohout\\_Josef.pdf](http://graphics.zcu.cz/files/DP_2002_Kohout_Josef.pdf)>
- [9] BOBACH, Tom; UMLAUF, Georg. *Natural Neighbor Interpolation and Order of Continuity* [online]. URL: <<http://www-umlauf.informatik.uni-kl.de/~bobach/work/publications/dagstuhl06.pdf>>
- [10] HARMAN, Chris; JOHNS, Mike. *Voronoi Natural Neighbors Interpolation* [online]. URL: <[http://web.cs.swarthmore.edu/~adanner/cs97/s08/papers/harman\\_johns.pdf](http://web.cs.swarthmore.edu/~adanner/cs97/s08/papers/harman_johns.pdf)>
- [11] LEE, D. T.; SCHACHTER, B. J. *Two Algorithms for Constructing a Delaunay Triangulation*. International Journal of Computer and Information Sciences, Vol. 9, No. 3, 1980 [online] URL: <[http://www.personal.psu.edu/faculty/c/x/cxc11/AERSP560/DELAUNAY/13\\_Two\\_algorithms\\_Delauney.pdf](http://www.personal.psu.edu/faculty/c/x/cxc11/AERSP560/DELAUNAY/13_Two_algorithms_Delauney.pdf)>
- [12] CGAL. *Documentation and User Manual* [online]. URL: <<http://doc.cgal.org/latest/Interpolation/index.html#secinterpolation>>

- [13] WIKIPEDIE. *Boris Delaunay* [online]. URL: <[http://en.wikipedia.org/wiki/Boris\\_Delaunay](http://en.wikipedia.org/wiki/Boris_Delaunay)>
- [14] WIKIPEDIE. *Natural neighbor* [online]. URL: <[http://en.wikipedia.org/wiki/Natural\\_neighbor](http://en.wikipedia.org/wiki/Natural_neighbor)>

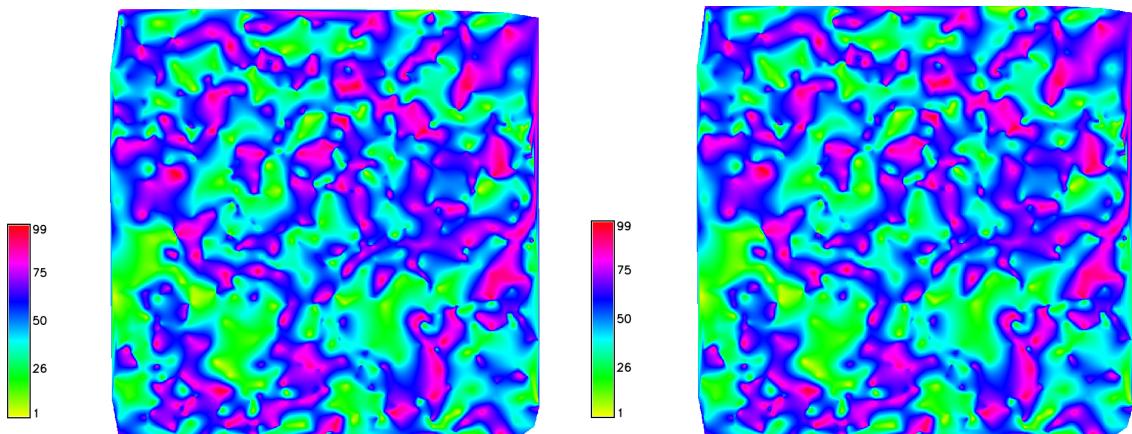
## A Srovnání GRASS GIS a Esri ArcGIS

Následující kapitola se bude věnovat srovnání GRASS GISu a ArcGISu při interpolaci metodou přirozeného souseda. GRASS GIS bude zastupovat Python modul *v.surf.nn*, o kterém byla řeč v kapitole 6.2, dále modul *v.surf.nn* využívající CGAL, viz kapitola 7.4.1. Za ArcGIS budeme zkoumat nástroj *Natural Neighbour*, který se nachází v toolboxu *Spatial Analyst*.

Pro porovnání byly zvoleny tři situace. Na území o velikosti deset × deset kilometrů, s rozlišením buňky deset metrů (počet buněk celkem tedy jeden milión) byly vygenerována bodová pole o počtu tisíc, sto tisíc a milion bodů. Nad těmito soubory byla po té provedena interpolace všemi třemi způsoby. Interpolace byla pro každou datovou sadu provedena v deseti opakováních, výsledný čas je průměrem. Testy byly provedeny na notebooku HP Pavilion dv6000 s procesorem Intel®Core™2 Duo CPU T5550 @ 1.83GHz × 2 a RAM pamětí 4 GB. Výsledky jsou uvedeny v tabulce 6.

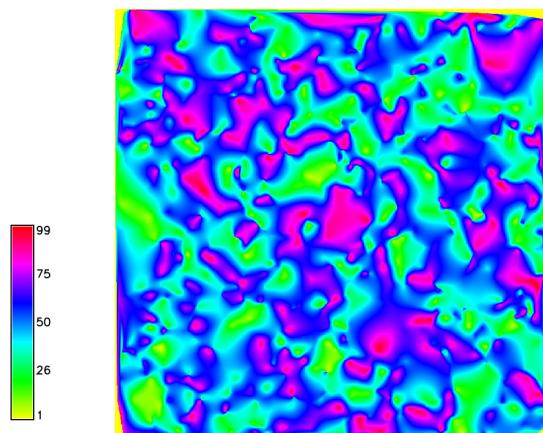
Počet bodů	<i>v.surf.nn</i>	<i>v.surf.nn</i>	ArcGIS
1 000	13,5 s	1 min 6 s	7,6 s
100 000	58,9 s	9 min 52 s	14,1 s
1 000 000	8 min 25 s	netestováno	2 min 13 s

Tabulka 6: Porovnání časů

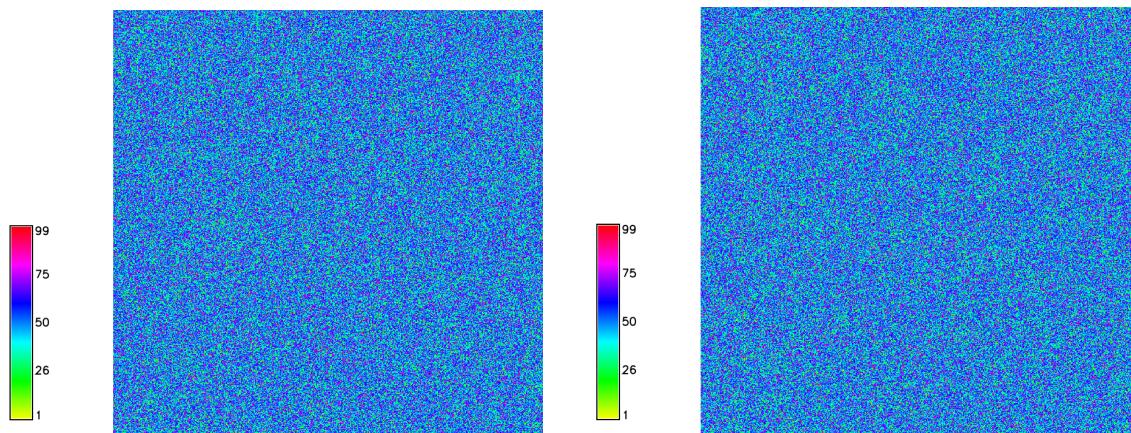
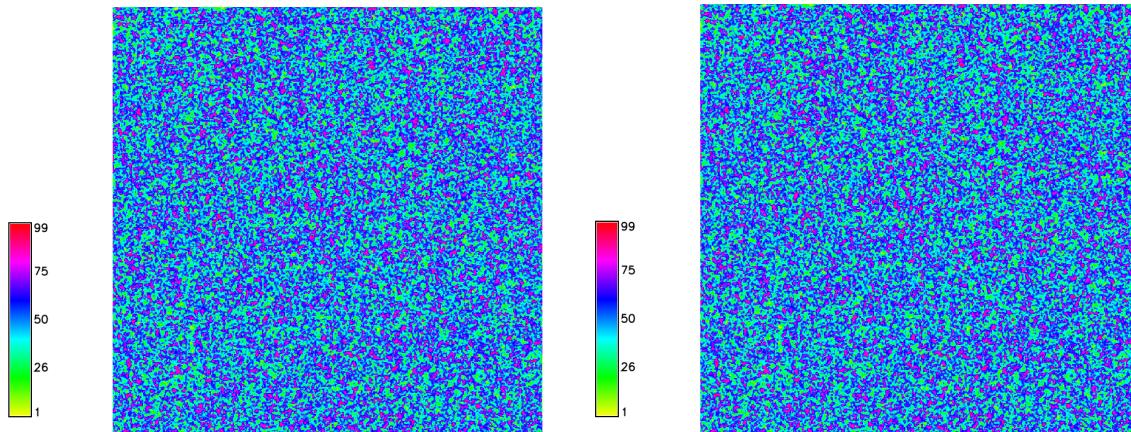


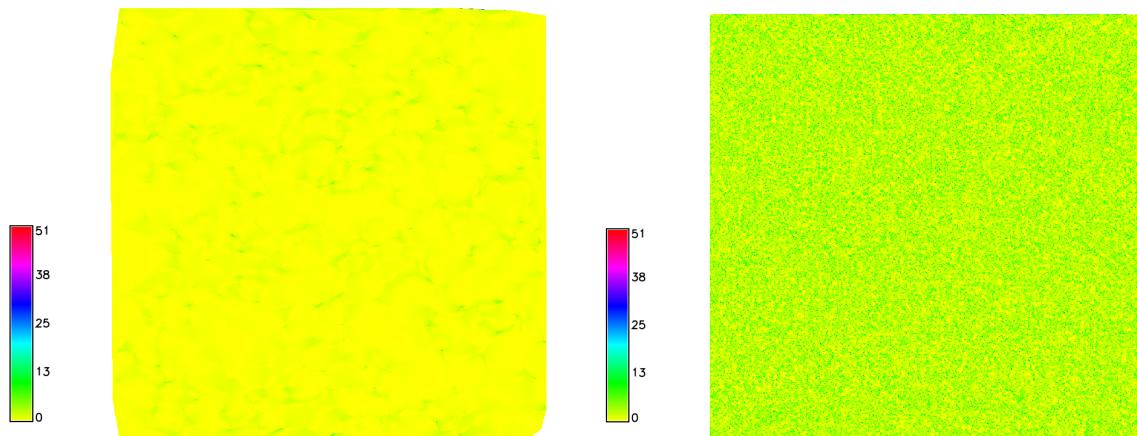
Obrázek 32: ArcGIS: Výstupní rastr pro 1 000 bodů

Obrázek 33: GRASS GIS, v.surf.nnbathy: Výstupní rastr pro 1 000 bodů



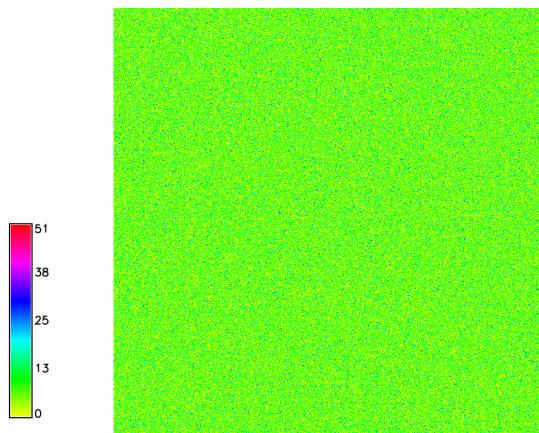
Obrázek 34: GRASS GIS, v.surf.nn: Výstupní rastr pro 1 000 bodů





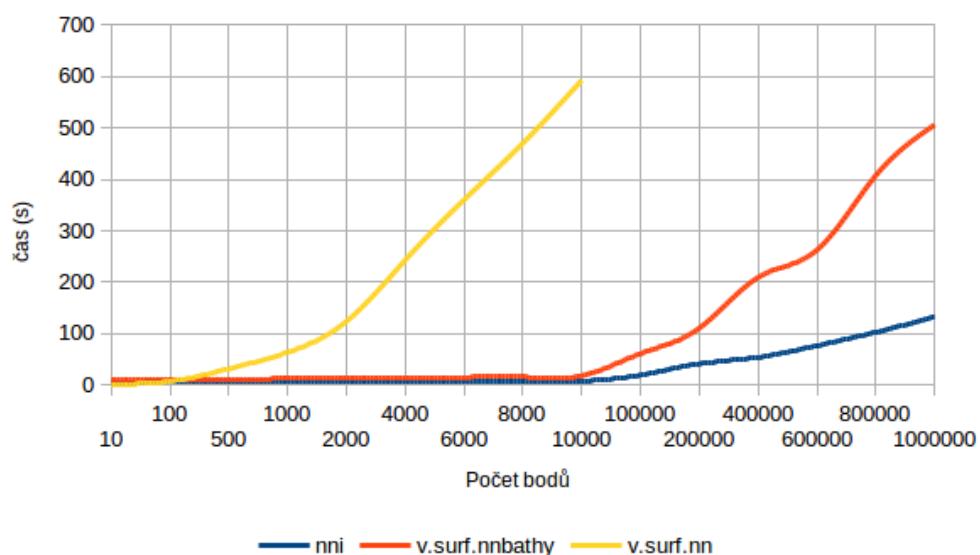
Obrázek 39: Rozdíl mezi výstupem GRASS GISu (v.surf.nnbathy) a Arc GISu pro 1 000 bodů

Obrázek 40: Rozdíl mezi výstupem GRASS GISu (v.surf.nnbathy) a Arc GISu pro 100 000 bodů

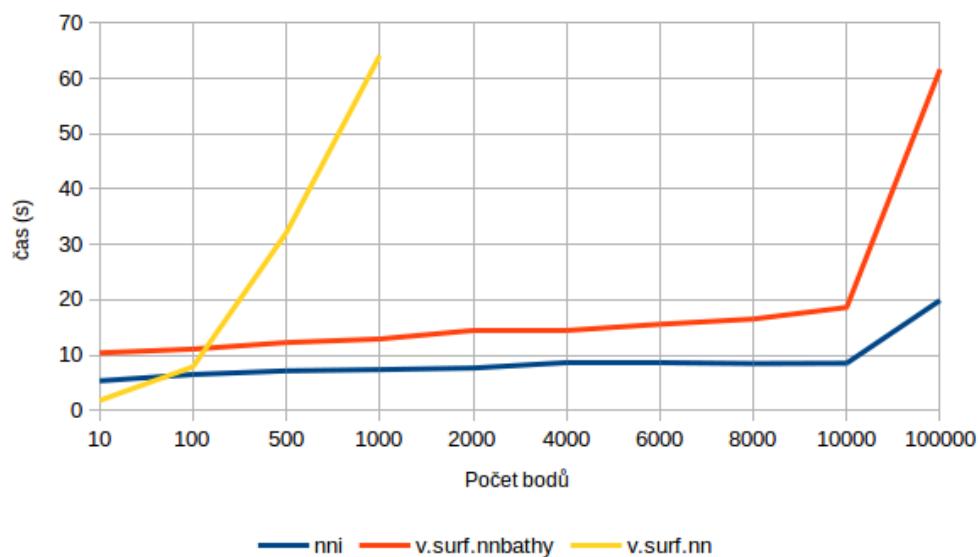


Obrázek 41: Rozdíl mezi výstupem GRASS GISu (v.surf.nnbathy) a Arc GISu pro 1 000 000 bodů

Pro tvorbu grafů bylo vygenerováno dalších patnáct datových sad v rozmezí od deseti až po milión bodů. Nad všemi těmi to sadami byly poté spuštěny interpolace pomocí modulu *v.surf.nnbathy* a *Natural neighbor interpolation (nni)* od ESRI ArcGISu. Pro modul *v.surf.nn* byly testy provedeny pouze pro datové sady do sta tisíc bodů. Už pro takto velkou datovou sadu modul počítal téměř deset minut, takže testy pro větší datové sady nebyly považovány za nutné.



Obrázek 42: Porovnání časů



Obrázek 43: Porovnání časů do 100000 bodů

## B Delaunay struktura

---

```

typedef struct {
    int vids[3];
} triangle;

typedef struct {
    int tids[3];
} triangle_neighbours;

typedef struct {
    double x;
    double y;
    double r;
} circle;

#if !defined(_ISTACK_STRUCT)
#define _ISTACK_STRUCT
struct istack;
typedef struct istack istack;
#endif

#if !defined(_DELAUNAY_STRUCT)
#define _DELAUNAY_STRUCT
struct delaunay;
typedef struct delaunay delaunay;
#endif

/** Structure to perform the Delaunay triangulation of a given array of
points.

*
* Contains a deep copy of the input array of points.
* Contains triangles, circles and edges resulted from the
triangulation.
* Contains neighbour triangles for each triangle.
* Contains point to triangle map.
*/
struct delaunay {
    int npoints;
    point* points;
    double xmin;
    double xmax;
    double ymin;
}

```

```

double ymax;

int ntriangles;
triangle* triangles;
circle* circles;
triangle_neighbours* neighbours; /* for delaunay_xytoi() */

int* n_point_triangles; /* n_point_triangles[i] is number of
                           * triangles i-th point belongs to */
int** point_triangles; /* point_triangles[i][j] is index of j-
                           th
                           * triangle i-th point belongs to */

int nedges;
int* edges; /* n-th edge is formed by points[edges[
                           n*2]]
                           * and points[edges[n*2+1]] */

/*
 * Work data for delaunay_circles_find(). Placed here for
   efficiency
 * reasons. Should be moved to the procedure if parallelizable code
 * needed.
*/
int* flags;
int first_id; /* last search result, used in start up
                  of a
                  * new search */

istack* t_in;
istack* t_out;

/*
 * to keep track of flags set to 1 in the case of very large data
   sets
*/
int nflags;
int nflagsallocated;
int* flagids;
};


```

---

## C Winged-edge struktura

---

```
struct vertex
{
    double x, y, z;
    struct edge *entry_pt;
    int index;
};

struct edge
{
    struct vertex *org;
    struct vertex *dest;
    struct edge *onext;
    struct edge *oprev;
    struct edge *dnext;
    struct edge *dprev;
};
```

---

## D Ukázka zdrojových kódů

V této příloze jsou uvedeny ukázky zdrojových kódů. Kompletní zdrojové kódy jsou uloženy na CD v adresáři *testing/scripts*.

### D.1 nnbathy.py

---

```

class Nnbathy:
    # base class
    def __init__(self, options):
        self._tmpxyz = grass.tempfile()
        self._xyzout = grass.tempfile()
        self._tmp = grass.tempfile()
        self._tmpcat = grass.tempfile()
        self.options = options
        self.region()
    pass

    def region(self):
        # set the computive region
        reg = grass.read_command("g.region", flags='p')
        kv = grass.parse_key_val(reg, sep=':')
        reg_N = float(kv['north'])
        reg_W = float(kv['west'])
        reg_S = float(kv['south'])
        reg_E = float(kv['east'])
        nsres = float(kv['nsres'])
        ewres = float(kv['ewres'])

        # set variables
        self.cols = int(kv['cols'])
        self.rows = int(kv['rows'])
        self.area = (reg_N-reg_S)*(reg_E-reg_W)
        self.ALG = 'nn'

        # set the working region for nnbathy (it's cell-center oriented
        )
        self.nn_n = reg_N - nsres/2
        self.nn_s = reg_S + nsres/2
        self.nn_w = reg_W + ewres/2
        self.nn_e = reg_E - ewres/2
        self.null = "NaN"

```

```

self.dtype = "double"

def compute(self):
    # computing
    grass.message("nnbathy" is performing the interpolation now. \
                  This may take some time... ')
    grass.verbose("Once it completes an 'All done.' \
                  message will be printed.")

    # nnbathy calling
    fsock = open(self._xyzout, 'w')
    grass.call([ 'nnbathy',
                 '-W', '%d' % 0,
                 '-i', '%s' % self._tmpxyz,
                 '-x', '%d' % self.nn_w, '%d' % self.nn_e,
                 '-y', '%d' % self.nn_n, '%d' % self.nn_s,
                 '-P', '%s' % self.ALG,
                 '-n', '%dx%d' % (self.cols, self.rows)],
               stdout=fsock)
    fsock.close()

def create_output(self):
    # create the output raster map
    # convert the X,Y,Z nnbathy output into a GRASS ASCII grid
    # then import with r.in.ascii
    # 1 create header
    header = open(self._tmp, 'w')
    header.write('north: %s\nsouth: %s\neast: %s\nwest: %s\nrows: %
                 s\ncols: %s\nstype: %s\nnull: %s\n\n' %
                 (self.nn_n, self.nn_s, self.nn_e, self.nn_w, self
                  .rows, self.cols, self.dtype, self.null))
    header.close()

    # 2 do the conversion
    grass.message("Converting nnbathy output to GRASS raster ... ")
    fin = open(self._xyzout, 'r')
    fout = open(self._tmp, 'a')
    cur_col = 1
    for line in fin:
        parts = line.split(" ")
        if cur_col == self.cols:
            cur_col = 0
            fout.write(str(parts[2]))
        else:

```

```

        fout.write(str(parts[2]).rstrip('\n')+' ')
        cur_col += 1
    fin.close()
    fout.close()

# 3 import to raster
grass.run_command('r.in.ascii', input=self._tmp,
                  output=self.options['output'], quiet=True)
grass.message("All done. Raster map <%s> created."
              % self.options['output'])

def __del__(self):
    # cleanup
    if self._tmp:
        os.remove(self._tmp)
    if self._tmpxyz:
        os.remove(self._tmpxyz)
    if self._xyzout:
        os.remove(self._xyzout)
    if self._tmpcat:
        os.remove(self._tmpcat)

class Nnbathy_raster(Nnbathy):
    # class for raster input
    def __init__(self, options):
        Nnbathy.__init__(self, options)
        self._load(options)

    def _load(self, options):
        # load input raster
        grass.run_command('r.stats', flags='1gn', input=options['input'],
                          output=self._tmpxyz, quiet=True, overwrite=
                          True)

class Nnbathy_vector(Nnbathy):
    # class for vector input
    def __init__(self, options):
        Nnbathy.__init__(self, options)
        self._load(options)

    def _load(self, options):

```

```

# load input vector, initial controls
if int(options[ 'layer' ]) == 0:
    _layer = ''
    _column = ''
else:
    _layer = int(options[ 'layer' ])
    if options[ 'zcolumn' ]:
        _column = options[ 'zcolumn' ]
    else:
        grass.message('Name of z column required for 2D vector
maps. ')
# convert vector to ASCII
if options[ 'kwhere' ]:
    grass.run_command("v.out.ascii", flags='r', overwrite=1,
                      input=options[ 'input' ], output=self.
                      _tmpcat,
                      format="point", separator="space", dp=15,
                      where=options[ 'kwhere' ], layer=_layer,
                      columns=_column)
else:
    grass.run_command("v.out.ascii", flags='r', overwrite=1,
                      input=options[ 'input' ], output=self.
                      _tmpcat,
                      format="point", separator="space", dp=15,
                      layer=_layer, columns=_column)

# edit ASCII file, crop out one column
if int(options[ 'layer' ]) > 0:
    fin = open(self._tmpcat, 'r')
    fout = open(self._tmpxyz, 'w')
    try:
        for line in fin:
            parts = line.split(" ")
            fout.write(parts[0]+ ' '+parts[1]+ ' '+parts[3])
    except StandardError, e:
        grass.fatal_error("Invalid input: %s" % e)
    fin.close()
    fout.close()
else:
    grass.message("Z coordinates are used.")

class Nnbathy_file:

```

---

```
# class for file input
def __init__(self, options):
    self.options = options
    self._load(options)

def _load(self, options):
    # load input file
    self._tmpxyz = options['file']
```

---

## D.2 v.surf.nnbathy.py

---

```
#!/usr/bin/env python
#####
# MODULE:      v.surf.nnbathy.py
#
# AUTHOR(S):   Adam Laza (mentor: Martin Landa)
#               (based on v.surf.nnbathy from GRASS 6)
#####
from nnbathy import Nnbathy_vector, Nnbathy_file

def main():
    # initial controls
    if (options['input'] and options['file']):
        grass.fatal("Please specify either the 'input' \
                    or 'file' option, not both.")

    if not(options['input'] or options['file']):
        grass.message("Please specify either the 'input' or 'file' \
                      option.")

    if (options['file'] and os.path.isfile(options['file'])):
        grass.message("File "+options['file']+ " does not exist.")

    # vector or file input?
    if (options['input']):
        obj = Nnbathy_vector(options)
    else:
        obj = Nnbathy_file(options)

    obj.compute()
    obj.create_output()
```

---

```
if __name__ == "__main__":
    options, flags = parser()
    main()
```

---

### D.3 r.surf.nnbathy.py

---

```

#!/usr/bin/env python
#####
# MODULE:      r.surf.nnbathy.py
#
# AUTHOR(S):   Adam Laza (mentor: Martin Landa)
#               (based on v.surf.nnbathy from GRASS 6)
#####
from nnbathy import Nnbathy_raster

def main():
    obj = Nnbathy_raster(options)
    obj.compute()
    obj.create_output()

if __name__ == "__main__":
    options, flags = parser()
    main()

```

---

### D.4 v.surf.nn

---

```

/******************
 * MODULE:      v.surf.nn
 *
 * AUTHOR(S):   Adam Laza
 *****************/
#include <cstdlib>
#include <vector>

extern "C" {
#include <grass/vector.h>
#include <grass/glocale.h>
#include <grass/raster.h>
#include <grass/gis.h>
}

#include "local_proto.h"

int main(int argc, char *argv[])
{

```

```

/* open input map */
Vect_open_old2(&In, opt.input->answer, "", opt.field->answer);
Vect_set_error_handler_io(&In, NULL);

field = Vect_get_field_number(&In, opt.field->answer);
column = opt.column->answer;

/* read points */
npoints = read_points(opt.input->answer, opt.field->answer,
                      opt.column->answer, function_values, points);
Vect_close(&In);
if (npoints < 1) {
    G_warning(_("No points loaded. Exiting."));
    exit(EXIT_SUCCESS);
}

/* get the window */
G_get_window(&window);
nsres = window.ns_res;
ewres = window.ew_res;
//G_message("cols %i, rows %i", window.cols, window.rows);

/* allocate buffers, etc. */
dcell = Rast_allocate_d_buf();

if ((maskfd = Rast_maskfd()) >= 0)
    mask = Rast_allocate_c_buf();
else
    mask = NULL;
fd = Rast_open_new(opt.output->answer, DCELL_TYPE);

/* perform NN interpolation */
G_message(_("Computing..."));

/* triangulation */
Delaunay_triangulation T;
typedef CGAL::Data_access< std::map<Point, Coord_type, K::Less_xy_2
>>
Value_access;
T.insert(points.begin(), points.end());

// coordinate computation in grid

```

```

double coor_x, coor_y;
coor_x = window.west;
coor_y = window.north;

for (int rows=0 ; rows<window.rows ; rows++) {
    G_percent(rows, window.rows, 2);

    if (mask)
        Rast_get_c_row(maskfd, mask, rows);

    coor_x = window.west;
    for (int cols=0 ; cols<window.cols ; cols++) {

        /* don't interpolate outside of the mask */
        if (mask && mask[cols] == 0) {
            Rast_set_d_null_value(&dcell[cols], 1);
            continue;
        }

        K::Point_2 p(coor_x, coor_y);
        std::vector< std::pair< Point, Coord_type > > coords;
        Coord_type norm = CGAL::natural_neighbor_coordinates_2(T, p
            , std::back_inserter(coords)).second;
        Coord_type res = CGAL::linear_interpolation(coords.begin()
            , coords.end(), norm, Value_access(function_values));
        G_debug(5, "x: %f y: %f -> res: %f (row=%d; col=%d)",
            coor_x, coor_y, res, rows, cols);
        coor_x += ewres;
        //std::cout << res << " ";
        dcell[cols] = (DCELL) res;
    }

    coor_y -= nsres;
    //std::cout << std::endl;

    Rast_put_d_row(fd, dcell);
}

G_percent(1, 1, 1);

Rast_close(fd);

/* writing history file */
Rast_short_history(opt.output->answer, "raster", &history);
Rast_command_history(&history);
Rast_write_history(opt.output->answer, &history);

```

```
G_done_msg("Raster map %s created.", opt.output->answer);  
exit(EXIT_SUCCESS);  
}
```

---

## E Seznam tabulek a obrázků

### Seznam tabulek

1	Jednoduchá trojúhelníková struktura . . . . .	32
2	Trojúhelníková struktura se sousedy . . . . .	32
3	Vertex based struktura . . . . .	33
4	Half-edge struktura . . . . .	35
5	Dart based struktura . . . . .	37
6	Porovnání časů . . . . .	62

### Seznam obrázků

1	Soubor bodů . . . . .	14
2	Konvexní obal nad doménou $\Omega$ . . . . .	14
3	Degenerace trojúhelníku . . . . .	15
4	Překrývající se trojúhelníky . . . . .	15
5	Nespojitá doména, triangulace obsahující díry . . . . .	16
6	Triangulace validní, ale neregulérní . . . . .	16
7	Triangulace nevalidní . . . . .	17
8	Regulérní triangulace . . . . .	17
9	Případ 1 . . . . .	20
10	Případ 2 . . . . .	20
11	Voronoiovy buňky . . . . .	22
12	Voronoiův diagram souboru bodů . . . . .	23
13	Voronoiův diagram a Delaunayova triangulace . . . . .	23
14	Paprskovitý (Radial sweep) algoritmus, zdroj: [1] . . . . .	26
15	Algoritmus inkrementální konstrukce (Step-by-Step), zdroj: [1] . . . . .	28
16	Rekurzivní dělení na dílčí podmnožiny, zdroj: [1] . . . . .	30
17	Propojování dílčích triangulací, zdroj: [1] . . . . .	30

---

18	Příklad jednoduché triangulace . . . . .	31
19	Datová struktura s použitím pseudo-uzlu, zdroj: [1] . . . . .	33
20	Half-edge struktura (Zdroj: <a href="http://pointclouds.org/blog/nvcs/martin/index.php">http://pointclouds.org/blog/nvcs/martin/index.php</a> ) . . . . .	34
21	Half-edge struktura, zdroj: [1] . . . . .	35
22	$\alpha$ -iterátory, zdroj: [1] . . . . .	36
23	Dart-based datová struktura, zdroj: [1] . . . . .	37
24	Původní VD . . . . .	39
25	Ukradené plochy . . . . .	39
26	Nově vzniklý VD . . . . .	40
27	Vektorová mapa na vstupu . . . . .	44
28	Výstupní rastrová mapa . . . . .	48
29	UML diagram . . . . .	50
30	Množina bodů pro triangulaci . . . . .	53
31	Triangulace pomocí v.delaunay . . . . .	53
32	ArcGIS: Výstupní rastr pro 1 000 bodů . . . . .	63
33	GRASS GIS, v.surf.nnbathy: Výstupní rastr pro 1 000 bodů . . . . .	63
34	GRASS GIS, v.surf.nn: Výstupní rastr pro 1 000 bodů . . . . .	63
35	ArcGIS: Výstupní rastr pro 100 000 bodů . . . . .	64
36	GRASS GIS, v.surf.nn.bathy: Výstupní rastr pro 100 000 bodů . . . . .	64
37	ArcGIS: Výstupní rastr pro 1 000 000 bodů . . . . .	64
38	GRASS GIS, v.surf.nnbathy: Výstupní rastr pro 1 000 000 bodů . . . . .	64
39	Rozdíl mezi výstupem GRASS GISu (v.surf.nnbathy) a Arc GISu pro 1 000 bodů . . . . .	65
40	Rozdíl mezi výstupem GRASS GISu (v.surf.nnbathy) a Arc GISu pro 100 000 bodů . . . . .	65
41	Rozdíl mezi výstupem GRASS GISu (v.surf.nnbathy) a Arc GISu pro 1 000 000 bodů . . . . .	65
42	Porovnání časů . . . . .	66
43	Porovnání časů do 100000 bodů . . . . .	66

## F Obsah CD

.	
text	
<b>LaTeX</b>	zdrojové soubory textu
adam-laza-bp-2015.pdf	tento text
adam-laza-bp-2015.xls	anotace práce
zadani.pdf	naskenované oficiální zadání této práce
src	
<b>nrbathy</b>	zdrojové kódy nrbathy
<b>cgal</b>	zdrojové kódy cgal
testing	
<b>scripts</b>	testovací skripty
<b>sample_data</b>	testovací data