Univerzita Karlova v Praze

Matematicko-fyzikální fakulta

Matyáš Dolák

Automatická optimalizace trojúhelníkové sítě pro výpočet vrstevnic

(Diplomová práce)

Kabinet software a výuky informatiky

Vedoucí diplomové práce: Doc. Dr. Ing. Ivana Kolingerová

Studijní program: Počítačová grafika

Prohlašuji, že jsem předloženou diplomovou práci vypracoval samostatně a s použitím odborné literatury a pramenů, jejichž úplný seznam je její součástí.

Na tomto místě bych také rád poděkoval vedoucí diplomové práce Doc. Dr. Ing. Ivaně Kolingerové za poskytnuté materiály, rady a připomínky.

Firmě Aquion, s. r. o. bych rád poděkoval za zapůjčení realných dat a za konzultace z hlediska praktického využití.

V Praze 2. 12. 2006

Obsah

Abstrakt

Název práce: Automatická optimalizace trojúhelníkové sítě pro tvorbu vrstevnic

Autor: Matyáš Dolák

Katedra (ústav): Kabinet software a výuky informatiky

Vedoucí diplomové práce: Doc. Dr. Ing. Ivana Kolingerová

e-mail vedoucí: kolinger@kiv.zcu.cz

Abstrakt: *Návrh a implementace metod pro automatické úpravy trojúhelníkové sítě pro tvorbu vrstevnic. Optimalizace jsou zkoumány na úrovni algoritmů pro triangulaci, přidávání povinných hran bez znalosti triangulace i se znalostí triangulace a na úrovni modifikace již hotových vrstevnic.*

Klíčová slova: Vrstevnice, Trojúhelníková síť, Optimalizace, Povinné hrany

---

Title: Automatic optimisation of triangle mesh for contour creation

Author: Matyáš Dolák

Department: Department of Software and Computer Science Education

Supervisor: Doc. Dr. Ing. Ivana Kolingerová

Supervisor's e-mail address: kolinger@kiv.zcu.cz

Abstract: *Design and implementation of methods for automatic triangle mesh editing for contour creation. Optimisations are investigated at the level of triangulation algorithms, constraint adding with or without the knowledge of triangulation and at the level of modifying existing contours.*

Keywords: Contour, Triangle mesh, Optimisation, Constraint

1 Úvod

Vrstevnice jsou jedním ze základních způsobů vyjádření třetí souřadnice ve dvojrozměrném výkresu. Nejčastěji se jimi zobrazuje výšková složka terénu, v praxi se však lze setkat i s jinými veličinami: teplota, hydrostatický tlak nebo velikost odběrů u vodovodních sítí, atmosférický tlak v meteorologii, hustota obyvatelstva v demografických aplikacích apod., pak mluvíme o isoliniích.

Výpočet vrstevnic z trojúhelníkových sítí pomocí běžných metod je zatížen některými chybami: tzv. fiktivní spočinky, sloučené kopce a tzv. zaškrcení (vizte [2]). Cílem této práce je navrhnout a implementovat algoritmy pro automatické opravy trojúhelníkové sítě tak, aby výstupem byly kvalitnější vrstevnice – bez těchto chyb. Zároveň je však nutné zachovat podporu ruční editace a tím možnost pro profesionálního kartografa dále opravit chyby, které metody zde popsané nestačí zachytit.

Tato práce navazuje na diplomové práce ing. Čermáka (Výpočet vrstevnic na trojúhelníkové síti, [1]) a ing. Strycha (Triangulace a editování vrstevnic, [2]). V první je navržen postup, jak z množiny bodů v trojrozměrném prostoru vytvořit vrstevnice pomocí triangulace. Druhá se zabývá možností vylepšování vzniklých vrstevnic ruční editací přidáváním povinných hran do triangulace.

Cílem této práce je navrhnout postupy pro automatické opravy trojúhelníkových sítí tak, aby vygenerované vrstevnice byly kvalitnější. Součástí práce je CD se softwarem implementujícím tyto algoritmy v prostředí Win32 a také série ukázkových dat.

První kapitoly práce obsahují souhrn předchozích poznatků, zabývají se hlavně samotným procesem vytváření vrstevnic a navrhují místa pro uplatnění automatických úprav.

Další část se zabývá podrobněji jednotlivými typy úprav, navrhuje algoritmy pro jejich provedení a předkládá jejich výsledky.

Třetí část popisuje software, který byl v rámci této diplomové práce naprogramován. Obsahuje uživatelskou i programátorskou dokumentaci, popis prostředí programu a také možností, jak jej dále rozvíjet.

Většina zde prezentovaných metod používá přidávání povinných hran. Předvedeny jsou však i další metody, které povinné hrany nepoužívají – například eliminace křížících se vrstevnic.

Ukazuje se, že automatická oprava chyb ve vrstevnicích má své meze a tedy je nutná kontrola a oprava zkušeným kartografem. Je však zbytečné, aby kartograf opravoval běžné chyby, které lze algoritmicky zachytit a opravit automaticky. Právě na takovéto úpravy je tato práce zaměřena.

2 Základní pojmy a konvence

Protože se v různých odvětvích může používat mírně odlišná terminologie, uveďme nejdříve terminologii a konvence používané v této práci.

Triangulace rovinné oblasti Ω je soubor $T = \{t_i\}$, $i = 1 \ldots N_T$ trojúhelníků takových, že:

- libovolná dvojice trojúhelníků se vzájemně protíná buď v jednom společném vrcholu, nebo v jedné společné hraně, nebo je jejich průnik prázdný
- sjednocení trojúhelníků T je souvislá množina v rovině

Oblast Ω je vždy polygonální doména; obecně nemusí být konvexní a může obecně obsahovat díry.

V oblasti výpočetní techniky se používá synonymum **Triangular Irregular Network**.

Triangulace nad množinou bodů M je taková triangulace, že vrcholy trojúhelníků jsou právě všechny body množiny M.

Delaunayova triangulace je taková triangulace, že kružnice opsaná každému trojúhelníku neobsahuje žádný vrchol jiného trojúhelníku.

Hltavá (žravá, greedy) triangulace je triangulace složená z nejkratších možných vzájemně se neprotínajících hran.

Povinná hrana pro triangulaci je předem dané omezení vynucující přítomnost dané hrany v triangulaci. Tato hrana obvykle porušuje podmínky kladené příslušnou triangulací, například pro Delaunayovu triangulaci trojúhelníky obsahující tuto hranu netestujeme, zda jejich opsaná kružnice neobsahuje žádný vrchol jiného trojúhelníku. Mluvíme pak o **Omezené Delaunayově triangulaci (CDT)**.

Lomová hrana je povinná hrana používaná v některých aplikacích pro specifické účely, nejčastěji vyjádření zlomu v ploše. Protože tato práce pracuje s nevyhlazeným terénem, lomová hrana je pro tyto účely ekvivalentní povinné hraně.

*Pozn.: V geodetické praxi se lze setkat se záměnou významu pojmů **povinná hrana** a **lomová hrana**.*

Fiktivní spočinek je jednou z nejčastějších chyb při automatickém generování vrstevnic (vizte podkapitolu Nesprávná triangulace kapitoly Kategorizace chyb). Jedná se o situaci, kdy výsledné vrstevnice zobrazují spočinek na místě, kde ve skutečnosti nemá být.

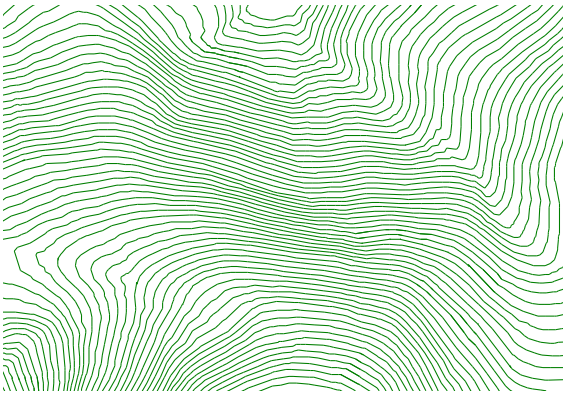
Plochý trojúhelník je takový trojúhelník triangulace, jehož všechny tři vrcholy mají stejnou výšku. Fiktivní spočinky vznikají téměř výhradně na plochých trojúhelnících.

3 Digitální model terénu

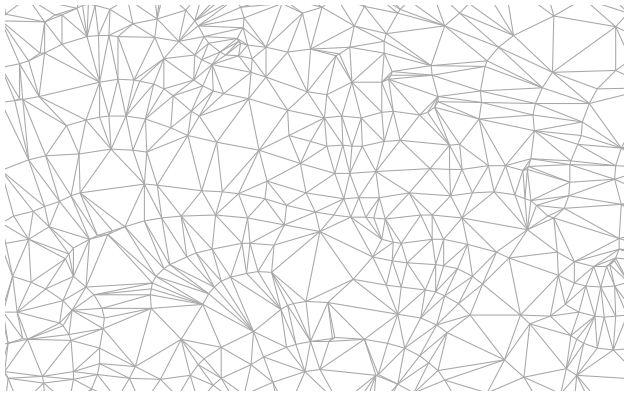
Pro zobrazení terénu se v kartografii nejčastěji používají vrstevnice. Kvůli jejich všeobecnému rozšíření a jednoduchému principu je naprostá většina lidí vnímá intuitivně; pro projektanty jsou nedílnou součástí práce.

Pro výpočetní účely se naopak používá reprezentace (nepravidelnou) trojúhelníkovou sítí, tzv. TIN (Triangular Irregular Network). Ta umožňuje relativně jednoduchý výpočet reprezentované veličiny pro libovolný zadaný bod, graficky však má její zobrazení v rovině téměř nulovou vyjadřovací schopnost. Lze ji však s výhodou použít pro vykreslení plochy pomocí dnešních grafických akcelerátorů, které jsou optimalizovány právě pro trojúhelníky.

Mezi těmito dvěma modely existuje vzájemný převod – vrstevnice lze navzorkovat body a mezi nimi vytvořit trojúhelníkovou síť a naopak v trojúhelníkové síti lze zjistit body, v nichž vrstevnice protíná hrany trojúhelníků a jejich pomocí aproximovat vrstevnice. Samozřejmě oba tyto převody způsobují ztrátu přesnosti, což ovšem relativně ku jejich vlastním nepřesnostem příliš nevadí. Vlastnímu převodu z TIN na vrstevnice se věnují práce [1, 2], ve kterých jsou popsány různé metody získávání vrstevnic.



Obr 3.1: Vrstevnice



Obr 3.2: TIN

V práci [1] je mimo jiné ověřena praktická možnost automatického generování vrstevnic z trojúhelníkových sítí. Ukazuje se, že reprezentace pomocí TIN je nejvhodnějším (a tedy zaslouženě nejčastějším) modelem terénu jak z hlediska výpočtů, tak z hlediska náročnosti na paměť. Vrstevnice vytvořené touto metodou jsou z vizuálního hlediska použitelné.

Práce [2] navazuje pozorováním několika druhů chyb, které při automatické tvorbě vznikají. Ukazuje, že nezáleží, zda je použita hltavá či Delaunayova triangulace, obě jsou z hlediska výsledných vrstevnic srovnatelné. Pro tuto práci nejdůležitějším poznatkem však je, že většinu chyb lze eliminovat pomocí přidávání povinných hran.

3.1 Typické úlohy

Tato práce je zaměřena na nalezení vrstevnic, tedy konverzi z množiny bodů do množiny isolinií. Úlohy, které jsou v této oblasti zpracovávány, jsou většinou tyto:

1. Geodet projde zadaný terén, vytvoří přesné zaměření bodů včetně jejich výšky, úlohou je spočítat z takto získaných dat vrstevnice. Data jsou v tomto případě výškově náhodná, většinou je k dispozici zhruba rovnoměrné pokrytí výpočetní plochy body. Zpracováním takovýchto dat vzniká nejméně chyb.
2. Výpočtem jsou zjištěny hodnoty veličiny v některých bodech, výstupem mají být isolinie veličiny. Typickým příkladem jsou meteorologické mapy (isobary), mapy velikostí odběru vody apod. Tato data jsou, co se výškového rozložení týká, náhodná, a tedy se zpracováním dojde k dobrým vrstevnicím i bez jakýchkoliv oprav.
3. Je dána (digitální) předloha vrstevnic na vstupu a výstupem má být mapa s podrobnějšími vrstevnicemi. Kartograf tedy provede nejdříve převod vrstevnic na body a pak zpětný převod bodů na vrstevnice s použitím menšího rozestupu vrstevnic. Data jsou v tomto případě výškově velmi přesně „kvantována“, což přináší velké problémy při tvorbě vrstevnic. Na tyto případy se tato práce obzvlášť zaměřuje.

Pozorováním bylo zjištěno, že právě diskrétní obor hodnot způsobuje největší problémy při tvorbě vrstevnic. Naopak faktory, které ovlivňují triangulaci pozitivně, jsou náhodné výšky a rovnoměrné pokrytí. Při náhodných výškách nedochází ke vzniku plochých trojúhelníků a rovnoměrné pokrytí zaručuje dobrou vypovídací schopnost datové reprezentace ve všech místech oblasti.

Samozřejmě ve speciálních případech není rovnoměrné pokrytí až tak důležité – například pro projektanty bývají důležité výškové poměry v obcích a na silnicích, které je spojují, ale není potřeba zvlášť velké přesnosti mimo tyto oblasti (vizte například data Vstup4 na přiloženém CD). Přesnější je tedy označení „data navzorkovaná úměrně vzhledem k požadované míře detailu, adaptivně.“

4 Existující metody

V literatuře je téma optimalizace trojúhelníkové sítě zastoupeno velmi omezeně. Ve většině dostupné literatury jsou obsaženy pouze algoritmy pro vytvoření vrstevnic z triangulace nebo optimalizace triangulace pro jiné účely, ne však již pro další úpravy.

Práce [2] a [3] zmiňují možnost automatických oprav a některé z nich jsou i nastíněny intuitivními postupy, chybí však přesnější rozbor. Základní myšlenkou je přidávání povinných hran, navíc práce [2] dokazuje, že vhodně přidané povinné hrany výslednou triangulaci velmi dobře vylepší. Analýzou možností se zabývá podkapitola Optimalizace nad triangulací kapitoly Optimalizace.

Existuje hodně postupů pro úpravy triangulace pomocí přidávání pomocných (Steinerových) bodů, nejznámějším příkladem je [6], další lze nalézt v [5], [7] a [8]. Tyto postupy jsou pro naše účely nevhodné. Převážná většina z nich totiž nebere ohled na třetí souřadnici, a to ani na možnost jejího dopočítání.

Ve [4] je zmíněn zajímavý algoritmus používající střední osy, myšlenka tohoto algoritmu je dále zkoumána v podkapitole Střední osa kapitoly Optimalizace a rozvinuta na implementovatelný optimalizační algoritmus.

5 Kategorizace chyb při automatickém výpočtu vrstevnic

Chyby vznikající při automatickém výpočtu vrstevnic lze podle vzniku rozdělit do těchto kategorií: chyby vzniklé nedostatečnými nebo nevhodnými daty, chyby vzniklé nesprávnou triangulací, chyby vzniklé numerickou nepřesností stroje, chyby vzniklé použitím nevhodné metody výpočtu a chyby vzniklé vyhlazováním.

5.1 Nedostatečná nebo nevhodná data

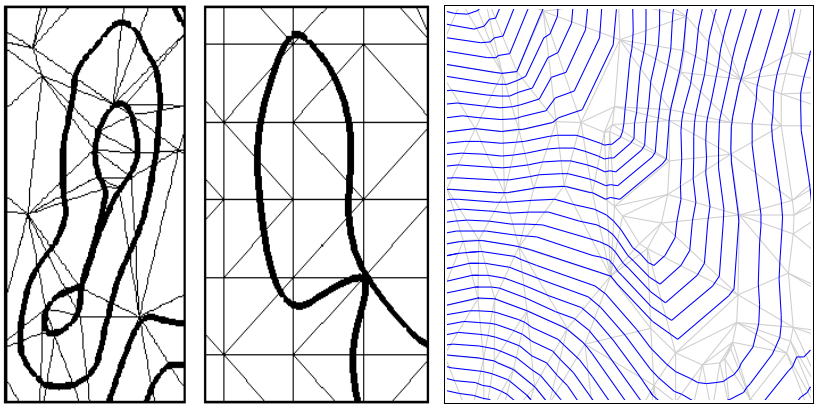
V praxi se lze často setkat s datovou množinou, která má velmi specifické pokrytí, většinou vedoucí k velmi špatným triangulacím. Například při zpracovávání projektů vodovodů a kanalizace bývají v datových podkladech zaměřeny pouze body na silnicích a v jejich blízkém okolí.

Do této kategorie lze zařadit i případy velmi řídké triangulace, kdy požadavky kartografů na přesnost výstupu řádově přesahují přesnost vstupu. Například při triangulaci území o rozloze okresu nelze požadovat přesnost reprezentace v řádu centimetrů, protože daná data prostě neobsahují dostatek podkladů.

Obecně tato problematická data nelze vylepšovat nějakým univerzálním algoritmem, každý případ je na to příliš specifický. Tato práce se těmto chybám věnovat dále nebude.

5.2 Nesprávná triangulace

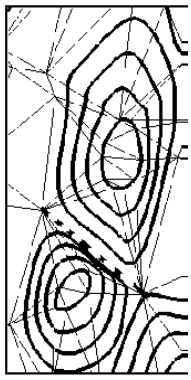
Práce [2] uvádí tři příklady chyb, které vznikají nesprávnou triangulací:



Obr 5.1: Nesprávná triangulace

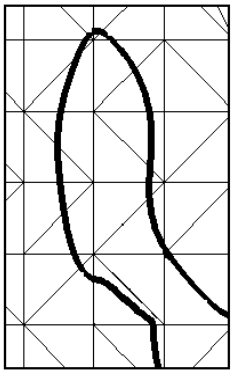
První případ ukazuje sloučení dvou vrcholků kopců, druhý zobrazuje vrstevnici procházející dvakrát blízkým okolím bodu a třetí případ je fiktivní spočinek.

První případ ukazuje, že nevhodnou volbou trojúhelníků lze docílit vrstevnic, které jsou sice podle definice správné, ale nepoužívají se a nejspíš ani neodpovídají realitě. Přitom jediná povinná hrana vedená napříč sloučenou oblastí dokáže vrstevnice opravit tak, jak by je kartograf očekával:



Obr 5.2: Oprava povinnou hranou

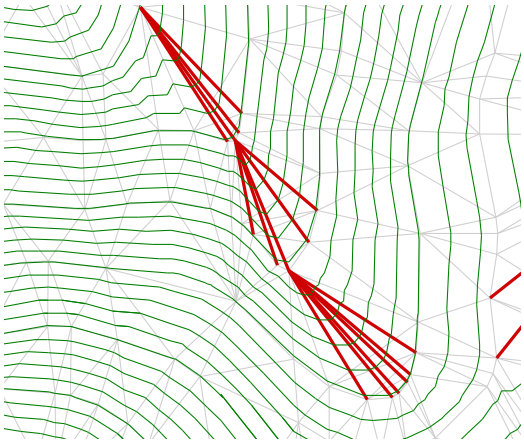
Druhá chyba je velmi podobna předchozí, má však jinou podstatu vzniku, ke spojení vrstevnice dochází pouze v okolí jediného bodu. Také v tomto případě pomůže jediná povinná hrana, která zajistí „překlopení“ hrany, která tuto abnormalitu způsobuje:



Obr 5.3: Oprava překlopením hrany

Nejviditelnější a zároveň také nejčastější chybou vrstevnic je právě fiktivní spočinek, tento druh chyby se tedy stal centrem pozornosti této práce. První dvě chyby je možné vhodně eliminovat na úrovni optimalizace vrstevnic, opravu fiktivních spočinků je ideální provádět v dřívějších fázích výpočtu (vizte kapitolu Optimalizace).

Fiktivní spočinek lze intuitivně opravit přidáním několika povinných hran (počet hran odpovídá počtu vrcholů plochých trojúhelníků:



Obr 5.4: Oprava fiktivních spočinků

Naším prvním úkolem je mimo jiné zjistit, zda některá triangulace je vhodnější, zda nějakým vedlejším účinkem eliminuje fiktivní spočinky. Porovnávány byly Delaunayova triangulace (DT) a hltavá triangulace (GT), jejichž implementaci laskavě poskytla vedoucí práce.

Podle [2] je z hlediska fiktivních spočinků vhodnější triangulací GT, sama o sobě vytváří méně fiktivních spočinků, než DT. Má však další problémy, mezi které patří nejen vysoká časová a paměťová náročnost, ale taky vedlejší dopady na triangulaci a výsledné vrstevnice. Tím, že vytváří velké množství hubených trojúhelníků, je zdrojem potenciálních chyb ve vrstevnicích. Často se pak stává, že vrstevnice se vlní tam, kde podle jiných triangulací vychází rovná.

Tyto argumenty ukazují na DT jako preferovanou triangulaci, ovšem s tím, že je nutné výslednou triangulaci před samotnou tvorbou vrstevnic upravit přidáním vhodných povinných hran.

To, co se jevilo jako nevýhoda DT, tedy tvorba plochých trojúhelníků s následným vznikem fiktivních spočinků, je možno chápat i jako výhodu, protože ploché trojúhelníky se dobře detekují, a tedy existuje poměrně jednoduchý způsob, jak určit místa, kde je zapotřebí triangulaci změnit. Pro použití dále popsané metody optimalizace je tato vlastnost klíčová.

5.3 Numerická nepřesnost stroje

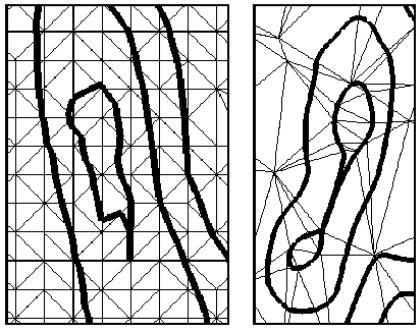
Tyto chyby jsou poměrně vzácné, vyskytují se jen ve velmi specifických případech vstupních dat, kdy například triangulace není jednoznačná nebo výškové rozdíly jednotlivých bodů jsou mimo zobrazovací schopnost datových typů dané platformy. Častým důsledkem chyb v této kategorii je vadná triangulace například kvůli chybné odpovědi na polohu bodu vůči přímce.

Tyto chyby nelze eliminovat jinak než použitím jiné platformy nebo přesnějších datových typů. Lze také použít knihovnu pro přesnou aritmetiku vyvinutou J. R. Shewchukem ([6]). Dále se tímto druhem chyb tato práce zabývat nebude.

5.4 Nevhodná metoda výpočtu

Tato kategorie je poněkud sporná, protože do ní lze zařadit i většinu chyb z ostatních kategorií. Uvedena je zde spíše proto, že vytváří na některé chyby nový pohled, pomocí kterého lze intuitivně navrhnout další metody optimalizace.

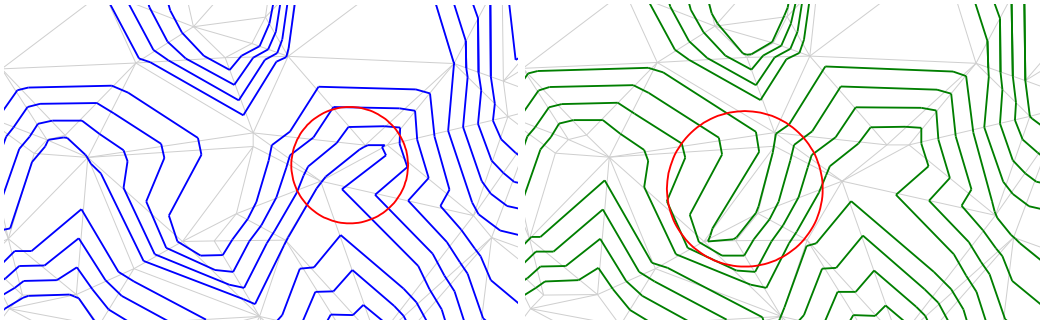
Nejvýznamnější zástupci této kategorie jsou na obrázku 6.5:



Obr 5.5: Chyby vzniklé nevhodnou metodou výpočtu

Prvním případem je tzv. „chobot" – vrstevnice sleduje jednu hranu triangulace a pak se po ní hned vrací zpět. Druhým případem je sloučení vrcholků kopců (tento případ byl podrobněji popsán v kategorii Nesprávná triangulace).

Obě tyto chyby vznikají v případě, že výška vrstevnice se liší od výšky bodu jen o malé ε nebo vůbec. Způsob počítání vrstevnic totiž pro tyto případy narazí na singularitu – u daného vrcholu není jasné, kterým trojúhelníkem má výpočet pokračovat. Této singularitě se proto algoritmus vyhýbá přičtením nebo odečtením malé konstanty od výšky bodu. Přičtením vznikne „chobot" na vrcholu kopce, odečtením vznikne v údolí; při použití opačné operace tato anomálie nevzniká. Toto pozorování umožňuje navrhnout metodu popsanou dále v kapitole Optimalizace nad vrstevnicemi.



Obr 5.6: choboty při přičtení a odečtení ε

5.5 Vyhlazování

I při samotném vyhlazování může dojít k nežádoucímu chování vrstevnic. Tyto případy jsou však poměrně vzácné. Navíc jejich výskyt je podmíněn ostatními typy chyb, nejčastěji špatnou triangulací, kdy vznikají velmi hubené trojúhelníky.

Jedná se o chybu, kdy jsou vrstevnice příliš blízko sebe a jsou hodně členité. Pak se může stát, že vyhlazování způsobí překřížení vrstevnic.

Eliminací tohoto typu chyby se teoreticky zabývá kapitola Optimalizace nad vrstevnicemi.

6 Optimalizace při generování vrstevnic

Průběh generování vrstevnic lze zachytit následujícím postupem (podle [2]):

1. Získání množiny bodů
2. Tvorba trojúhelníkové sítě
3. Vrstevnice jako lomené čáry
4. Vyhlazení vrstevnic

Z hlediska vylepšování je možné vložit optimalizace jako mezikroky po každém stupni generování. Samozřejmě čím dříve optimalizaci umístíme, tím méně informací bude tato optimalizace mít o probíhajícím procesu (např. mezi kroky 1 a 2 nebude mít optimalizace k dispozici vrstevnice, takže nesmí používat algoritmus, který by jich jakkoliv využíval). Toto omezení lze částečně obejít tím, že se některé kroky procesu zopakují. Cenou za to však bude větší náročnost, výpočetní i paměťová, celého programu.

Optimalizacemi před a během získávání množiny bodů se tato práce nezabývá, protože tyto již většinou nespadají do kategorie algoritmických úloh. Z hlediska typických úloh se jedná o dobré zaškolení geodeta, případně vhodné rozmístění měřících stanic.

6.1 Optimalizace na množině bodů

Na množině bodů lze provádět před samotnou triangulací jen poměrně omezené úpravy. Naopak výhodou je, že není třeba žádné předzpracování dat, takže tyto metody nevyžadují opakování náročných operací, jako je třeba triangulace či počítání sousedů trojúhelníků. Ovlivnit následující triangulaci lze v zásadě jen čtyřmi způsoby: přidáním bodu, odebráním bodu, posunutím bodu a vytvořením povinné hrany.

Odebírání bodu by v některých speciálních případech mohlo mít pozitivní dopad, nicméně vzhledem k již tak dosti zjednodušenému modelu a běžnému nedostatku vstupních dat se jím tato práce zabývat nebude.

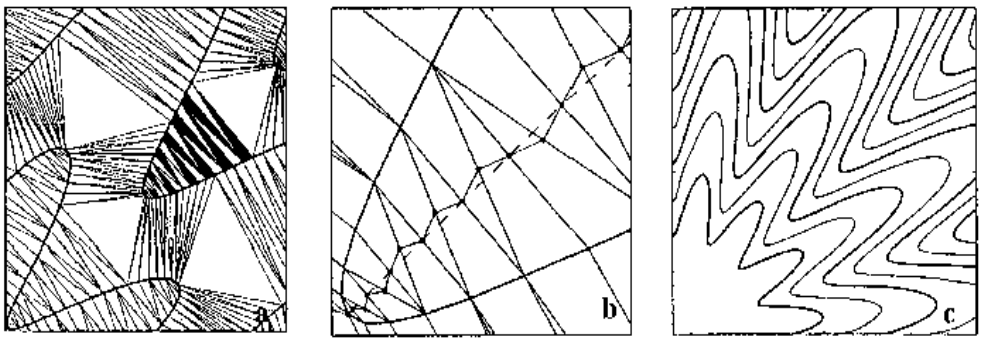
Přidávání bodu by mohlo být z hlediska triangulací velmi prospěšné, ale obecně je problematické definovat způsob, jakým vybrat, kam bod přidat a jakou hodnotu mu přiřadit. Případné heuristiky vycházejí ze znalosti topologie alespoň okolí tohoto bodu, které však v tomto stupni ještě neznáme.

Posouvání bodu naráží na podobné problémy, jako přidávání, tedy hlavně výpočet nové třetí souřadnice. V praxi se navíc poměrně často požaduje, aby v místě zadaných bodů reprezentace přesně odpovídala příslušným bodům, což by po posunu nemuselo platit.

Zůstává možnost přidávání povinných hran. Jak je ukázáno již v práci [2], povinnými hranami lze vrstevnice radikálně zlepšit. Problém s neznalostí okolí bodu ovšem setrvává i zde, takže kritéria přidávání hran musí být pečlivě zvažována.

6.1.1 Střední osa

Práce [2] zmiňuje zajímavou metodu, tzv. střední osy. Při ní se přidávají body takové, které leží na střední ose fiktivního spočinku. Předpokladem je již existující triangulace, na které se detekují fiktivní spočinky. Pak se procházením plochých trojúhelníků v jednotlivých spočincích vytvoří odhad tzv. střední osy této oblasti pomocí středů společných hran těchto trojúhelníků. Na jednom konci se osa dotýká vrstevnice, na druhém je potřeba najít pokračování k další vrstevnici. Tím získáme výšky osy na koncích a můžeme lineárně interpolovat výšky bodů na této ose.



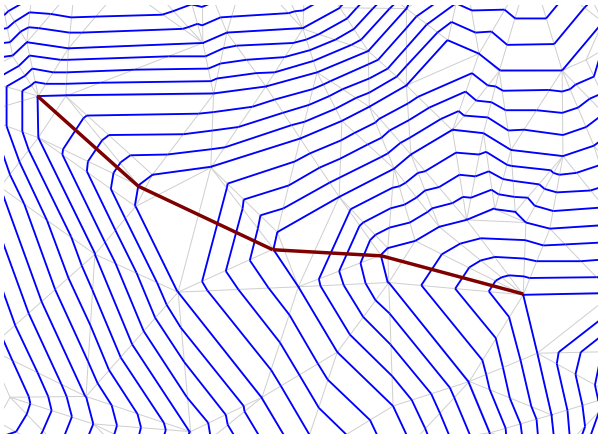
Obr 6.1: Střední osa

Na obrázku je původní triangulace (a), konstrukce střední osy (b) a výsledné vrstevnice po retriangulaci (c).

Metoda má velmi příznivé výsledky, trpí ovšem nedostatky, z nichž největšími jsou výpočet správného vedení střední osy v případě kombinovaných spočinků a velký nárůst datové množiny. Princip eliminace pomocí této metody se však stal základní myšlenkou dalších algoritmů uvedených dále.

6.1.2 Automatická tvorba hřbetnic a údolnic

Úplně nejlepším postupem by bylo, kdyby se podařilo přidávat povinné hrany tak, aby kopírovaly hřbetnice a údolnice (tj. pomyslné čáry spojující místa s největší křivostí vrstevnic, vizte [4]). V takovém případě by se totiž ve výsledných vrstevnicích eliminovaly fiktivní spočinky.



Obr 6.2: Ideální hřbetnice

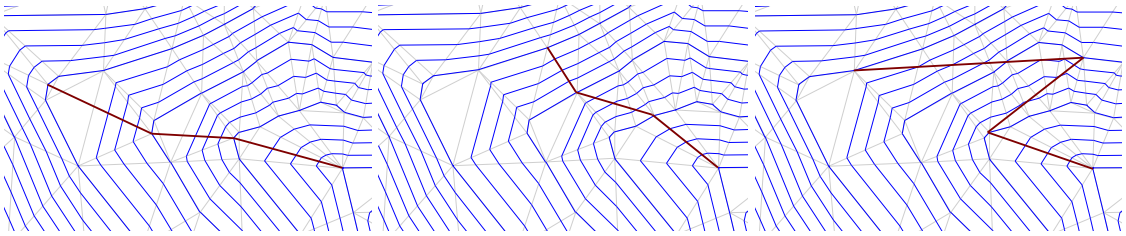
Hlavní idea algoritmu je tedy takováto:

1. Nějakým způsobem vybrat startovací body, nejlépe na dně nebo vrcholu hřbetnice nebo údolnice.
2. Z těchto startovacích bodů vést povinné hrany tak, aby jejich sklon byl co nejblíže nulovému, ale přitom nenulový.
3. Touto hranou získáme nový startovací bod, pro který opakujeme bod 2

Intuitivně by tento algoritmus mohl být úspěšný, praxe však ukazuje několik podstatných nedostatků. Výběr bodů pro start algoritmu je velmi problematický. Přitom špatně vybrané body mohou velmi negativně ovlivnit části sítě, které by jinak byly triangulovány dobře – povinné hrany se od tohoto bodu snaží jít k některé údolnici, což vytváří velmi hubené trojúhelníky. Na vrstevnicích se to projeví fiktivním zavlněním.

Tento problém by bylo možné obejít tím, že by se nejdříve několika kroky algoritmu bez přidávání hrany dorazilo do bodu, který již leží na příslušné údolnici nebo hřbetnici, a pak od tohoto bodu by se pokračovalo podle původního algoritmu.

Na mnohem větší obtíže však narazíme při pokusu o přesnější specifikaci kroku č. 2. Použijeme-li totiž opravdu doslova hranu s nenulovým sklonem co nejbližším nule, získáme s největší pravděpodobností hranu do úplně jiné části sítě. Je tedy potřeba omezit hledání jen na blízké okolí startovacího bodu. Zde ovšem narážíme na to, jak definovat blízké okolí. Určitě jej nelze specifikovat jako vzdálenost a ani jako omezení počtu nejbližších bodů, které jsou prohledávány, protože v obou případech lze jednoduše zkonstruovat protipříklad, na kterém toto kritérium nebude fungovat. Navíc na úspěchu tohoto kritéria stojí a padá použitelnost celého algoritmu:



Obr 6.3: Vliv kritéria na generované spádnice

Na první ukázce je vidět úspěšnou aplikaci, kdy se podařilo vystihnout vzdálenost prohledávání a je vytvořena ideální údolnice. Na druhé ukázce je vidět neúspěšné nastavení – pokud nejsou prohledány body dostatečně daleko, začnou nacházené hrany utíkat od hřbetnice. Třetí ukázka zobrazuje naopak příliš široký okruh vyhledávání, kdy je nalezen bod úplně mimo oblast zájmu.

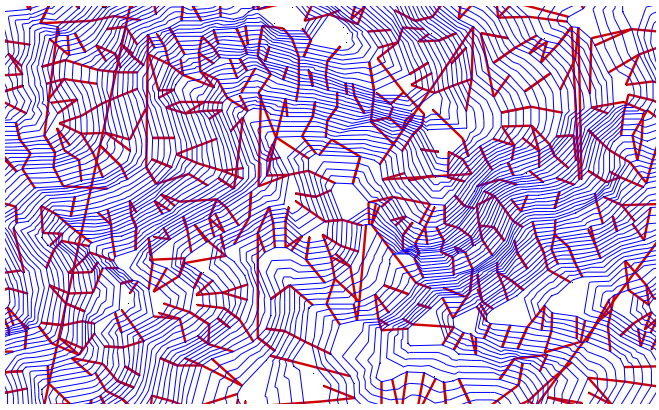
Finálním problémem algoritmu je stanovení kritéria, kdy cyklus 2-3 zastavit.

Pozn.: V případě, že vstupními daty je digitalizovaná vrstevnice (tedy přesněji když je obor hodnot výšky diskrétní), bylo by možné krok 2 upravit tak, že by nejdříve našel nejbližší bod o rozdílné výšce a pak prohledával pouze body o této výšce („na sousední vrstevnici“), nicméně ani tak se nezbavíme problému definice okolí pro prohledávání – vrstevnic o takovéto výšce může být v zadaných datech několik.

6.1.3 Ke každému bodu nejbližší vyšší a nižší

Tento algoritmus se řadí do kategorie „hrubou silou“. Vychází z poznatku, že obecně při triangulaci jsou nejproblematičtější tzv. ploché trojúhelníky, a nejjednodušším pokusem o jejich eliminaci je každému bodu přiřadit povinnou hranu k jeho nejbližším sousedům s nižší a s vyšší souřadnicí z.

Bohužel ani takto jednoduchý algoritmus není možné bezhlavě aplikovat. Problém nastává u lokálních extrémů – např. u lokálního maxima je nejbližším vyšším bodem pouze některé vzdálené lokální maximum; ke globálnímu maximu dokonce takový bod neexistuje.



Obr 6.4: Ukázka výstupu algoritmu

Na ukázce na obr. 6.3 si nelze nevšimnout několika extrémně dlouhých povinných hran, které jsou právě důsledky lokálních extrémů. Samozřejmě takovéto extrémy by bylo možno odfiltrovat, ale pouze některou ze statistických metod, které nemají stoprocentní účinnost. A protože tyto hrany mají velmi nepříznivé důsledky pro generované vrstevnice, musíme i tuto metodu zavrhnout.

Pozn.: Navíc algoritmus v principu nezabraňuje vzniku křížících se povinných hran, což obecně mohou triangulační algoritmu ignorovat, ale je tady vidět potenciální nebezpečnost.

6.2 Optimalizace nad triangulací

Optimalizace na této úrovni již má k dispozici triangulaci a může ji tedy využívat – nejčastěji k určování lokality bodu nebo k vyhledávání a eliminaci plochých trojúhelníků. Nevýhodou je potřeba již spočítané triangulace a většinou také potřeba jejího přepočítávání po spočtených úpravách.

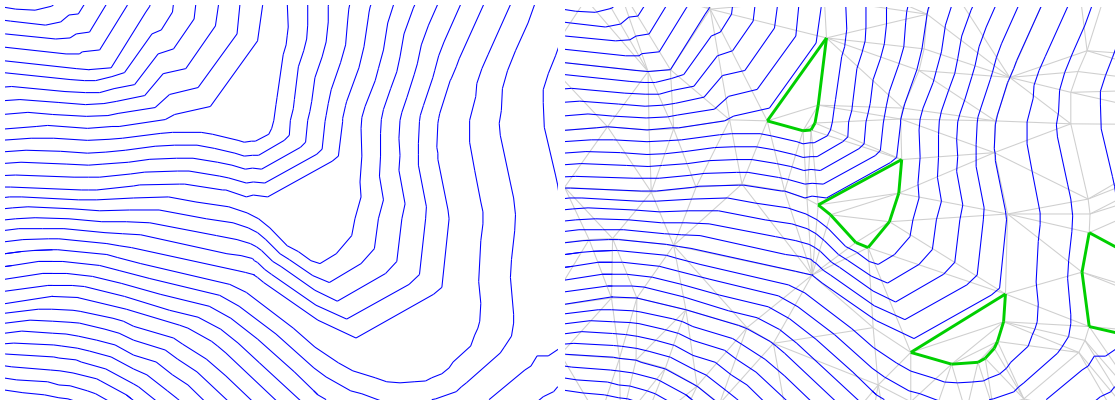
Dvě z metod, které zde budou popsány, používají jako prostředek k eliminaci přidávání povinných hran. Jak uvádí již práce [2], lze tímto způsobem opravit velkou část chyb, při zkoušení těchto metod však vychází najevo, že ne všechny chyby se takto opravit dají. Navíc povinné hrany v triangulaci způsobují až několikanásobného zpomalení výpočtu. Lze však říci, že i přes tyto nevýhody jsou tyto metody velmi úspěšné a vyplatí se je používat.

6.2.1 Eliminace fiktivních spočinků

Cílem je získat metodu, která je dostatečně účinná a zároveň obsahuje minimální (ideálně žádné) vedlejší účinky v podobě zavlečených chyb. Bylo proto potřeba navrhnout co nejrobustnější pravidla pro přidávání povinných hran, aby byly přidány pouze takové hrany, které opravdu triangulaci zlepší.

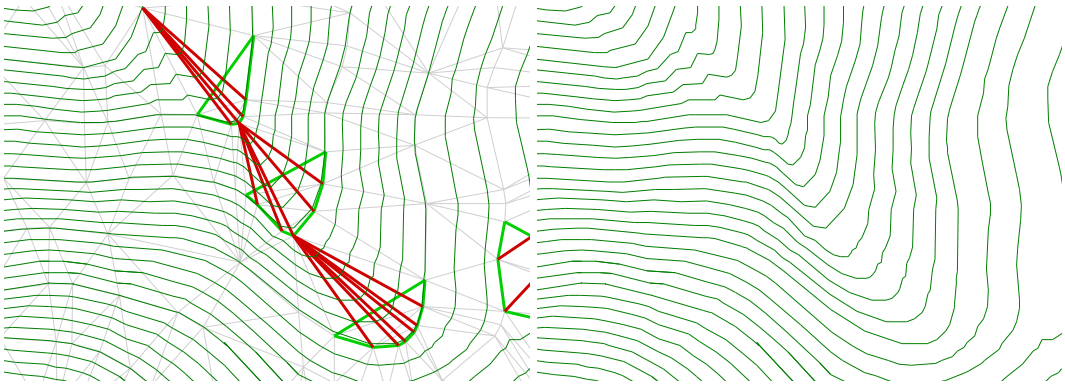
Tato metoda se pro jednoduché spočinky projevuje jako nejúspěšnější. Po jejím použití jsou všechny jednoduché fiktivní spočinky eliminovány, zůstávají pouze některé složitější útvary jako několikanásobné spojené spočinky apod. Výhodou je relativně jednoduchá implementace a malá časová náročnost.

Podívejme se na typický fiktivní spočinek, který vznikl právě kvůli plochým trojúhelníkům:



Obr 6.5: Fiktivní spočinek s vyznačenými plochými trojúhelníky

Tedy nejčastěji se lze setkat s vrstevnicí ve tvaru písmene U s rozevřenými rameny, uvnitř kterého jsou ploché trojúhelníky. Ideálním řešením by v tomto případě bylo přidat povinné hrany mezi každým bodem těchto plochých trojúhelníků a bodem v sousedním trojúhelníku proti nejdelší hraně skupiny plochých trojúhelníků (oproti použití hřbetnic, kdy přidáváme pouze jednu hranu):

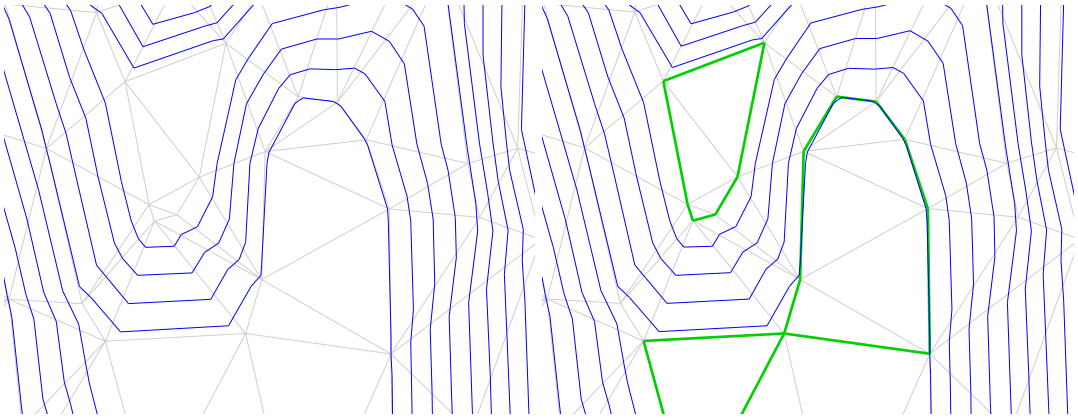


Obr 6.6: Oprava fiktivních spočinků povinnými hranami a výsledné vrstevnice

Algoritmus 1:

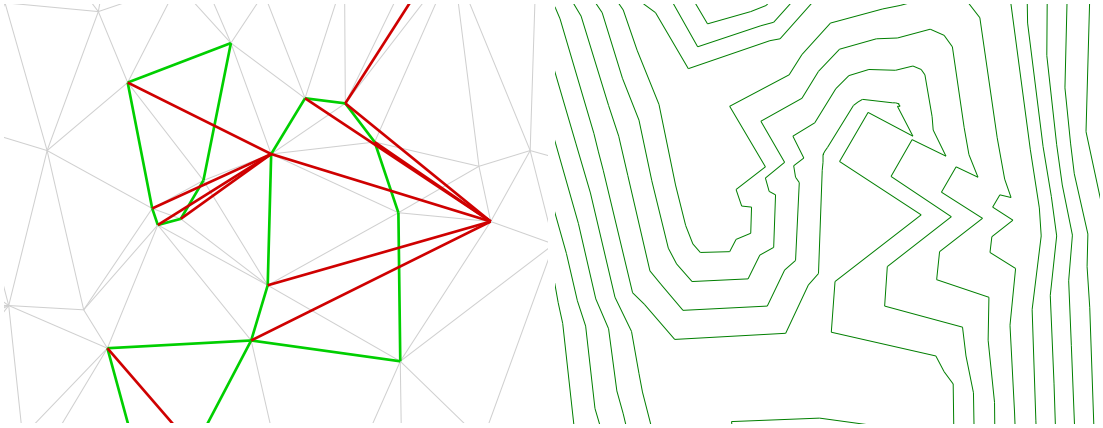
1. Vyhledej skupinu plochých trojúhelníků
2. Vyber nejdelší hranu skupiny, s bodem v sousedním trojúhelníku proti ní vytvoř povinné hrany pro všechny body plochých trojúhelníků

Zatím by se mohlo zdát, že tato metoda musí dávat zaručeně lepší výsledky, než původní neoptimalizovaná triangulace. Bohužel se však v reálných sítích vyskytují složitější fiktivní spočinky, na které je potřeba algoritmus upravit.



Obr 6.7.: Fiktivní spočinek s atypickou nejdelší hranou

Na obrázku vidíme opět fiktivní spočinek ve tvaru písmene U s rozevřenými rameny, ale nejdelší hrana je tentokrát na jednom z ramen. V tomto případě tedy navrhovaný algoritmus přidá povinné hrany, které výsledné vrstevnice určitě nezlepší:



Obr 6.8: Nesprávně přidané povinné hrany

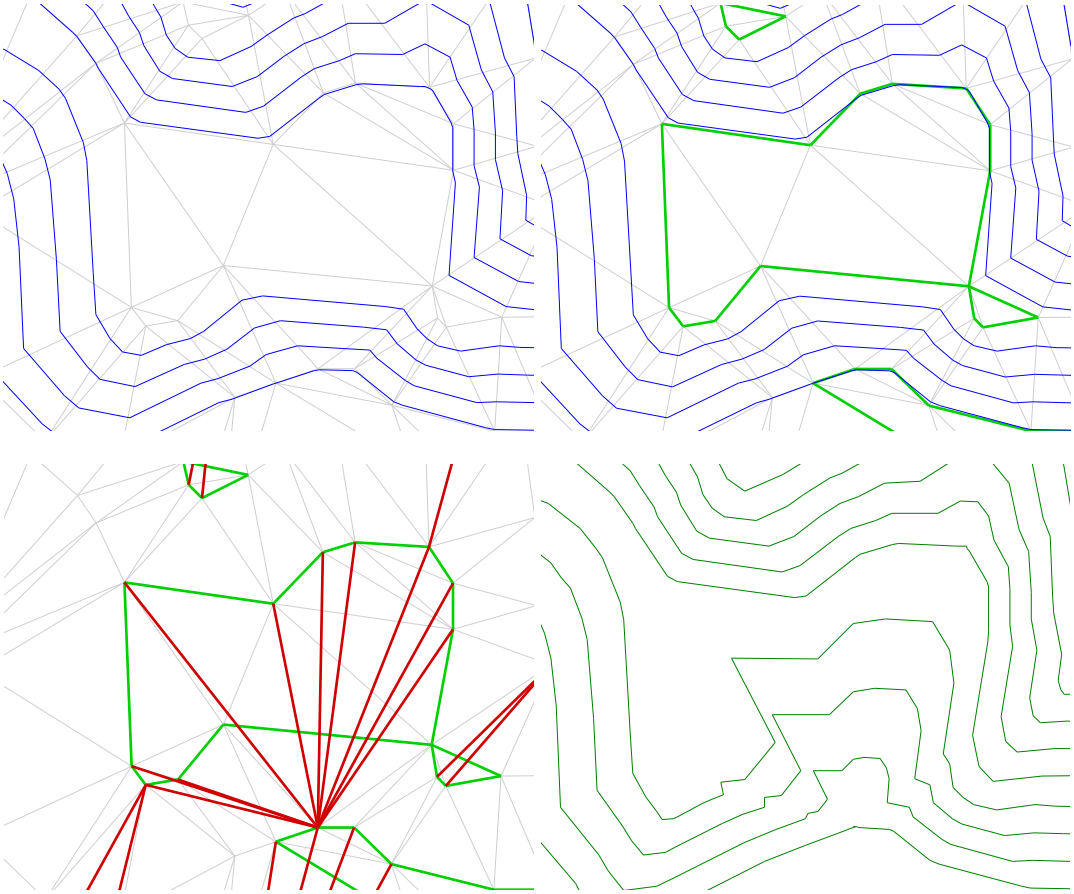
Řešení existují trojího druhu: buď se program pokusí najít bod, ze kterého jsou všechny povinné hrany správné, nebo přidá jen dobré hrany, nebo takový fiktivní spočinek neopraví. Samozřejmě je nejdříve potřeba takovouto situaci detekovat, což je ale jednoduché – otestujeme, zda se některá z povinných hran kříží s hraniční hranou plochých trojúhelníků.

Protože naším cílem je robustní algoritmus, první z řešení není použito. Ukazuje se totiž, že existují ještě komplikovanější případy, na kterých by mohlo hledání správného bodu být komplikované a hlavně s nezaručenými výsledky. Doplníme tedy algoritmus:

Algoritmus 2:

1. Vyhledej skupinu plochých trojúhelníků
2. Vyber nejdelší hranu skupiny, s bodem v sousedním trojúhelníku proti ní vytvoř povinné hrany pro všechny body plochých trojúhelníků
3. Kříží-li některá z povinných hran hraniční hranu skupiny, odeber ji.

Problémem zůstávají spočinky, které samy o sobě nejsou konvexní. Povinné hrany pak nekříží žádnou z hraničních hran a přesto jsou nesprávné:



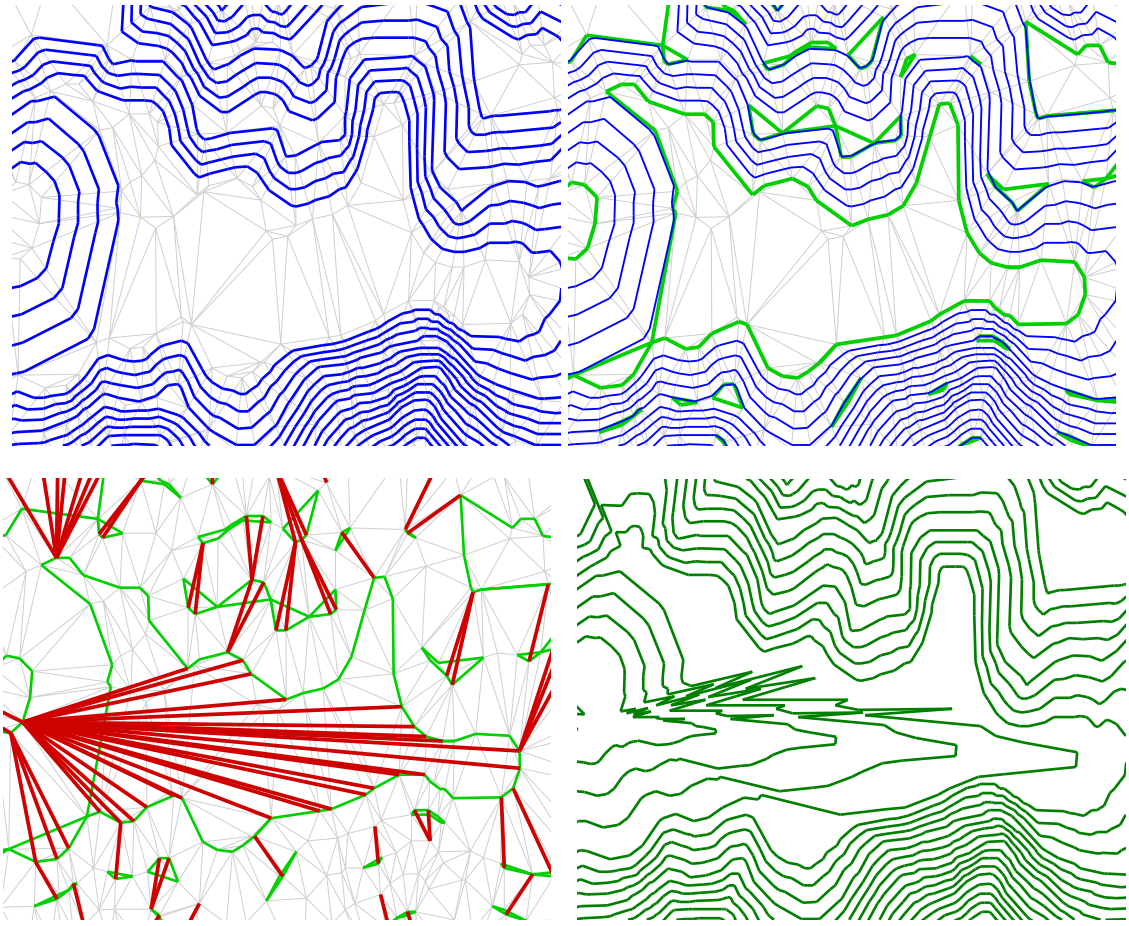
Obr 6.9: Chybně opravený nekonvexní spočinek

Je tedy potřeba další úprava algoritmu. Přidávané hrany je nutné dále filtrovat a povolit pouze takové, které procházejí příslušnou nejdelší hranou. Další krok ve vývoji algoritmu tedy vypadá takto:

Algoritmus 3:

1. Vyhledej skupinu plochých trojúhelníků
2. Vyber nejdelší hranu skupiny, s bodem v sousedním trojúhelníku proti ní vytvoř povinné hrany pro všechny body plochých trojúhelníků
3. Kříží-li některá z povinných hran hraniční hranu skupiny, odeber ji.
4. Nekříží-li některá z povinných hran nejdelší hraniční hranu skupiny, odeber ji.

I tento algoritmus však naráží na problémy v jednom z nejhorších typu spočinků – kombinovaných, které se vyskytují například na dně údolí:



Obr 6.10: Spočinek neopravitelný povinnou hranou

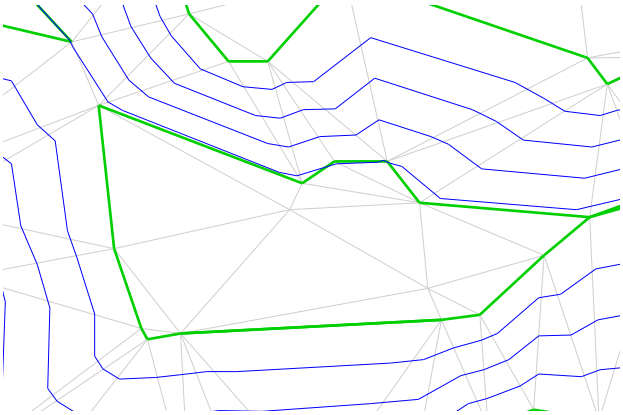
Z tohoto případu je vidět, že fiktivní spočinky takovéhoto typu ani nelze opravit povinnou hranou Naskýtá se otázka, zda se vůbec jedná o spočinky fiktivní, nebo reálné. Je tedy vhodné neopravovat je vůbec. Jejich detekce není jednoduchá, uvědomíme-li si však, že úpravy vzniklé použitím pravidel 3 a 4 našeho algoritmu také produkují nepříliš kvalitní opravy, je vhodné úplně ignorovat všechny skupiny plochých trojúhelníků, kterých se tyto dva body týkají. Do tohoto hodnocení právě spadají i poslední zmiňované – kombinované – spočinky. Finální algoritmus tedy vypadá takto:

Algoritmus 4:

1. Vyhledej skupinu plochých trojúhelníků
2. Vyber nejdelší hranu skupiny, s bodem v sousedním trojúhelníku proti ní vytvoř povinné hrany pro všechny body plochých trojúhelníků
3. Kříží-li některá z povinných hran hraniční hranu skupiny, nebo nekříží-li některá z povinných hran nejdelší hranu skupiny, odeber všechny povinné hrany přidané v kroku 2 a skupinu dále nezpracovávej.

I přes toto relativně velké omezení je algoritmus poměrně úspěšný v eliminaci fiktivních spočinků v běžných datech.

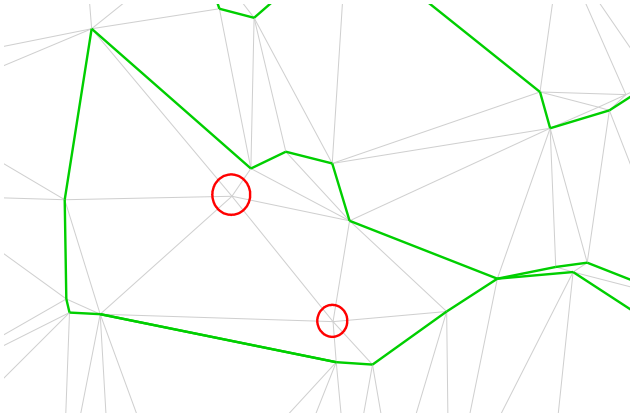
Samozřejmě lze nalézt případy, kdy se tento algoritmus nechová zcela korektně. Jsou to zejména případy, kdy jsou fiktivní spočinky spojeny ve větší celek. Jeden z takových případů ukazuje následující obrázek:



Obr 6.11: Spojené fiktivní spočinky

Podobné příklady je možné filtrovat, avšak pouze velmi restriktivním způsobem – neoptimalizovat skupiny plochých trojúhelníků, ve kterých je alespoň jeden plochý trojúhelník, jehož všichni tři sousedé jsou ploché trojúhelníky. Zdá se, že by toto omezení vyřadilo zpracování většiny fiktivních spočinků, protože často je vrstevnice uvnitř spočinku mírně prohnutá, čímž se vytvoří jeden nebo dva takové trojúhelníky.Zkoušením na reálných datech bylo zjištěno, že tento filtr je jen o trochu více omezující, než uvedený krok 3 algoritmu 4. Konkretní hodnoty jsou v kapitole Výsledky v praxi.

Pozn.: Pozorný čtenář si zajisté postřehl, že se v algoritmech vyhýbáme používání kritéria „konvexní oblast". Je to ze dvou důvodů: zjišťování konvexnosti je náročné a numericky nestabilní a hlavně ve skupině plochých trojúhelníků může být několik bodů i uvnitř skupiny:



Obr 6.12: Body uvnitř oblasti

Vyhledávání

Předpokladem pro správnou funkci vyhledávání fiktivních spočinků, který jsme zatím opomíjeli, je skutečnost, že jsou v triangulaci reprezentovány plochými trojúhelníky. Z tohoto hlediska je tedy nutné porovnat metody triangulace.

Delaunayova triangulace tomuto algoritmu vyhovuje velmi dobře, ploché trojúhelníky vytváří přesně tam, kde je člověk intuitivně očekává, a tedy to, co bylo původně její nevýhodou, se díky této optimalizaci stává její výhodou.

Naopak hltavá triangulace fiktivní spočinky téměř nevyrábí, ovšem (podle [2]) za cenu toho, že výsledné vrstevnice neodpovídají zadání. Navíc ploché trojúhelníky vznikají i v místech, kde by fiktivní spočinek běžně nenastal, a tedy optimalizace takové skupiny by vedla k chybně interpretovaným datům.

Je proto zřejmě lepší použít Delaunayovu triangulaci a výsledné ploché trojúhelníky, které jsme předtím považovali za chyby, nyní s výhodou použít právě pro vyhledávání a následnou optimalizaci.

6.2.2 Modifikace metody středové osy.

V předchozí kapitole jsme narazili na problémy v případě, že je fiktivních spočinků na jednom místě více a dotýkají se. Minulá metoda je uměla detekovat, ale neuměla je správně opravit. Nyní se pokusíme alespoň o částečné řešení tohoto speciálního druhu.

Základní myšlenka byla prezentována již v podkapitole o metodě středních os. Je možné sledovat trasu střední osy a pak na ni přidávat body. V našem případě nebudeme však přidávat body, ale povinné hrany, použijeme ovšem algoritmus procházení trojúhelníků pro hledání střední osy.

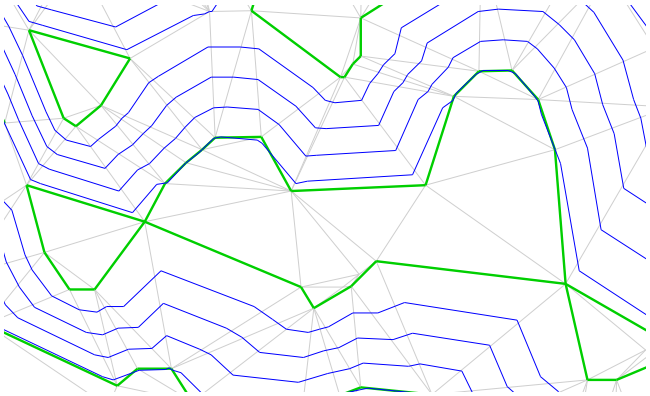
Algoritmus 5:

1. Vyhledej skupinu sousedících plochých trojúhelníků
2. Vyber startovací trojúhelník a směr (Heuristika 1)
3. Od vybraného trojúhelníku postupuj po plochých sousedech podle nějakého kritéria (Heuristika 2)
4. Pro trojúhelníky zpracované v krocích 2 a 3 vytvoř povinné hrany
5. Nevyhovující povinné hrany odfiltruj (Heuristika 3)
6. Opakuj kroky 2 – 4 pro zatím nezpracované trojúhelníky

Samozřejmě tento návrh je nutné doplnit příslušnými heuristikami 1, 2 a 3. Postupně zde budou ukázány různé možnosti, které byly ozkoušeny.

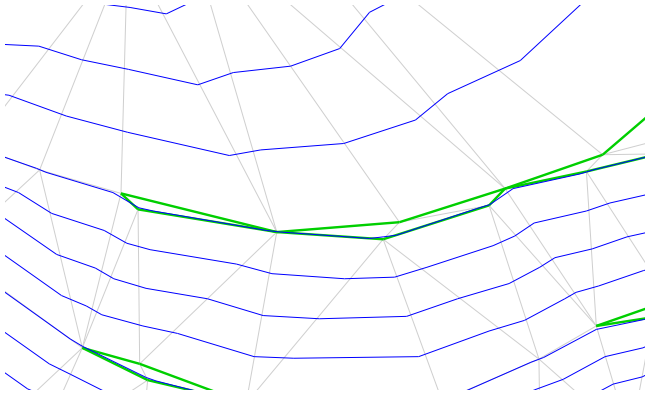
Výběr startovacího trojúhelníku (Heuristika 1)

Pro první heuristiku, výběr startovacího trojúhelníku a směru, existují dva možné postupy. Můžeme se pokoušet najít trojúhelník ve „špičce“ fiktivního spočinku, nebo u jeho „kořene“. Ukazuje se, že postupovat od špičky ke kořeni je jednodušší, trojúhelníky ve špičce lze podstatně lépe odhadovat.



Obr 6.13: Ukázka kombinovaného spočinku

Podle situace na obrázku je zřejmé, že jako kořeny nelze vyhledávat například pouze trojúhelníky se dvěma hranami na hranici, z nichž jedna je nejdelší. Samotné vyhledávání podle nejdelší hraniční hrany také může vést ke špatným výsledkům. Existují však speciální situace, kdy je vhodné vyhledávat i podle nejdelší hraniční hrany – když je původní vrstevnice zvlněná:



Obr 6.14: Ukázka zvlněné původní vrstevnice

V tomto případě stačí hledat trojúhelníky s nejdelší hranou na hranici a přeskočit krok 3 – vytvořit pouze jedinou povinnou hranu.

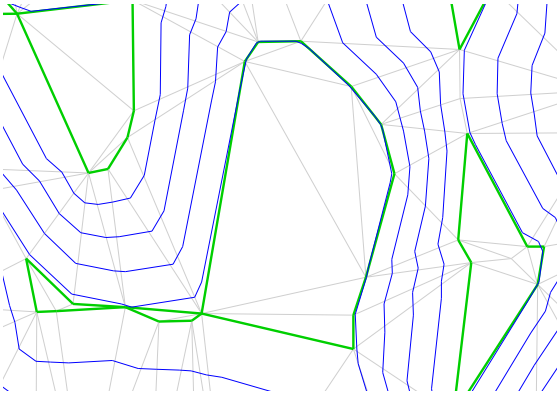
Naopak vezmeme-li trojúhelníky ve špičce, vidíme, že se můžeme pokusit o několik různých kritérií. Můžeme třeba vzít trojúhelník s nejkratší hranou, nebo ten s nejkratší hraniční hranou. Další možností je vzít trojúhelník s minimálním obsahem. Obě tato kritéria jsou jednoduchá na vyhodnocení a jejich výsledky jsou uspokojivé.

Ještě lepších výsledků však dosáhneme, uvědomíme-li si, že většina hledaných trojúhelníků má dvě hrany na hranici oblasti. Můžeme tedy tyto trojúhelníky zpracovat jako první a další až v případě jejich vyčerpání.

Procházení sousedů (Heuristika 2)

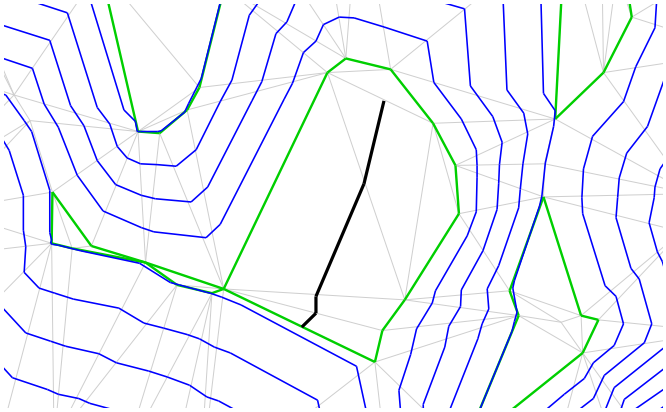
Předpokládejme, že máme správně vybrané startovací trojúhelníky ve špičkách fiktivních spočinků. Dalším krokem je tedy projít sousedy a vybrat takové, které tvoří jeden spočinek.

První pokus byl velmi jednoduchý: vybereme nejdelší hranu a tou pokračujeme dále. (Samozřejmě je nutné ošetřit případ, kdy jsme do trojúhelníka nejdelší hranou přišli) Toto kritérium bohužel nefunguje v situacích, kdy jsou fiktivní spočinky úzké a dlouhé – trojúhelníky v nich pak jsou poskládané nejdelšími hranami na boční straně spočinku:



Obr 6.15: Nejdelší hrana na boční straně spočinku

Další jednoduché kritérium si všímá toho, že osa spočinku by měla mít jen minimální ohyb, v běžných případech je úplně rovná. Budeme tedy postupovat po trojúhelnících tak, aby úhly jejich středních příček byly co nejbližší přímému úhlu:



Obr 6.16: Ukázka středních příček

Toto kritérium se potýká hned se dvěma problémy. Prvním je problém zvolení startovacího směru. Jako nejrozumnější řešení se jeví zvolit jako startovací směr polopřímku vycházející z vrcholu nad nejdelší hranou a procházející středem nejdelší hrany. Druhým problémem je fakt, že startovací trojúhelník často neleží přesně v ose, ale bokem od ní.

Jako nejvhodnější se ukázala kombinace obou metod. Stanovíme maximální poměr délek stran, při jeho překročení se automaticky vydáme nejdelší stranou, jinak pokračujeme podle úhlového kritéria.

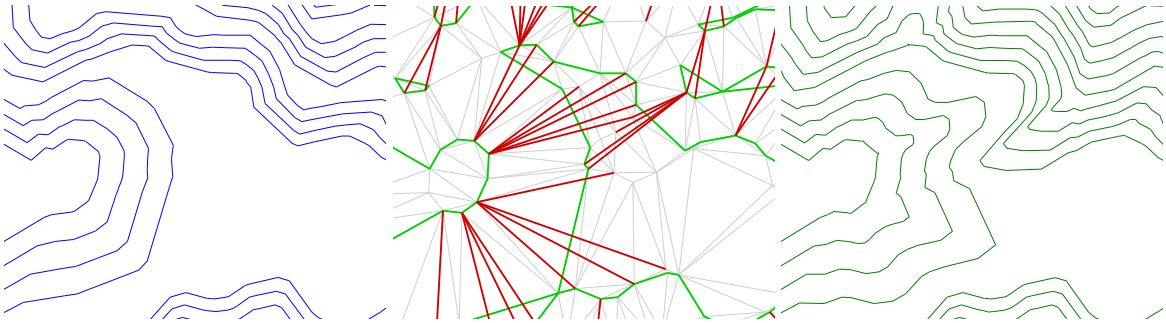
Můžeme si také uvědomit, že má-li trojúhelník pouze dva ploché sousedy, tak je pokračování jednoznačně určeno druhým sousedem, než ze kterého jsme přišli. Je však nutné dávat pozor, abychom se v případě kombinovaných spočinků „nestočili“ do jiného spočinku. Musíme tedy kontrolovat délku stran a k tomuto kritériu přikročit pouze tehdy, je-li odchozí hrana delší než příchozí.

Filtrace nevyhovujících hran (Heuristika 3)

Stejně jako v předchozí metodě je i zde velká pravděpodobnost, že vytvoříme hrany, které spíše výsledné vrstevnice pokazí, než aby je zlepšily. Toto platí hlavně u spočinků vznikajících v místě spojení dvou údolí. Vzhledem k použitým metodám však existuje i možnost, že projdeme spočinek ze dvou startovacích trojúhelníků do jiných koncových a vytvoříme dvě sady křížících se povinných hran.

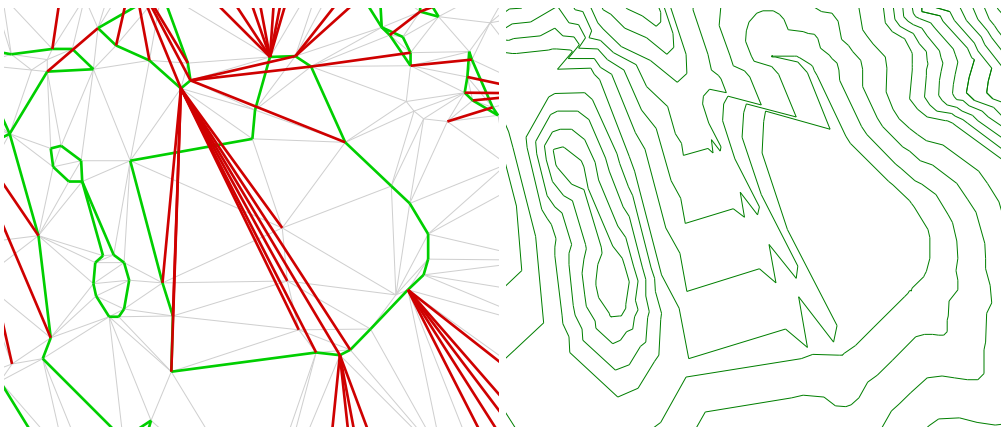
První filtr tedy odstraňuje křížící se povinné hrany. Protože tyto jsou většinou příznakem složitého spočinku, je lepší odstranit obě dvě hrany. Aby filtr fungoval korektně, je nutné hrany nejdříve pouze označit jako neplatné a až po skončení filtrace je fakticky odstranit.

Narozdíl od jednoduché eliminace však v případě křížících se hran nebudeme ignorovat celou oblast, protože můžeme opravit alespoň malou část této oblasti:



Obr 6.17: Částečně opravený složitý spočinek

Druhý filtr musí zabránit vzniku hran, které by způsobily přílišné chvění výsledných vrstevnic:

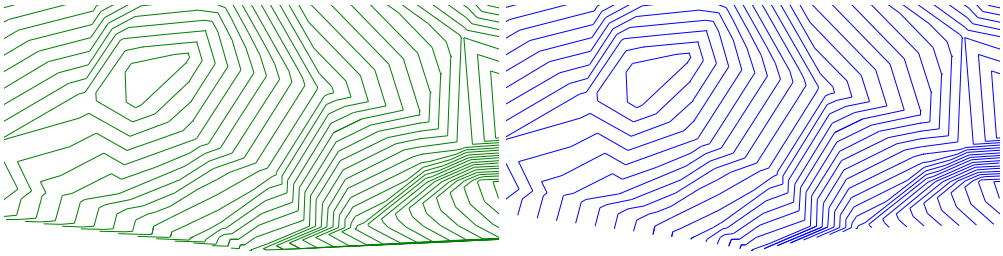


Obr 6.18: Povinné hrany způsobující chvění vrstevnice

Toho dosáhneme jednoduše tím, že pro každou hranu zjistíme, pod jakým úhlem je vidět ze všech bodů oblasti. Je-li tento úhel příliš velký, hranu odstraníme.

6.2.3 Okraje triangulace

Protože výstupem základní triangulace je konvexní oblast, na jejích okrajích se často vyskytují velmi hubené trojúhelníky, které jsou ve většině případů nežádoucí. Je proto vhodné je eliminovat. Jak je ale detekovat? Jako nejjednodušší způsob se jeví postupně eliminovat trojúhelníky na okraji triangulace, které mají některý z vnitřních úhlů příliš malý. Experimentálně bylo ověřeno, že odstranění trojúhelníků s úhlem menším než 3 stupně odstraní nevhodné chování triangulace a přitom dopad na již správné vrstevnice je minimální:



Obr 6.19: Před a po odstranění okrajů triangulace

6.3 Optimalizace nad vrstevnicemi

Pokud máme k dispozici již spočtené vrstevnice, je možné je analyzovat a případně opravovat chyby přímo na nich. Nevýhodou jsou poměrně špatně definovatelná kritéria pro opravy, špatná implementovatelnost a často také velká výpočetní náročnost postupu. Výhodou je práce přímo s výstupními daty, tedy jednodušší intuitivní náhled na algoritmy.

6.3.1 Křížení vrstevnic

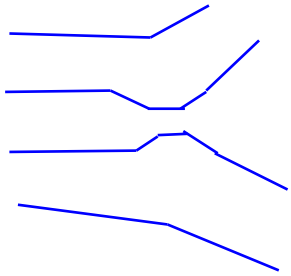
Jak již bylo naznačeno v podkapitole Vyhlazování kapitoly Kategorizace chyb, lze se ve výjimečných případech setkat s negativními důsledky vyhlazování – vrstevnice se překříží. Tento stav je poměrně jednoduše zjistitelný. Detekce je výpočetně poměrně náročná – křížení každého segmentu vrstevnice s každým dalším segmentem každé vrstevnice, tedy $O(n^2)$, kde n je celkový počet segmentů všech vrstevnic. S pomocí optimalizací například kvadrantovým stromem lze však náročnost v průměrných případech úspěšně zredukovat.

Zajímavější je způsob korekce této chyby. Nejjednodušším řešením je vyměnit překřížené části, jak je naznačeno na následujícím obrázku. Je však vidět, že tímto způsobem zůstanou na vrstevnicích zúžená místa, která stále nevypadají dobře:

Obr 6.20: Jednoduché odkřížení vrstevnic

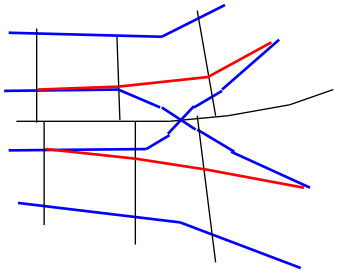
Pozn.: Obrázky v této kapitole nejsou skutečně spočítané vrstevnice, ale pouze náčrtky zjednodušené pro lepší pochopení. Samozřejmě při překřížení vrstevnice musí dojít i k deformaci okolních vrstevnic, což by obrázek velmi znepřehlednilo.

Je tedy potřeba opravit větší okolí překřížení. K tomu lze ale použít jen různé statistické heuristiky, u kterých hrozí jednak neúspěch a jednak kolize s jinými vrstevnicemi v oblasti, pokud by jisté zúžení nastávalo i v okolních vrstevnicích:



Obr 6.21: Složitější okolí pro odkřížení

Poměrně dobré výsledky by mohla dávat následující metoda: Vytvoříme střední osu křížících se vrstevnic. Na tuto osu povedeme kolmice od vedlejších vrstevnic. Protože osa by měla odpovídat zhruba poloviční výšce jednak mezi křížícími se vrstevnicemi a také mezi jejich sousedy, můžeme místo křížení nahradit aproximací vrstevnic pomocí bodů na kolmicích ve vzdálenosti 1/3 od osy a 2/3 od další vrstevnice:



Obr 6.22: Aproximace ze střední osy

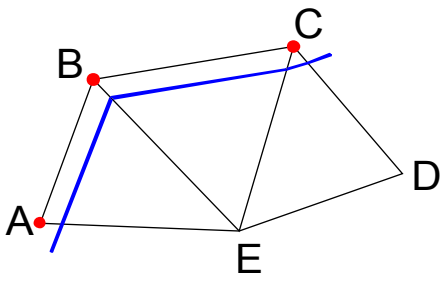
6.3.2 Oprava pomocí ε-filtrace

Po fiktivních spočincích jsou v žebříčku nejčastějších chyb v automaticky generovaných vrstevnicích nejvýše tzv. sloučené vrcholky kopců a „choboty“ (pro jednoduchost nazývejme tyto chyby dále jen „choboty“). Tyto chyby jsou popsány v podkapitole Nevhodná metoda výpočtu kapitoly Kategorizace chyb. Jejich vznik se váže k řešení singulárních případů při převodu triangulace na vrstevnice – je-li výška vrcholu triangulace rovna výšce vrstevnice, algoritmus upraví výšku vrcholu přičtením nebo

odečtením malé konstanty. Právě v jednom z těchto případů vznikne „chobot“, ve druhém jsou vrstevnice vygenerovány správně. Kdybychom věděli, ve kterém případě chobot vzniká, použili bychom opačný případ pro vygenerování správnější vrstevnice.

Prvním problémem při implementaci této optimalizace je vůbec detekce chobotů. Je potřeba navrhnout algoritmus, který dokáže rozhodnout, zda a ve kterém místě se na vrstevnici nachází chobot.

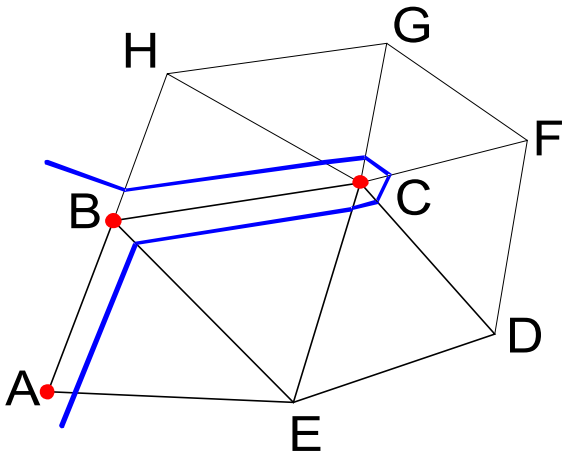
Předpokládejme, že se tento problém bude řešit přímo při počítání vrstevnic z triangulace. Máme tedy k dispozici triangulaci a pomocí ní vytváříme vrstevnice. Můžeme tedy označit „podezřelé body“, ve kterých se kvůli eliminaci singularit musí posunout výška vrcholu. Při průchodu triangulace pro každou vrstevnici si budeme značit pro každý bod také informaci, zda na hraně, na které vznikl, je i podezřelý bod. Pokud ano, uložíme si tuto informaci a k podezřelému bodu si poznačíme bod na předchozí hraně, ze kterého jsme k tomuto bodu přišli. Obrázek 6.22 zobrazuje příklad. Body A, B a C jsou podezřelé, vrstevnice postupně prochází úsečkami AE, BE, CE a CD.



Pořadí	body
1	**A**E
2	**B**E
3	**C**E
4	**C**D

Obr 6.23.: Následnost bodů

Po nalezení následnosti na celé vrstevnici projdeme seznam a budeme hledat situace, kdy jsme nejprve přešli od podezřelého bodu B k podezřelému bodu C a poté od podezřelého bodu C přešli zpět k podezřelému bodu B. Situace je schematicky načrtnuta na obrázku 6.23. Podezřelé jsou opět body A, B a C, detekovaný úsek obsahuje body B a C. Algoritmus tedy může u těchto dvou bodů obrátit ε-posun.



Pořadí	body
1	**B**E
2	**C**E
3	**C**D
4	**C**F
5	**C**G
6	**C**H
7	**B**H

Obr 6.24.: Detekovaný chobot

Tímto způsobem zjistíme body, jejichž směr ε-posunu je potřeba pro tuto vrstevnici změnit. Opravíme tedy posun a spustíme znovu výpočet vrstevnice, tentokrát už bez kontroly následnosti.

Druhá metoda detekce nepotřebuje triangulaci, ale pouze původní body; vychází z toho, že máme dvě sady vrstevnic, pro +ε i -ε. Je však potřeba ošetřit problémy v případě neuzavřených vrstevnic.

Pro každou vrstevnici zjistíme, které body původní množiny jsou uvnitř vrstevnice a které vně. Porovnáním těchto množin pro odpovídající vrstevnici v opačném posunu získáme seznam bodů, jejichž posun je potřeba změnit. V tomto případě by bylo možné postupovat i bez nového výpočtu vrstevnic tak, že by se použily části vrstevnic takové, které jsou bez chobotů. Implementace však je náročná pro sloučené vrcholky kopců, kdy se z jedné vrstevnice stanou dvě. Bližší informace o použitelných algoritmech lze získat v pracích [7, 8].

6.3.3 Vyhlazování

Vyhlazování se používá hlavně pro odstranění nežádoucího efektu lomového průběhu linie prostých vrstevnic. Vyhlazování probíhá postupně na jednotlivých vrstevnicích, tedy se jedná o lokální úpravu. Může k tomu být použito různých algoritmů. Práce [1] a [2] implementují vyhlazování pomocí kombinace kubické interpolace a váhového průměrování, nebo pomocí B-spline. První z těchto metod byla shledána adekvátní, žádné její další úpravy nejsou potřeba.

7 Implementace

Součástí této práce je pokračování práce na programu ing. Strycha. Program umožňoval vypočítat triangulaci bodové sítě, pomocí ní spočítat vrstevnice, vyhladit je a posléze triangulaci ručně editovat vkládáním povinných hran.

Program Moduler vznikal v průběhu této práce jako prostředí pro testování různých algoritmů. Aby bylo možné zkoušet co nejširší spektrum úprav, byl navržen jako modulární systém založený na principu toku dat. Jednotlivé moduly definují své vstupy a výstupy v předem definovaných datových typech, uživatel pak pouze spojuje vstupy jednotlivých modulů s výstupy jiných.

Aby byly výsledky této práce použitelné ve větší šíři, byla jako jazyk programu zvolena angličtina.

K dispozici bylo:

- předchozí práce ing. Strycha ve formě samostatného programu
- implementace Delaunayovy triangulace
- implementace výpočtu vrstevnic z triangulace
- implementace vyhlazování

Naprogramováno bylo:

- Modulární prostředí pro grafickou editaci výpočetního postupu
- triangulace, výpočet vrstevnic a vyhlazování přepsány jako jednotlivé moduly
- výpočet vrstevnic upraven pro použití metody ε-filtrace
- zobrazení výstupu buď pomocí nativních Win32 funkcí nebo pomocí OpenGL
- moduly pro optimalizace triangulace popsané v této práci
- moduly pro generování ukázkových a náhodných dat
- moduly pro statistiku triangulace

7.1 Modulární prostředí

Samotný program Moduler slouží pouze jako kostra a sjednocující aplikace pro jednotlivé moduly. Každý modul pak vykonává jednotlivé dílčí kroky výpočtu vrstevnic. K tomuto modelu bylo přistoupeno na základě požadavku na nejrůznější možnosti kombinace výpočetních a optimalizačních kroků, které by v tradičním monolitickém programu nebylo možné jednoduše podchytit. Předchozí práce [2] od modulárního modelu upustila kvůli požadavku na interaktivitu; tato práce se naopak snaží interaktivitu nahradit automatickými výpočty.

Aplikace Moduler obsahuje tyto části: manažer pluginů a jejich výpočetních modulů, manažer sdílené paměti, plánovač výpočtu a grafický editor. Manažer pluginů se stará o dynamické načítání knihoven s výpočetními moduly, o jejich inicializaci, registraci, a uvolnění. Manažer sdílené paměti má za úkol alokovat a uvolňovat paměť podle požadavku výpočetních modulů; tím, že je globální a sdílený, lze docílit toho, že jednotlivé moduly si mohou předávat data v paměti bez potřeby callback-mechanismů pro pozdější uvolnění paměti. Plánovač výpočtů má na starost spouštění výpočtů jednotlivých modulů sériově za sebou podle orientovaného grafu jejich závislostí. Aktuální implementace používá jediné vlákno výpočtu, je však možné ji rozšířit na simultánní zpracování úloh ve více vláknech a tím podporu víceprocesorových strojů (tam, kde to charakter výpočtu dovoluje).

Jednotlivé moduly jsou standardní soubory typu dynamic-link-library s příponou ml. Každý modul může obsahovat více výpočetních modulů, při své inicializaci všechny své moduly musí registrovat. Rozhraní bylo navrženo minimalisticky a tak, aby moduly mohly být naprogramovány v libovolném programovacím jazyce, který umožňuje vytvářet nativní dynamické knihovny pro platformu Win32. Rozhraní nepoužívá žádné pomocné prostředí (např. typu OLE), takže jeho naprogramování je velmi přímočaré.

7.2 Výpočetní moduly

Tím, že jednotlivé výpočty byly separovány do modulů, lze mimo jiné docílit i jejich znovu-použitelnost v jiných projektech, což bylo dokázáno ve spolupráci s komerčním software SiteFlow, který v současné době využívá modul triangulace.

Algoritmy převzaté z práce ing. Strycha byly od základu přepsány, používají jednoduchý objektový kód a jsou samostatně zapouzdřeny, tedy i na úrovni zdrojového kódu je lze jednoduše znovu použít.

Delaunayova triangulace byla modifikována pro ladící účely tak, aby křížící se hrany nepovažovala za kritickou chybu, ale ignorovala je. Tím je zjednodušeno zpracování dat i moduly, které nemají zaručeno, že se jimi produkované povinné hrany nekříží, a také případné kombinování více sad povinných hran z různých algoritmů jednoduchým sjednocením. Navíc byla pomocí různých optimalizací zrychlena o cca 50% proti původní implementaci.

Zobrazení výstupu bylo naprogramováno ve dvou různých variantách, jedna nativní Win32 a druhá používající OpenGL. První varianta zobrazuje výstup pouze v půdorysu, tedy dvourozměrně, za to však nabízí možnosti identifikace jednotlivých objektů pomocí kurzoru myši. Druhá varianta zobrazuje data trojrozměrně, podobně jako v práci ing. Strycha [2], oproti ní však má rozdílné ovládání myší, které je bližší technické praxi (program AutoCAD)

8 Výsledky v praxi

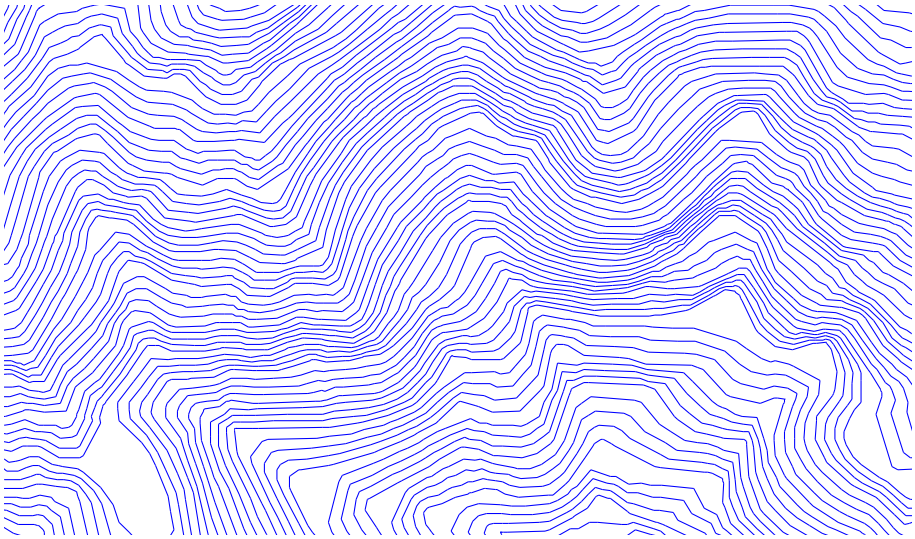
Algoritmy 4 (jednoduchá eliminace) a 5 (modifikace středové osy) byly zkoušeny na několika reálných datových množinách. Výsledky jsou velice příznivé. Na obrázcích níže vidíme grafické výstupy algoritmů. Je zřejmé, že došlo k celkovému zlepšení triangulace – sice ne do úplně ideální podoby; ale velká část práce kartografa byla úspěšně nahrazena automatickým zpracováním. Pro porovnání počtu skupin plochých trojúhelníků (což je víceméně měřítko počtu fiktivních spočinků) vizte následující tabulku:

Data	Hory.tin		Šumava 1.tin		Vstup4.tin	
Počet bodů	19819		1472		12057	
Počet skupin před	2031		56		69	
Počet skupin po (Alg. 4)	583	28 %	15	26 %	23	33 %
Počet skupin po (Alg. 4 mod)	593	29 %	16	28 %	23	33 %
Počet po 1. iteraci Alg 5	591	29 %	22	39 %	43	62 %
Počet po 2. iteraci Alg. 5	403	19 %	16	28 %	35	50 %

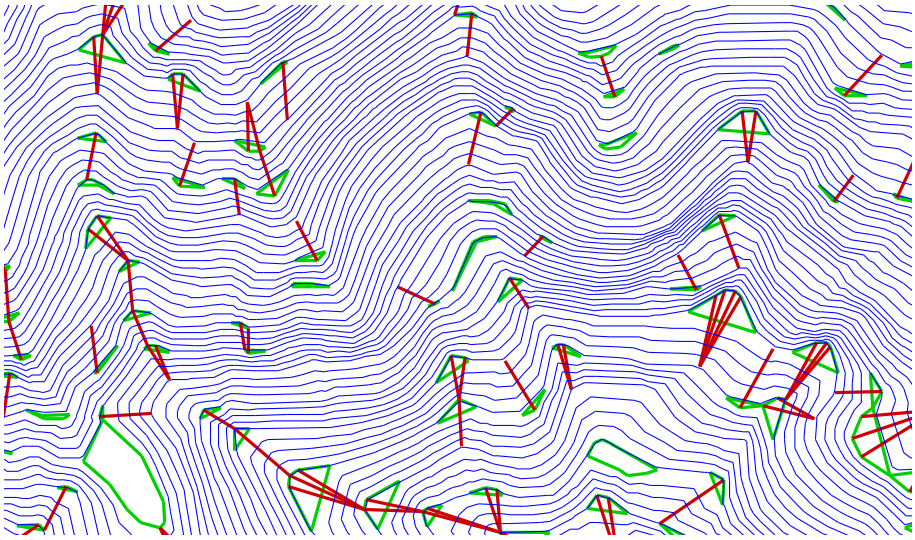
Datové množiny Hory a Šumava vznikly digitalizací vrstevnic, Vstup4 obsahuje digitalizované vrstevnice tří obcí a geodetické zaměření silnic mezi nimi. Algoritmus 4 mod označuje použití Algoritmu 4 a navíc odstranění všech povinných hran v oblastech s bodem uvnitř skupiny plochých trojúhelníků (volba *Only for simple areas*)

Prostým porovnáním statistik by se mohlo zdát, že po první iteraci je metoda jednoduché eliminace úspěšnější, protože počet spočinků po optimalizaci je menší. Je však nutno vzít v úvahu také subjektivní kvalitu přidávaných povinných hran, která je u jednoduché eliminace značně horší. Vyšší počet zbylých skupin při použití metody středové osy (Alg. 5) je dán hlavně filtracemi, které byly na výsledné povinné hrany uplatněny. Časté jsou taky případy, že se jedna skupina rozdělí na dvě menší. Proto byla ozkoušena i druhá iterace stejné metody. Výsledky jsou lepší než při jednoduché eliminaci, jediným problémem byl vznik křížících se povinných hran.

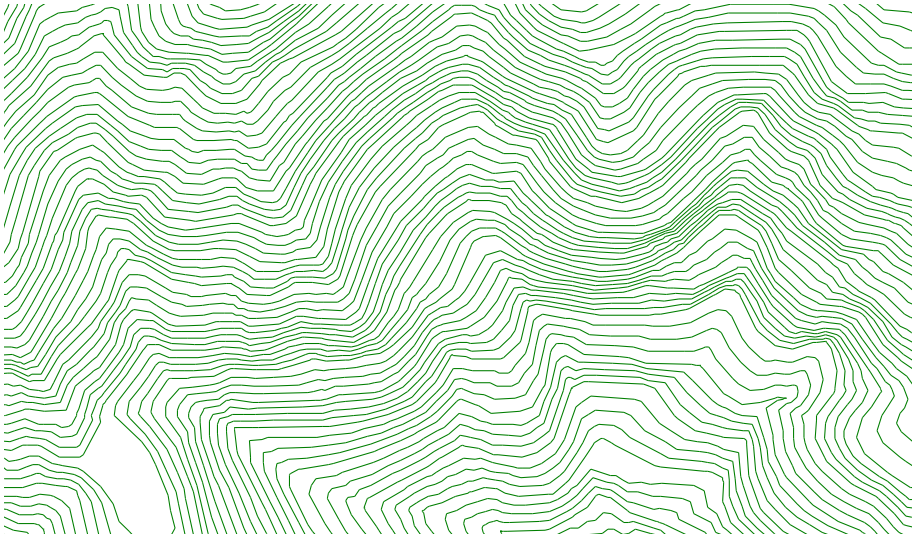
Výsledky Alg. 5 jsou velice příznivé – eliminoval všechny jednoduché spočinky a většinu kombinovaných spočinků tak, jak by postupoval člověk-kartograf. Velkým úspěchem je i fakt, že množství nesprávně vytvořených hran je naprosto minimální, pohybuje se v jednotkách tisícin počtu původních fiktivních spočinků.



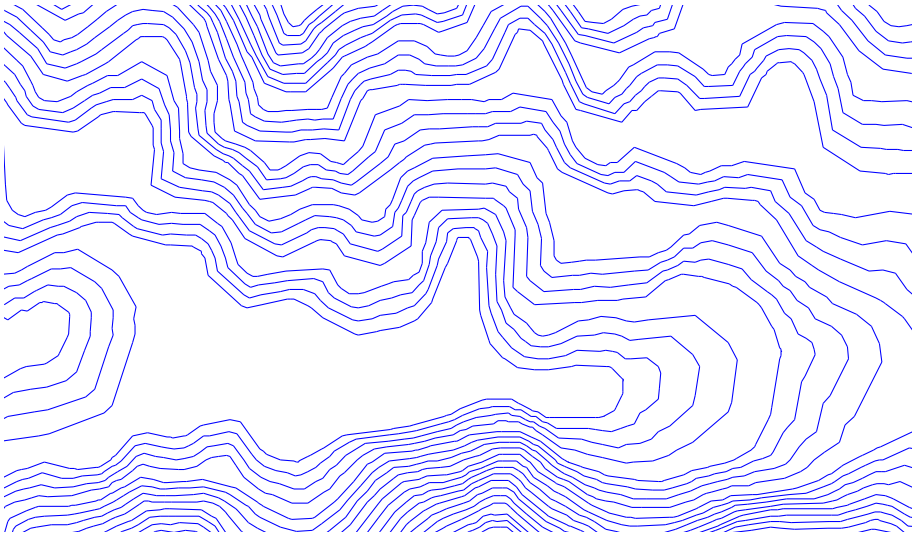
Obr 8.1.: Hory.tin, vrstevnice před optimalizací



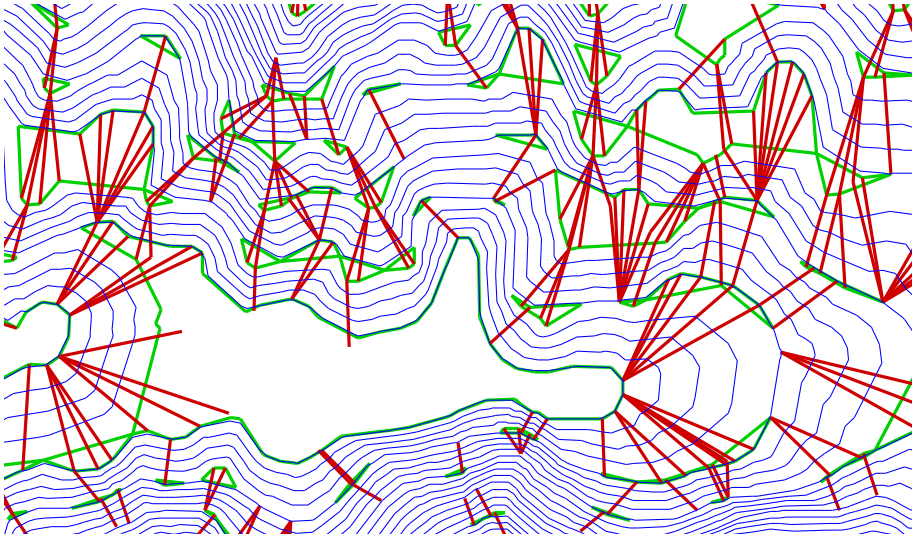
Obr 8.2.: Hory.tin, vrstevnice před optimalizací a opravy Alg. 4



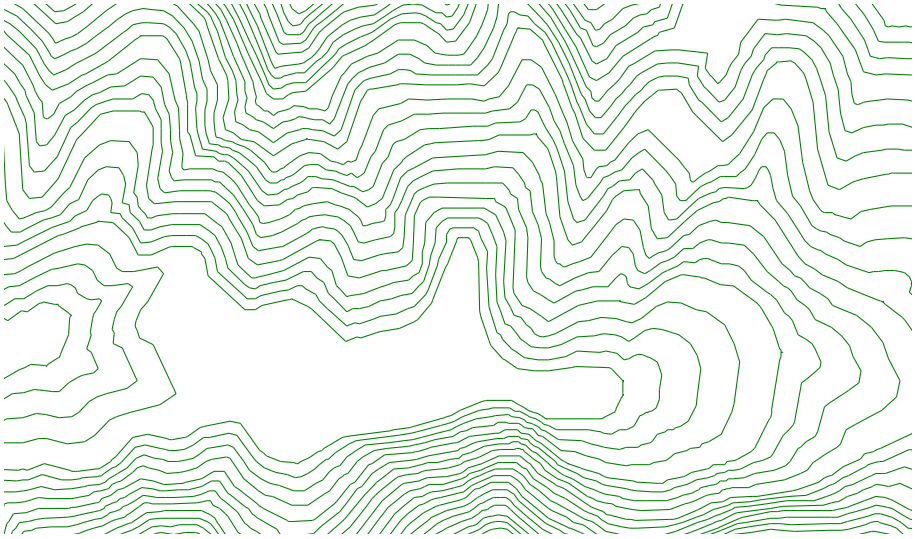
Obr 8.3.: Hory.tin, vrstevnice po optimalizaci Alg. 4



Obr 8.4.: Hory.tin, vrstevnice před optimalizací



Obr 8.5.: Hory.tin, vrstevnice před optimalizací a opravy Alg. 5



Obr 8.6.: Hory.tin, vrstevnice po optimalizaci Alg. 5

9 Shrnutí

Na závěr shrňme získané poznatky o optimalizacích trojúhelníkových sítí. Bylo zde navrhnuto několik metod, které mohou zlepšit kvalitu generovaných vrstevnic. Další metody byly ozkoušeny s negativními výsledky, nastíněny jejich problémy a možné případné směry řešení.

Dvě prezentované metody – jednoduchá eliminace spočinků a modifikovaná metoda střední osy – byly shledány schopnými nahradit podstatnou část práce kartografa pomocí automatického přidávání povinných hran. Ukázalo se, že však nelze všechny případy vyřešit pouze přidáváním povinných hran, pro ještě lepší výsledky ve speciálních případech by bylo zapotřebí i zahušťovat datovou množinu.

Další dvě metody (křížení vrstevnic a ε–filtrace), ač neověřeny implementací, byly alespoň teoreticky analyzovány a byla naznačena možnost jejich implementace.

Pro účely této práce bylo naprogramováno rozšiřitelné programové prostředí, které usnadňuje kombinování jednotlivých prezentovaných metod a také porovnávání jejich výsledků. Většina obrázků v této práci vznikla právě použitím tohoto programu a příslušných optimalizačních modulů.

10 Prameny

[1] Čermák P.: *Výpočet vrstevnic na trojúhelníkové síti (diplomová práce)*, Západočeská univerzita v Plzni, 2002

[2] Strych V: *Triangulace a editování vrstevnic (diplomová práce)*, Západočeská univerzita v Plzni, 2003

[3] Kolingerová I., Strych V. , Čada V.: *Using Constraints in Delaunay and Greedy Triangulation for Contour Lines Improvement*, in: M.Bubak et al. (Eds.): ICCS 2004, LNCS 3039, pp.123-130, 2004, Springer Verlag Berlin Heidelberg, 2004

[4] Brandli M., Schneider B.: *Shape Modeling And Analysis Of Terrain*, International Journal of Shape Modeling 1 (2), 167-189, 1994

[5] www.wikipedia.org, sekce *Triangulation*

[6] Shewchuk J. R., *Triangle* program and documentation

[7] Ganapathy S., Dennehy T. G.: *A New General Triangulation Method For Planar Contours*, Computer Graphics Volume 16, Number 3, July 1982

[8] Boissonat J., Geiger B.: *Three-dimensional Reconstruction Of Complex Shapes Based On The Delaunay Triangulation*, INRIA research report, April 1992.

11 Příloha A – uživatelská příručka programu Moduler

Výpočet vrstevnic lze abstrahovat jako posloupnost jednotlivých kroků, které přidávají a případně modifikují data od vstupních k výstupním. Každý krok je definován svým algoritmem, typem dat, které potřebuje na svém vstupu a typem dat, která produkuje. Protože způsobů, jak získat z daných vstupních dat výstupní, může být více, každý takový algoritmus byl izolován do tzv. Výpočetního modulu. Ten tedy obsahuje deklaraci vstupních dat, výstupních dat, vlastního algoritmu a případně i nastavení parametrů algoritmu.

Jako příklad uveďme Delaunayovu triangulaci, která jako vstupní data očekává množinu bodů v trojrozměrném prostoru, seznam povinných hran, na svém výstupu produkuje seznam hran, seznam trojúhelníků a seznam vzájemných sousedů trojúhelníků. Algoritmus nemá žádné parametry, tedy nepotřebuje žádná nastavení.

Dva druhy modulů zaslouží ještě další pozornost: načítání a ukládání (příp. zobrazování) dat. Tyto sice neobsahují algoritmy, ale protože jejich struktura je jinak velmi podobná výpočetním modulům, jsou mezi ně pro jednoduchost zařazeny.

Jako příklad vezměme OpenGL renderer, který zobrazuje data pomocí OpenGL. Jeho vstupy jsou body, hrany, trojúhelníky, vrstevnice, povinné hrany a zvýrazněné body, hrany nebo trojúhelníky. Samozřejmě není nutné všechna tato data dodávat, modul pro každé zobrazované elementy inteligentně rozhodne, zda má pro jejich zobrazení dostatek informací. Stačí tedy například zadat body, trojúhelníky a vrstevnice, pak budou zobrazeny pouze tyto položky.

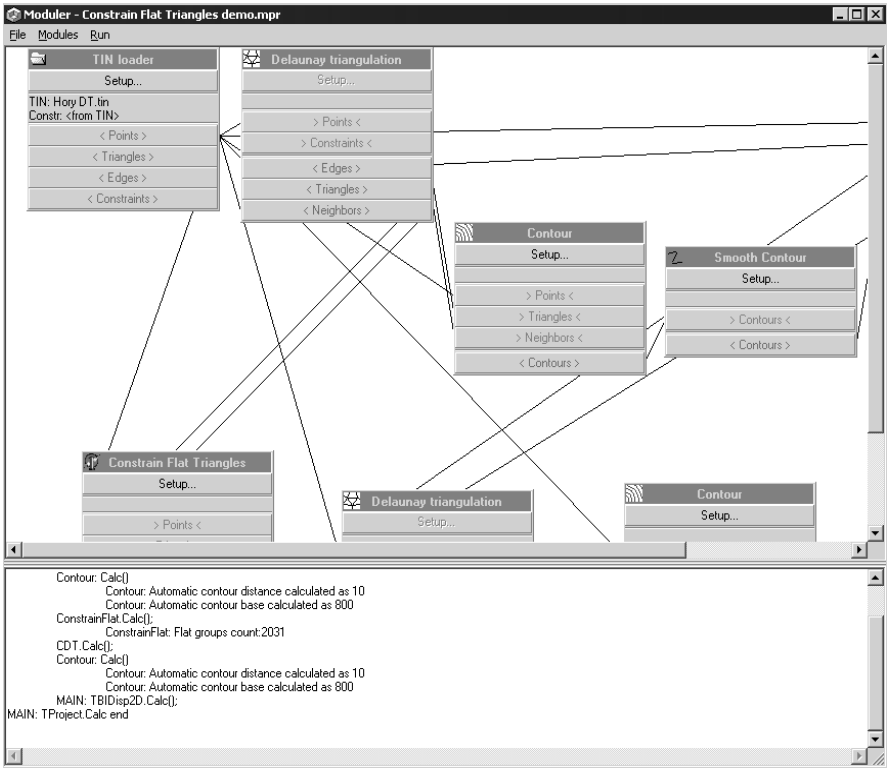
Moduler je tzv. SDI aplikace (*Single Document Interface*), tedy pracuje se v něm s jediným otevřeným dokumentem. Dokumentem se rozumí seznam instancí výpočetních modulů a seznam propojení jejich vstupů a výstupů. Typická práce v Moduleru tedy sestává z přidávání a odebírání modulů a editace datového toku mezi jejich výstupy a vstupy.

Po spuštění programu Moduler.exe se zobrazí hlavní okno (Obr. 11.1).

V horní části je standardně umístěno menu, prostřední největší část je pracovní plocha a spodní část je okno výpisu. Mezi pracovní plochou a oknem výpisu je rozdělovač, pomocí nějž lze změnit poměr velikosti těchto dvou oken.

Menu slouží pro ovládání aplikace, pomocí něj lze načíst, uložit nebo založit nový dokument, otevřít 5 naposled otevřených dokumentů, ukončit program; přidat nebo smazat modul, případně zobrazit seznam možných modulů, a spustit výpočet.

V pracovní ploše je zobrazen obsah dokumentu, tj. jednotlivé výpočetní moduly a vazby mezi jejich vstupy a výstupy. Zde probíhá většina editační práce v programu. Veškeré ovládání se provádí myší, pro některé akce existují klávesové zkratky, které budou popsány později.



Obr 11.1: Hlavní okno aplikace

Okno výpisu slouží pro zobrazování stavu a běhu programu. Při spuštění výpočtu sem zapisují jednotlivé moduly informaci o svém běhu, stejně tak jako běhové chyby, ke kterým může v průběhu výpočtu dojít (jako typického zástupce jmenujme křížící se povinné hrany, které modul triangulace ignoruje, ale sem zapíše poznámku o jejich existenci). V případě neošetřené výjimky se sem zapíše výpis zásobníků a další ladící informace.

Každá instance výpočetního modulu je reprezentována panelem s modrým záhlavím, červenými tlačítky, které reprezentují vstupy, a zelenými tlačítky, které reprezentují výstupy. Propojení vstupů s výstupy se zobrazuje jako černá čára mezi příslušnými tlačítky.

Modul lze přemístit pomocí táhnutí myší za jeho záhlaví. Pracovní plocha se automaticky roztáhne, je-li potřeba více místa pro zobrazení modulů.

Nové propojení se vytvoří kliknutím na vstup a dalším kliknutím na příslušný výstup (příp. v obráceném pořadí). Samozřejmě každý vstup může dostávat data pouze z jediného výstupu, Moduler tedy automaticky smaže případnou hranu, která již k danému vstupu před vložením existovala.

Stejným postupem lze propojení smazat, tj. kliknutím na vstup a dalším kliknutím na vstup. Pokud již takové propojení existuje, je vymazáno.

Spuštění výpočtu se provede z menu Run položkou Run (nebo klávesou F9). Moduler pak prochází jednotlivé instance modulu, zjistí závislosti jejich vstupů a výstupů. Pokud nastane konflikt (kružnice v orientovaném grafu), ohlásí chybu a skončí. V případě, že konflikty nenastaly, postupně se spustí jednotlivé instance výpočetních modulů.

11.1 Příklad

Vezměme jako příklad jednoduchou úlohu: Máme množinu bodů v prostoru a chceme zobrazit vrstevnice, které jsou těmito body tvořeny. Spočítáme je pomocí jednoduché Delaunayovy triangulace bez optimalizací, zobrazit je chceme trojrozměrně.

Založíme tedy nový dokument a vložíme do něj postupně moduly TIN loader, Delaunay triangulation, Contour a OpenGL renderer.

V nastavení modulu TIN loader je potřeba zvolit jméno souboru se vstupními daty (formát dat vizte Příloha A), klikneme tedy na tlačítko „Setup…“ modulu TIN loader a vyplníme jméno souboru v kolonce TIN filename. Případně lze použít tlačítko Browse (otevřená složka) pro nalistování souboru. Dialog zavřeme tlačítkem OK.

Dále vytvoříme vazby: Body na výstupu modulu TIN loader chceme poslat na vstup modulu Delaunay triangulation. Tuto vazbu vytvoříme kliknutím na zelené tlačítko Points modulu TIN loader a pak na červené tlačítko Points modulu Delaunay triangulation. Stejným způsobem vytvoříme vazbu mezi zelenými Points modulu TIN loader a červenými Points modulu Contour. Protože chceme zobrazovat pouze vrstevnice, nepošleme body modulu OpenGL renderer.

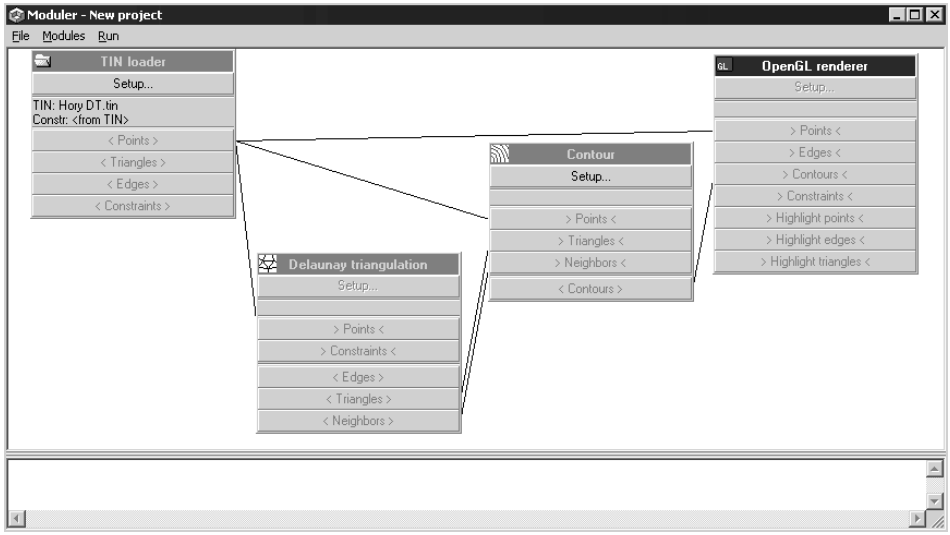
Modul Delaunay triangulation nemá žádné nastavitelné parametry, proto jeho tlačítko „Setup…“ není aktivní.

Dalším krokem je vytvořit vrstevnice z triangulace získané v modulu Delaunay triangulation. Modul Contour, který počítá vrstevnice, potřebuje na vstupu body a trojúhelníky; dostane-li i sousedy trojúhelníků, ušetří na jejich výpočtu. Propojíme tedy výstup trojúhelníků a sousedů modulu Delaunay triangulation se vstupy trojúhelníků a sousedů modulu Contour.

Výpočet modulu Contour lze ovlivnit parametry, můžeme je tedy nastavit v dialogu vyvolaném stiskem tlačítka „Setup…“. Můžeme zvolit, v jakém rozestupu se vrstevnice mají spočítat. Buď nastavíme pevný rozestup (*Manual*), nebo necháme automaticky spočítat rozestup tak, aby byl výsledný počet vrstevnic větší nebo roven zadanému číslu (*Automatic*).

Prozatím námi zkonstruovaný řetězec spočítá vrstevnice, ale nemá žádný výstup. K tomu použijeme modul OpenGL renderer, který zobrazí výsledky výpočtu trojrozměrně. Propojme tedy jeho vstup Contours s výstupem Contours modulu Contour.

Nyní máme hotový návrh výpočtu (Obr. 11.2), ten si můžeme uložit do dokumentu pomocí menu *File*, položky *Save*. Hlavně ale můžeme výpočet spustit pomocí menu *Run*, položky *Run*. Zobrazí se okno s průběhem výpočtu, do okna výpisu se zapíší spouštěné moduly a nakonec se zobrazí okno modulu OpenGL renderer s výslednými vrstevnicemi.



Obr 11.2.: Příklad jako projekt v Moduleru

Další příklady projektů i vstupních dat jsou uloženy na CD, které je součástí této práce, ve složce Data.

11.2 Přehled menu

Menu	Položka	Akce
File	New	Založí nový dokument. Pokud stávající není uložen, zeptá se, zda uložit.
	Open…	Načte uložený dokument. Zobrazí dialog pro výběr dokumentu uloženého na disku. Není-li stávající dokument uložen, zeptá se, zda uložit.
	Save	Uloží dokument na disk. Jedná-li se o nový dokument, zeptá se nejdříve na požadované umístění a jméno.
	Save as…	Uloží dokument na disk pod zadaným jménem.
	1 - 5	Obsahuje jména 5 naposled používaných dokumentů, příslušný dokument po kliknutí načte. Není-li stávající dokument uložen, zeptá se, zda uložit.
	Exit	Ukončí Moduler. Není-li stávající dokument uložen, zeptá se, zda uložit.
Modules	Add…	Přidá novou instanci výpočetního modulu. Zobrazí dialog s výběrem typu modulu. Nový modul přidá do pracovní plochy vpravo nahoru.
	Delete current…	Smaže aktivní instanci výpočetního modulu. Před smazáním si vyžádá potvrzení.
	List available…	Zobrazí seznam všech dostupných typů výpočetních modulů.
Run	Run	Spustí výpočet.

11.3 Přehled klávesových zkratek

Klávesa	Ekvivalent	Akce
Ctrl + N	File – New	Založí nový dokument
Ctrl + O	File – Open…	Načte uložený dokument
Ctrl + S	File – Save	Uloží dokument
Alt + X	File – Exit	Ukončí Moduler
Ins	Modules – Add…	Přidá instanci výpočetního modulu
Del	Modules – Delete current…	Smaže aktivní instanci výpočetního modulu
F9	Run – Run	Spustí výpočet

11.4 Přehled výpočetních modulů

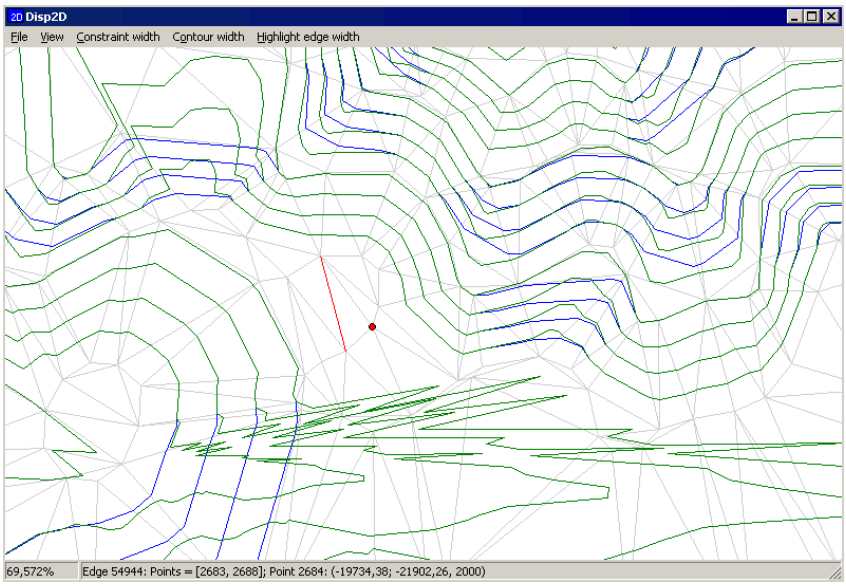
2D renderer

Základní zobrazovací modul. Zobrazuje data ve 2D pohledu shora. Je implementován přímo v Moduleru. Vstupy: Body (*Points*), hrany (*Edges*), povinné hrany (*Constraints*), vrstevnice (*Contours*), zvýrazněné body (*Highlight points*), zvýrazněné hrany (*Highlight edges*), trojúhelníky (*Triangles*), zvýrazněné trojúhelníky (*Highlight triangles*), druhé vrstevnice (*Contours 2*), druhé zvýrazněné trojúhelníky (*Highlight triangles 2*). Výstupy: žádné. Nastavení: žádné.

Zobrazí data dodaná na vstupu v půdorysu. Zobrazuje pouze zadané položky, není potřeba dodávat úplně všechna data. Např. pokud dodáme body a hrany, zobrazí se pouze triangulace (pomocí hran) a případně na požádání i body. Naopak můžeme nechat zobrazit pouze vrstevnice bez jakýchkoliv dalších objektů.

Zvýrazněné objekty představují možnost, jak zobrazit výsledky různých detekčních algoritmů. Například modul FlatTriangles detekuje ploché trojúhelníky, které pak lze pomocí vstupu Highlight triangles zobrazit proti ostatní triangulaci zvýrazněně. Typicky se tato možnost používá pro povinné hrany, které se mohou zvýraznit buď jako Highlight edges, nebo jako Constraints (obojí fungují stejně).

Vstup *Contours 2* lze použít pro srovnání výsledků dvou různých cest výpočtů vrstevnic; vrstevnice obou skupin lze zobrazit současně, vybrat jednu skupinu, případně i nechat zobrazovač rychle přepínat mezi skupinami.



Obr 11.3: Ukázkové okno 2D rendereru

V horní části okna je menu, pomocí kterého lze zvolit zobrazení jednotlivých datových prvků a tloušťky čar pro povinné hrany, vrstevnice a zvýrazněné hrany. Navíc lze použít příkaz *Cycle contours*, které zapne nebo vypne rychlé přepínání zobrazení vrstevnic 1 a 2. Pomocí *File*, *Save as metafile...* lze obrázek uložit jako vektorový metasoubor EMF nebo WMF.

Prostřední, největší část okna slouží k zobrazení. Ovládá se pomocí myši takto:

Levé tlačítko + posun vlevo, vpravo – rotace obrazu

Ctrl + levé tlačítko nebo prostřední tlačítko nebo levé + pravé tlačítko – posun obrazu

Pravé tlačítko + posun vlevo, vpravo nebo kolečko myši – zoom

Po stabilizaci kurzoru se zvýrazní bod, který je nejblíže kurzoru (jsou-li zadány body na vstupu); hrana, která je nejblíže kurzoru (jsou-li ve vstupních datech hrany); do trojúhelníku, jehož střed je nejblíže kurzoru, se vykreslí smyčka ukazující na jeho primární vrchol a zároveň se očísluje pořadí jeho hran (žlutě) a bodů (světle modře).

Vespod v okně je stavový řádek, kde se vypisuje aktuální měřítko a pak informace o zvýrazněných objektech – číslo zvýrazněného trojúhelníku, čísla jeho bodů, číslo zvýrazněné hrany, čísla jejích bodů, číslo zvýrazněného bodu a jeho tři souřadnice.

Constrain Flat Triangles

Implementuje výpočet povinných hran pro eliminaci fiktivních spočinků (vizte podkapitolu Eliminace fiktivních spočinků kapitoly Optimalizace). Je implementován v modulu ConstrainFlat.ml Vstupy: body (*Points*), trojúhelníky (*Triangles*), sousedé trojúhelníků (*Neighbors*). Výstupy: povinné hrany (*Constraints*), hraniční hrany (*Group edges*). Nastavení: použitá metoda výpočtu.

Implementuje metodu optimalizace popsanou v kapitole Eliminace fiktivních spočinků. Pro porovnání zůstalo zachováno všech pět verzí algoritmu, mezi kterými lze přepínat v nastavení:

- *All* – vynutí přidání všech povinných hran (Alg. 1)
- *Only non-intersecting* – přidá hrany, které nekříží hraniční hrany skupiny (Alg. 2)
- *Only convex* – přidá pouze hrany, které navíc kříží příslušnou startovací hranu (Alg. 3)
- *Only for convex areas* – přidá hrany pouze z oblastí, které nejsou postiženy předchozími dvěma výjimkami (a tedy jejich hranice jsou konvexní) (Alg. 4)
- *Only for simple areas* – přidá hrany pouze z předchozích oblastí, ve kterých se navíc nevyskytuje žádný plochý trojúhelník se třemi plochými sousedy. (Alg. 4 mod)

Na svém vstupu vyžaduje body a trojúhelníky; pokud jsou zadáni i sousedé trojúhelníků (např. z výstupu triangulace), ušetří se na jejich výpočtu. První iterace garantuje vytvoření povinných hran bez křížení, druhá iterace již tuto garanci nemá.

Contour

Počítá vrstevnice z trojúhelníkové sítě. Vstupy: body (*Points*), trojúhelníky (*Triangles*), sousedé trojúhelníků (*Neighbors*). Výstupy: vrstevnice (*Contours*). Nastavení: vertikální vzdálenost vrstevnic, směr posunu.

Spočítá vrstevnice jako průsečíky s hranami trojúhelníkové sítě. Pomocí sousednosti trojúhelníků postupuje po vrstevnici. Singulární případy (vrcholy triangulace ve stejné výšce jako vrstevnice) postupuje tak, že přičte nebo odečte (podle nastavení směru posunu) malou konstantu.

Nastavení vertikální vzdálenosti je možno provést ručně (Manual) nebo nechat modul automaticky spočítat rozestup tak, aby bylo ve výstupu minimálně N hladin (Auto). V případě automatického nastavení se rozestupy nastavují jako čísla tvaru 10^n, 2×10^n, 5×10^n (tedy např. 0.1, 0.2, 0.5, 1, 2, 5, 10, 20, 50 atp.)

Na vstupu modul vyžaduje body a trojúhelníky; pokud jsou zadáni i sousedé trojúhelníků, ušetří se na jejich výpočtu paměť i výpočetní síla.

Delaunay triangulation

Počítá Delaunayovu triangulaci s povinnými hranami. Vstupy: body (*Points*), povinné hrany (*Constraints*). Výstupy: hrany (*Edges*), trojúhelníky (*Triangles*), sousedé trojúhelníků (*Neighbors*). Nemá další nastavení.

Triangulace se interně počítá pomocí DAG-lokalizace. Velkého zrychlení je dosaženo pomocí speciální hromadné alokace DAG uzlů.

V případě, že je nalezena povinná hrana křížící se s jinou povinnou hranou, je ta pozdější z nich ignorována a vygeneruje se varování do okna výpisu.

Edge joiner

Spojí dvě množiny hran a / nebo povinných hran nad stejnou množinou bodů. Vstupy: 2x hrany (*Edges*), 2x povinné hrany (*Constraints*). Výstupy: hrany (*Edges*), povinné hrany (*Constraints*). Nastavení: žádné.

Umožňuje spojit dvě množiny hran nebo povinných hran nad stejnou množinou bodů. Hrany na vstupu 2 připojí za hrany na vstupu 1. Neprovádí žádné kontroly křížení!

Pomocí tohoto jednoduchého modulu lze například spojit výsledky dvou iterací modulu Medial Axis Flat nebo Constrain Flat Areas.

Modul slouží také jako proof-of-concept modularity, na rozdíl od ostatních modulů je naprogramován v prostředí Microsoft Visual C++ 6. Navíc je obsažen ve stejné dynamické knihovně jako modul Point joiner, tedy předvádí, jak spojit více modulů do jediné DLL knihovny.

FallLine

Automatická tvorba hřbetnic a údolnic. Vstupy: body (*Points*), Indicie (*Indices*). Výstup: Povinné hrany (*Constraints*). Nastavení: žádné.

Implementuje algoritmus hledání spádnic navržený v podkapitole Automatická tvorba hřbetnic a údolnic. Z bodů označených pomocí Indicií vytvoří povinné hrany k blízkým vrcholům s co nejmenším nenulovým sklonem. Vyhledá se 6 (konstanta `NUM_NEAREST`) nejbližších bodů, z těch se vybere ten s nejmenším nenulovým sklonem.

Flat triangles

Detekce plochých trojúhelníků. Vstupy: body (*Points*), trojúhelníky (*Triangles*). Výstupy: ploché trojúhelníky (*Triangles*), ploché trojúhelníky se 3 plochými sousedy (*Locked flat triangles*). Nastavení: žádné.

Detekuje ploché trojúhelníky a umožní jejich vizualizaci v rendereru pomocí vstupu Highlight triangles. Navíc umožňuje detekovat i ploché trojúhelníky se třemi plochými sousedními trojúhelníky. Tyto tzv. flat-locked trojúhelníky jsou příznakem složitějších fiktivních spočinků, algoritmus Constrain flat triangles umožňuje ignorovat všechny skupiny, které takovéto trojúhelníky obsahují.

LenCut

Odebere nejdelší hrany. Vstupy: body (*Points*), hrany (*Edges*). Výstupy: hrany (*Edges*). Nastavení: žádné.

Najde nejdelší hranu a podle ní stanoví práh pro odebírání jako 80 procent (konstanta `CutThreshold`) její délky. Pracuje jako jednoduchý statistický modul. Ukazuje se, že tyto hrany odpovídají nežádoucí.trojúhelníkům na okraji triangulace.

Hranový vstup přijímá jak hrany triangulace, tak povinné hrany. Délky jsou počítány 2D.

LenHist

Histogram délky hran. Vstupy: body (*Points*), hrany (*Edges*). Výstupy: žádné. Nastavení: žádné.

Spočte a zobrazí histogram délek hran. Počítá délky jak 2D (zobrazeny žlutě), tak 3D (zobrazeny červeně).

LocalExtrems

Vybere body lokálních extrémů. Vstup: body (*Points*). Výstupy: Lokální maxima (*Local maxima*), lokální minima (*Local minima*), lokální extrémy (*Local extrems*). Nastavení: Grid pro výpočet.

Vstupní body rozdělí do mřížky podle nastavených rozměrů a z každé buňky mřížky vybere maximum a minimum, pokud existují.

Medial Axis Flat

Spočítá povinné hrany modifikovanou metodou středové osy. Vstupy: body (*Points*), trojúhelníky (*Triangles*), sousedé trojúhelníků (*Neighbors*). Výstupy: povinné hrany (*Constraints*), startovací trojúhelníky (*Starting triangles*), hranice skupin plochých trojúhelníků (*Group edges*). Nastavení: metoda výběru startovacích trojúhelníků, parametry vyhledávání pokračování, omezení počtu hledání.

Implementuje modifikovanou metodu středové osy, jak byla popsána v kapitole Optimalizace. Základním výstupem jsou povinné hrany, další výstupy jsou pro prezentační a ladící účely. Na vstupu vyžaduje alespoň body a trojúhelníky; jsou-li zadáni i sousedé, ušetří se na jejich výpočtu. První iterace garantuje vytvoření povinných hran bez křížení, druhá iterace již tuto garanci nemá.

V nastavení lze povolit zjednodušené procházení pro 2 ploché sousedy (Direct walk). Parametrem Maximum angle se nastaví práh pro filtraci povinných hran; je-li z některého bodu skupiny vidět povinnou hranu pod úhlem větším, hrana nebude přidána. Length decision threshold coeff slouží pro zadání prahového poměru délek stran trojúhelníka, při jeho překročení se pro procházení použije hrana automaticky bez ohledu na úhlové kritérium. Položka Starting triangles umožňuje nastavit metodu volby startovacího trojúhelníku:

- *Shortest (any) edge* –trojúhelník s nejkratší (libovolnou) stranou
- *Shortest boundary edge* –trojúhelník s nejkratší hranou na hranici skupiny

- *Minimum area* – trojúhelník s nejmenším obsahem
- *Two boundary edges, min area* – trojúhelník se dvěma hranami na hranici skupiny s nejmenším obsahem, poté pouze s nejmenším obsahem.
- *Two boundary edges, shortest boundary edge* – trojúhelník se dvěma hranami na hranici skupiny s nejkratší hranou na hranici skupiny, poté pouze s nejkratší hranou na hranici skupiny
- *Two boundary edges, shortest non-boundary edge (N/I)* – neimplementováno
- *Two boundary edges, min area only* – trojúhelník se dvěma hranami na hranici skupiny s nejmenším obsahem
- *Two boundary edges, shortest boundary edge, only* – trojúhelník se dvěma hranami na hranici skupiny s nejkratší hranou na hranici skupiny
- *Two boundary edges, shortest non-boundary edge (N/I)* – neimplementováno
- *The longest edge on the boundary* – trojúhelník s nejdelší hranou na hranici skupiny
- *The longest edge on the boundary, single* – trojúhelník s nejdelší a zároveň jedinou hranou na hranici skupiny
- *Two boundary edges, min area, then longest on boundary* – trojúhelník se dvěma hranami na hranici skupiny s nejmenším obsahem, pak trojúhelník s nejdelší hranou na hranici skupiny.

Při výpočtu vypisuje do okna výpisu počet detekovaných skupin plochých trojúhelníků.

OpenGL renderer

Zobrazuje výstupní data trojrozměrně pomocí OpenGL. Vstupy: body (*Points*), hrany (*Edges*), vrstevnice (*Contours*), povinné hrany (*Constraints*), zvýrazněné body (*Highlight points*), zvýrazněné hrany (*Highlight edges*), zvýrazněné trojúhelníky (*Highlight triangles*). Výstup: žádný. Nastavení: žádné. Ke své činnosti potřebuje nainstalovanou podporu OpenGL v systému, jmenovitě knihovnu OpenGL32.dll

Využívá hardwarové akcelerace pro vykreslení 3D bodů, triangulace i vrstevnic. Zobrazuje pouze zadané položky, není potřeba dodávat úplně všechna data. Např. pokud dodáme body a hrany, zobrazí se pouze triangulace (pomocí hran) a případně na požádání i body. Naopak můžeme nechat zobrazit pouze vrstevnice bez jakýchkoliv dalších objektů. Stejně jako v modulu 2D renderer lze navíc zobrazit zvýrazněné prvky – body, hrany i trojúhelníky.

Ovládání zobrazení pomocí myši:

- levé tlačítko a táhnutí – posun obrazu ve směru táhnutí
- pravé tlačítko a táhnutí – rotace obrazu

Point joiner

Spojí dvě množiny bodů spolu s jejich hranami, povinnými hranami a trojúhelníky. Vstupy: 2x body (*Points*), 2x hrany (*Edges*), 2x povinné hrany (*Constraints*), 2x trojúhelníky (*Triangles*). Výstupy: body (*Points*), hrany (*Edges*), povinné hrany (*Constraints*), trojúhelníky (*Triangles*). Nastavení: žádné.

Umožňuje spojit dvě množiny bodů. Ke každé množině lze navíc specifikovat i její hrany, povinné hrany a trojúhelníky, ty budou taktéž spojeny a správně přečíslovány. Pozor, tento modul předpokládá, že vstupy 2 jsou relativní vůči bodům na vstupu 2, takže jej nelze použít například pro sloučení dvou sad povinných hran, pro tento účel je potřeba použít modul Edge joiner.

Modul slouží také jako proof-of-concept modularity, na rozdíl od ostatních modulů je naprogramován v prostředí Microsoft Visual C++ 6. Navíc je obsažen ve stejné dynamické knihovně jako modul Edge joiner.

Poisson Generator

Generátor bodů se (pseudo)-poissonovským rozdělením. Vstup: žádný. Výstup: body (*Points*). Nastavení: počet bodů (*Output points*), počet generátorů (*Generators*), rozměry X, Y, Z (*X, Y, H bounds*)

Nejdříve náhodně zvolí všechny souřadnice generátorů, pak volí náhodně souřadnice X a Y bodů. Pro každý spočítá vzdálenost od každého generátoru a tu použije jako váhu pro průměrování souřadnic generátorů, čímž dostane souřadnici Z nového bodu. Momentální implementace používá jako váhovou funkci $f(dist) = \min(100\,000; 10\,000 / dist^2)$.

Výstupem jsou body ve tvaru parabolických kopců s různými výškami.

Remove boundary triangles

Odstraňuje hubené trojúhelníky na okraji triangulace. Vstupy: body (*Points*), trojúhelníky (*Triangles*), sousedé trojúhelníků (*Neighbors*). Výstup: trojúhelníky (*Triangles*). Nastavení: velikost minimálního úhlu ve stupních.

Projde trojúhelníky na okraji triangulace a je-li některý z jejich vnitřních úhlů menší, než zadaný práh, je tento trojúhelník odstraněn. Tato eliminace je opakována, dokud je v jednom cyklu alespoň jeden trojúhelník odstraněn.

Saver

Ukládá spočítaná data. Vstupy: body (*Points*), trojúhelníky (*Triangles*), povinné hrany (*Constraints*), vrstevnice (*Contours*). Výstup: žádný. Nastavení: soubory pro ukládání dat.

Uloží body, triangulaci nebo vrstevnice do příslušných souborů. Používá stejný formát jako TIN Loader, takže lze data po zpracování uložit a pak libovolně načítat např. pro nejrůznější statistiky.

Lze použít jména souborů s relativní cestou. Moduler při otevření projektu nastaví aktuální cestu na složku s projektem, takže všechny relativní cesty se vztahují k této složce.

Popisy použitých formátů souborů jsou k dispozici v příloze C.

Smooth contour

Vyhlazení vrstevnic. Vstup: vrstevnice (*Contours*). Výstup: vrstevnice (*Contours*). Nastavení: metoda pro vyhlazení a její případné parametry.

Umožňuje vyhladit vrstevnice jednou ze čtyř metod:

- Densening with interpolation
- McMaster 5-point average with slide
- Densening with unbound interpolation
- Plain densening

TIN loader

Načítá data ze souboru TIN a CTR. Vstup: žádný. Výstupy: body (*Points*), trojúhelníky (*Triangles*), hrany (Edges), povinné hrany (*Constraints*). Nastavení: jména souborů s daty.

Body, trojúhelníky a povinné hrany jsou načteny ze souboru TIN. Pokud je použit výstup hran, jsou tyto spočítány ze znalosti trojúhelníků.

Lze použít jména souborů s relativní cestou. Moduler při otevření projektu nastaví aktuální cestu na složku s projektem, takže všechny relativní cesty se vztahují k této složce.

UpDownEach

Implementuje metodu „Ke každému bodu nejbližší vyšší a nižší“ z podkapitoly Optimalizace na množině bodů. Vstup: body (*Points*). Výstup: povinné hrany (*Constraints*). Nastavení: žádné.

Ke každému body vygeneruje povinnou hranu k nejbližšímu nižšímu a nejbližšímu vyššímu bodu. K vyhledávání používá optimalizaci mřížkou, prochází sousední buňky, dokud nenajde vyšší i nižší bod, nebo zkontroluje alespoň 20 bodů, nebo narazí na konec mřížky.

12 Příloha B – programátorská dokumentace

Modularita výpočetního modelu aplikace Moduler přináší poměrně velké možnosti z uživatelského hlediska, z hlediska programátorského je však o něco náročnější. Kromě rozdělení výpočtu do jednotlivých dynamických knihoven je potřeba respektovat vzájemný interface s hlavním programem.

Jako prostředí pro vývoj aplikace včetně modulů bylo použito nástroje Borland Delphi 6, nicméně celý interface je navržen tak, aby bylo možné programovat další doplňky v téměř libovolném prostředí, jedinou podmínkou je možnost tvorby dynamických knihoven Win32 se zadanými exportovanými funkcemi. Jako proof-of-concept byl modul Joiners naprogramován v Microsoft Visual C++ 6.

Hlavní program při svém startu najde ve své startovací složce všechny soubory s příponou „ml“ a pokusí se v nich najít funkce `MLCreate` a `MLFree`. Pokud je nalezne, zařadí modul do seznamu aktivních knihoven a zavolá jeho funkci `MLCreate`.

12.1 Interface

Funkce `MLCreate` dostane jako parametr ukazatel na funkce rozhraní, tedy na strukturu `RInterface` obsahující ukazatele na funkce, které modul může volat pro komunikaci s hlavním programem. Tento ukazatel si modul musí uschovat pro pozdější použití.

Uvnitř funkce `MLCreate` by modul měl zaregistrovat všechny typy výpočetních modulů, jejichž implementaci nabízí. Výpočetní modul je definován jako seznam jmen a datových typů vstupů a výstupů a call-back funkce pro reakce na události (`RModInfo`).

Funkce `MLFree` je použita pro finální úklid při ukončování činnosti modulu. Většinou je však nevyužita, moduly se bez ní dokážou jednoduše obejít.

12.2 Call-back funkce

Call-back funkce jsou volány hlavním programem při každé manipulaci s instancemi. Nejdříve je pomocí `OnCreate` instance vytvořena, případně `OnLoad` načte konfiguraci ze souboru. V průběhu výpočtu je volána nejdříve funkce `OnSetInput` pro každý vstup, pak jednou funkce `OnRequest` a nakonec `OnCalc`. Je-li položka `OnSetup` nastavena (není rovna NULL), Moduler zobrazuje u daného typu modulu aktivní tlačítko Setup, po jehož stisknutí se vyvolá přes call-back `OnSetup` dialogové okno s nastavením modulu.

12.3 Výpočet

Struktura výpočtu je reprezentována jako orientovaný graf. Aby výpočet mohl proběhnout, musí tento graf být acyklický. Pak tedy musí existovat uzel grafu (instance

výpočetního modulu), jehož všechny vstupy jsou nepřiřazeny. Na tento modul spustíme výpočet a ve struktuře jej „odtrhneme“ z grafu. Protože výsledný pod-graf je opět orientovaný acyklický, opět existuje uzel, jehož všechny vstupy jsou již spočítány. Můžeme tedy iterovat, dokud nebudou spočítány všechny moduly.

Výpočet probíhá na základě dostupnosti dat. Cyklicky jsou testovány všechny dosud nespočítané instance, program zkontroluje, zda všechny jejich vstupy jsou již k dispozici, a pokud ano, zavolá na instanci výpočet. Zjišťování výsledků výpočtu probíhá až v okamžiku, kdy jsou potřeba jako vstup jinému modulu.

12.4 Datové typy

Data jsou předávána jako zapouzdřené datové typy. Každý typ je označen při registraci výpočetního modulu. Aplikace automaticky hlídá správnost přiřazení datových typů mezi vstupy a výstupy.

Jsou podporovány tyto datové typy: Body (`RMPoints`), Hrany (`RMEdges`), Trojúhelníky (`RMTriangles`), Sousedé trojúhelníků (`RMTriangleNeighbors`), Indicie (`RMIndices`) a Vrstevnice (`RMContours`). Výpočetní modul nesmí uvolňovat paměť použitou pro vstupní data; naopak mechanismus alokace výstupních dat si řídí sám pomocí kombinace `OnGetOutput` a `OnFreeOutput`.

12.5 Log

Kterýkoliv modul smí psát informace do logu, což je spodní část okna aplikace, pomocí funkce `logWrite`. Pro dodržení indentace jsou k dispozici funkce `logPush` (přidá úroveň) a `logPop` (ubere úroveň).

Standardním chováním modulů je při zahájení výpočtu vypsat hlášku o startu výpočtu a přidání indentace. Při ukončení výpočtu se indentace zase ubírá. To umožňuje uživateli mít v případě výskytu chyby přehled o tom, kde chyba nastala.

Je-li aplikace psaná v Delphi 6, lze využít i přiložený soubor jclDebug a při startu aplikace registrovat exception handler `RInterface.dbgExceptionNotify`. Tím se umožní přesnější sledování výjimek, včetně podrobného výpisu zásobníku se jmény souborů a čísly řádků.

13 Příloha C – formát souboru TIN a CTR

Formát dat předchozích prací byl označen příponou DAT, která ovšem v dnešním prostředí působí značné problémy svou všeobecností. Proto byla použita nová přípona TIN, která lépe vystihuje obsah souboru. Pro účely kompatibility program Moduler dokáže otevírat i soubory s příponou DAT.

Soubor TIN je textově orientovaný soubor obsahující povinně seznam bodů v trojrozměrném prostoru a volitelně seznam trojúhelníků, jejich sousedů a povinných hran. Všechna čísla jsou uváděna decimálně, používá se desetinná tečka, jako konec řádku je použit dvojznak CRLF.

Na prvním řádku je počet bodů (celé číslo), označme jej B

Dalších B řádků obsahuje souřadnice bodů, každý řádek obsahuje právě tři souřadnice – právě jeden bod. Souřadnice jsou odděleny libovolným bílým znakem.

Následuje řádek s počtem trojúhelníků (celé číslo) T. Pokud soubor neobsahuje trojúhelníky, T = 0

Dalších T řádků definuje indexy bodů trojúhelníků, na každém řádku po třech celých číslech pro právě jeden trojúhelník. Původní software předpokládal pravotočivou orientaci trojúhelníků, Moduler již toto omezení nevyžaduje.

Na následujícím řádku je zopakován počet trojúhelníků v případě, že soubor obsahuje sousedy; pokud sousedy neobsahuje, je na tomto řádku nula. Toto celé číslo označme S

Dalších S řádků obsahuje definice sousedů, což jsou indexy do předchozího pole trojúhelníků. Moduler tyto informace ignoruje, podstatný je pouze počet řádků.

Následuje počet povinných hran, celé číslo C.

Dalších C řádků definuje povinné hrany jako dvojici indexů do pole prvků, každá dvojice na zvláštním řádku.

Formát CTR byl přidán jako formát specifický pro Moduler, jsou v něm uloženy vrstevnice. Na prvním řádku je celkový počet vrstevnic, pak následují jednotlivé vrstevnice. Na prvním řádku vrstevnice je počet bodů (int) a výška (float), následují souřadnice X a Y (float) jednotlivých bodů, každý bod na jednom řádku.

Ukázkový TIN soubor:

5	*počet vrcholů*
0.42868 0.20106 -0.86000	*definice XYZ souřadnic vrcholů*
0.91568 0.38039 0.04672	
0.55284 0.86402 0.42103	
0.93631 0.48820 1.67458	
0.49317 0.46177 1.43138	
4	*počet trojúhelníků*
0 1 4	*definice trojúhelníků*
1 3 4	
0 4 2	
4 3 2	
0	*počet sousedů (nepoužit)*
1	*počet povinných hran*
3 4	*definice povinné hrany*

14 Příloha D – zadání práce

1. Prostudujte dostupnou literaturu o reprezentaci terénu pomocí trojúhelníkových sítí, o automatizovaném výpočtu vrstevnic na trojúhelníkové síti a o možnostech jejich vyhlazování, seznamte se s výsledky a programovým vybavením vytvořeným v rámci předchozích DP.
2. Prověřte vlastnosti Delaunayovy, žravé, příp. dalších triangulací pro výpočet vrstevnic a základní kategorie chyb vznikajících při automatickém výpočtu vrstevnic. V případě potřeby navrhněte vlastní kategorizaci.
3. Navrhněte kritéria a metody pro plně či částečně automatickou eliminaci chyb.
4. Navrhněte vhodné metody úpravy vyhlazení vrstevnic z předchozí práce.
5. Implementujte navržené metody v prostředí MS Windows tak, aby vhodně doplnily stávající programové vybavení, a experimentálně ověřte jejich funkčnost včetně testování na reálných datech. K programovému vybavení vytvořte též dokumentaci.