ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA STAVEBNÍ, OBOR GEODÉZIE A KARTOGRAFIE
KATEDRA GEOMATIKY

název předmětu

ALGORITMY V DIGITÁLNÍ KARTOGRAFII A GIS

číslo úlohy	název úlohy
3	Digitální model terénu

školní rok	studijní skup.	číslo zadání	zpracoval	datum	klasifikace
2019/2020	60		Tomáš Bouček, Jan Šikola	6.2. 2019	

Obsah

1. Zadání

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

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

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

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

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

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

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

Zhodnocení činnosti algoritmu včetně ukázek proveďte alespoň na **3 strany** formátu A4.

2. Údaje o bonusových úlohách

V rámci bonusových úloh byla zpracována *triangulace nekonvexní oblasti zadané polygonem, automatický popis vrstevnic* a *algoritmus pro automatické generování terénních tvarů (kupa, údolí, spočinek, hřbet).*

3. Popis a rozbor problémů

Cílem úlohy bylo vytvořit aplikaci pomocí QT Creatoru, která by nad množinou bodů vytvořila digitální model terénu pomocí Delaunay triangulace. Body je možno zadávat buď kliknutím myši do prostoru aplikace nebo je náhodně vygenerovat pomocí vytvořeného generátoru terénních tvarů. Uživatel může *DMT* vizualizovat pomocí vrstevnic, sklonu či orientace terénu.

4. Popis algoritmů

4.1 Delaunay triangulace

K vytvoření Delaunay triangulace byla použita metoda inkrementální konstrukce, která je založena na postupném přidávání bodů do již vytvořené množiny *DT*. Nad existující Delaunayovskou hranou *e* je hledán bod *p* minimalizující poloměr $k_i = (e, p_i)$. Delaunayovská hrana je orientovaná, a proto bod *p* hledáme pouze nalevo od ní. Pokud se nalevo od hrany žádný takový bod nevyskytuje, dojde k prohození orientace hrany.

Do množiny *DT* jsou vždy přidávány tři hrany tvořící trojúhelník. Při konstrukci je dále využívána množina *AEL* (*Active Edge List*), která obsahuje hrany, ke kterým hledáme body *p*. Algoritmus běží, doku není množina *AEL* prázdná. Hrany se z *AEL* odstraňují v momentu, kdy se k příslušné hraně najde bod *p*.

Nalezení bodu *p* probíhá následujícím způsobem. Nejprve se určí poloha bodu od dané hrany.

$$\vec{u} = (x_2 - x_1, y_2 - y_1)$$
$$\vec{v} = (x_p - x_1, y_p - y_1)$$

$$t = \begin{vmatrix} u_x & u_y \\ v_x & v_y \end{vmatrix} \ .$$

Pokud t < 0, pak se bod *p* nachází v levé polorovině. Určení poloměru se provede následovně:

$$m = 0{,}5 \cdot \frac{k_{12} \cdot (-k_4) + k_{11} \cdot k_5 - (k_{10} + k_4 \cdot k_5) \cdot k_6}{x_3 \cdot (-k_4) + x_2 \cdot k_5 + x_1(-k_6)}$$

$$n = 0{,}5 \cdot \frac{k_1 \cdot (-k_9) + k_2 \cdot k_8 + k_3 \cdot (-k_7)}{y_1 \cdot (-k_9) + y_2 \cdot k_8 + y_3 \cdot (-k_7)}$$

$$r = \sqrt{(x_1 - m)^2 + (y_1 - n)^2} \ ,$$

kde

$k_1 = x_1^2 + y_1^2$ $\quad k_2 = x_2^2 + y_2^2$ $\quad k_3 = x_3^2 + y_3^2$ $\quad k_4 = y_1 - y_2$

$k_5 = y_1 - y_3$ $\quad k_6 = y_2 - y_3$ $\quad k_7 = x_1 - x_2$ $\quad k_8 = x_1 - x_3$

$k_9 = x_2 - x_3$ $\quad k_{10} = x_1^2$ $\quad k_{11} = x_2^2$ $\quad k_{12} = x_3^2$

Samotný algoritmus Delaunay triangulace má následující podobu:

1) Nalezení pivota p_1, $p_1 = min(x)$, nalezení bodu p_2 nejbližšího k pivotu
2) Vytvoření hrany $e = (p_1, p_2)$
3) Nalezení optimálního Delaunay bodu *p*
4) Pokud *p* nenalezen, prohoď orientaci *e*, tj. $e = (p_2, p_1)$, znovu krok 3
5) Vytvoř zbývající hrany trojúhelníku, tj. $e_2 = (p_2, p)$ a $e_3 = (p, p_1)$
6) AEL ← *e*, AEL ← e_2, AEL ← e_3
7) DT ← *e*, DT ← e_2, DT ← e_3

dokud AEL není prázdná:
    AEL → e, $e = (p_1, p_2)$ // vezmi první hranu z AEL
    Prohození orientace e, tj. $e = (p_2, p_1)$
    Nalezení optimálního Delaunay bodu p
    Pokud p existuje
        $e_2 = (p_2, p)$, $e_3 = (p, p_1)$
        DT ← e, DT ← e_2, DT ← e_3
        Prohození orientace e_2 a e_3, *tj.* $e_2' = (p, p_2)$, $e_3' = (p_1, p)$
        Pokud e_2' je v AEL
            AEL → e_2'
        Jinak AEL ← e_2
        Pokud e_3' je v AEL
            AEL → e_3'
        Jinak AEL ← e_3

4.2 Tvorba vrstevnic

Vrstevnice byly tvořeny pomocí lineární interpolace. Ta je založena na analytické geometrii, kde je úkolem najít průsečnici roviny tvořenou trojúhelníkem a vodorovné roviny s výškou *h*. Koncové body *A, B* této průsečnice se určí z podobnosti trojúhelníků.

$$x_A = \frac{x_3 - x_1}{z_3 - z_1} \cdot (z - z_1) + x_1$$

$$y_A = \frac{y_3 - y_1}{z_3 - z_1} \cdot (z - z_1) + y_1$$

$$x_B = \frac{x_2 - x_1}{z_2 - z_1} \cdot (z - z_1) + x_1$$

$$y_B = \frac{y_2 - y_1}{z_2 - z_1} \cdot (z - z_1) + y_1$$

Spojením těchto bodů vznikne hrana tvořící v daném trojúhelníku vrstevnici o dané výšce. Vrstevnice byly kresleny pro následující případy:

- Strana trojúhelníku leží ve vodorovné rovině
- Vodorovná rovina protíná trojúhelník ve vrcholu a protilehlé straně
- Vodorovná rovina protíná trojúhelník ve dvou stranách

4.3 Sklon terénu

Sklonem terénu se rozumí odchylka normálového vektoru roviny trojúhelníku od normálového vektoru vodorovné roviny.

$$n_x = u_y \cdot v_z - v_y \cdot u_z$$

$$n_y = -(u_x \cdot v_z - v_x \cdot u_z)$$

$$n_z = u_x \cdot v_y - v_x \cdot u_y$$

$$n_t = \sqrt{n_x^2 + n_y^2 + n_z^2}$$

$$\varphi = acos\left(\frac{n_z}{n_t}\right) \ .$$

4.4 Orientace terénu

Orientací terénu se rozumí natočení terénu vůči světovým stranám.

$$n_x = u_y \cdot v_z - v_y \cdot u_z$$

$$n_y = -(u_x \cdot v_z - v_x \cdot u_z)$$

$$\alpha = atan\left(\frac{n_x}{n_y}\right) \ .$$

Při počítání orientace terénu byla v použita funkce *atan2*. Následné vykreslení je provedeno ve stupních šedi pro získání lepšího dojmu z plastičnosti. Stupně šedi jsou generovány tak, jako by Slunce svítilo od severu.

5. Řešení bonusových úloh

5.1 Triangulace nekonvexní oblasti

Nekonvexní oblast je do aplikace zadávána ručně uživatelem a reprezentována znázorněným polygonem. Následně se pomocí Winding Number Alorithm naleznou všechny body ležící v tomto polygonu. Vytvoří se triangulace a pomocí funkce na zjišťování existence průsečíku dvou úseček se vyloučí takové trojúhelníky, které se alespoň jednou svou stranou protínají se stranou zadaného polygonu.

Test existence průsečíku dvou úseček:

$$\vec{u} = (x_2 - x_1, y_2 - y_1) \qquad \vec{v} = (x_4 - x_3, y_4 - y_3) \qquad \vec{w} = (x_1 - x_3, y_1 - y_3)$$

$$k_1 = v_x \cdot w_y - v_y \cdot w_x \qquad k_2 = u_x \cdot w_y - u_y \cdot w_x \qquad k_3 = v_y \cdot u_x - v_x \cdot u_y$$

$$\alpha = \frac{k_1}{k_3} \qquad \beta = \frac{k_2}{k_3}$$

Pokud $\alpha \in \langle 0,1 \rangle$ a $\beta \in \langle 0,1 \rangle$, tak průsečík daných dvou úseček existuje.

5.2 Automatický popis vrstevnic

Vrstevnice jsou tvořené hranami, ty jsou tvořené dvěma body o souřadnicích *X*, *Y*, a *Z*. Odtud se získá požadovaná hodnota, která má být znázorněna a umístí se na střed dané vrstevnice. Toho se docílí zprůměrováním souřadnic X, *Y* počátečního a koncového bodu. Pomocí funkce *drawText* se popisky vykreslí.

5.3 Algoritmy pro generování terénních tvarů

Náhodné body

Souřadnice *X*, *Y* jsou generovány pomocí funkce *QRandomGenerator*, a jejich rozsah je dán velikostí okna. Souřadnici *Z* je následně přidělena náhodná hodnota pomocí funkce *rand*.

GenerateRandomPoints

1) Deklarace vektoru bodů
2) for (int i = 0; i < n; i++)   //*n* je počet udávající, kolik bodů se má vygenerovat
3) Vygenerování souřadnic x, y, z
4) Přidělení souřadnic bodu
5) Přidání bodu do vektoru

Kupa

Generování kupy vychází z generování náhodných bodů, kde se převezmou souřadnice *X* a *Y*. Následně je spočítán bod umístěný v těžišti všech bodů a přidělí se mu výška *Z*. Náhodně vygenerovaným bodům se poté přidělí náhodně výšky tak, že čím dál jsou od středu, tím je jejich výška menší.

Výpočet těžiště:

$$x_T = \frac{\sum_{i=1}^{n} x_i}{n} \quad ,$$

$$y_T = \frac{\sum_{i=1}^{n} y_i}{n} \quad ,$$

kde

n je celkový počet bodů.

*Generate*Kupa

1) Vygenerování náhodných bodů
2) Výpočet těžiště
3) Přidělení výšky těžišti bodů
4) Přidání těžiště do vektoru bodů
5) Pro každý bod ve vektoru bodů:
6) Určení vzdálenosti bodu od těžiště
7) Přiřazení výšky bodu nepřímo úměrné jeho vzdálenosti od těžiště   //“čím dál, tím míň“

Hřbet

Znovu se vychází z náhodných bodů. Generátor náhodně určí, zda bude hřbet ve směru osy X nebo osy Y a podle toho setřídí body. Prvnímu a poslednímu bodu v množině se následně přidělí výšky. Dále se spočítá bod v těžišti bodů a přidělí se mu výška prvního bodu. U zbylých náhodně vygenerovaných bodů se zjistí jejich poloha vůči počátečnímu a koncovému bodu hřbetu (určí se jejich vzdálenost) a podle toho, k jakému bodu jsou blíže, se jim přidělí náhodná výška (menší, než je výška počátečního bodu hřbetu).

generateRidge

1) Vygenerování náhodných bodů
2) Náhodné určení orientace hřbetu (horizontálně či vertikálně vůči Canvasu)
3) Pokud horizontálně
4)   Setřídění bodů podle X
5) jinak
6)   Setřídění bodů podle Y
7) Přidělení výšky prvnímu a poslednímu bodu ze setříděných bodů
8) Výpočet těžiště bodů + přidělení výšky
9) Přidání těžiště do vektoru bodů
10) Pro všechny body: //i = 1, i < velikost vektoru bodů - 1
11)   d1   //vzdálenost bodu od prvního bodu
12)   d2   //vzdálenost bodu od předposledního bodu
13)   pokud d1 < d2
14)   přidělení výšky nepřímo úměrné vzdálenosti bodu od prvního bodu
15)   jinak
16)   přidělení výšky nepřímo úměrné vzdálenosti bodu od posledního bodu

Údolí

Údolí se tvoří stejně jako hřbet, jen s tím rozdílem, že prvnímu a poslednímu bodu setříděné množiny bodů je přiřazena nízká výška a zbylým bodům se úměrně na vzdálenosti přiřazuje náhodná vyšší výška.

generateValley

1) Vygenerování náhodných bodů
2) Náhodné určení orientace údolí (horizontálně či vertikálně vůči Canvasu)
3) Pokud horizontálně
4)   Setřídění bodů podle X
5) jinak
6)   Setřídění bodů podle Y
7) Přidělení výšky prvnímu a poslednímu bodu ze setříděných bodů
8) Výpočet těžiště bodů + přidělení výšky
9) Přidání těžiště do vektoru bodů
10) Pro všechny body: //i = 1, i < velikost vektoru bodů - 1
11)   d1   //vzdálenost bodu od prvního bodu
12)   d2   //vzdálenost bodu od předposledního bodu
13)   pokud d1 < d2
14)   přidělení výšky přímo úměrné vzdálenosti bodu od prvního bodu

15) jinak
16) přidělení výšky přímo úměrné vzdálenosti bodu od posledního bodu

Spočinek

Byla nastavena primární výška dvou okrajovým bodům. Zbytku datasetu byla udělena výška úměrně podle vzdálenosti od dvou okrajových bodů. Dále byla zjištěna výška bodu v polovině datasetu a tato výška byla přidělena následujícím *n/4* bodům datasetu.

generateSettling

1) Vygenerování náhodných bodů
2) Náhodné určení orientace spočinku vůči Canvasu
3) Pokud horizontálně
4) Setřídění bodů podle X
5) jinak
6) Setřídění bodů podle Y
7) Nastavení výšky spočinku, tj. přidělení výšky prvnímu a druhému bodu
8) Pro všechny body: //i = 2
9) Vypočtení vzdálenosti bodu od prvního bodu
10) Přidělení výšky bodu nepřímo úměrně vzdálenosti bodu od spojnice prvních dvou bodů
11) Pro čtvrtinu bodů z druhé poloviny datasetu:
12) points[i].z() = points[points.size()/2].z()
13) Pro zbývající body:
14) points[i].z() = points[i].z() + dz // dz = z1 – z2

// z1 = points[points.size()/2].getZ();
// z2 = points[points.size()/2 + n/4].getZ();

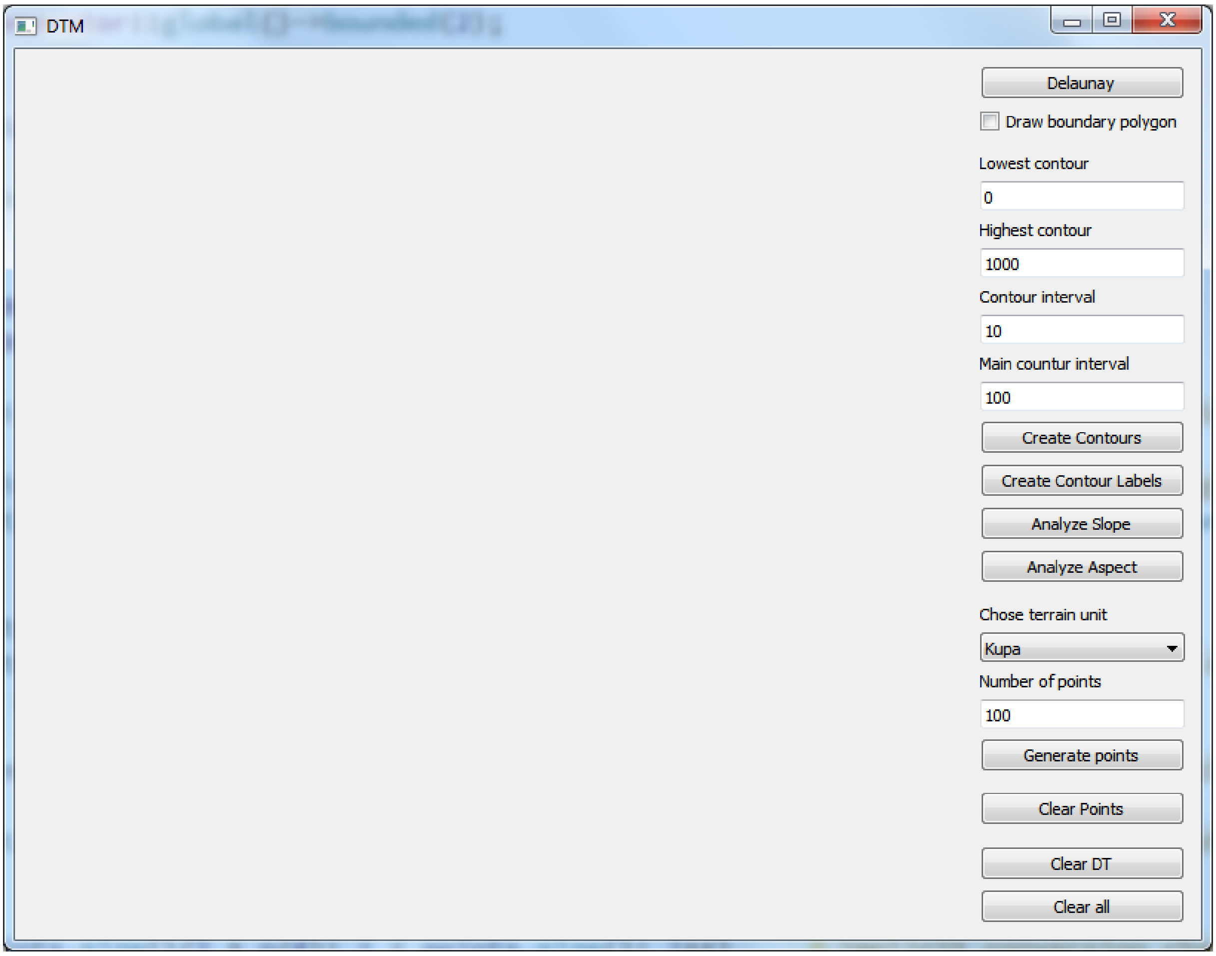
6. Vstupní data

Vstupními daty mohou být považovány body kliknuté uživatelem do plochy aplikace, případně body vygenerovány generátorem terénních tvarů. Druhou možností je vložení bodů ze souboru v textovém formátu, kde body jsou odděleny řádky. Každý řádek obsahuje tři hodnoty, kde souřadnice jsou seřazeny *x y z*.

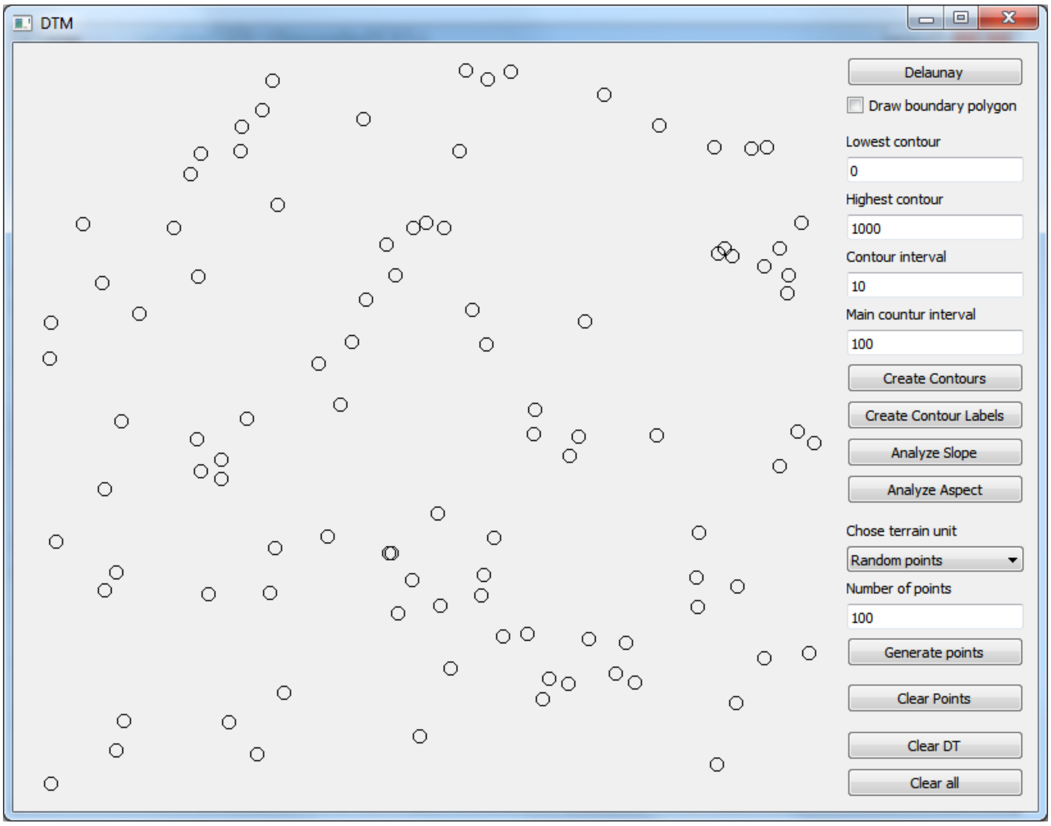
7. Výstupní data

Za výstupní data můžeme považovat grafické znázornění vrstevnic a jejich popisky, dále pak vizualizace sklonu a orientace terénu, případně i samotnou Delaunay triangulaci.

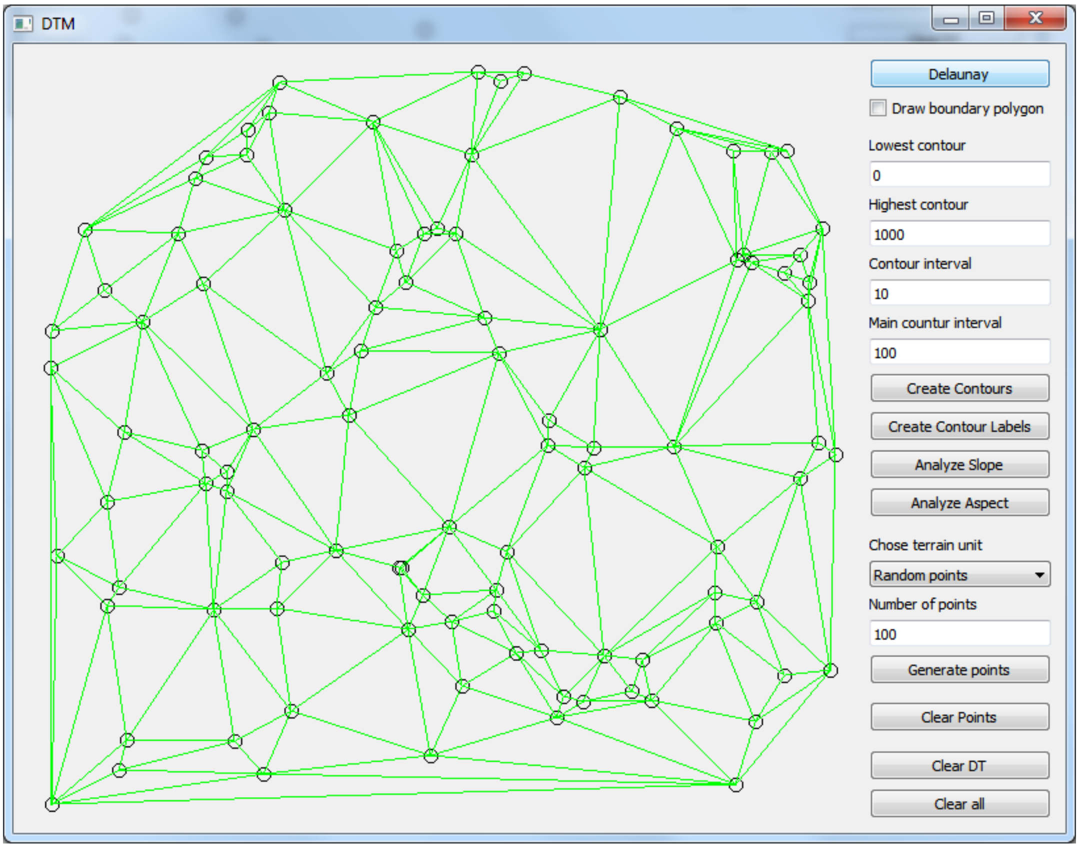
8. Obrázky vytvořené aplikace



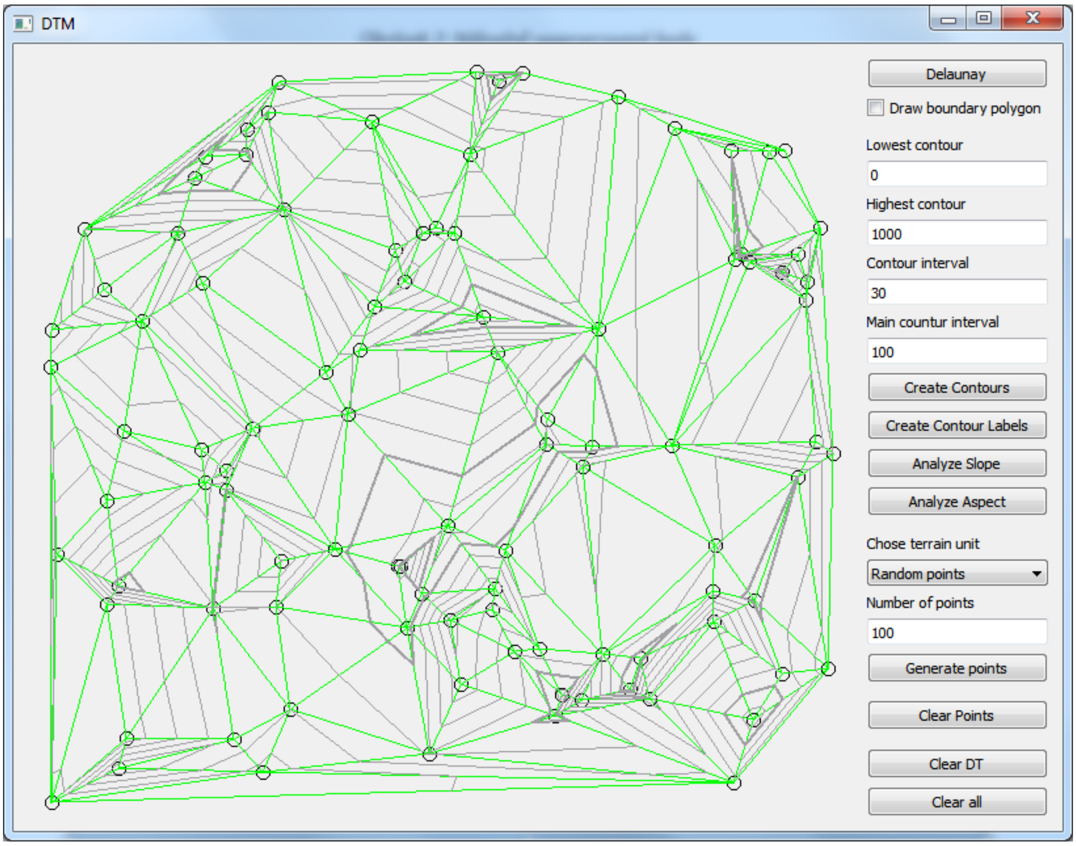
Obrázek 1: Aplikace po spuštění



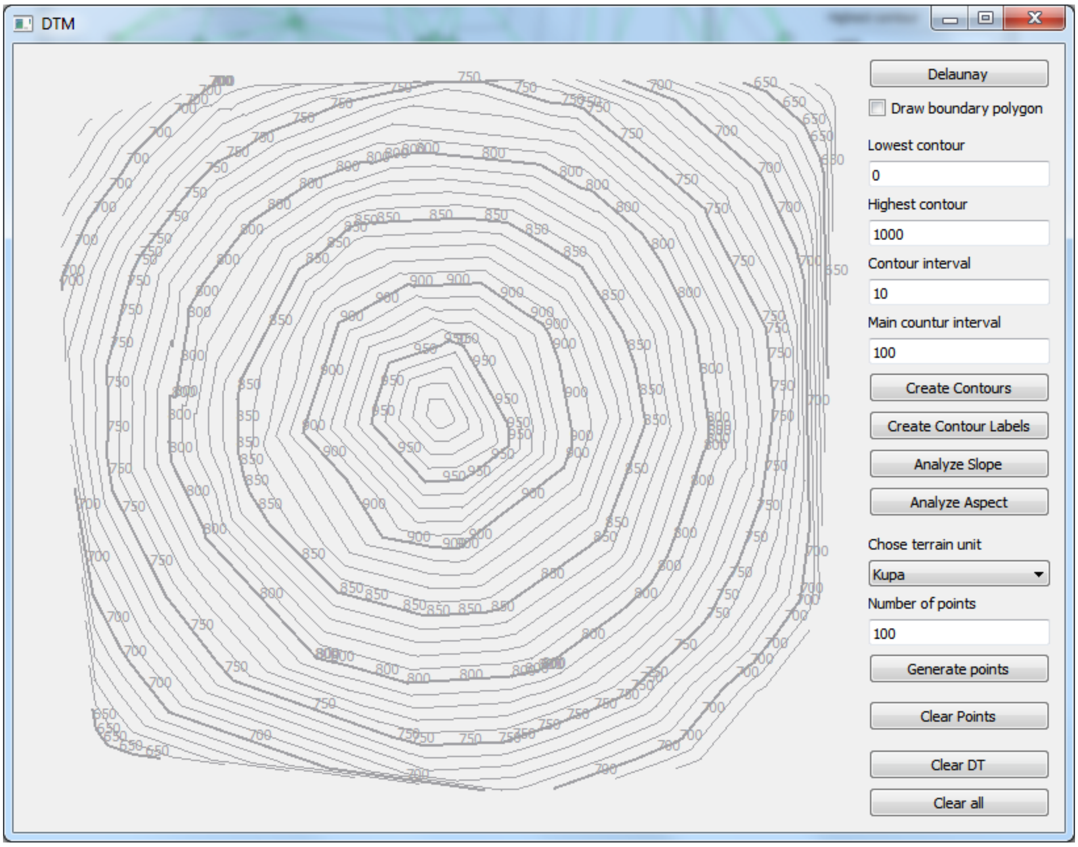
Obrázek 2: Náhodně vygenerované body



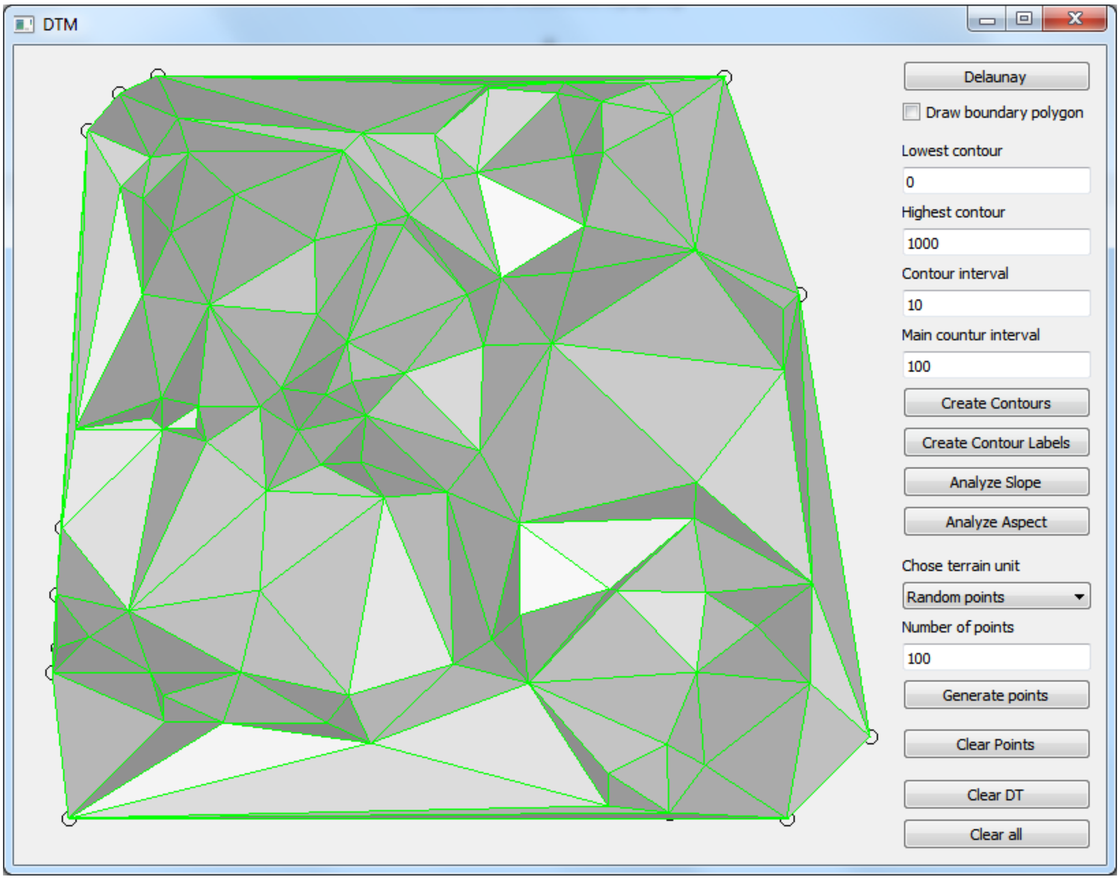
Obrázek 3: Vytvořená triangulace po stisknutí tlačítka Delaunay



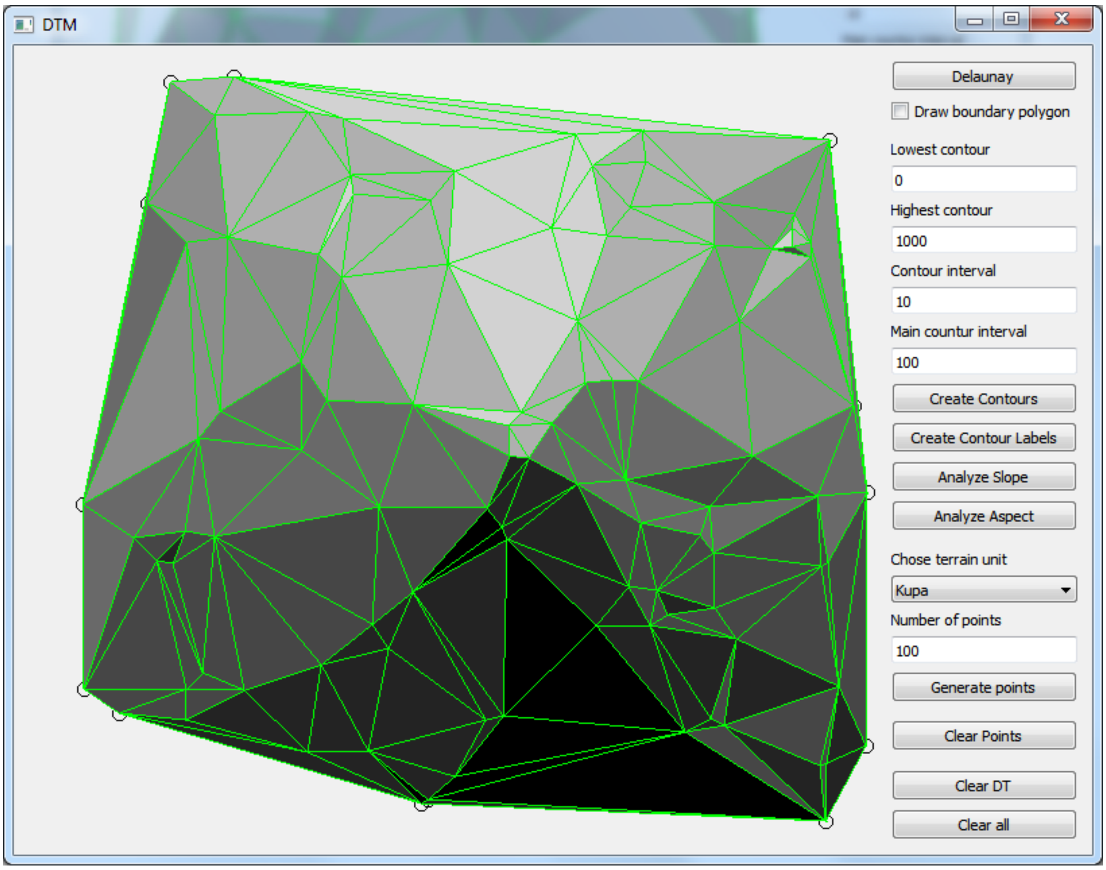
Obrázek 4: Vygenerované vrstevnice



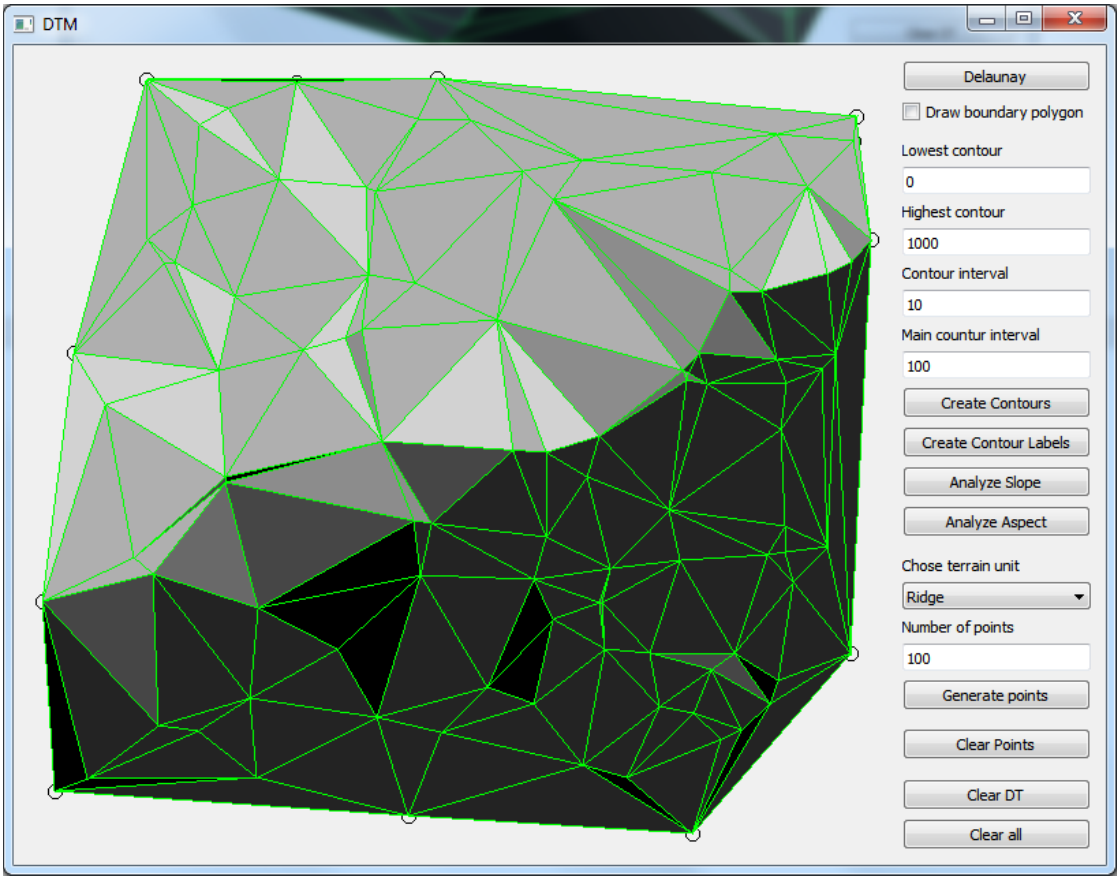
Obrázek 5: Vrstevnice s popisky hlavních vrstevnic na kupě



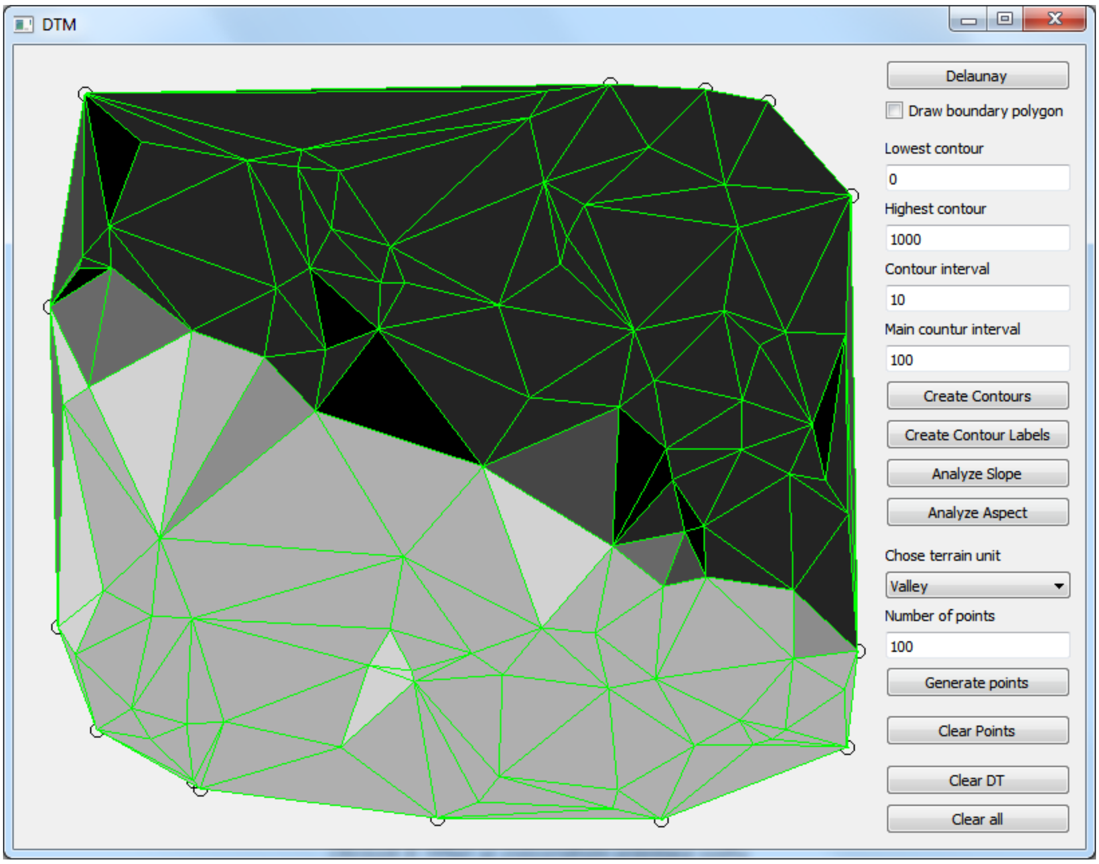
Obrázek 6: Vizualizace sklonu terénu



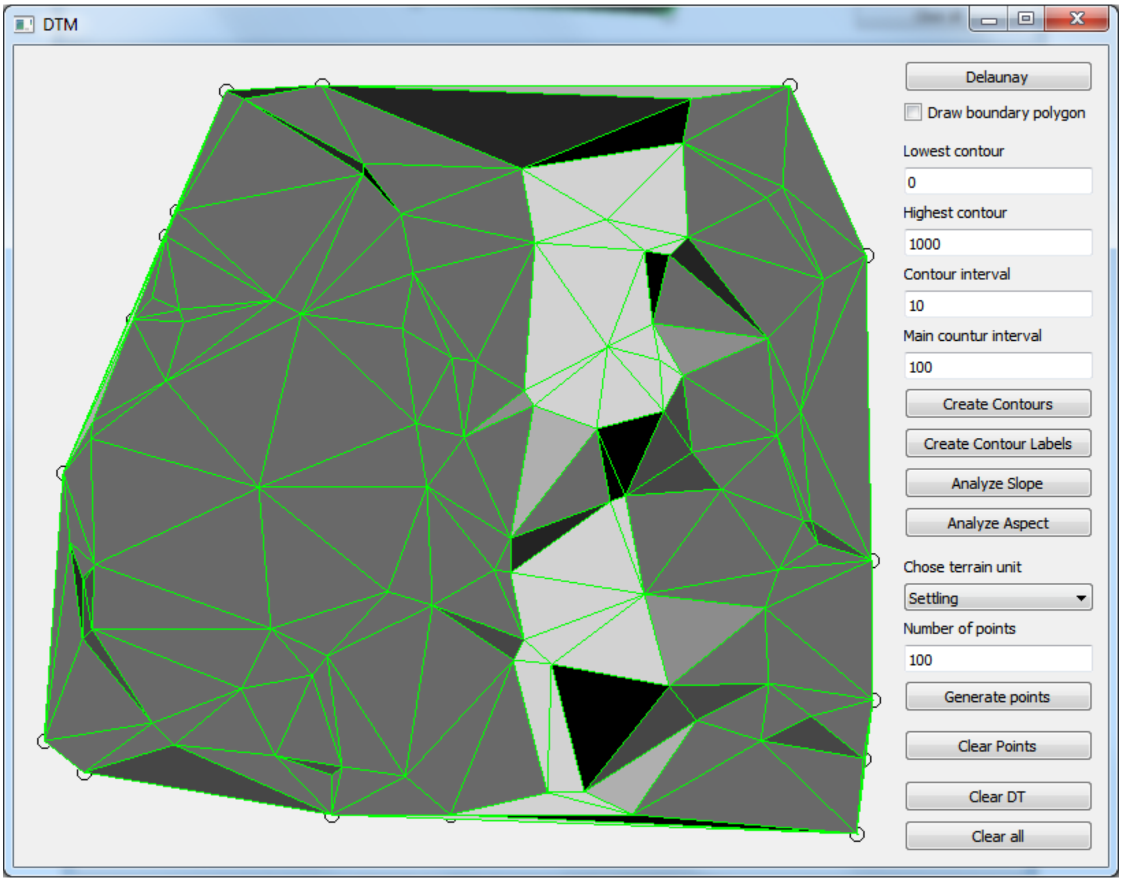
Obrázek 7: Kupa a její orientace svahu



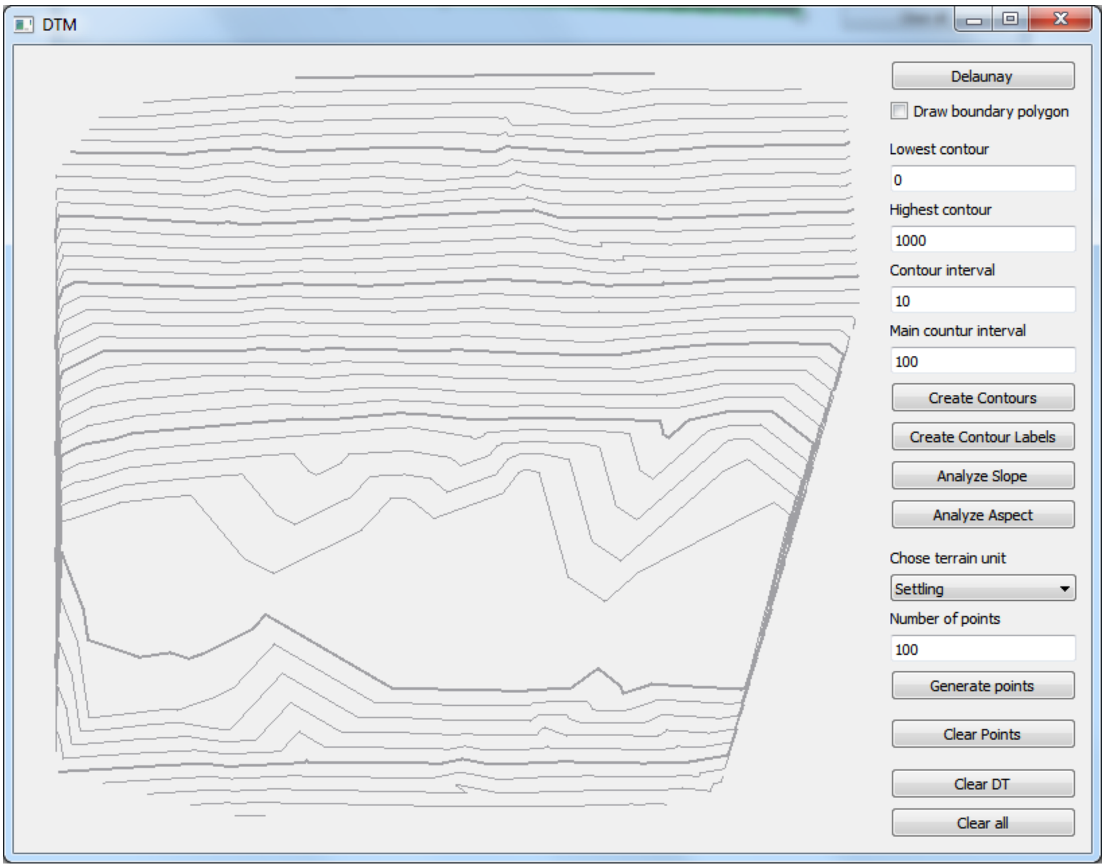
Obrázek 8: Hřbet se znázorněním orientace svahu



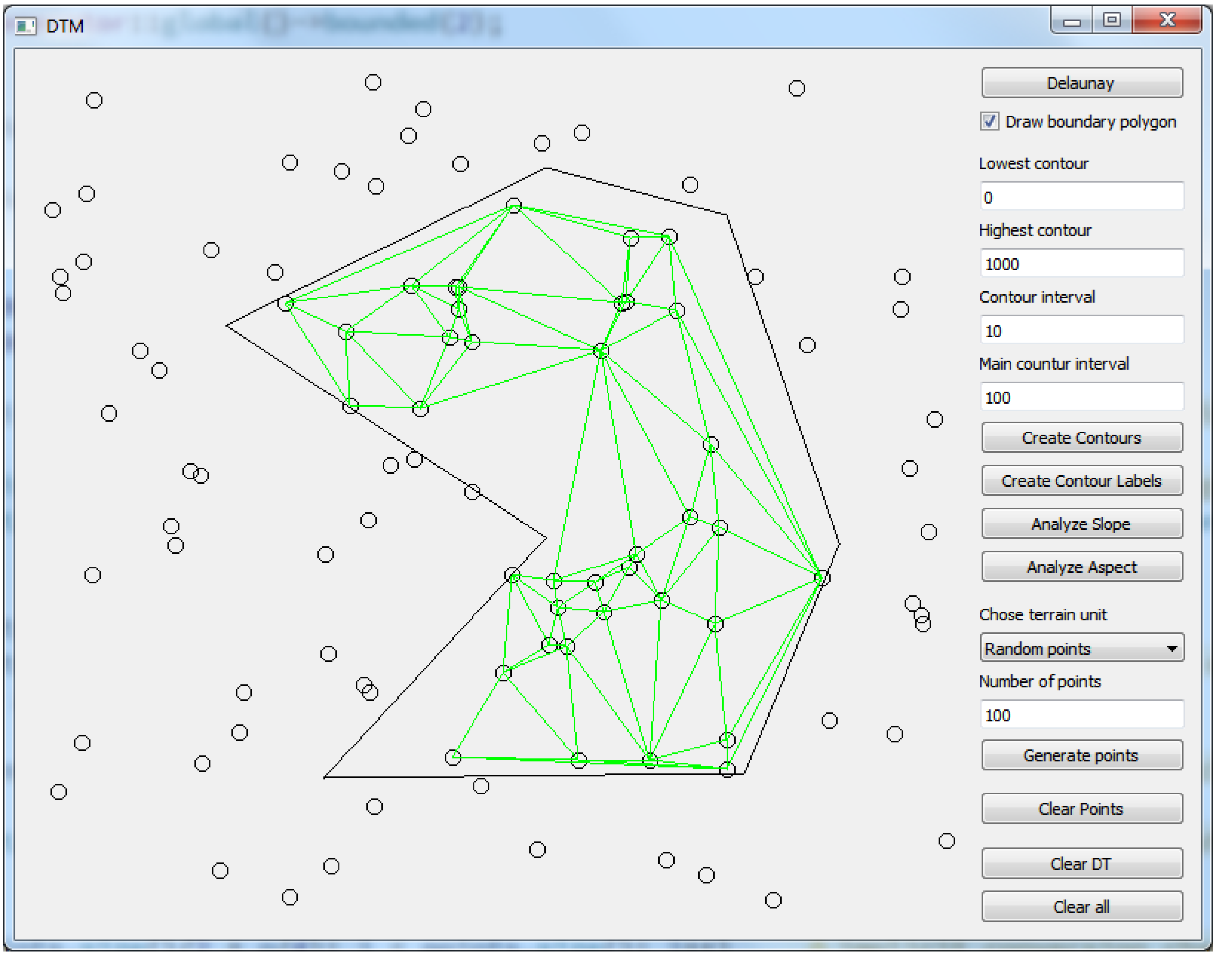
Obrázek 9: Údolí se znázorněním orientace svahu



Obrázek 10: Spočinek se znázorněním orientace svahu



Obrázek 11: Spočinek s vrstevnicemi (jiný než předchozí případ)



Obrázek 12: Triangulace nekonvexní oblasti

9. Dokumentace

Třída Algorithms:

```
static int getPointLinePosition(QPoint3D &q,QPoint3D &p1,QPoint3D &p2);
```

- funkce na určení polohy bodu vůči přímce, tj. jestli se nachází v levé nebo v pravé polorovině, případně na přímce

```
static double getCircleRadius(QPoint3D &p1, QPoint3D &p2, QPoint3D &p3,
QPoint3D &c);
```

- funkce na získání středu kružnice

```
static int getNearestpoint(QPoint3D &p, std::vector<QPoint3D> &points);
```

- funkce pro nalezení nejbližšího bodu

```
static double distance2Points(QPoint3D &p1, QPoint3D &p2);
```

- funkce na vzdálenost mezi dvěma body

```
static double getPointLineDistance(QPoint3D &q, QPoint3D &p1, QPoint3D &p2);
```

- funkce na výpočet vzdálenosti bodu od přímky

```
    static int intersectOfTwoLinesExists(QPoint3D &p1, QPoint3D &p2, QPoint3D &p3,
QPoint3D &p4);
```

- funkce na zjištění existence průsečíku dvou úseček

```
    static int positionPointPolygonWinding(QPoint3D &q, std::vector<QPoint3D>
&pol);
```

- funkce zjišťující polohu bodu vůči polygonu pomocí Winding Number Algorithm

```
    static double getAngle2Vectors(QPoint3D &p1,QPoint3D &p2,QPoint3D &p3,QPoint3D
&p4);
```

- funkce na výpočet úhlu mezi dvěma vektory

```
    static int getDelaunayPoint(QPoint3D &s, QPoint3D &e, std::vector<QPoint3D>
&points);
```

- funkce na určení Delaunayského bodu vhodného pro Delaunay triangulaci

```
    static std::vector<Edge> DT(std::vector<QPoint3D> &points);
```

- funkce generující Delaunay triangulaci. Vstupem je množina bodů.

```
    static QPoint3D getContourPoint(QPoint3D &p1, QPoint3D &p2, double z);
```

- funkce při určení bodu ležícího na vrstevnici potřebná pro následnou tvorbu vrstevnic

```
    static std::vector<Edge> createContourLines(std::vector<Edge> &dt, double
z_min, double z_max, double dz);
```

- funkce generující vrstevnice

```
    static double calculateSlope(QPoint3D &p1, QPoint3D &p2, QPoint3D &p3);
```

- funkce pro výpočet sklonu terénu

```
    static double calculateAspect(QPoint3D &p1, QPoint3D &p2, QPoint3D &p3);
```

- funkce pro výpočet orientace svahu

```
    static std::vector<Triangle> analyzeDTM(std::vector<Edge> &dt);
```

- funkce analyzující terén podle sklonu a orientace

```
    static std::vector<QPoint3D> generateKupa(int n, int height, int width);
```

- funkce na generování kupy

```
    static std::vector<QPoint3D> generateRidge(int n, int height, int width);
```

- funkce na generování hřbetu

```
    static std::vector<QPoint3D> generateValley(int n, int height, int width);
```

- funkce na generování údolí

```
    static std::vector<QPoint3D> generateSettling(int n, int height, int width);
```

- funkce na generování spočinku

```
static std::vector<QPoint3D> generateRandomPoints(int n, int height, int width);
```

- funkce na generování náhodných bodů

Třída Draw:

```
private:
    std::vector<QPoint3D> points;
```

- deklarace proměnné pro vygenerování bodů a práce s nimi

```
    std::vector<Edge> dt;
```

- deklarace proměnné pro delaunay triangulaci

```
    std::vector<Edge> contours, label_contours, main_contours;
```

- deklarace proměnných pro vrstevnice, popisky vrstevnic a hlavní vrstevnice

```
    std::vector<Triangle> dtm, aspect_dtm;
```

- deklarace proměnných pro digitální model terénu a pro digitální model terénu určený pro práci s orientací svahu

```
    int ch;
```

- deklarace proměnné pro ukládání stavu checkBoxu

```
    std::vector<QPoint3D> polygon;
```

- deklarace proměnné pro práci s nakresleným polygonem

```
public:
    void setPoints(std::vector<QPoint3D> &points_){points = points_;}
```

- funkce pro nastavení bodů

```
    std::vector<Edge>& getDt() {return dt;}
```

- funkce pro získání Delaunay triangulace

```
    void setDt(std::vector<Edge> &dt_){dt=dt_;}
```

- funkce pro nastavení Delaunay triangulace

```
    void setContours(std::vector<Edge> &contours_){contours = contours_;}
```

- funkce pro nastavení vrstevnic

```
    void setMainContours(std::vector<Edge> &main_contours_){main_contours = 
main_contours_;}
```

- funkce pro nastavení hlavních vrstevnic

```
    void setLabelContours(std::vector<Edge> &label_contours_){label_contours = 
label_contours_;}
```

- funke pro nastavení popisků vrstevnic

```
    void setCheck(int ch_){ch = ch_;}
```

- funkce pro nastavení stavu checkBoxu

```
    int getCheck(){return ch;}
```

- funkce pro získání stavu checkBoxu

```
std::vector<QPoint3D> getBoundaryPolygon(){return polygon;}
```

- funkce pro získání ohraničujícího polygonu

```
void setBoundaryPolygon(std::vector<QPoint3D> &polygon_){polygon = polygon_;}
```

- funkce pro nastavení ohraničujícího polygonu

```
std::vector<Edge>& getMainContours() {return main_contours;}
```

- funkce pro získání hlavních vrstevnic

```
void paintEvent(QPaintEvent *event);
```

- funkce pro vykreslování grafických výstupů aplikace (body, vrstevnice, sklon, orientace, atd.)

```
void mousePressEvent(QMouseEvent *event);
```

- funkce pro odečet souřadnic sejmutého bodu

```
std::vector<QPoint3D> getPoints(){return points;}
```

- funkce pro získání bodů

```
void clearPoints(){points.clear(); }
```

- funkce pro vymazání bodů

```
void clearDT(){dt.clear();}
```

- funkce pro vymazání Delaunay triangulace

```
void clearSlope(){dtm.clear();}
```

- funkce pro vymazání sklonu terénu

```
void clearAspect(){aspect_dtm.clear();}
```

- funkce pro vymazání orientace terénu

```
void clearContours(){contours.clear(); label_contours.clear();
main_contours.clear();}
```

- funkce pro vymazání vrstevnic, popisků vrstevnic a hlavních vrstevnic

```
void clearPolygon(){polygon.clear();}
```

- funkce pro vymyzání ohraničujícího polygonu

```
int getDtSize(){return dt.size();}
```

- funkce pro získání velikosti množiny Delaunay trojúhelníků

```
void setDTM(std::vector<Triangle> &dtm_){dtm = dtm_;}
```

- funkce pro nastavení DTM

```
void setAspectDTM(std::vector<Triangle> &aspect_dtm_){aspect_dtm = aspect_dtm_;}
```

- funkce pro nastavení DTM pro orientaci svahu

Program dále pracuje s vytvořenou třídou *Edge*, která slouží k práci s hranami. Umožňuje získat/nastavit počáteční a koncový bod hrany, případně změnit orientaci. Další vytvořenou třídou je *QPoint3D*, která je odvozená od třídy *QPointF*. Z ní dědí souřadnice *X* a *Y*, souřadnice *Z* byla námi doimplementována. Třída *Triangle* slouží pro práci s trojúhelníky. Jeden trojúhelník je pak tvořen třemi body (vrcholy) a dále informací o sklonu a orientaci svahu. S těmito parametry je možno pracovat pomocí „*get*“ a „*set*“ funkcí. Třídy *SortbyY* a *SortbyX* slouží k setřiďování bodů podle dané souřadnice.

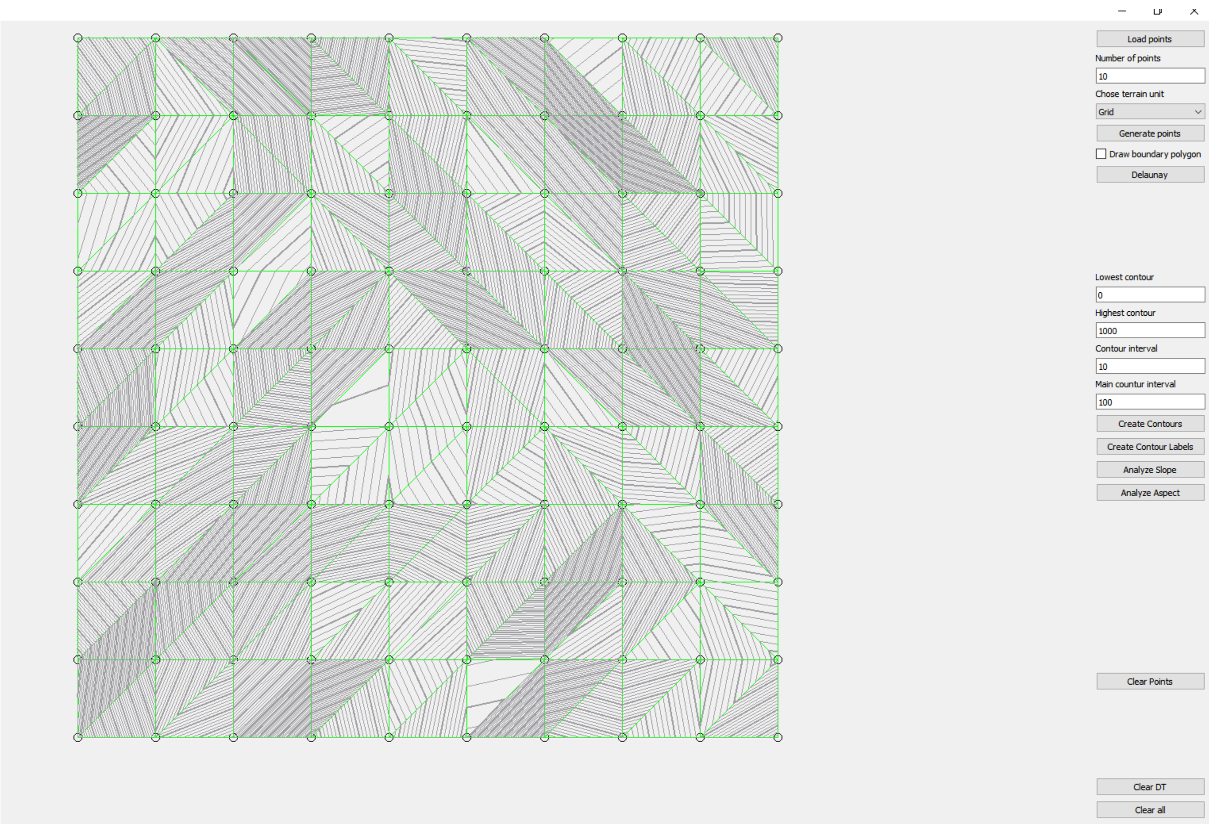
10. Náměty na vylepšení

- Při vykreslování vrstevnic by mohly být hladké přechody mezi jednotlivými stranami triangulační sítě.
- Při testování na reálných datech je značný rozdíl mezi vrstevnicemi. Absence povinných hran.

11. Závěr

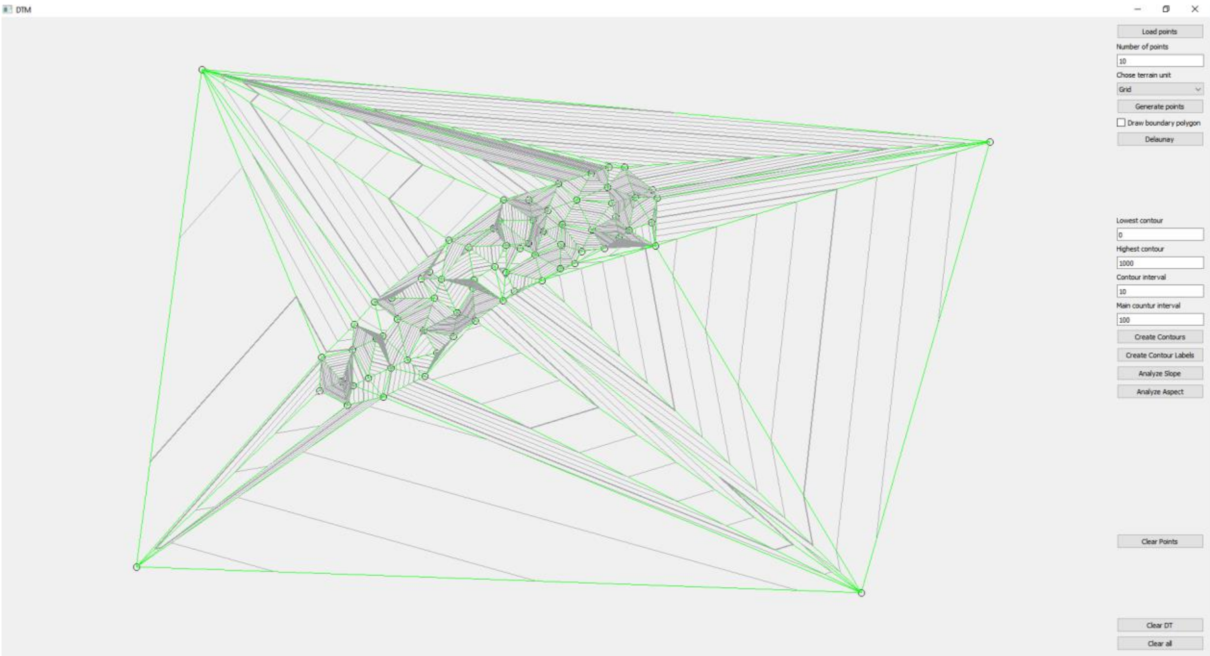
Byla vytvořena aplikace, která na množině bodů umí vygenerovat Delaunay triangulaci, vrstevnice, sklon a orientaci svahu. Vygenerované vrstevnice jsou správné, ale ne kartograficky korektní. To samé platí pro jejich popisky. Uživatel má možnost vygenerovat si různé terénní tvary (kupa, hřbet, údolí, spočinek). Dále je možno zadat ohraničující polygon, ve kterém se následně vytvoří triangulace.

Delaunay triangulace bohužel nedává vždy optimální výsledky. Jedním z případů je tvorba Delaunay triangulace v pravidelné mřížce, kde výpočet nemá jednoznačné řešení vzhledem ke stejným rozestupům mezi jednotlivými body. Vygenerované vrstevnice z takto vytvořené triangulace působí zvláštně a nereprezentují skutečný stav terénu.



Obrázek 13: Triangulace v gridu

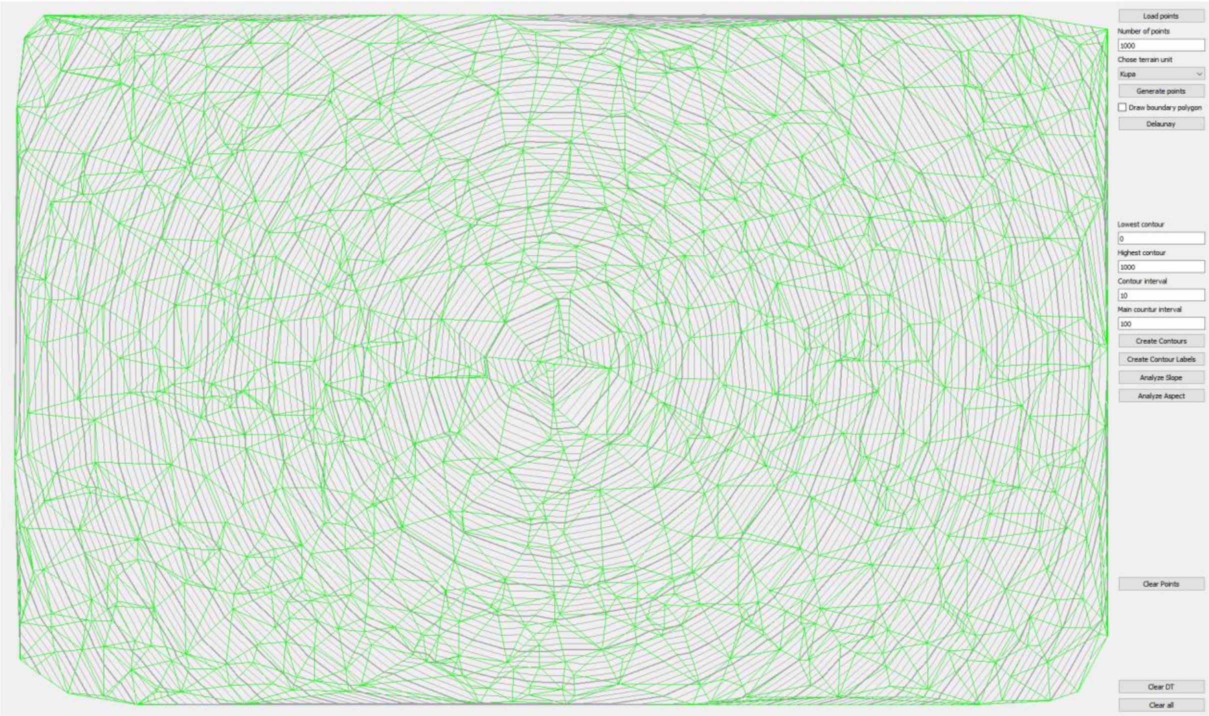
Dalším případem nevhodně vytvořené triangulace je situace, kdy chceme vytvářet triangulaci z dat s nestejnou hustotou bodů v rámci jednoho území. Vznikají tak velké, a především úzké trojúhelníky ve kterých vytvořené vrstevnice nemusí vůbec odpovídat realitě. Řešením by bylo nastavit v aplikaci nějakou mez, která by určovala, zda se jedná o odlehlé body a do výpočtu by následně nebyly uvažovány. S tvorbou úzkých trojúhelníků vzniká také nepřehlednost, kdy v takto vytvořených trojúhelnících bývají velmi často vrstevnice „napasované“ na sebe a splývají tak v jednu plochu. To je také dáno vysokým sklonem trojúhelníků a bylo by tedy vhodné nahradit vrstevnice v takovýchto případech jiným kartografickým znázorněním.



Obrázek 14: Úzké trojúhelníky

Dalším nevhodným výsledkem, a týká se to spíše již tvorby vrstevnic, jsou přechody vrstevnic mezi jednotlivými trojúhelníky, jak je patrné na obou předchozích obrázcích. Vrstevnice v těchto místech často tvoří ostré „špičky“, což má za následek pocit, že je terén tvořen na sebe přiloženými pláty. To samozřejmě není pravda, a proto by bylo vhodné výsledné vrstevnice na přechodech mezi jednotlivými trojúhelníky vhodně zaoblovat a vytvořit tak hladké přechody, jako je tomu i ve skutečném terénu.

Naopak případ, kdy naše aplikace dává správné výsledky, je generování kupy. Vrstevnice vzniklé z Delaunay triangulace vytvořené na takovémto terénním tvaru jsou hladké a je z nich krásně patrná samotná kupa i s vrcholem. Nutno podotknout, že vygenerovaná kupa má téměř dokonalý tvar a vrstevnice tak tvoří soustředné kružnice s konstantními rozestupy, nicméně jako ukázka správně vygenerované triangulace slouží dobře.



Obrázek 15: Korektně vygenerovaná triangulace