



Z Á P A D O Č E S K Á U N I V E R Z I T A

Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

Triangularizační metody v rovině

vedoucí : Doc. Ing. Václav Skala, CSc.
autor : David Blažek
ZČU Plzeň, 1995/96

<i>Úvod</i>	1
1 Základní pojmy z geometrie	2
2 Řešení problému triangularizace	4
2.1 Triangulace a triangularizace	4
2.2 Kritéria pro výběr optimální triangulace	5
2.3 Optimální triangulace	7
2.4 Delaunayova triangulace	10
2.4.1 Algoritmy lokálního zdokonalování.....	11
2.4.2 Algoritmy postupného vkládání.....	11
2.4.3 Přírůstkové algoritmy.....	11
2.4.4 Algoritmy využívající převodu do vyšších dimenzí.....	12
2.4.5 Rozděl-a-panuj.....	12
3 Triangularizace v algoritmech	13
3.1 Nenasytná triangularizace	13
3.1.1 Segmentový strom.....	14
3.2 Vázaná triangularizace	16
3.2.1 Řetězcová metoda lokalizace bodu v PSLG.....	16
3.2.2 Triangularizace monotónního polygonu.....	25
3.3 Voronoioův diagram	27
3.3.1 Některé vlastnosti Voronoioiva diagramu.....	29
3.3.2 DCEL (Double-Connected-Edge-List).....	33
3.4 Konstrukce Voronoioiva diagramu metodou rozděl-a-panuj	35
3.4.1 Konstrukce řetězce σ	40
3.5 Přírůstková konstrukce Voronoioiva diagramu	46
3.6 Přírůstková konstrukce Voronoioiva diagramu s důrazem na topologickou stabilitu	47
3.6.1 Přístup s důrazem na topologickou stabilitu.....	49
3.6.2 Použití numerických hodnot jako informace nižší priority.....	52
3.6.3 Zdokonalení numerických výpočtů.....	55
3.6.4 Závěrem.....	58
3.7 Konstrukce Voronoioiva a Delaunayova diagramu pomocí topologicky orientovaného algoritmu rozděl-a-panuj	58
3.7.1 Obvyklý algoritmus rozděl-a-panuj pro konstrukci Delaunayova diagramu.....	60
3.7.2 Algoritmus cestou kombinatorické geometrie.....	63
3.7.3 Použití numerických výpočtů.....	69
3.7.4 Závěrem.....	70
3.8 Delaunayova přírůstková triangularizace	71
3.8.1 Datová struktura „okřídlené hrany“.....	74
3.8.2 Lokalizace bodu metodou zjemňování triangulace.....	77
3.9 Triangularizační algoritmus DeWall (Delaunay Wall)	82
3.9.1 Realizace.....	82
3.9.2 Závěrem:.....	86
3.10 Delaunayova triangularizace využívající pravidelnou mříž	86
3.10.1 Předzpracování.....	86
3.10.2 Datová struktura.....	88
3.10.3 Triangularizační proces.....	89
3.10.3.1 Krok 1: Nalezení startovního generátoru a první hrany.....	89
3.10.3.2 Krok 2: Konstrukce trojúhelníků.....	91
3.10.3.3 Krok 3: Spojování sestavených trojúhelníků.....	93

Chyba! Neznámý argument přepínače.

3.10.3.4 Delaunayovo kritérium	100
3.10.4 Algoritmus	101
3.10.5 Konstrukce konvexní obálky	103
3.10.6 Degenerace.....	103
3.10.7 Datová struktura „seznam vrcholů“	103
3.10.8 Implementace metody Delaunayovy triangularizace využívající pravidelnou mříž	108
3.10.8.1 Inicializace.....	112
3.10.8.2 Detekce třetího bodu.....	112
3.10.8.3 Správa seznamu vrcholů a řízení triangulace.....	117
3.10.9 Výsledky testování.....	118
3.10.10 Závěrem	121
4 Triangularizace v praxi.....	122
Závěr.....	125
Nenasytná triangularizace - základní verze	125
Nenasytná triangularizace - modifikovaná verze (Gilbert, 1979).....	125
Vázaná triangularizace.....	126
Voronoiův diagram metodou rozděl-a-panuj.....	126
Přírůstková konstrukce Voronoiova diagramu	126
Přírůstková konstrukce Voronoiova diagramu s důrazem na topologickou stabilitu	127
Delaunayova triangularizace metodou rozděl-a-panuj	127
Delaunayova triangularizace topologicky orientovanou metodou rozděl-a-panuj	127
Delaunayova přírůstková triangularizace	128
Triangularizační algoritmus DeWall (Delaunay Wall).....	128
Delaunayova triangularizace využívající pravidelnou mříž.....	128
Použitá literatura.....	130

Úvod

Triangularizace je jedním z nejčastějších úkolů výpočtové geometrie a trigonometrická síť je velmi často používána v široké škále aplikací mnoha oborů. Jako vývoj mnoha dalších oborů, tak i vývoj triangularizačních metod zaznamenal s nástupem výpočetní techniky prudké zrychlení. Výkon dnešních počítačů však mnohonásobně převyšuje výkon počítačů používaných ještě před několika lety a to při snižování nákladů, a stále rychleji a rychleji roste. Tento fakt způsobuje, že činností využívajících výpočetní techniku stále přibývá a tak přibývá i aplikací využívajících trigonometrické sítě. Každá aplikace však řeší specifický problém a tak se stále objevují i nové a nové triangularizační metody a jejich variace. Přestože bouřlivý rozvoj informatiky stále zkracuje vzdálenosti mezi námi a zdroji informací, není jednoduché vývoj daného problému sledovat.

Cílem této diplomové práce je tedy zmapovat terén klasických i nově prezentovaných metod využívaných při počítačové realizaci triangularizace a usnadnit tak čitateli zorientování se v problému a následný výběr algoritmu vhodného pro řešení úloh čitateľovy oblasti zájmu.

Při přípravě této práce jsem zpracovávané metody setřídil tak, aby jejich pořadí vyhovovalo mému záměru zvolna postupovat od teorie k praxi. Na začátku jsou tedy definovány základní pojmy výpočtové geometrie používané v dalších kapitolách, následují pravidla a definice potřebné k zvládnutí problematiky triangularizace a hlavní principy na kterých jsou triangularizační algoritmy založeny. Dále práce pokračuje prezentací jednotlivých metod a algoritmů od nejjednodušších ke složitějším s postupným doplňováním řešení praktických záležitostí až k ukázce využití triangularizační techniky v praxi.

Závěr práce je věnován porovnání a zhodnocení vlastností a výkonností představených technik.

1 Základní pojmy z geometrie

Označení E^d znamená d -rozměrný *Euklidovský prostor*, tj. prostor „ d -tic“ (x_1, \dots, x_d) reálných čísel x_i , $i = 1, \dots, d$. Dále si velmi zhruba zopakujeme některé základní objekty užívané ve výpočtové geometrii a používané níže v této práci.

Bod. D -tice (x_1, \dots, x_d) označuje bod p v E^d .

Přímka. Jsou dány dva odlišné body q_1 a q_2 v E^d . Lineární kombinací

$$\alpha q_1 + (1 - \alpha)q_2 \quad (\alpha \in \mathbb{R})$$

je přímka v E^d .

Úsečka. Jsou dány dva odlišné body q_1 a q_2 v E^d . Přidáme-li k výrazu $\alpha q_1 + (1 - \alpha)q_2$ podmínku $0 \leq \alpha \leq 1$, dostaneme úsečku spojující dva body q_1 a q_2 :

$$\alpha q_1 + (1 - \alpha)q_2 \quad (\alpha \in \mathbb{R}, 0 \leq \alpha \leq 1).$$

Konvexní oblast. Oblast D v E^d je konvexní, jestliže pro každé dva body q_1 a q_2 v D platí, že úsečka q_1q_2 leží celá v oblasti D .

Pozn. Průnik dvou konvexních množin je konvexní množina.

Konvexní obálka. Konvexní obálka množiny bodů S v E^d je hranice obklopující nejmenší konvexní oblast v E^d obsahující celou množinu S .

Mnohoúhelník (polygon). V E^2 je polygon definován jako konečná množina úseček tak, že každý koncový bod úsečky je sdílen právě dvěma hranami polygonu a v žádných dvou podmnožinách hran polygonu se žádná hrana neopakuje. Počet vrcholů a hran je stejný.

Jednoduchý polygon. Řekneme, že polygon je *jednoduchý*, jestliže žádné dvě po sobě nejdoucí hrany nesdílí společný vrchol. Jednoduchý polygon rozděluje rovinu na dvě disjunktní oblasti, vnitřní (ohraničenou) a vnější (neohraničenou), které jsou odděleny polygonem (Jordan curve theorem). (Pozn.: termín polygon se často užívá pro označení sjednocení vnitřní oblasti a její hranice.)

Jednoduchý polygon je konvexní, jestliže je jeho vnitřní oblast konvexní množina.

Polygon hvězdicového tvaru. Jednoduchý polygon P je *hvězdicového tvaru*, existuje-li bod z , který neleží vně P takový, že pro všechny body p polygonu P platí, že úsečka zp leží celá uvnitř polygonu P . (To znamená, že každý

konvexní polygon je zároveň hvězdicového tvaru.) Oblast, kde se může bod z nacházet se nazývá jádro mnohoúhelníku P . (To znamená, že každý konvexní polygon je sám sobě jádrem.)

Rovinný graf. Graf $G = (V, E)$ (V je množina vrcholů, E je množina spojnic mezi těmito vrcholy) je rovinný, může-li být, aniž by došlo k překřížení některých spojnic, zasazen do roviny. Je-li množina E množinou úseček (hran), mluvíme o tzv. *přímkovém rovinném grafu*, neboli PSLG (planar straight-line graph).

Triangulace. Rozdělení roviny nazveme triangulací, jestliže všechny její ohraničené oblasti jsou trojúhelníky. Triangulace konečné množiny S bodů je rovinný graf na S s maximálním počtem hran (triangulace S je obdržena spojením bodů množiny S neprotínajícími se úsečkami tak, aby každá oblast uvnitř konvexní obálky tvořila trojúhelník).

2 Řešení problému triangularizace

2.1 Triangulace a triangularizace

📖 **Definice** [PREP85]: Necht' je v rovině dáno N bodů. Spojme je neprotínajícími se úsečkami tak, aby každá vnitřní oblast konvexní obálky tvořila trojúhelník. Množinu těchto trojúhelníků nazýváme triangulací.

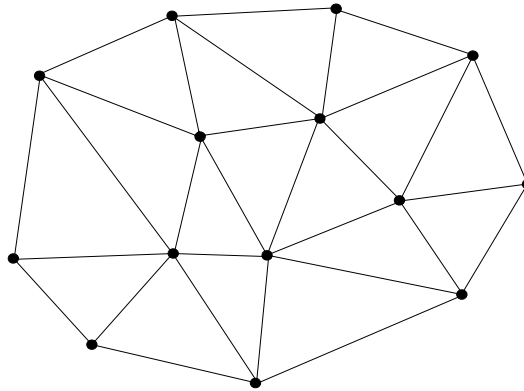
📖 **Definice** [HOSCH]: Jestliže množina $T = \{(\alpha_j, \beta_j, \gamma_j) : \alpha_j, \beta_j, \gamma_j \in \{1, \dots, N\}\}$ skládající se z M celočíselných trojic $(\alpha_j, \beta_j, \gamma_j)$ tvoří triangulaci množiny bodů $P = \{P_i = (x_i, y_i), i = 1(1)N\}$, pak platí:

- body $P_{\alpha_j}, P_{\beta_j}, P_{\gamma_j}$ tvoří pro každé $j = 1(1)M$ rohy jednoho trojúhelníku T_j ,
- každý trojúhelník je definován třemi body množiny T , které jsou současně rohy tohoto trojúhelníka,
- průnik vnitřních oblastí dvou trojúhelníků T_j, T_k , kde $j \neq k$, je prázdný,
- sjednocení všech trojúhelníků tvoří konvexní obálku bodů množiny P .

🔁 **Tvrzení:** Triangulace N vrcholů obsahuje maximálně $3N - 6$ hran.

Důkaz: Nejmenší možný počet bodů v triangulaci je tři. Taková triangulace (vlastně jeden trojúhelník) obsahuje tři hrany. Přidáním jednoho bodu přibydou nejméně dvě hrany (v případě, že tento bod se stane bodem konvexní obálky) a nejvíce tři hrany. Z toho vyplývá, že triangulace může obsahovat maximálně $3N - 6$ hran.

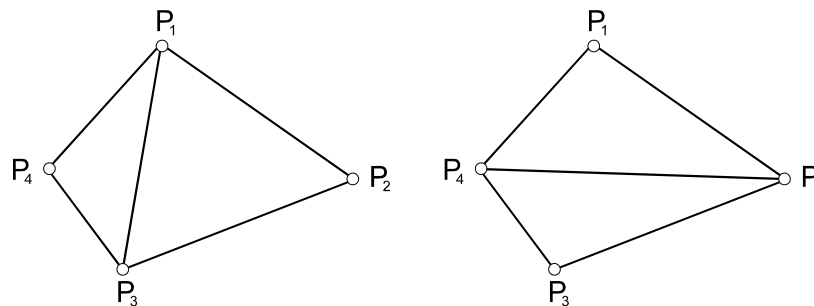
Triangularizace je tedy proces vedoucí k nalezení soustavy trojic bodů tvořících co možná neoptimálnější trojúhelníkovou síť. Slovo neoptimálnější tady znamená nejlépe vyhovující pro danou aplikaci, pro kterou je trojúhelníková síť tvořena (mohou být požadovány trojúhelníky co nejméně „podlouhlé“, či vytvořená síť má zachovávat některé hrany - např. pro účely kartografie při zpracovávání mimoúrovňového terénu, atd.).



Obr. 1: Příklad triangulace množiny bodů.

2.2 Kritéria pro výběr optimální triangulace

V rovině jsou dány čtyři různé body P_1, P_2, P_3 a P_4 , pro které platí, že žádné tři z těchto čtyř bodů neleží na přímce. Triangulaci tohoto jednoduchého případu můžeme provést dvěma způsoby a otázka zní jak vybrat tu výhodnější (viz. obrázek 2).



Obr. 2: Dvě možné varianty triangulace čtyř bodů.

Definice: Kritérium nejkratších úhlopříček:

Triangulace T je lepší než triangulace T' (z hlediska kritéria nejkratší diagonály) právě tehdy, když platí: $d < d'$, kde d je délka diagonály P_1P_2 triangulace T a d' je délka diagonály P_2P_4 triangulace T' .

Toto kritérium je sice příjemné pro svoji snadnou implementaci, ale není často používané, neboť se nevyhýbá generování, ve většině případů nežádoucích, dlouhých tenkých trojúhelníků. Z hlediska tohoto problému se jako výhodnější jeví následující Lawsonovo kritérium.

Definice: Max-Min úhlové kritérium:

Triangulace T je lepší než triangulace T' (z hlediska Max-Min úhlového kritéria) právě tehdy, když platí: $a(T) > a(T')$, $a(T) = \min\{a(T_j): T_j \in T\}$

a $a(T) = \min\{a(T_j): T_j \in T\}$, tj. nejmenší úhel triangulace T je větší než nejmenší úhel triangulace T' .

Nedostatky tohoto kritéria tkví v problémech při zpracovávání velmi malých úhlů o velikostech blízkých nule. Barnhill a Little proto navrhuji následující kritérium.

 **Definice:** Min-Max úhlové kritérium:

Triangulace T je lepší než triangulace T' (z hlediska Min-Max úhlového kritéria) právě tehdy, když platí: $a(T) < a(T')$, $a(T) = \max\{a(T_j): T_j \in T\}$ a $a(T') = \max\{a(T'_j): T'_j \in T'\}$, tj. největší úhel triangulace T je menší než největší úhel triangulace T' .

 **Definice:** Max-Min radius kritérium:

Triangulace T je lepší než triangulace T' (z hlediska Max-Min radius kritéria) právě tehdy, když platí: $r > r'$, $r = \min\{r(t_1), r(t_2)\}$ a $r' = \min\{r(t'_1), r(t'_2)\}$, kde $r(t_i)$ (resp. $r(t'_i)$) je poloměr kružnice opsané trojúhelníku t_i (resp. t'_i) triangulace T (resp. T'), tj. nejmenší poloměr kružnice opsané trojúhelníku v triangulaci T je větší než nejmenší poloměr kružnice opsané trojúhelníku v triangulaci T' .

 **Definice:** Min-Max radius kritérium:

Triangulace T je lepší než triangulace T' (z hlediska Min-Max radius kritéria) právě tehdy, když platí: $r < r'$, $r = \max\{r(t_1), r(t_2)\}$ a $r' = \max\{r(t'_1), r(t'_2)\}$, kde $r(t_i)$ (resp. $r(t'_i)$) je poloměr kružnice opsané trojúhelníku t_i (resp. t'_i) triangulace T (resp. T'), tj. největší poloměr kružnice opsané trojúhelníku v triangulaci T je menší než největší poloměr kružnice opsané trojúhelníku v triangulaci T' .

 **Definice:** Max-Min obsahové kritérium:

Triangulace T je lepší než triangulace T' (z hlediska Max-Min obsahového kritéria) právě tehdy, když platí: $P > P'$, $P = \min\{P(t_1), P(t_2)\}$ a $P' = \min\{P(t'_1), P(t'_2)\}$, kde $P(t_i)$ (resp. $P(t'_i)$) je obsah trojúhelníka t_i (resp. t'_i) triangulace T (resp. T'), tj. nejmenší z obsahů obou trojúhelníků triangulace T je větší než nejmenší z obsahů obou trojúhelníků triangulace T' .

 **Definice:** Max-Min výškové kritérium:

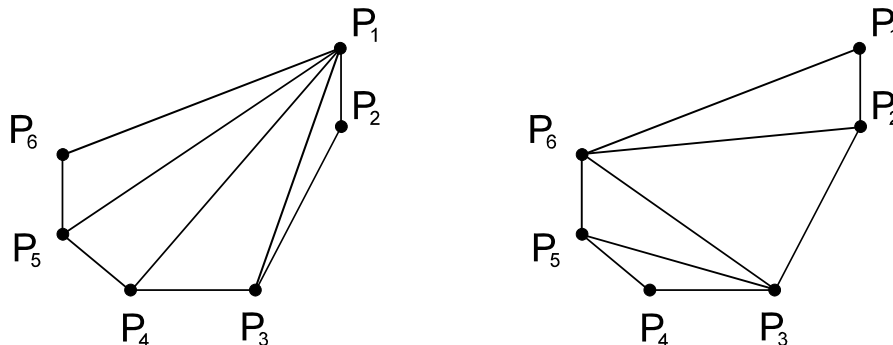
Triangulace T je lepší než triangulace T' (z hlediska Max-Min výškového kritéria) právě tehdy, když platí: $v > v'$, $v = \min\{v_i(T)\}$ a $v' = \min\{v_i(T')\}$, kde $v_i(T)$ (resp. $v_i(T')$) je výška i -tého trojúhelníka triangulace T (resp. T'), tj. nejmenší z výšek obou trojúhelníků triangulace T je větší než nejmenší z výšek obou trojúhelníků triangulace T' .

2.3 Optimální triangulace

Pomocí výše uvedených kritérií pro triangulaci čtyřúhelníku můžeme triangulovat také větší množiny bodů např. tak, že nejdříve provedeme triangulaci první čtveřice bodů a přidáváním dalších (např. k jedné straně vzniklého polygonu nejbližších) bodů se dopracujeme k optimální triangulaci. Na základě těchto kritérií můžeme dojít k lokálně optimálním triangulacím. Nyní si klademe otázku jak pomocí těchto kritérií docílit triangulací optimálních globálně.

Definice: Triangulace T se nazývá lokálně optimální z hlediska některého kritéria K tehdy, když každý čtyřúhelník definovaný dvěma trojúhelníky z T sdílejícími jednu hranu, která leží mezi nimi, je optimálně triangulován podle kritéria K .

Množině bodů může být přiřazeno několik lokálně optimálních triangulací (viz. obrázek 3).



Obr. 3: Dvě lokálně optimální triangulace podle Min-Max úhlového kritéria.

Ke globálně optimální triangulaci je možné přejít porovnáním uspořádaných M-tic, které vplynuly z lokálních rozhodovacích kritérií: Je-li číslo $a(T_j)$ měřítkem (podle výše uvedených kritérií) jakosti trojúhelníku T_j (dlouhý a tenký nebo rovnostranný), pak můžeme pro každou triangulaci T množiny bodů P definovat uspořádanou M-tici $a(T) = (a_1, \dots, a_M)$ složenou z podle velikosti seříděných složek $a_i = a_i(T_j)$. Kvalita dvou triangulací T a T' nyní může být vyhodnocena porovnáním vektorů $a(T)$ a $a(T')$, takže např. $a(T) < a(T')$ tehdy, když existuje celé číslo k takové, že $a_i = a'_i$ pro $i = 1, \dots, k - 1$ a $a_k < a'_k$.

Definice: Triangulace T množiny bodů P se nazývá globálně optimální (z hlediska kritéria K) tehdy, když pro výše uvedený vektor měřítek $a(T)$ kvality triangulace platí: $a(T) \geq a(T')$ (popř. \leq) pro každou triangulaci T' množiny bodů P .

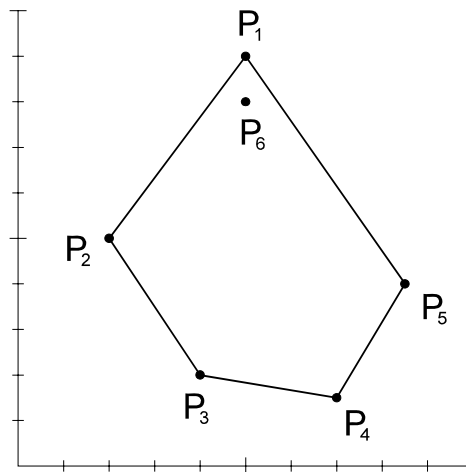
Globálně optimální triangulace je také lokálně optimální a je jednoznačná (až na případy, kdy vektor měřítek $a(T)$ zůstává konstantní, čímž můžeme dostat více globálně optimálních triangulací).

Příklad [HOSCH]:

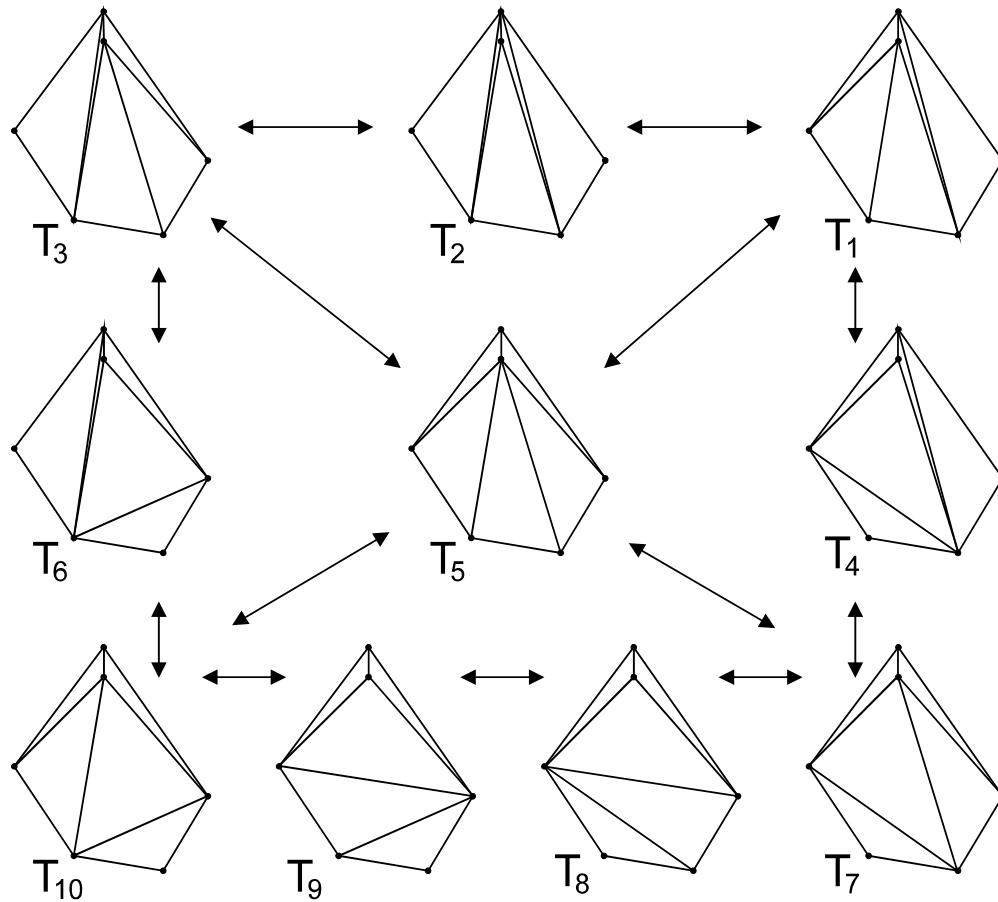
V rovině je definováno šest bodů (viz. obrázek 4):

$$\begin{array}{lll} P_1 = (5, 9) & P_2 = (2, 5) & P_3 = (4, 2) \\ P_4 = (7, 1.5) & P_5 = (8.5, 4) & P_6 = (5, 8) \end{array}$$

Těchto šest bodů může být triangularizováno deseti různými způsoby (triangulace T_i , $i = 1, \dots, M = 10$, viz. obrázek 5).



Obr. 4: Šest bodů k příkladu



Obr. 5: Deset možných způsobů triangulace šesti bodů z příkladu

Šipky na obr. znázorňují, že jimi označené sousední triangulace se liší pouze v jedné hraně, tj. v odlišné triangulaci jednoho čtyřúhelníka.

Vektory $a(T_i)$ triangulací T_i pak vypadají následovně:

Max-Min úhlové kritérium:

$$a(T_1) = (0.04, 0.14, 0.35, 0.46, 0.62)$$

$$a(T_2) = (0.02, 0.04, 0.35, 0.46, 0.50)$$

$$a(T_3) = (0.02, 0.11, 0.42, 0.46, 0.50)$$

$$a(T_4) = (0.04, 0.14, 0.35, 0.37, 0.66)$$

$$a(T_5) = (0.11, 0.14, 0.42, 0.46, 0.62)$$

$$a(T_6) = (0.02, 0.11, 0.50, 0.58, 0.88)$$

$$a(T_7) = (0.11, 0.14, 0.37, 0.42, 0.66)$$

$$a(T_8) = (0.11, 0.14, 0.37, 0.46, 0.70)$$

$$a(T_9) = (0.11, 0.14, 0.57, 0.58, 0.70)$$

$$a(T_{10}) = (0.11, 0.14, 0.58, 0.62, 0.88)$$

Min-Max úhlové kritérium:

$$a(T_1) = (2.84, 2.36, 1.99, 1.77, 1.57)$$

$$a(T_2) = (2.98, 2.84, 1.99, 1.91, 1.57)$$

$$a(T_3) = (2.98, 2.42, 1.91, 1.88, 1.57)$$

$$a(T_4) = (2.84, 2.36, 2.32, 1.99, 1.40)$$

$$a(T_5) = (2.42, 2.36, 1.88, 1.77, 1.57)$$

$$a(T_6) = (2.98, 2.42, 1.95, 1.91, 1.27)$$

$$a(T_7) = (2.42, 2.36, 2.32, 1.88, 1.40)$$

$$a(T_8) = (2.42, 2.36, 2.32, 1.50, 1.50)$$

$$a(T_9) = (2.42, 2.36, 1.95, 1.74, 1.50)$$

$$a(T_{10}) = (2.42, 2.36, 1.95, 1.77, 1.27)$$

Setříděním vektorů $a(T_i)$ pro Max-Min úhlové kritérium dostaneme pořadí: $a(T_2) < a(T_3) < a(T_6) < a(T_4) < a(T_1) < a(T_7) < a(T_8) < a(T_5) < a(T_9) < a(T_{10})$, tj. jako globálně optimální podle Max-Min úhlového kritéria se jeví triangulace T_{10} .


Setříděním vektorů $a(T_i)$ pro Min-Max úhlové kritérium dostaneme pořadí: $a(T_5) > a(T_9) > a(T_{10}) > a(T_8) > a(T_7) > a(T_1) > a(T_4) > a(T_3) > a(T_6) > a(T_2)$, tj. jako globálně optimální podle Min-Max úhlového kritéria se jeví triangulace T_5 . Avšak porovnáme-li ještě jedny vektory

$a(T_5) = (2.42, 2.36, 1.88, 1.77, 1.57)$ a $a(T_9) = (2.42, 2.36, 1.95, 1.74, 1.50)$, vidíme, že třetím prvkem je výhodnější $a(T_5)$ ($1.88 < 1.95$), ovšem čtvrtým i pátým prvkem je $a(T_9)$ opět lepší a proto vybereme jako globálně optimální triangulaci T_9 .

Max-Min úhlové kritérium a Min-Max úhlové kritérium vedou často ke stejným triangulacím. Srovnávací test s množinami náhodně generovaných čísel ukázal, že jen několik málo čtyřúhelníků obou triangulací je triangularizováno odlišně. Tato odlišnost se podle Nielsona dá vyčíslit přibližně deseti procenty (10% odlišných trojúhelníků).

Triangulace konstruovaná Lawsonovou metodou používající Max-Min úhlové kritérium přináší výsledky ekvivalentní Delaunayho triangulaci, která je často užívaným nástrojem pro generování vysoce kvalitní trojúhelníkové sítě a je vhodná pro většinu aplikací. Delaunayho triangulace je duální k Dirichletovu mozaikování, které je také často nazýváno Thiessenovými, či Voronoiovými diagramy.

2.4 Delaunayova triangulace

 **Definice:** [CIG92] *Delaunayova triangulace* $(DT(P))$ množiny bodů P daných v E^2 je množina vrcholů a hran (spojujících tyto vrcholy v síť trojúhelníků) takových, že platí:

1. Každý bod množiny P je vrcholem nejméně jednoho trojúhelníka z $DT(P)$ a naopak každý vrchol $DT(P)$ je bodem množiny P .
2. Průnik dvou různých hran $DT(P)$ je buď prázdný, nebo je jím společný vrchol $DT(P)$, tj. žádné dvě hrany $DT(P)$ se nekříží.
3. Pro každé tři vrcholy V_1, V_2 a $V_3 \in DT(P)$ tvořící trojúhelník $T_i \in DT(P)$ platí, že vnitřek kružnice $K(V_1, V_2, V_3)$ opsané trojúhelníku T_i neobsahuje žádný jiný bod množiny P .

Jak už bylo uvedeno výše, pro konstrukci Delaunayovy triangulace může být využito její duality k Voronoiovým diagramům. My se však nyní budeme zabývat přímými metodami generování Delaunayovy triangulace.

Algoritmy těchto metod mohou být rozděleny takto:

- algoritmy lokálního zdokonalování
- algoritmy postupného vkládání
- přírůstkové algoritmy
- algoritmy využívající převodu do vyšších dimenzí
- rozděl-a-panuj

2.4.1 Algoritmy lokálního zdokonalování

Tyto algoritmy nejdříve rychle vytvoří inicializační trojúhelníkovou síť, kterou pak použitím kritérií pro lokální optimum optimalizují.

2.4.2 Algoritmy postupného vkládání

Algoritmy postupného vkládání v prvním kroku vytvoří trojúhelník, který obsahuje celou množinu zpracovávaných bodů P . Potom vkládají v každém kroku po jednom bodu z P tak, že trojúhelník obsahující právě vkládaný bod je rozdělen na pod-trojúhelníky, jejichž jedním z vrcholů je právě tento nový bod. Všechny trojúhelníky sousedící s nově vzniklými jsou rekurzivně testovány na rovnostrannost a je-li to nutné, hrana oddělující tyto trojúhelníky je prohozena (pro dosažení rovnostrannosti, tj. splnění třetí podmínky Delaunayovy triangulace). Tato metoda je ve své nejprostší verzi velmi jednoduchá pro implementaci a zvládá generování trojúhelníkové sítě množiny bodů v E^d (samozřejmě pak nemluvíme o dělení na trojúhelníky, ale na d -simplexy (trojúhelník je 2-simplex, tetrahedron je 3-simplex). Navíc tento algoritmus má střední výpočtovou složitost řádu $O(n \log n + n^{\lfloor d/2 \rfloor})$, tedy v rovině $O(n + n \log n)$.

2.4.3 Přírůstkové algoritmy

Tyto algoritmy používají třetí kritérium Delaunayovy triangulace — „prázdné kružnice opsané“ — ke konstrukci trojúhelníkové sítě postupným přidáváním trojúhelníků, jejichž kružnice jim opsané neobsahují žádný jiný bod množiny P . Např. vychází z jednoho inicializačního trojúhelníka, k jehož straně přidají bod z P vyhovující kritériu „prázdné kružnice opsané“, který se stane vrcholem nového trojúhelníka sousedícího s inicializačním trojúhelníkem, pro který proces opakujeme až do posledního bodu z P . Efektivnost těchto algoritmů v prostých verzích je nízká (výpočtová složitost se pohybuje kolem $O(n^2)$ v rovině a $O(n^3)$ v E^3), ale mohou být použity efektivní akcelerační techniky.

2.4.4 Algoritmy využívající převodu do vyšších dimenzí

Tyto metody transformují bod z E^d do E^{d+1} (v našem případě z E^2 do E^3) a pak počítají konvexní obálku transformovaných bodů. Delaunayova triangulace je pak obdržena jednoduchou projekcí této konvexní obálky zpět do E^d (do E^2).

2.4.5 Rozděl-a-panuj

Tento algoritmus se ukázal jako jeden z nejlepších a proto i nejpoužívanějších pro zpracovávání množiny bodů v rovině, kde vykazuje velmi dobré výpočtové složitosti (jak střední případ, tak i nejhorší případ). Tato metoda je založena na rekurzivním dělení zpracovávané množiny bodů na podmnožiny, ve kterých jsou pak vytvořeny lokální trojúhelníkové sítě, které jsou poté ve spojovací fázi sjednoceny ve výslednou trojúhelníkovou síť. Algoritmy rozděl-a-panuj prezentované v odborné literatuře jsou navrženy pro řešení problému pouze v E^2 . To proto, že právě fáze spojování dílčích triangulací je dobře řešitelná v E^2 díky jednoznačnému uspořádání hran náležících k vrcholu. To již v E^d ($d > 2$) zachováno není a spojovací fáze je velmi obtížně realizovatelná.

3 Triangularizace v algoritmech

3.1 Nenasytá triangularizace

Tato metoda známá po názvem „nenasytá“ je metodou, která „nikdy nenapraví to, co již jednou udělá“. Algoritmus přidá k triangulaci v každém kroku jednu hranu a tento proces skončí v momentě, kdy se počet přidaných hran shoduje s předpokládaným počtem (zjištěným z velikosti zpracovávané množiny P a její konvexní obálky), nebo kdy jiný test diagnostikuje kompletnost sítě. Lokální optimum při přidávání hran zajišťuje kritérium výběru té nejkratší možné hrany z těch, které náleží danému vrcholu a nekříží žádnou z již zařazených hran.

Toto bylo přiblížení této techniky. Vlastní implementace vypadá následovně:

Inicializace:

Označme nejdříve množinu zpracovávaných bodů S a její velikost N .

Vygenerujme všech $\binom{N}{2}$ hran mezi body množiny S a seřídíme je podle

vzrůstající délky (do zásobníku hran). Triangulaci inicializujeme jako prázdnou.

Tělo:

Ze zásobníku vyjmeme nejkratší hranu. Nekříží-li tato hrana žádnou hranu v triangulaci, přidáme ji do zásobníku. V opačném případě ji zahodíme. Tento proces korektně končí v případě, že triangulace je kompletní (sledováním počtu přidaných hran), nebo když je zásobník hran prázdný.

Tato metoda není jenom jednoduchá pro implementaci, ale i pro analýzu.

Inicializační třídění hran podle délky spotřebuje $O(N^2 \log N)$ operací. Dále

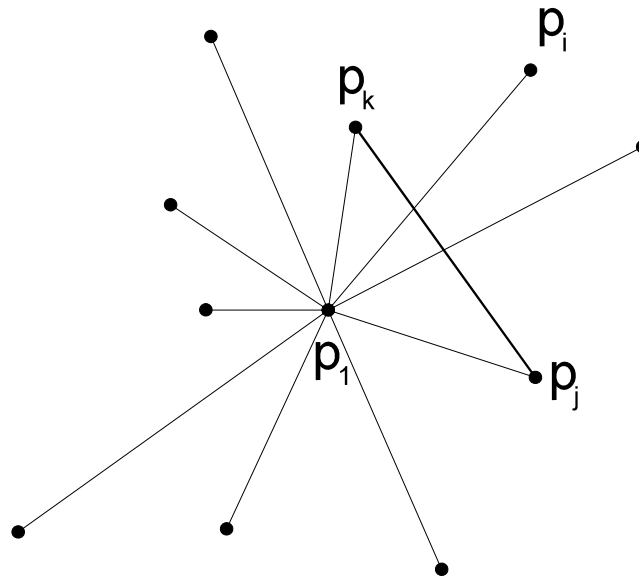
vybíráme $\binom{N}{2}$ hran ze zásobníku a každou testujeme na křížení s hranami

triangulace v nějakém čase $\varphi(N)$, jehož velikost je závislá na technice použité pro provedení tohoto testu. Celková výpočtová složitost je tedy

$O(N^2 \log N + N^2 \varphi(N))$.

Nejjednodušší test křížení hran je test vybrané hrany ze zásobníku s každou hranou triangulace. Jelikož každý průsečíkový test proběhne v konstantním čase, můžeme pro φ psát: $\varphi(k) = C_1 k$ (C_1 je konstanta). Tím se ve výsledku dostaneme na celkovou složitost $O(N^3)$.

Nyní si ukažme mnohem efektivnější řešení (Gilbert, 1979) [PREP85]. Cílem je zde udržovat rovnováhu mezi prací rozhodovací úlohy (zda vybraná hrana náleží triangulaci) a výběrovou úlohou (prohlížení hran podle vzrůstající délky).



Obr. 1: Hvězdice paprsků z p_1 (HVĚZDICE(p_1))

Předpokládejme, že právě vybraná hrana je p_1p_i (viz. obr.). Dále uvažujme množinu hran spojujících p_1 se všemi ostatními body v S a nazvěme ji hvězdici paprsků z p_1 , označme HVĚZDICE(p_1). Tyto paprsky jsou seřazeny, řekněme proti směru chodu hodinových ručiček, a rozdělují rovinu (úhel 2π s průcholem v p_1) na $(N - 1)$ sektorů, či úhlových intervalů. Jestliže je hrana p_kp_j přidělena do triangulace, překleneje množinu po sobě jdoucích sektorů v hvězdici paprsků z každého bodu p_l , kde $l \neq k, j$. Jak už bylo řečeno v definici, žádné dvě hrany v triangulaci se neprotínají, proto triangulační hrany ležící v každém sektoru HVĚZDICE(p_1), pro $l = 1, \dots, N$, jsou seřazené. Předpokládejme, že bod p_i leží v sektoru $p_jp_1p_k$ HVĚZDICE(p_1). Při rozhodování, jestli má být hrana p_1p_i přidána do triangulace, musíme testovat, zda se nekříží s nějakou hranou patřící k triangulaci a protínající tento sektor, a především s tou, která leží v sektoru k bodu p_1 nejbližší. To znamená, že test křížení je redukován na speciální případ problému určení polohy bodu v rovině, ovšem za předpokladu, že pro každý sektor hvězdice paprsků pro každý bod z S udržujeme jednu hranu triangulace, nazývanou *spanning edge*. Vyhledávací datová struktura podporující tento test (využití *spanning edge*) musí být dynamická, jelikož musí být aktualizována pokaždé, když je nová hrana přidána do triangulace. Vhodnou datovou strukturou se pro tento účel jeví *segmentový strom*.

3.1.1 Segmentový strom

Segmentový strom [PREP85] je datová struktura vytvořená pro správu intervalů čísel na reálné ose, jejichž extrémy náleží pevně množině dělení této

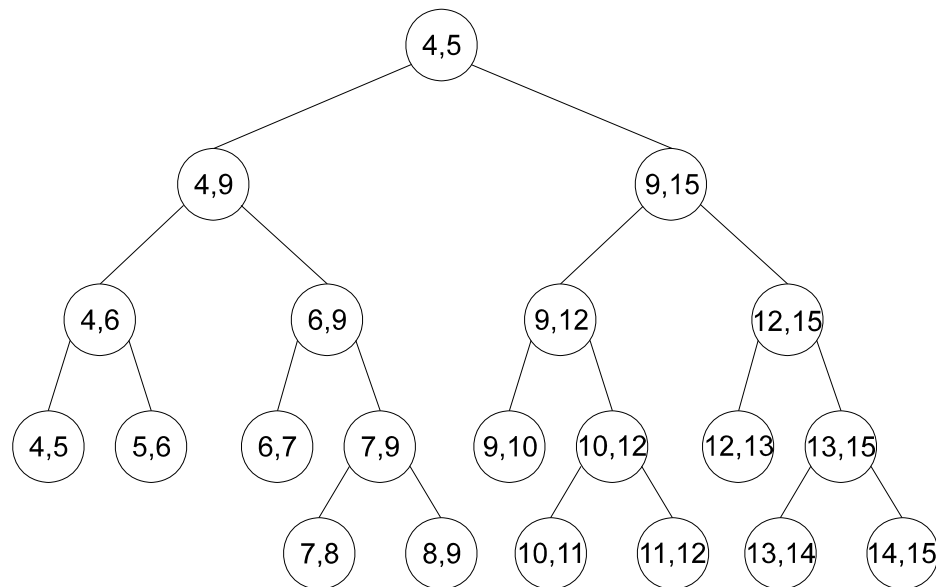
osy. Protože tato množina dělení je pevná, *segmentový strom* je statická struktura, která nepodporuje vkládání, ani mazání těchto dělení. Ty však mohou být normalizovány jejich záměnou za hodnotu jejich pořadí (zleva do prava) na číselné ose. Tak docílíme toho, že *dělení* můžeme považovat za celá čísla z intervalu $[1, N]$.

Segmentový strom je vlastně binární vyhledávací strom. Pro celá čísla l a r taková, že $l < r$, je *segmentový strom* $T(l, r)$ rekursivně vytvořen následujícím způsobem:

$T(l, r)$ se skládá z

- ◊ kořene v s parametry $B[v] = l$ (Beginning) a $E[v] = r$ (End), kde $r - l > 1$
- ◊ levého podstromu $T(l, \lfloor (B[v] + E[v])/2 \rfloor)$
- ◊ pravého podstromu $T(\lceil (B[v] + E[v])/2 \rceil, r)$.

Kořeny těchto podstromů jsou nazvány jako *levý syn* stromu T (zn. $LSON[v]$) a *pravý syn* stromu T (zn. $RSON[v]$). Parametry $B[v]$ a $E[v]$ definují interval $[B[v], E[v]] \subseteq [l, r]$ sdružený s uzlem v . Na obrázku 2 je příklad *segmentového stromu* $T(4, 15)$.



Obr. 2: Segmentový strom $T(4, 15)$

Intervaly z množiny $\{[B[v], E[v]]: v \text{ je uzal stromu } T(l, r)\}$ se nazývají *standardní intervaly*. *Standardní intervaly* příslušející listům stromu $T(l, r)$ se nazývají *elementární intervaly*. Je zřejmé, že *segmentový strom* je vyvážený, a jeho hloubka je $\lceil \log_2(r - l) \rceil$.

V případě *nenasytné triangularizace* jsou sektory $HVĚZDICE(p_i)$ organizovány jako $(N - 1)$ listů *segmentového stromu* T_1 . Každý uzal stromu T_1 je asociován s hranou $e(v)$, která je nejbližší hranou triangulace k bodu p_i . (Zřejmé $e(v)$ může být i „prázdnou hranou“.) Hrana $e(v)$ protínající sektor či

množinu po sobě následujících sektorů danou uzlem segmentového stromu, je určena třetím parametrem uzlu (číslo, nebo ukazovátka do seznamu hran). Testujeme-li hranu $p_i p_j$, procházíme segmentový strom T_j sledováním cesty určené bodem (paprskem HVĚZDICE(p_i)) p_j a v každém uzlu této cesty testujeme p_j proti hraně $e(v)$ (existuje-li). Tento proces je ukončen, když je hrana $p_i p_j$ odmítnuta. V opačném případě, kdy v žádném z uzlů této cesty neprotíná $p_i p_j$ hranu $e(v)$, je přijata. V tomto případě musí být ještě každá HVĚZDICE(p_l), pro $l \neq i, j$, aktualizována vložení $p_i p_j$. Každá HVĚZDICE(p_l) je aktualizována v čase $O(\log N)$. Hvězdic je celkem $(N - 2)$, proto celkový čas aktualizace je $O(N^2 \log N)$. To znamená, že „*nenasytná triangularizace*“ množiny N bodů může být sestrojena v čase $O(N^2 + N^2 \log N)$.

3.2 Vázaná triangularizace


Mnohé z možných řešení problému triangularizace mohou mít tzv. vázaný charakter, tj. část množiny triangulačních hran může být předurčena již v samotné formulaci problému. To si můžeme ukázat na případě, kdy řešíme úlohu triangularizace vnitřní oblasti jednoduchého mnohoúhelníku.

Právě výše uvedená technika (nenasytná triangularizace) může být použita pro řešení takových úloh, ale její nevýhodou je velmi nízká výkonnost a velká paměťová náročnost, pro které je v praxi nepříliš často využívána. Proto byla vyvinuta nová triangularizační technika, kterou si nyní přiblížíme.

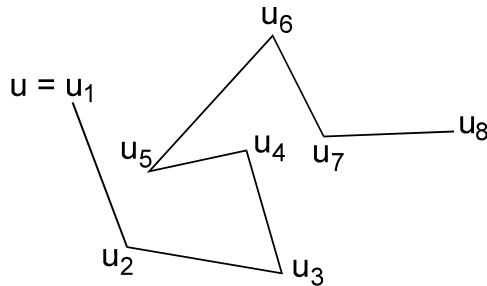
Jako obvykle je dána množina S , která je tvořena N body v rovině. Navíc je dána množina E obsahující M navzájem se neprotínajících hran (jejichž vrcholy jsou některé body z S), které se musí objevit v trojúhelníkové síti. Dále nepatrně upravíme předešlé předpoklady tak, že oblast určená k triangularizaci je nejmenší pravoúhelník, označme jej $PRAV(S)$, opisující množinu S , jehož strany jsou rovnoběžné s osami souřadného systému. (V tomto pravoúhelníku se nacházejí alespoň dva vrcholy s největší, a alespoň dva vrcholy s nejmenší y -ovou souřadnicí.)

Úkolem této metody je vhodně dekomponovat $PRAV(S)$ na jednodušší polygonální oblasti, které již mohou být snadno triangularizovány. Klíčem zde bude pojem „*řetězec*“, který je definován v následující pokapitole:

3.2.1 Řetězcová metoda lokalizace bodu v PSLG

 **Definice** [PREP85]: *Řetězec* $C = (u_1, \dots, u_p)$ je PSLG daný množinou vrcholů $\{u_1, \dots, u_p\}$ a množinou hran $\{(u_i, u_{i+1}) : i = 1, \dots, p - 1\}$, kde PSLG je přímkový rovinný graf (planar straight-line graph).

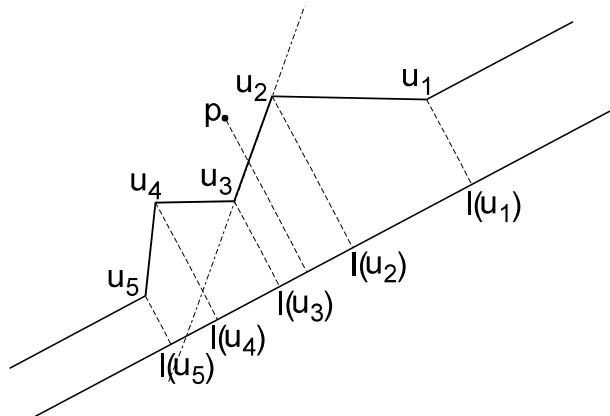
Tento řetězec je tedy speciálním případem PSLG, jinak zvaným též *polygonální čára*, viz. obrázek 1.



Obr. 1: Příklad obecného řetězce

Definice [PREP85]: Řekneme, že řetězec C je monotóní vzhledem k přímce l , jestliže průsečíkem libovolné kolmice k l a řetězce C je právě jeden bod.

To znamená, že pro řetězec $C = (u_1, \dots, u_p)$ platí, že množina vrcholů $\{u_1, \dots, u_p\}$ se při ortogonální projekci zobrazí na přímce l do seříděné množiny bodů $\{l(u_1), \dots, l(u_p)\}$, viz obr. 2.



Obr. 2: Příklad řetězce monotóního vzhledem k přímce l .

☞ **Poznámka:** Je-li dán navíc bod p , pak jeho obraz $l(p)$ vzniklý promítnutím na přímku l leží v některém z intervalů $(l(u_i), l(u_{i+1}))$, který můžeme snadno najít binárním vyhledáváním. Potom lze jedním testem určit polorovinu tvořenou přímkou $l(u_i)l(u_{i+1})$, která obsahuje bod p . (Toto spolu s následujícím lze využít pro řešení problému určení polohy bodu v rovině.)

Definice [PREP85]: Řekneme, že polygon P je monotóní vzhledem k přímce l , jestliže je jednoduchý a jeho hranice je sjednocením dvou řetězců monotóních vzhledem k l .

Předpokládejme, že $L = \{C_1, \dots, C_r\}$ je množina řetězců C_1, \dots, C_r monotóních vzhledem k jedné přímce l , pro niž dále platí:

◇ *Tvrzení 1:*

Daný rovinný graf G je podmnožinou sjednocení prvků množiny L . (Hrana grafu může náležet více než jednomu řetězci z L .)

◇ *Tvrzení 2:*

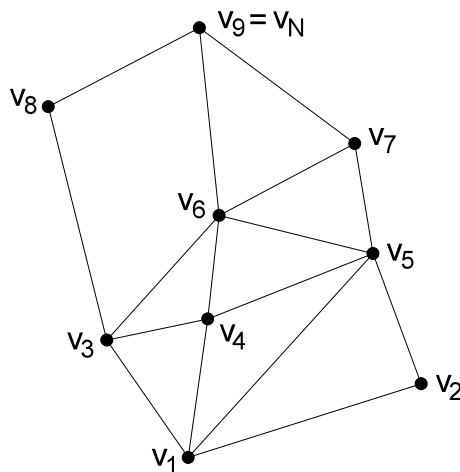
Pro každé dva řetězce C_i a C_j z množiny L platí, že vrcholy řetězce C_i , které nejsou zároveň vrcholy řetězce C_j , leží v téže polorovině vzniklé rozdělením roviny řetězcem C_j .

(To znamená, že řetězce C_1, \dots, C_r jsou setříděné, čehož lze dobře využít při vyhledávacích úlohách aplikací binárního vyhledávání.)

Důležitá otázka však zní: je možné daný PSLG interpretovat jako množinu monotóních řetězců? (A tím potažmo i monotóních polygonů?)
Odpověď: ne obecně, avšak po zavedení pojmu *regulární vrchol* a *regulární PSLG* (viz. níže) je možné jednoduše transformovat náš PSLG na takový graf, na který je již procedura vytvářející množinu monotóních řetězců (monotóních polygonů) aplikovatelná.

📖 **Definice [PREP85]:** Předpokládejme, že G je PSLG daný množinou vrcholů $\{v_1, \dots, v_N\}$, kde vrcholy v_1, \dots, v_N jsou indexovány tak, že $i < j$ právě tehdy, když je buď $y(v_i) < y(v_j)$, anebo $x(v_i) > x(v_j)$ pro případ, že $y(v_i) = y(v_j)$. Řekneme, že vrchol v_j je regulární, existují-li celá čísla i a k , $i < j < k$, taková, že (v_i, v_j) a (v_j, v_k) jsou hrany grafu G . Řekneme, že graf G je regulární, jestliže je regulární i každý vrchol v_j , kde $1 < j < N$ (kromě extrémních vrcholů v_1 a v_N).

Příklad regulárního grafu vidíme na obrázku 3.



Obr. 3: Příklad regulárního PSLG

Nyní si ukážeme, že regulární PSLG je interpretovatelný jako množina řetězců monotóních vzhledem k y -ové ose souřadnicového systému. (Od této chvíle budeme za přímku l považovat osu y .)

Představme si, že hrana (v_i, v_j) je orientována z v_i do v_j pro $i < j$. Potom můžeme mluvit o množině „vstupních“ hran (resp. vrcholů) $IN(v_j)$ a o množině „výstupních“ hran (resp. vrcholů) $OUT(v_j)$. Předpokládejme, že hrany z množiny $IN(v_j)$ jsou seřazeny proti směru chodu hodinových ručiček, zatímco hrany z množiny $OUT(v_j)$ po směru chodu hodinových ručiček. Vycházejce z definice regularity konstatujeme, že ani jedna z těchto množin není, pro každý vrchol vyjma extrémních, prázdná. Tento fakt nám umožňuje ukázat, že pro každý vrchol v_j ($j \neq 1$) můžeme zkonstruovat řetězec monotóní vzhledem k ose y (y -monotóní) z v_1 do v_j , což je triviální pro $j = 2$. Jelikož v_j je regulární, pak podle definice regularity existuje nějaké $i < j$ takové, že (v_i, v_j) je hrana grafu G . Z předchozího víme, že řetězec C jdoucí z v_1 do v_i je y -monotóní. Zřetěžením C a hrany (v_i, v_j) je očividně opět y -monotóní řetězec.

K dokončení důkazu ještě musíme ukázat, že výše uvedená tvrzení 1 a 2 platí. Označme $W(e)$ váhu hrany e (tj. počet řetězců, kterým e náleží). Dále zaveďme:

$$W_{IN}(v) = \sum_{e \in IN(v)} W(e)$$

$$W_{OUT}(v) = \sum_{e \in OUT(v)} W(e)$$

Proměnná $W_{IN}(v)$ je tedy součet vah všech vstupních hran do vrcholu v a podobně $W_{OUT}(v)$ je součet vah všech výstupních hran z vrcholu v .

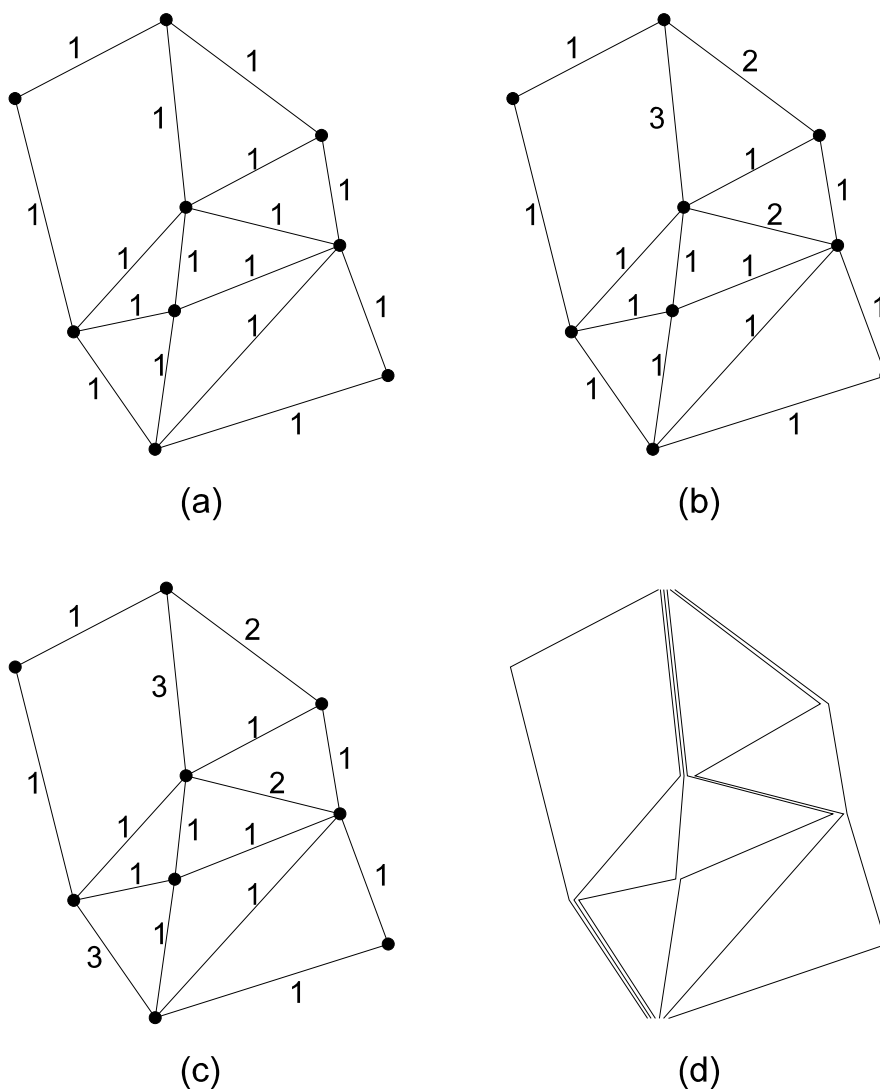
Nyní je třeba už jenom ukázat, že:

- (1) každá hrana má kladnou váhu, tedy $W(e) > 0$
- (2) pro každý vrchol v_j ($j \neq 1, N$) platí: $W_{IN}(v_j) = W_{OUT}(v_j)$.

Podmínka (1) zaručuje, že každá hrana náleží alespoň jednomu řetězci (Tvrzení 1), zatímco podmínka (2) garantuje, že $W_{IN}(v_j)$ řetězců prochází vrcholem v_j a mohou být nadefinovány tak, že se navzájem neprotínají (Tvrzení 1).

Zajištění rovnosti $W_{IN} = W_{OUT}$ může být dosaženo dvěma průchody grafem G . Nejdříve nastavíme pro každou hranu e $W(e) = 1$. Pak dostaneme, po prvním průchodu z v_1 do v_N , že $W_{IN}(v_i) \leq W_{OUT}(v_i)$ pro každý vrchol v_i , vyjma extrémů v_1 a v_N . Druhým průchodem, z v_N do v_1 , zajistíme, že $W_{IN}(v_i) \geq W_{OUT}(v_i)$, to znamená žádanou rovnováhu. Zaveďme ještě $v_{IN}(v) = |IN(v)|$ a $v_{OUT}(v) = |OUT(v)|$ a uveďme vzniklý algoritmus (v „pseudopascalu“) [PREP85]:

```
begin
  for každou hranu  $e$  do  $W(e) := 1$ ; (* inicializace *)
  for  $i := 2$  until  $N - 1$  do
    begin
       $W_{IN}(v_i) :=$  součet všech hran směřujících do  $v_i$ ;
       $d_1 :=$  nejlevější hrana vycházející z  $v_i$ ;
      if ( $W_{IN}(v_i) > v_{OUT}(v_i)$ ) then  $W(d_1) := W_{IN}(v_i) - v_{OUT}(v_i) + 1$ 
      end (* konec prvního průchodu *)
    for  $i := N - 1$  until  $2$  do
      begin
         $W_{OUT}(v_i) :=$  součet všech hran vycházejících z  $v_i$ ;
         $d_2 :=$  nejlevější hrana přicházející do  $v_i$ ;
        if ( $W_{OUT}(v_i) > W_{IN}(v_i)$ ) then  $W(d_2) := W_{OUT}(v_i) - W_{IN}(v_i) + W(d_2)$ 
        end (* konec druhého průchodu *)
      end.
```

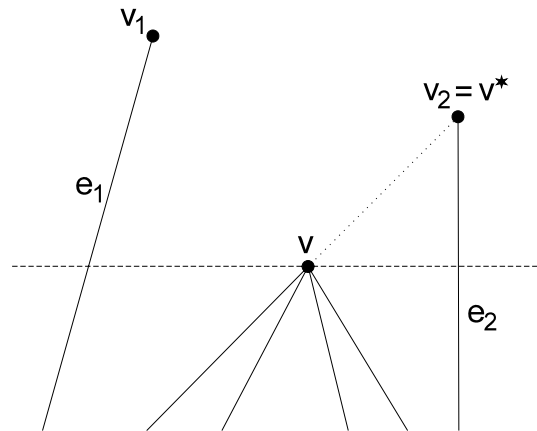


Obr. 4: Konstrukce váh hran naším algoritmem po inicializaci, obr. (a), po prvním průchodu, obr. (b), po druhém průchodu, obr. (c), a výslednou množinu řetězců L , obr. (d).

Výpočetní složitost tohoto algoritmu je lineární funkcí počtu hran a počtu vrcholů grafu G . Procedura může být modifikována tak, že konstrukce množiny řetězců L (např. přiřazení hran řetězcům) je provedena během druhého průchodu. Na obrázku 4 vidíme určování váh hran algoritmem po inicializaci, obr. (a), po prvním průchodu, obr. (b), po druhém průchodu, obr. (c), a výslednou množinu řetězců L , obr. (d). Tento obrázek tedy ukazuje, jak lze regulární PSLG interpretovat jako množinu monotóních řetězců.

Nyní si ukážeme, jak transformovat libovolný PSLG na regulární PSLG, tedy „regularizační“ proceduru, která je základem pro vhodnou dekompozici $PRAV(S)$:

Vezměme neregulární vrchol v grafu G takový, ze kterého nevychází žádná hrana (viz. obrázek 5).



Obr 5. Příklad neregulárního vrcholu v .

Horizontální (na obr. čárkovaná) přímka procházející vrcholem v obvykle protíná dvě hrany e_1 a e_2 grafu G ležící po obou stranách vrcholu v . (Alespoň jedna z těchto protínaných hran musí existovat, neboť v není regulární vrchol.) Necht' v_i je horní koncový bod hrany e_i ($i = 1, 2$) a v^* je v_i s menší y -ovou souřadnicí (v našem případě v_2). Pak úsečka vv^* neprotíná žádnou hranu v G a tak může být přidána do PSLG, čímž je „regularizován“ vrchol v . Tento krok bude základním stavebním kamenem naší regularizační procedury, ve které použijeme metodu „plane sweep“. Tj. množinu vrcholů budeme procházet shora dolů, přičemž budeme regularizovat vrcholy nemající žádnou výstupní hranu a poté, ve druhém cyklu, zdola nahoru, přičemž budeme regularizovat vrcholy nemající žádnou vstupní hranu. Dvě základní datové struktury metody „plane sweep“ jsou:

1. tzv. *event point schedule*, v našem případě posloupnost vrcholů (v_N, v_{N-1}, \dots, v_1) v prvním cyklu, ve druhém cyklu v opačném pořadí
2. tzv. *sweep-line status*, v našem případě zleva do prava seřazené posloupnosti průsečíků každé *sweep-line* s PSLG. Navíc, v každém intervalu je udržován vrchol s menší y -ovou souřadnicí z obou extrémů intervalu. Tato struktura může být realizována jako vyvážený binární vyhledávací strom.

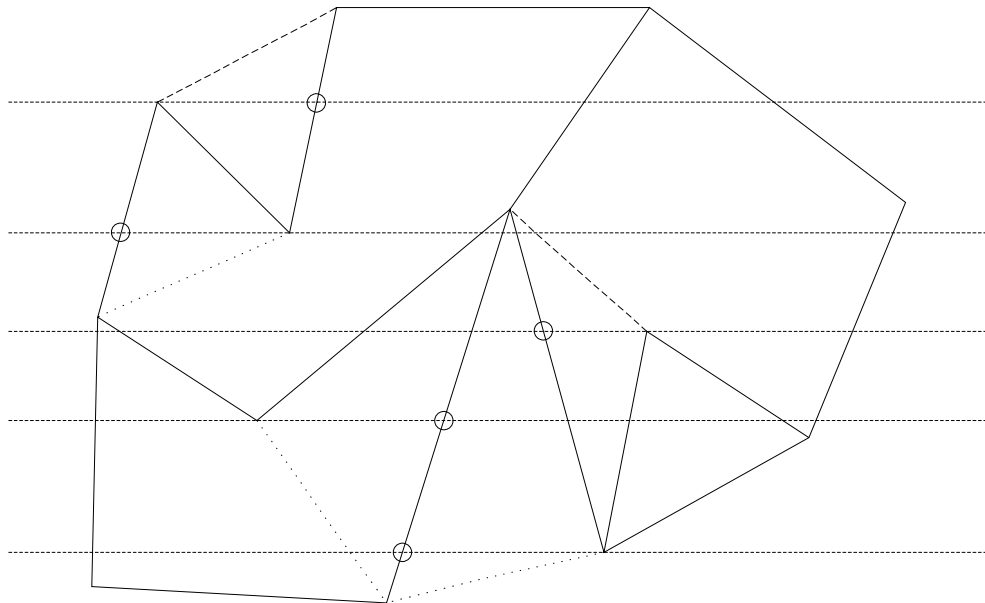
První cyklus algoritmu bude vypadat následovně:

- (1) lokalizujeme vrcholu v v intervalu na příslušné *sweep-line*, tj. v datové struktuře *sweep-line status*
- (2) jestliže není vrchol v regulární, sestrojíme hranu vv^* , tj. hranu z v do vrcholu asociovaném s intervalem z kroku (1) a tu přidáme jako novou hranu do PSLG

(3) v případě přidání nové hrany v kroku (2) aktualizujeme datovou strukturu *sweep-line status* pro vrcholy ležící mezi v a v^* (připomeňme, že vrcholy jsou seříděné podle y)

Druhý cyklus bude podobný, pouze s tím rozdílem, že budeme postupovat od v_1 k v_N . Určení vrcholu v^* může být v rámci regularity upraveno pro dosažení „lepších trojúhelníků“ jinak, než pouze podle menší y -ové souřadnice (např. v^* bude vrchol s bližší y -ovou souřadnicí).

Příklad takto regularizovaného PSLG je uveden na následujícím obrázku 6.



Obr. 6: Regularizovaný PSLG. Čárkované hrany jsou přidány do PSLG v prvním cyklu, tečkované v cyklu druhém. Zakroužkované jsou průsečíky *sweep-line* s PSLG, které určují hranu s vrcholem v^* .

Vzhledem k inicializačnímu třídění vrcholů podle y -ové souřadnice a lokalizaci každého vrcholu ve *sweep-line status* ($O(\log N)$), je výpočtová složitost regularizace $O(N \log N)$.

Tj. PSLG o N vrcholech může být regularizován v $O(N \log N)$ při paměťové složitosti řádu $O(N)$.

Shrneme-li tyto poznatky, máme množiny S a E definující PSLG. Do něj regularizační procedura (viz. výše) přidává v čase $O(N \log N)$ nové hrany tak, že:

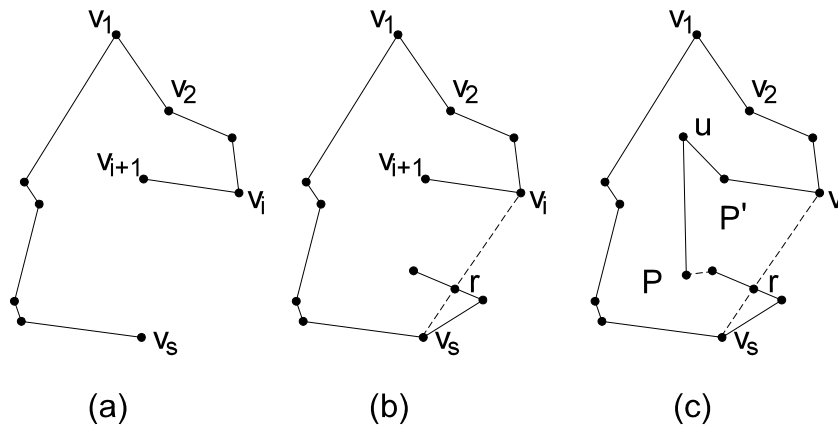
1. Žádné dvě hrany se neprotínají.
2. Každý vrchol (vyjma vrcholu s největší y -ovou souřadnicí) je přímo spojen s alespoň jedním vrcholem s větší y -ovou souřadnicí.
3. Každý vrchol (vyjma vrcholu s nejmenší y -ovou souřadnicí) je přímo spojen s alespoň jedním vrcholem s menší y -ovou souřadnicí.

Tvrdíme, že každý polygon vzešlý z regularizační procedury je monotóní. Důkaz je založen na pojmu „*vnitřní hrot*“:

Řekneme, že vrchol v jednoduchého polygonu je *vnitřním hrotem* tohoto polygonu, jestliže vnitřní úhel náležící vrcholu v je větší než π a oba sousední vrcholy zároveň buď mají větší, nebo menší y -ovou souřadnici než v . Z bodů 2 a 3 výše vyplývá, že žádný vrchol regularizovaného grafu nemůže být *vnitřním hrotem*.

☞ **Tvrzení:** Je-li P jednoduchý polygon neobsahující žádný vnitřní hrot, pak je P monotóní vzhledem k ose y .

Důkaz: Necht' v_1, v_2, \dots, v_m je posloupnost vrcholů polygonu P seřazená po směru chodu hodinových ručiček a v_1 a v_s jsou vrcholy s největší a nejmenší y -ovou souřadnicí, viz. obrázek 7.



Obr. 7: (a) v_i je první vrchol posloupnosti $v_1v_2 \dots$, pro který platí: $y(v_{i+1}) > y(v_i)$.
(b) Přímka z v_i do v_s protíná P v bodě r .
(c) Vrchol u s největší y -ovou souřadnicí v P' je *vnitřní hrot* polygonu P .

Jestliže P není monotóní, pak alespoň jeden ze dvou řetězců z v_1 do v_s vytvářejících hranici polygonu P nemá své vrcholy setříděny podle zmenšující se souřadnice y (není pouze klesající). Za tento řetězec uvažujme řetězec procházející vrcholem v_2 (pro druhý řetězec platí následující symetricky). Vrchol v_i , $1 < i < s$, je prvním vrcholem v posloupnosti, pro který $y(v_{i+1}) > y(v_i)$. Poznamenejme, že posloupnost vrcholů $(v_{i-1}v_iv_{i+1})$ tvoří v případě že v_i je vnitřní hrot *pravou otáčku*.

☞ **Poznámka:** Posloupnost vrcholů $(v_1v_2v_3)$, $v_i = [x_i, y_i]$, tvoří *pravou otáčku* platí-li:

$$\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} < 0$$

a levou otáčku, je-li tento determinant kladný.

Dále sestrojme přímku procházející vrcholy v_i a v_s . Bod r ($r \neq v_i$) je pak prvním průsečíkem této přímky ve směru od v_i k v_s s polygonem P . Úsečka $v_i r$ rozděluje zevnějšek polygonu P na dvě části. Jednou z nich je polygon P' , jak ukazuje obrázek v části (c). Kromě bodu r jsou všechny vrcholy polygonu P' zároveň vrcholy polygonu P . Necht' u je ten vrchol s největší y -ovou souřadnicí ze všech vrcholů polygonu P' . Pak u je také vrcholem polygonu P a zároveň vnitřním hrotem polygonu P .

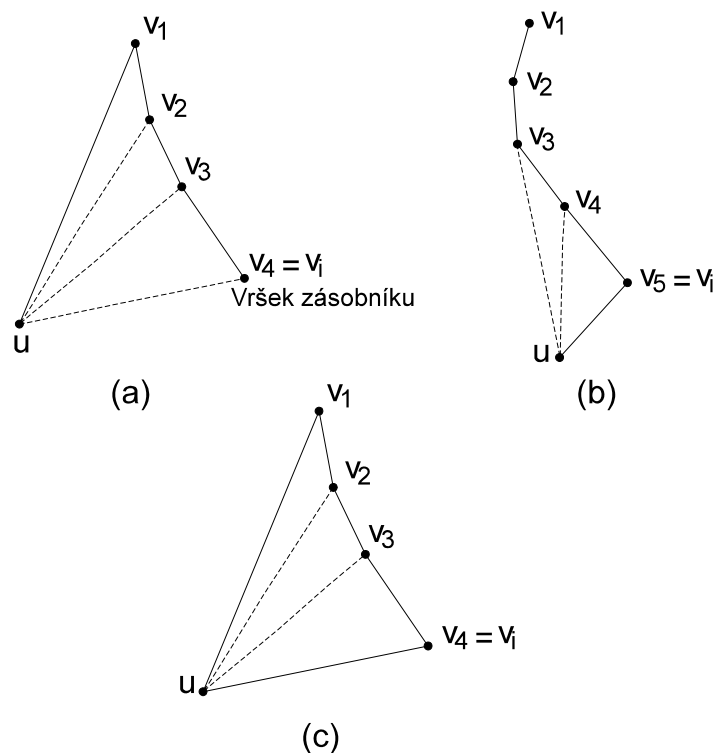
Výše uvedené tvrzení a fakt, že žádný polygon sestrojený regularizační procedurou neobsahuje vnitřní hrot dokazují, že každý polygon vzešlý z regularizační procedury je monotóní. Nyní si ukážeme, jak triangularizovat monotóní polygon.

3.2.2 Triangularizace monotóního polygonu

Jelikož P je monotóní polygon vzhledem k ose y , spojením dvou monotóních řetězců tvořících obálku tohoto polygonu můžeme v čase $O(N)$ seřadit vrcholy polygonu P sestupně podle y -ové souřadnice. Necht' u_1, u_2, \dots, u_N je výsledná posloupnost. Monotónie pak znamená, že pro každý vrchol u_i ($1 \leq i \leq N-1$) existuje i vrchol u_j ($i < j$) takový, že hrana $u_i u_j$ je hranou polygonu P .

Triangularizační algoritmus zpracovává v každém kroku po jednom vrcholu podle zmenčující se y -ové souřadnice. V tomto procesu se generují *diagonály* polygonu P . Každá diagonála „odkrojuje“ trojúhelník a zanechává k triangularizaci polygon s o jednu menším počtem hran. Algoritmus používá zásobník obsahující vrcholy, které již byly navštíveny, avšak ještě nebyly dosaženy diagonálou. Vršek i dno zásobníku jsou algoritmu přístupné (dno pouze pro čtení).

Zásobník obsahuje vrcholy v_1 (vršek zásobníku), v_2, \dots, v_i tvořící řetězec v obálce polygonu P , kde $y(v_1) > y(v_2) > \dots > y(v_i)$, přičemž je-li $i \geq 3$ platí pro $j = 1, \dots, i-2$, že úhel $(v_j v_{j+1} v_{j+2}) \geq 180^\circ$ (viz. obrázek 8).



Obr. 8: Tři případy hlavního kroku algoritmu triangularizace monotónního polygonu. Čárkovaně jsou vyznačeny přidávané diagonály.

Algoritmus pro triangularizaci monotónního polygonu vypadá takto:

- I. (Inicializační krok) Vrcholy u_1 a u_2 jsou vloženy do zásobníku.
- II. (Tělo) Necht' u je zpracovávaný vrchol. Pak:
 - Je-li u vrchol sousedící s vrcholem v_1 , nikoliv však s vrcholem v_i (vršek zásobníku), přidáme diagonály uv_2, uv_3, \dots, uv_i . V zásobníku nahradí v_i vrcholem u (na obr. (a)).
 - Je-li u vrchol sousedící s vrcholem v_i , nikoliv však s vrcholem v_1 , pak dokud platí, že $i > 1$ a úhel $(uv_i v_{i-1}) < 180^\circ$, přidáme diagonálu uv_{i-1} a vyjmemme ze zásobníku v_i . Jestliže tato dvojpodmínka přestane platit (nebo jestliže neplatila již od začátku), přidáme u do zásobníku (na obr. (b)).
 - Sousedí-li u jak s vrcholem v_1 , tak i s vrcholem v_i , přidáme diagonály $uv_2, uv_3, \dots, uv_{i-1}$ (na obr. (c)).

Korektnost tohoto algoritmu je závislá na faktu, že přidávaná diagonála leží celá uvnitř polygonu. Vezměme například diagonálu uv_2 v případě (i) obr. (a). Žádný z vrcholů v_3, \dots, v_i nemůže ležet uvnitř nebo na hranici trojúhelníku (u, v_1, v_2) , neboť vnitřní úhly vrcholů v_2, v_3, \dots, v_{i-1} jsou nejméně 180° , což zaručuje, že vrchol u leží v opačné polorovině dané přímkou $v_1 v_2$ než v_3, v_4, \dots, v_i . Žádný jiný vrchol polygonu uvnitř nebo na hranici trojúhelníku

neleží, neboť všechny tyto vrcholy mají menší y -ovou souřadnici než má vrchol u .

Z hlediska výkonnosti, inicializační spojení je provedeno v čase $O(N)$. V triangularizačním procesu je každý vrchol přidáván do zásobníku a zpracováván právě jednou (kromě možnosti, v případě (ii), kdy není splněna podmínka $i > 1$ a současně úhel $(uv_i v_{i-1}) < 180^\circ$). Tato úloha je proveditelná také v $O(N)$. To tedy znamená:

☞ **Tvrzení:** Monotóní polygon tvořený N stranami může být triangularizován v optimálním čase $O(N)$.

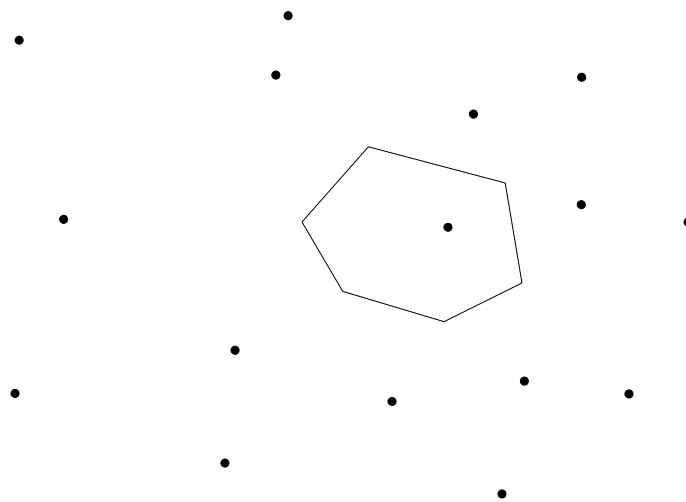
Kombinací tohoto závěru a předešlého výsledku (že $\text{PRAV}(S)$ může být dekomponován na monotóní polygony v čase $O(N \log N)$) dostáváme:

☞ **Tvrzení:** Vázaná triangularizace množiny N bodů může být sestrojena v optimálním čase $O(N \log N)$.

3.3 Voronoiův diagram

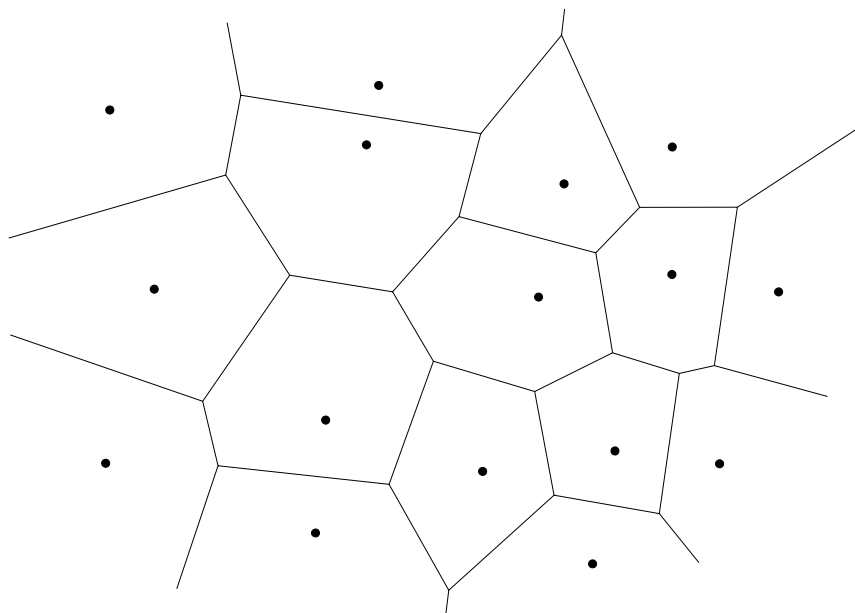
Voronoiův diagram je velmi důležitým pojmem výpočtové geometrie a používá se pro mnoha řešení mnoha problémů. Pomocí Voronoiova diagramu lze snadno určit polohu bodu, sestrojít konvexní obálku množiny bodů, či využít faktu, že Voronoiův diagram je duální k Delaunayově triangulaci, a je jí tedy ekvivalentní.

Nechť S je množina N bodů v rovině. Hledáme oblast obsahující bod p_i ($p_i \in S$) takovou, že libovolný bod z této oblasti je blíže k p_i než ke kterémukoli jinému bodu z S . Cítíme, že řešení tohoto problému rozdělí rovinu na regiony (každý region je oblast kolem bodu, ve které se může nacházet jiný bod, který je prvním zaručeně bližší než jakýkoli jiný bod mimo region). Nechť p_i a p_j jsou dva body v rovině. Oblastí, v níž se nalézají body bližší k p_i než k p_j , je právě ta polorovina vzniklá z roviny rozdělením kolmicí na $p_i p_j$ procházející středem $p_i p_j$, která obsahuje bod p_i . Označme tuto polorovinu jako $H(p_i, p_j)$. Oblast, obsahující body bližší k p_i než ke kterémukoli jinému bodu, je průnikem $N - 1$ polorovin a má tvar konvexního polygonu, který má nejvýše $N - 1$ stran. Označme ji $V(i)$. Tento útvar se nazývá Voronoiův polygon pro vrchol p_i a je ukázán na obrázku 1.



Obr. 1: Voronoiův polygon

Těchto N Voronoiových polygonů vytváří v rovině síť, které budeme říkat *Voronoiův diagram* a budeme ji označovat $Vor(S)$.



Obr. 2: Voronoiův diagram

Vrcholy tohoto diagramu se nazývají *Voronoiovy vrcholy* a úsečkám, které je spojují, říkáme *Voronoiovy hrany*.

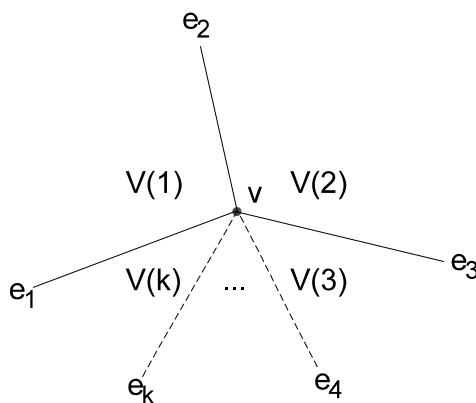
Každý z N vrcholů množiny S náleží právě jednomu Voronoiovu polygonu. To tedy znamená, že leží-li bod o souřadnicích (x, y) v polygonu $V(i)$, pak je tento bod nejbližším sousedem vrcholu p_i .

3.3.1 Některé vlastnosti Voronoiova diagramu

Předpoklad A: Žádné čtyři body z množiny S neleží na jedné kružnici.

☞ **Tvrzení:** Každý vrchol Voronoiova diagramu je průnikem právě tří hran tohoto diagramu.

Důkaz: Je zřejmé, že každý vrchol je průnikem množiny hran. Necht' je tedy e_1, e_2, \dots, e_k ($k \geq 2$) ve směru chodu hodinových ručiček seřazená posloupnost hran směřujících do vrcholu v , jak ukazuje obr. 3.



Obr. 3: Vrchol v a s ním spojené Voronoiovy hrany

Hrana e_i je společná pro Voronoiovy polygony $V(i - 1)$ a $V(i)$, pro $i = 2, \dots, k$ a hrana e_1 je společná pro $V(k)$ a $V(1)$. To znamená, že vrchol v je stejně vzdálený od p_1, p_2, \dots, p_k . To ale znamená, že p_1, p_2, \dots, p_k jsou kocirkulární, což je pro $k \geq 4$ v rozporu s předpokladem A. Proto $k \leq 3$. Nyní vezmeme případ, kdy $k = 2$. Pak hrana e_1 je společná pro $V(1)$ a $V(2)$, stejně jako e_2 . Obě tedy leží na kolmici půlící úsečku $p_1 p_2$, což znamená, že se ve v neprotínají. Dostáváme tedy, že $k = 3$.

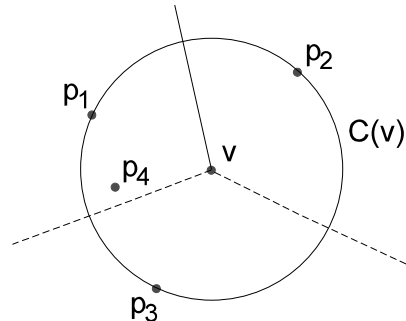
Jinak řečeno, Voronoiovy vrcholy jsou středy kružnic, které jsou dány třemi body množiny S a Voronoioův diagram je regulární třetího stupně.

☞ **Poznámka:** Graf G je *regulární*, jsou-li všechny jeho vrcholy stejného stupně.

Takovou kružnici se středem ve vrcholu v označme $C(v)$.

☞ **Tvrzení:** Pro každý vrchol v Voronoiova diagramu sestrojeného na množině bodů S platí, že kružnice $C(v)$ neobsahuje jiný bod množiny S .

Důkaz: Necht' p_1, p_2 a p_3 jsou tři z bodů množiny S , které leží na kružnici $C(v)$, jak je ukázáno na obrázku 4.

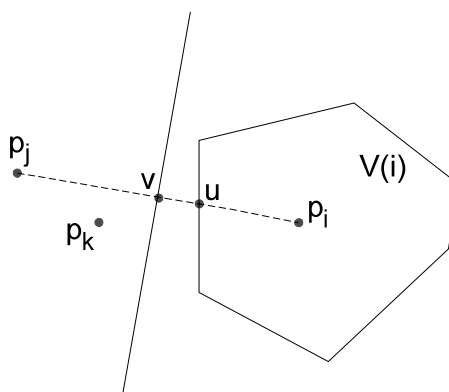


Obr. 4: Kružnice $C(v)$ nemůže obsahovat jiný bod množiny S .

Obsahuje-li kružnice $C(v)$ nějaký jiný bod z S , řekněme p_4 , pak p_4 leží blíže k v než p_1, p_2 a p_3 . To ale znamená (podle definice Voronoiova polygonu), že v musí ležet v polygonu $V(4)$ a ne v polygonech $V(1), V(2)$, ani $V(3)$. To je ale rozpor, neboť v zadání je v pro $V(1), V(2)$ a $V(3)$ společné.

☞ **Tvrzení:** Každý nejbližší soused vrcholu p_i definuje hranu Voronoiova polygonu $V(i)$.

Důkaz: Necht' p_j je nejbližší soused bodu p_i a necht' v je střed úsečky $p_i p_j$, viz. obrázek 5.

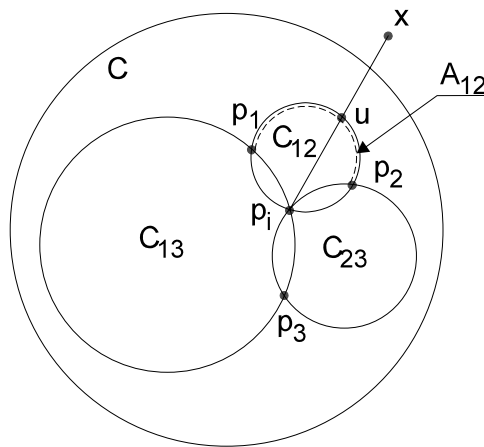


Obr. 5: Každý nejbližší soused vrcholu p_i definuje hranu Voronoiova polygonu $V(i)$.

Předpokládejme, že v neleží na hranici $V(i)$. Pak úsečka $p_i v$ protíná nějakou hranu polygonu $V(i)$, řekněme Voronoiovu hranu danou body p_i a p_k , v bodě u . Potom $délka(p_i u) < délka(p_i v)$, takže $délka(p_i p_k) \leq 2 * délka(p_i u) < 2 * délka(p_i v) = délka(p_i p_j)$, což znamená, že p_k leží blíže k p_i než p_j .

↪ **Tvrzení:** Polygon $V(i)$ je neohraničen (otevřen) jen a pouze tehdy, když bod p_i je bodem konvexní obálky množiny S .

Důkaz: Nepatří-li bod p_i konvexní obálce množiny S , leží uvnitř nějakého trojúhelníka $p_1p_2p_3$. Předpokládejme, že kružnice C_{12} prochází body p_i, p_1 a p_2 , kružnice C_{13} body p_i, p_1 a p_3 a konečně na kružnici C_{23} leží body p_i, p_2 a p_3 , jak ukazuje obrázek 6.



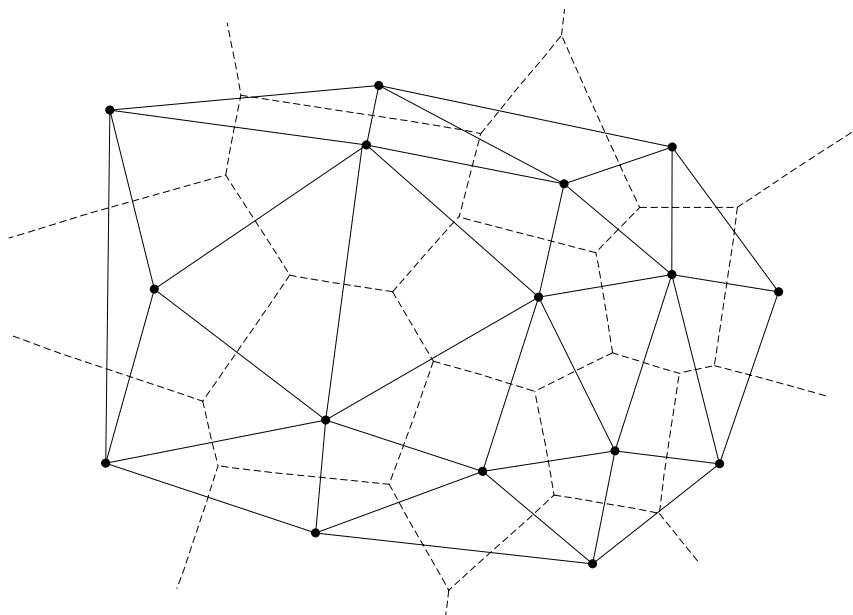
Obr. 6: K důkazu výše uvedeného tvrzení.

Každá z těchto kružnic má konečný poloměr. Dále A_{12} je kruhový oblouk na kružnici C_{12} mezi body p_1 a p_2 takový, že neobsahuje bod p_i . Lze snadno ukázat, že každý bod ležící na oblouku A_{12} je blíže k p_1 , nebo k p_2 , než k p_i . Totéž lze ukázat i pro kružnice C_{13} a C_{23} . Necht' C je kružnice obklopující C_{12} , C_{13} i C_{23} . Tvrdíme, že každý bod x ležící vně C je bližší k jednomu bodu z p_1 , p_2 , nebo p_3 , než k p_i . Opravdu, vezměme úsečku xp_i . Tato úsečka protíná jednu ze tří stran trojúhelníka $p_1p_2p_3$, řekněme p_1p_2 . To znamená, že protíná také oblouk A_{12} v bodě u . Bod u je ale bližší buď k p_1 nebo k p_2 než k p_i . Jelikož x je bližší k p_1 , p_2 , nebo k p_3 , než k p_i , polygon $V(i)$ leží celý uvnitř kružnice C a je uzavřený. Voronoiův polygon $V(i)$ je tedy uzavřený a necht' e_1, e_2, \dots, e_k ($k \geq 3$) je posloupnost jeho hran. Každá hrana e_h ($h = 1, \dots, k$) leží na kolmici půlící úsečky $p_i p_h$, kde $p_h \in S$ a $h \neq i$. Z toho plyne, že p_i je vnitřním bodem polygonu $p_1p_2 \dots p_k$, tj. bod p_i není bodem konvexní obálky množiny S .

Jelikož jen neuzavřený (neohraničený) polygon může mít „paprsky“ jako své hrany, pak tyto „paprsky“ oddělují dvojici sousedů konvexní obálky množiny S .

Nyní vezměme náš Voronoiův diagram a úsečkou spojme každé dva sousední body tak, že jí protneme hranu společnou příslušným Voronoiovým

polygonům. Dostaneme tak planární graf sestrojený na množině bodů S duální k Voronoiovu diagramu, viz. obrázek 7.



Obr. 7: Planární graf duální k Voronoiovu diagramu.

☞ **Tvrzení:** Graf duální k Voronoiovu diagramu je triangulace množiny bodů S .

Důkaz: Abychom dokázali, že graf duální k Voronoiovu diagramu je triangulace, musíme ukázat, že konvexní obálka množiny S je rozdělena na trojúhelníky dané body z S . Tyto trojúhelníky jsou konstruovány následovně. Nechť v je Voronoiov vrchol sdílený polygony $V(1)$, $V(2)$ a $V(3)$. $T(v)$ je potom trojúhelník tvořený vrcholy p_1 , p_2 a p_3 . Tvrdíme, že leží-li celý trojúhelník $T(v)$ uvnitř konvexní obálky S , pak není degenerovaný (tj. p_1 , p_2 a p_3 nejsou kolineární). V případě, že by p_1 , p_2 a p_3 byli kolineární, pak by úsečky p_1p_2 , p_2p_3 a p_1p_3 leželi na jedné přímce a jejich osy by byly rovnoběžné. Vrchol v ovšem leží na jejich průsečíku, tj. v nekonečnu. To znamená, že $V(1)$, $V(2)$ a $V(3)$ jsou neuzavřené, tj. kolineární body p_1 , p_2 a p_3 jsou body konvexní obálky, což je v rozporu s tvrzením, že $T(v)$ leží uvnitř této obálky množiny S .

☞ **Tvrzení:** Voronoiov diagram sestrojený na množině N bodů se skládá z nejvýše $2N - 5$ vrcholů a $3N - 6$ hran.

Důkaz: Každé hraně duálního grafu náleží právě jedna Voronoiova hrana. Triangulace duální k tomuto Voronoiovu diagramu má nejvýše $3N - 6$ hran a $2N - 4$ oblastí. Proto počet Voronoiových hran je také nejvýše $3N - 6$,

zatímco počet vrcholů diagramu je nejvýše $2N - 5$, neboť k Voronoiovu polygonu je duální pouze ohraničená oblast triangulace.

☞ Poznámka: Každý Voronoiov polygon může tvořit nejvýše $N - 1$ hran. Protože hran je však celkem nejvýše $3N - 6$ a každá z nich je sdílena právě dvěma polygony, průměrný počet hran Voronoiova polygonu nepřevyšuje číslo šest.

Z toho plyne, že zavedením duality může být (jak už bylo uvedeno na začátku kapitoly) využito Voronoiova diagramu pro konstrukci Delaunayovy triangulace.

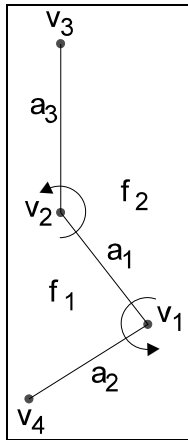
3.3.2 DCEL (Double-Connected-Edge-List)

Ještě než se budeme zabývat samotnou konstrukcí Voronoiova diagramu, seznámíme se s datovou strukturou, na kterou se v dalších kapitolách budeme odvolávat.

DCEL (Double-Connected-Edge-List) je datová struktura zvláště vhodná k popisu planárních grafů. Planární graf $G = (V, E)$ je mapováním každého vrcholu V do bodu v rovině a každé hrany E do jednoduché křivky spojující dva obrazy dvou vrcholů náležících hraně E tak, že žádné dva obrazy hran se neprotínají (s výjimkou jejich koncových bodů). Navíc platí, že pro planární grafy jsou těmito křivkami úsečky, tedy opět hrany.

Nechť $V = \{v_1, \dots, v_N\}$ a $E = \{e_1, \dots, e_M\}$. Hlavní složkou DCEL planárního grafu (V, E) je *uzel hran*. Každá hrana je v DCEL reprezentována právě jednou. Každý uzel se skládá ze čtyř polí V^1, V^2, F^1, F^2 a dvou ukazatelů P^1 a P^2 . Proto je DCEL reprezentována šesti poli o M položkách. Nyní si vysvětleme význam těchto polí. Pole V^1 obsahuje počáteční bod hrany, pole V^2 koncový bod hrany a tak je hrana zorientována. Pole F^1 a F^2 obsahují názvy oblastí ležících po levé straně a po pravé straně hrany vzhledem k její orientaci. A konečně ukazatelé P^1 a P^2 ukazují na, ve smyslu pořadí proti chodu hodinových ručiček, první hranu sousedící s hranou (V^1, V^2) vycházející z V^1 a z V^2 . Pro názvy oblastí a vrcholů můžeme použít typ integer. Příklad DCEL je ukázán na obrázku 8.

	V^1	V^2	F^1	F^2	P^1	P^2
a_1	1	2	1	2	a_2	a_3
a_2	4	1	1			
a_3	2	3		2		



Obr. 8: Příklad DCEL

To znamená, že např. pro graf o N vrcholech a F oblastech, můžeme použít pole hlaviček vrcholů $HV[1:N]$ a seznam oblastí $HF[1:F]$. Tyto dvě pole mohou být vyplněny v čase $O(N)$ průchodem polí V^1 a F^1 . Následující procedura $VRCHOL(j)$ vrátí posloupnost hran spjatých s vrcholem v_j jako posloupnost adres uložených v poli A .

```

procedure VRCHOL( $j$ )
begin
   $a := a;$ 
   $a_0 := a;$ 
   $A[1] := a;$ 
   $i := 2;$ 
  if ( $V^1[a] = j$ ) then  $a := P^1[a]$  else  $a := P^2[a];$ 
  while ( $a \neq a_0$ ) do
    begin
       $A[i] := a;$ 
      if ( $V^1[a] = j$ ) then  $a := P^1[a]$  else  $a := P^2[a];$ 
       $i := i + 1;$ 
    end
  end.

```

Výpočtová složitost této procedury je nepochybně přímo úměrná počtu hran náležících vrcholu v_j . Podobně můžeme sestavit proceduru $OBLAST(j)$ vracející posloupnost hran ohraničujících oblast f_j , a to pouze záměnou HV a V^1 za HF a F^1 . Procedura $VRCHOL(j)$ vrací hrany kolem v_j v pořadí proti chodu hodinových ručiček, zatímco procedura $OBLAST(j)$ vrací hrany kolem oblasti f_j v pořadí ve směru chodu hodinových ručiček.

3.4 Konstrukce Voronoiova diagramu metodou rozdě-l-a-panuj

Ačkoliv Voronoiovy diagramy uvádím pro zcela jiné účely, bylo by škoda nezmínit se o tom, že těchto diagramů se užívá v řadě jiných odvětvích. Například v archeologii jsou Voronoiovy diagramy používány k mapování schopnosti užívat nástroje starověkými kulturami a pro studium působení soupeřících obchodních center. V ekologii se jedná o závislost přežití organismů na počtu „sousedů“ soupeřících o potravu a světlo a také o využití Voronoiových diagramů druhů lesů a pralesů a lokalit obývaných různými druhy živočišné říše pro výzkum následků přelidnění. Struktura molekul je dána kombinací účinků elektrických sil a sil krátkého dosahu působení, které jsou zkoumány konstrukcí podrobných, důkladně vypracovaných Voronoiových diagramů.

Konstrukcí Voronoiova diagramu $\text{Vor}(S)$ na množině bodů S míníme vytvoření popisu diagramu, jako planárního grafu, skládajícího se z následujících částí:

1. Souřadnice Voronoiových vrcholů;
2. Množina hran spojujících vždy dva Voronoiovy vrcholy a dvě hrany, seřazené proti směru chodu hodinových ručiček, které jsou jejich následníky v každém koncovém vrcholu hrany (DCEL, nebo-li Double-Connected-Edge-List). Tato seříděnost zároveň definuje proti směru chodu hodinových ručiček orientované cykly hran v každém vrcholu a po směru chodu hodinových ručiček orientované cykly hran kolem každé oblasti (Voronoiových polygonů).

Nejjednodušší přístup ke konstrukci Voronoiova diagramu je konstrukce jeho polygonů po jednom v každém kroku. Jelikož každý Voronoiov polygon je průnikem $N - 1$ polorovin, může být sestrojen v čase $O(N \log N)$ a celý diagram tedy v celkovém čase $O(N^2 \log N)$. Dále ukážeme, že celý diagram může být sestrojen v optimálním čase $O(N \log N)$, což znamená, že konstrukce diagramu je asymptoticky ne více složitá než nalezení jednoho z jeho polygonů!

Tato složitost je důvodem toho, že Voronoiov diagram je eminentním uchazečem o metodu *rozděl-a-panuj*. Hrubé nastínění algoritmu:

procedure VORONOI_DIAGRAM

- krok (1) Rozděl S na dvě podmnožiny S_1 a S_2 přibližně stejně velké.
- krok (2) Sestroj rekurzivně $\text{Vor}(S_1)$ a $\text{Vor}(S_2)$.
- krok (3) Spojením $\text{Vor}(S_1)$ a $\text{Vor}(S_2)$ vrať $\text{Vor}(S)$.

Později si ukážeme, že krok (1) může být proveden v čase $O(N)$. Označíme-li $T(N)$ celkový čas potřebný pro běh algoritmu, pak krok (2) je

proveden přibližně v čase $2T(N/2)$. $\text{Vor}(S_1)$ a $\text{Vor}(S_2)$ mohou být spojeny do $\text{Vor}(S)$ v lineárním čase. Potom optimální algoritmus má složitost $O(N \log N)$.

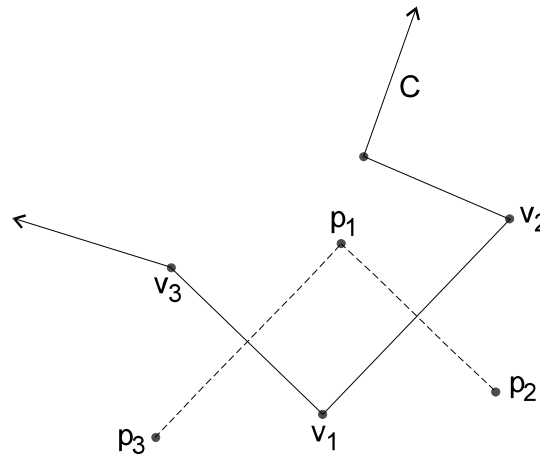
Definice: Necht' $\{S_1, S_2\}$ je dělení množiny S . Pak $\sigma(S_1, S_2)$ značí množinu Voronoiových hran, které jsou společné pro pár Voronoiových polygonů $V(i)$ a $V(j)$, kde $p_i \in S_1$ a $p_j \in S_2$.

Tvrzení: $\sigma(S_1, S_2)$ je množina hran *podgrafu* Voronoiova diagramu $\text{Vor}(S)$ o těchto vlastnostech:

- (i) $\sigma(S_1, S_2)$ se skládá z hranově disjunktích cyklů a řetězců. Jestli-že řetězec obsahuje právě jednu hranu, pak je touto hranou přímka. V opačném případě jsou jeho dvě extrémní hrany (počáteční a koncová) polopřímkami.
- (ii) Jsou-li S_1 a S_2 odděleny lineárně (tj. náleží-li dělicímu řetězci více než jeden bod, pak všechny tyto body leží v jednom dělení), pak je $\sigma(S_1, S_2)$ jednoduchý monotóní řetězec.

Důkaz:

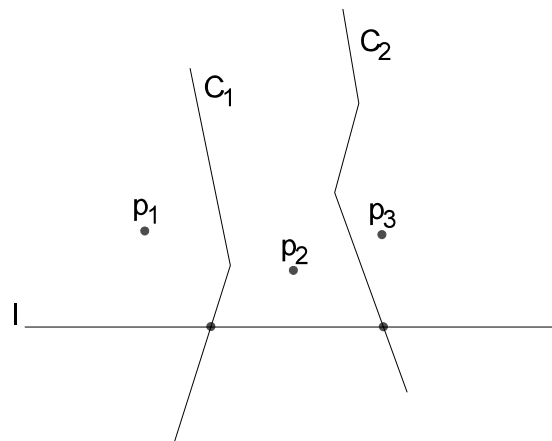
- (i) Představme si, že ve Voronoiově diagramu $\text{Vor}(S)$ obarvíme všechny polygony $\{V(i): p_i \in S_1\}$ červeně a všechny polygony $\{V(j): p_j \in S_2\}$ zeleně. Dostaneme tak dvoubarevnou diagram. Je na první pohled vidět, že hranice mezi polygony různých barev jsou hranově disjunktí cykly a řetězce. Poznamenejme, že dvě části řetězce $\sigma(S_1, S_2)$ mohou sdílet vrchol pouze v případě, že tento vrchol je nejméně čtvrtého stupně. Protože však všechny vrcholy Voronoiova diagramu jsou třetího stupně (viz. výše), tvrdíme, že tyto části jsou i vrcholově disjunktí. Každá část řetězce $\sigma(S_1, S_2)$ rozděluje rovinu na dvě oblasti. Řetězec se tedy skládá buď ze samotné přímky, nebo obsahuje polopřímky tvořící počáteční a koncovou hranu.
- (ii) Předpokládejme, že S_1 a S_2 jsou odděleny vertikální čarou m a C je část řetězce $\sigma(S_1, S_2)$. Vycházejte z bodu q , začneme traverzovat (pohybovat se křížením směrů) po C ve směru klesající souřadnice y , a tak budeme pokračovat, dokud nenačítáme na vrchol v_1 , ve kterém C začne stoupat vzhůru, jak ukazuje obrázek 1.



Obr. 1: K důkazu, že $\sigma(S_1, S_2)$ jednoduchý monotóní řetězec.

Hrana v_1v_2 je kolmou osou p_1p_2 a hrana v_1v_3 je kolmou osou p_1p_3 . Protože $y(v_3) > y(v_1)$ a $y(v_2) > y(v_1)$ platí, že $x(p_3) \leq x(p_1) \leq x(p_2)$. Avšak, podle tvaru C , p_2 a p_3 náležejí stejné množině, což je v rozporu s předpokladem, že S_1 a S_2 jsou odděleny vertikální čarou. To znamená že vrchol v_1 nemůže existovat a řetězec C je vertikálně monotóní.

Nyní předpokládejme, že $\sigma(S_1, S_2)$ obsahuje nejméně dva (vertikálně monotóní) řetězce C_1 a C_2 . Horizontální přímka l protíná každý z řetězců C_1 a C_2 v jednom bodě. Průsečík této přímky s C_1 leží díky monotónosti nalevo od průsečíku s C_2 , viz. obrázek 2.



Obr. 2: K důkazu, že $\sigma(S_1, S_2)$ se skládá z jednoho monotóního řetězce.

Pak existují tři body p_1, p_2 a p_3 takové, že $x(p_1) < x(p_2) < x(p_3)$, a p_1 a p_3 náležejí stejné množině dělení $\{S_1, S_2\}$, což je v rozporu s předpokladem, že S_1 a S_2 jsou odděleny vertikálně. To znamená, že $\sigma(S_1, S_2)$ je tvořen jediným monotóním řetězcem.

Jsou-li S_1 a S_2 odděleny lineárně, můžeme místo o řetězci $\sigma(S_1, S_2)$ mluvit pouze o řetězci σ . Je-li dělicí čára m vertikální, můžeme jednoznačně říci, že σ dělí rovinu na *levou* část π_L a *pravou* část π_R .

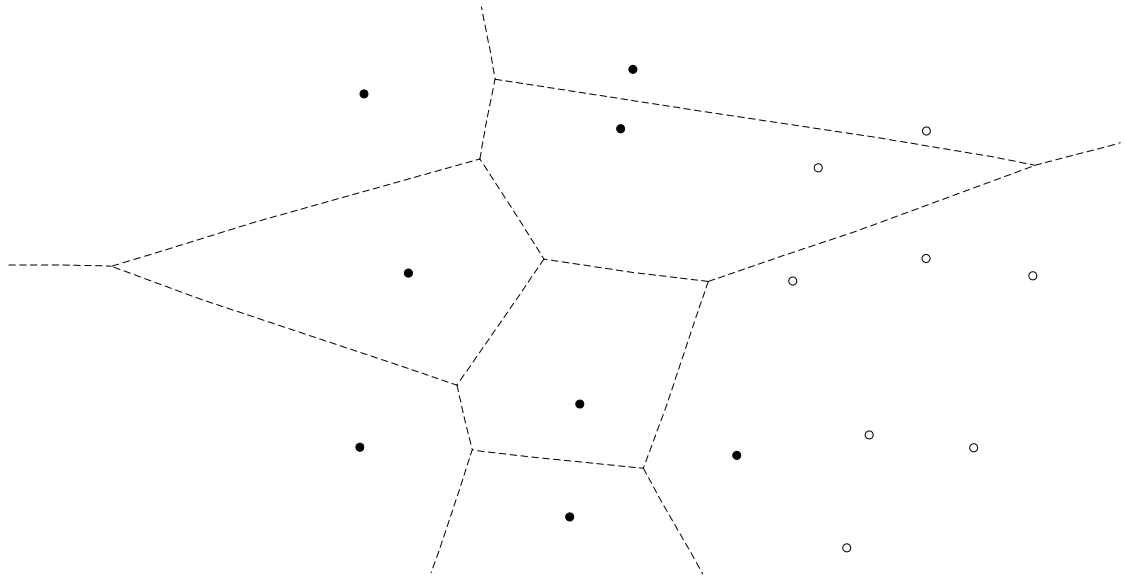
↪ **Tvrzení:** Jsou-li S_1 a S_2 lineárně odděleny vertikální čarou tak, že S_1 je nalevo od S_2 , pak Voronoiův diagram $\text{Vor}(S)$ je sjednocením $\text{Vor}(S_1) \cap \pi_L$ a $\text{Vor}(S_2) \cap \pi_R$.

Důkaz: Všechny body množiny S vpravo od σ jsou body množiny S_2 . Pak všechny hrany Voronoiova diagramu $\text{Vor}(S)$ vpravo od σ oddělují Voronoiovy polygony $V(i)$ a $V(j)$, kde jak p_i , tak p_j náleží množině S_2 . To znamená, že každá hrana z $\text{Vor}(S)$ v π_R se buď shoduje, nebo je částí nějaké hrany z $\text{Vor}(S_2)$. Analogicky všechny body množiny S vlevo od σ jsou body množiny S_1 a všechny hrany Voronoiova diagramu $\text{Vor}(S)$ vlevo od σ oddělují Voronoiovy polygony $V(i)$ a $V(j)$, kde jak p_i , tak p_j náleží množině S_1 . Z toho vyplývá, že každá hrana z $\text{Vor}(S)$ v π_L se buď shoduje, nebo je částí nějaké hrany z $\text{Vor}(S_1)$.

Nyní můžeme přistoupit k úpravám našeho hrubého algoritmu pro konstrukci Voronoiova diagramu metodou rozděl-a-panuj. Algoritmus je doplněn takto:

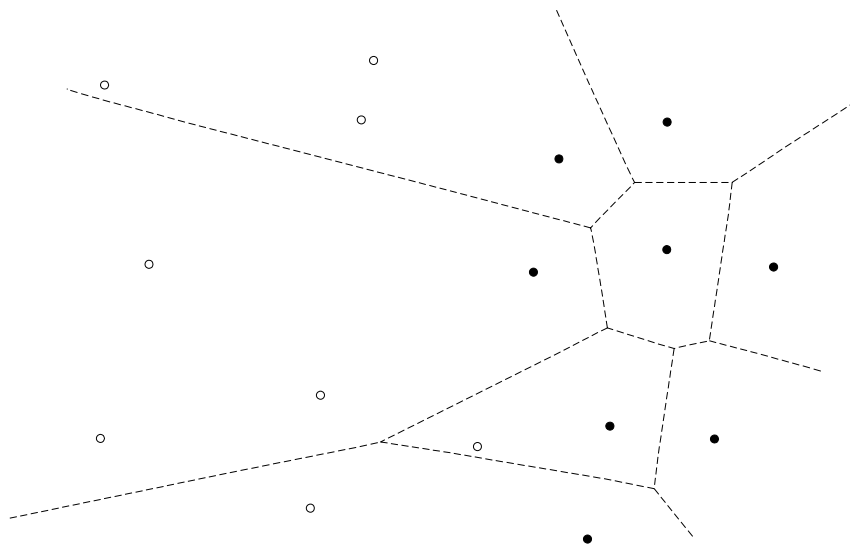
procedure VORONOI_DIAGRAM

- krok (1) Rozděl S na dvě podmnožiny S_1 a S_2 přibližně stejně velké podle x -ové souřadnice.
- krok (2) Sestroj rekurzivně $\text{Vor}(S_1)$ a $\text{Vor}(S_2)$.
- krok (1a) Sestroj polygonální řetězec σ oddělující S_1 a S_2
- krok (3b) Zruš všechny hrany $\text{Vor}(S_2)$ ležící vlevo od σ a stejně tak všechny hrany $\text{Vor}(S_1)$ ležící vpravo od σ . Výsledkem je $\text{Vor}(S)$, tedy Voronoiův diagram va množině bodů S .

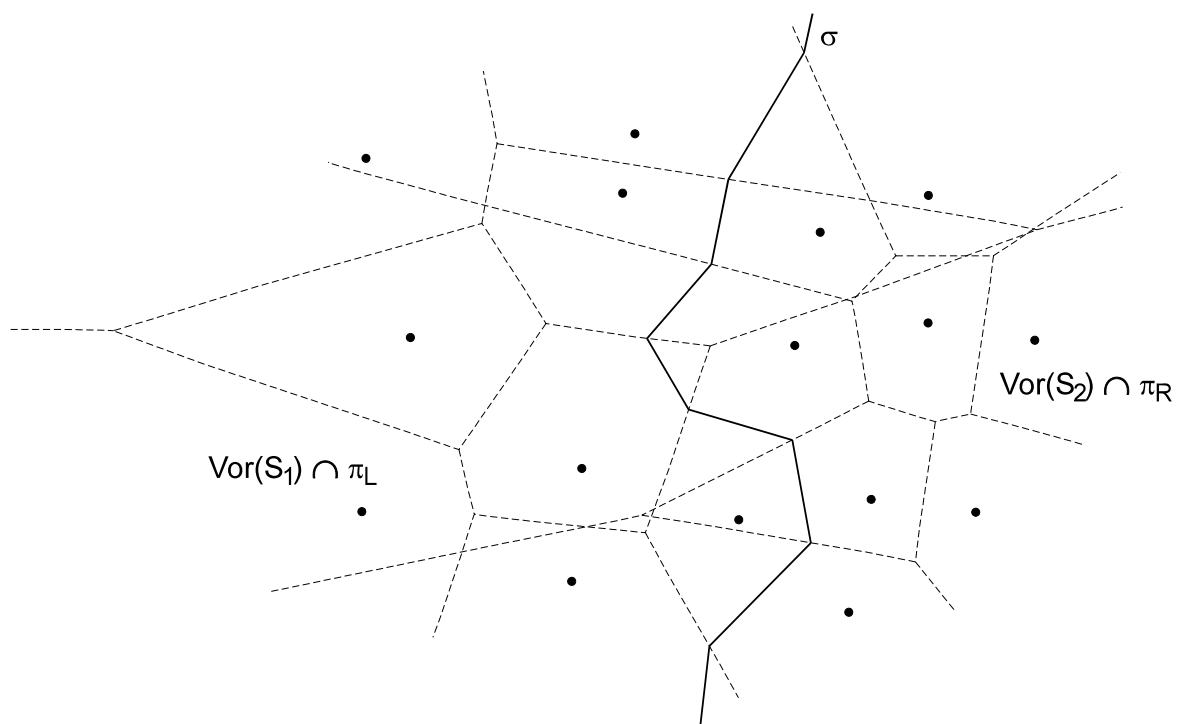


Obr. 3: Voronoiův diagram levé části množiny S , tedy $\text{Vor}(S_1)$.

Úspěšnost této procedury závisí na tom, jak rychle jsme schopni sestrojít řetězec σ , neboť krok (3b) již problém nepředstavuje. Na obrázku výše, je ukázán příklad $\text{Vor}(S_1)$ a na následujících obrázcích pak příklad $\text{Vor}(S_2)$ a výsledný $\text{Vor}(S)$.



Obr. 4: Voronoiův diagram pravé části množiny S , tedy $\text{Vor}(S_2)$.



Obr. 5: Sloučení $\text{Vor}(S_1)$ a $\text{Vor}(S_2)$ pomocí řetězce σ .

Inicializační dělení množiny S v mediánu x -ových souřadnic může být provedeno v čase $O(N)$ standardním algoritmem pro určení mediánu.

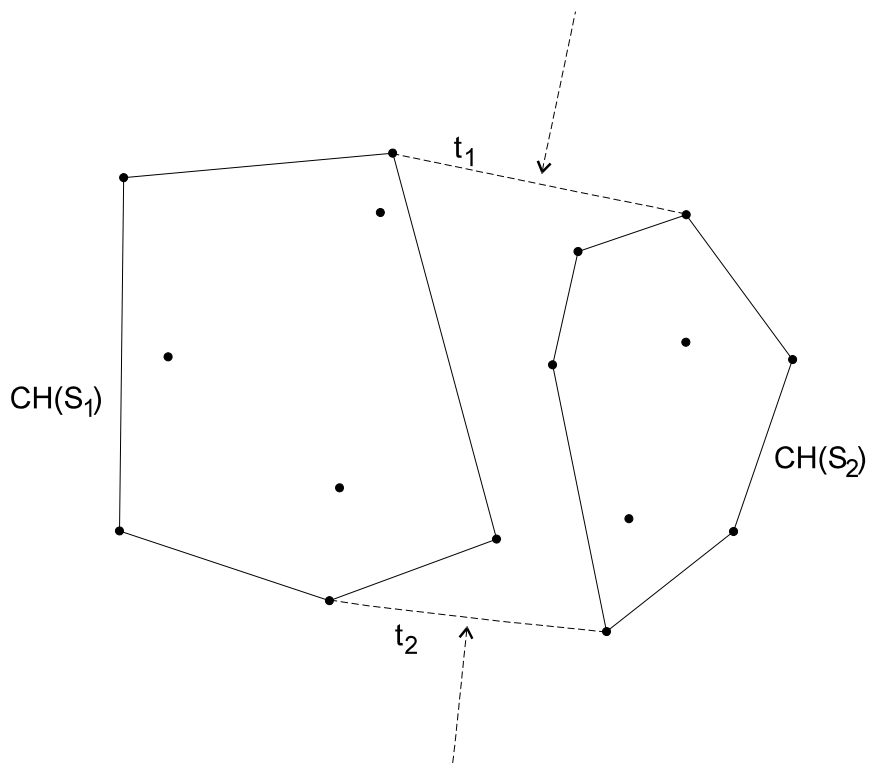
(\approx Poznámka: Medián lze snadno určit tak, že nejdříve pole setřídíme a pak vybereme jeho prostřední prvek. Je však zřejmé, že tento způsob nebude nejoptimálnější. Mnohem lepším způsobem je např. metoda dělení posloupnosti na úseky.)

Krok (3b) může být proveden v čase $O(|S_1| + |S_2|) = O(N)$.

To znamená, že nyní zbývá nalézt vhodný, dostatečně výkonný způsob konstrukce řetězce σ .

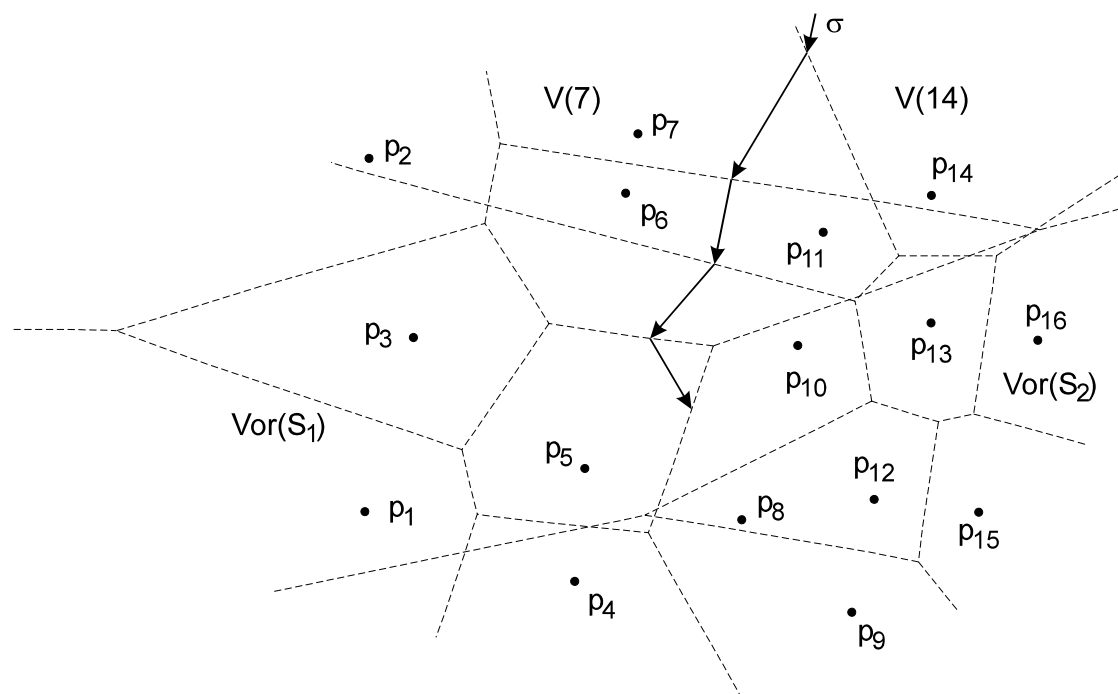
3.4.1 Konstrukce řetězce σ

Prvním krokem při konstrukci řetězce σ je nalezení jeho polopřímkových koncových (resp. počáteční a koncová) hran (*paprsků*). Všimněme si, že každý takový paprsek řetězce σ je kolmou osou pomocného segmentu mezi $\text{CH}(S_1)$ a $\text{CH}(S_2)$. Jelikož S_1 a S_2 jsou lineárně odděleny, vznikají právě dvě pomocné polygonové oblasti $\text{CH}(S_1)$ a $\text{CH}(S_2)$. Existují-li konvexní obálky těchto dvou oblastí, pak výše zmíněné pomocné segmenty, označme je t_1 a t_2 , jsou sestrojitelné nejhůře v lineárním čase a naše paprsky řetězce σ mohou být rychle nalezeny, jak ukazuje obrázek 6.



Obr. 6: Hledání paprsků řetězce σ .

Známe-li paprsky řetězce σ , pokračujeme v konstrukci hranou po hraně, dokud nenarazíme na koncový paprsek. Tuto konstrukci přibližuje následující obrázek 7.



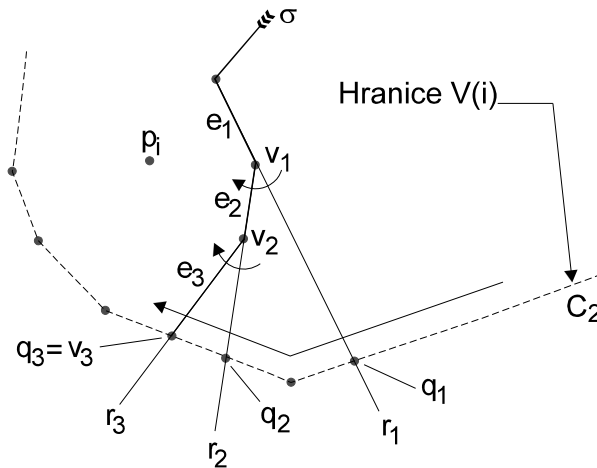
Obr. 7: Konstrukce řetězce σ cik-cak pochodem.

Horní paprsek (vstupní hrana) je kolmá osa bodů p_7 a p_{14} . Představme si bod z pohybující se po paprsku z nekonečna směrem dolů. Nejdříve bod z leží v nějakém polygonu jak diagramu $\text{Vor}(S_1)$, tak i $\text{Vor}(S_2)$ (v našem případě ve $V(7)$ a $V(14)$). Pokračuje, dokud nepřekříží hranu jednoho z těchto polygonů, kdy změní směr pohybu. V našem případě bod z nejdříve narazí na hranu diagramu $\text{Vor}(S_2)$. To znamená, že nyní je bod z blíže k vrcholu p_{11} než k vrcholu p_{14} , tedy dál se bude pohybovat po ose kolmé na p_7p_{11} . Tak bude pokračovat, dokud nenarazí na osu kolmou na p_6p_7 Voronoiova diagramu $\text{Vor}(S_1)$ a dál bude pohybovat po ose kolmé na p_6p_{11} . Takovým způsobem pokračuje bod z v traverzování dokud nenarazí na dolní (výstupní) paprsek řetězce σ .

Za předpokladu, že řetězec σ je traverzován ve směru rostoucí y -ové souřadnice, si ukážeme lepší mechanismus tohoto pochodu, který k stávající hraně e a stávajícímu vrcholu v (v je horní koncový vrchol hrany e) najde následující hranu e' a vrchol v' . Odděluje-li hrana e polygony $V(i)$ a $V(j)$, pro $p_i \in S_1$ a $p_j \in S_2$, pak v' bude buď průsečíkem hrany e a hranice polygonu $V(i)$ v diagramu $\text{Vor}(S_1)$, nebo průsečíkem hrany e a hranice polygonu $V(j)$ v diagramu $\text{Vor}(S_2)$, podle toho, který je k v blíže. Takže procházením hranic $V(i)$ ve $\text{Vor}(S_1)$ a $V(j)$ ve $\text{Vor}(S_2)$ můžeme snadno určit vrchol v' . Naneštěstí, řetězec σ může zůstat uvnitř daného polygonu $V(i)$ po několikerém otočení před tím, než protne některou z jeho hran. Kdybychom však měly pokaždé prohlížet všechny polygony $V(i)$, museli bychom provést příliš velké množství

porovnání (dostáváme se ke kvadratickému času). Naštěstí struktura Voronoiova diagramu skýtá mnohem efektivnější postup.

Předpokládejme, že řetězec σ se skládá z posloupnosti více než jedné hrany e_1, e_2, \dots, e_k polygonu $V(i)$, kde $p_i \in S_1$ (viz. obrázek, kde $k = 3$).



Obr. 8: Průsečíky q_1, q_2 a q_3 jsou seřazeny na C_2 .

Protáhněme hrany e_h směrem od vrcholu v_h do nekonečna. Vznikne tak polopřímka (paprsek) r_h , která protíná hranici polygonu $V(i)$ v diagramu $\text{Vor}(S_1)$ v bodě q_h . Dále označme část řetězce σ tvořenou posloupností hran e_1, e_2, \dots, e_k jako C_1 a část hranice polygonu $V(i)$ vpravo od řetězce σ jako C_2 . Podřetězce C_1 i C_2 jsou konvexní (C_2 je částí konvexního polygonu $V(i)$, třeba i neuzavřeného). Jelikož úhly z $v_h q_h$ do $v_h q_{h+1}$ jsou stejného znaménka, průsečíky q_1, q_2, \dots, q_k jsou na C_2 seřazeny. To znamená, že C_2 , tedy část hranice polygonu $V(i)$ patřící diagramu $\text{Vor}(S_1)$, je třeba procházet ve směru chodu hodinových ručiček, bez vracení se zpět. Stejně tak lze ukázat, že hranici polygonu $V(j)$ v diagramu $\text{Vor}(S_2)$ je třeba procházet naopak proti směru chodu hodinových ručiček, také bez vracení se zpět.

Přesněji, předpokládáme, že oba diagramy $\text{Vor}(S_1)$ i $\text{Vor}(S_2)$ jsou dány strukturou DCEL, kde (viz. kapitola DCEL) hranice oblastí ve $\text{Vor}(S_1)$ jsou orientovány po směru chodu hodinových ručiček a hranice oblastí ve $\text{Vor}(S_2)$ jsou orientovány proti směru chodu hodinových ručiček. DCEL diagramu $\text{Vor}(S_1)$ obsahuje ukazatel $\text{NEXT}_1[\]$ určený pro procházení hranice oblasti ve směru chodu hodinových ručiček a podobně DCEL diagramu $\text{Vor}(S_2)$ obsahuje ukazatel $\text{NEXT}_2[\]$ určený pro procházení hranice oblasti proti směru chodu hodinových ručiček. V algoritmu jsou udržovány tři hrany: hrana e_L ve $\text{Vor}(S_1)$, hrana e_R ve $\text{Vor}(S_2)$ a „aktivní hrana“ e řetězce σ (ve skutečnosti přímka, na které leží hrana e). Pro hranu e dále udržuje její počáteční vrchol v (vrchol v^* na počátečním paprsku e^* vhodně zvolíme). Nakonec udržuje dva body z množiny S : bod $p_L \in S_1$ a bod $p_R \in S_2$, pro které platí, že aktivní hrana e leží na ose půlící úsečku $p_L p_R$. Průsečík hran e a e' označíme jako $I(e, e')$.

Výraz $I(e, e') = \Lambda$ bude znamenat, že se e a e' neprotínají. Proměnnými t_1 a t_2 označíme, již výše použité a vysvětlené, pomocné hrany, přičemž $t_1 = pq$. Implementace kroku (3a) bude tedy vypadat následovně:

```

begin
1.   $p_L := p$ ;
2.   $p_R := q$ ;
3.   $e := e*$ ; (* Znak * je použito jako indexu *)
4.   $v := v*$ ;
5.   $e_L :=$  první hrana (otevřené) hranice polygonu  $V(p_L)$ ;
6.   $e_R :=$  první hrana (otevřené) hranice polygonu  $V(p_R)$ ;
7.  repeat
8.    while ( $I(e, e_L) = \Lambda$ ) do  $e_L := \text{NEXT}_1[e_L]$ ;
      (*Průchod hranicí polygonu  $V(p_L)$ *)
9.    while ( $I(e, e_R) = \Lambda$ ) do  $e_R := \text{NEXT}_2[e_R]$ ;
      (*Průchod hranicí polygonu  $V(p_R)$ *)
10.   if ( $I(e, e_L)$  je blíže k  $v$  než  $I(e, e_R)$ ) then
      begin
11.      $v := I(e, e_L)$ ;
12.      $p_L :=$  bod z  $S$  ležící na opačné straně hrany  $e_L$ ;
13.      $e :=$  osa půlící hranu  $p_L p_R$ ;
14.      $e_L :=$  opačná hrana k  $e_L$ 
      (*hrana z  $V(p_L)$ , ale opačným směrem než stará  $e_L$ *)
      end else
      begin
15.      $v := I(e, e_R)$ ;
16.      $p_R :=$  bod z  $S$  ležící na opačné straně hrany  $e_R$ ;
17.      $e :=$  osa půlící hranu  $p_L p_R$ ;
18.      $e_R :=$  opačná hrana k  $e_R$ 
      (*hrana z  $V(p_R)$ , ale opačným směrem než stará  $e_R$ *)
      end
19.   until ( $p_L p_R = t_2$ )
end.

```

Nyní několik slov k algoritmu. Na řádcích 1 až 6 probíhá inicializace, po které, na řádcích 7 až 19, prochází řetězec σ . Samotné procházení obálek polygonů $V(p_L)$ a $V(p_R)$ bez vracení se, jak jsme si přiblížili výše, se realizuje řádky 8 a 9. Řádky 11 až 14 a 15 až 18 aktualizují vztahy mezi parametry $\{v, p_L, p_R, e_L, e_R\}$ měnící se během průchodu, a to v konstantním čase. Jelikož $\text{Vor}(S_1)$ dohromady s $\text{Vor}(S_2)$ neobsahují více než $3N - 6$ hran a ne více než $O(N)$ vrcholů v řetězci σ , pak složitost celé konstrukce řetězce σ může být pouze lineární funkcí.

Nyní si připomeňme, že při konstrukci Voronoiova diagramu musíme zrušit všechny hrany $\text{Vor}(S_1)$, které leží vpravo od řetězce σ a stejně tak všechny hrany $\text{Vor}(S_2)$, které leží vlevo od řetězce σ . To učiníme při průchodu po či proti směru chodu hodinových ručiček po obvodu polygonů $V(p_L)$ a $V(p_R)$ během konstrukce σ . Proces spojování $\text{Vor}(S_1)$ s $\text{Vor}(S_2)$ do $\text{Vor}(S)$

tedy proběhne v lineárním čase. Tuto kapitolu nyní uzavřeme následujícím tvrzením:

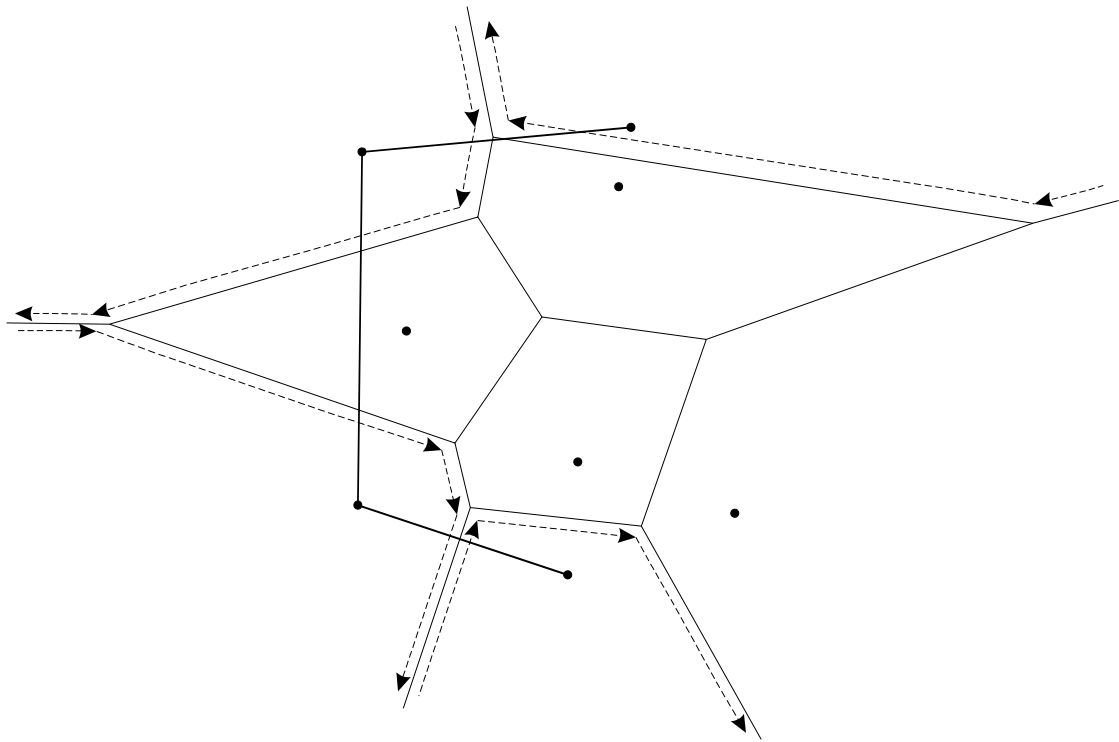
☞ **Tvrzení:** Konstrukce Voronoiova diagramu sestrojeného na množině N bodů v rovině je výpočtové složitosti $O(N \log N)$.

Důkaz: Čas, závislý na proceduře pro rekurzivní spojování dílčích Voronoiových diagramů, je popsateľným opakováním vztahu $T(N) = 2T(N/2) + O(N) = O(N \log N)$ (kde $T(N)$ je celkový čas potřebný pro běh algoritmu, $2T(N/2)$ je čas potřebný pro rekurzivní konstrukci $\text{Vor}(S_1)$ a $\text{Vor}(S_2)$ a konečně $O(N)$ vystihuje lineární čas pro spojení $\text{Vor}(S_1)$ s $\text{Vor}(S_2)$ do $\text{Vor}(S)$).

☞ **Tvrzení:** Triangulace duální k Voronoiovu diagramu je taková triangulace, že uvnitř kružnice opsané každému jejímu trojúhelníku se nenachází jiný bod množiny S , tj. je to Delaunayova triangulace.

Důkaz: Důkaz vychází ze samotné podstaty Voronoiova diagramu, jehož každý vrchol v (střed kružnice opsané trojúhelníku $p_i p_j p_k$) je sdílen právě třemi Voronoiovými polygony $V(i)$, $V(j)$ a $V(k)$, přičemž polygon $V(i)$ neobsahuje žádný jiný bod než p_i a stejně tak polygon $V(j)$ (resp. $V(k)$) neobsahuje žádný jiný bod než p_j (resp. p_k).

☞ **Poznámka:** Jak už bylo jednou řečeno, triangularizace je jen jednou z mnoha úloh řešitelných prostřednictvím Voronoiova diagramu. Další takovou úlohou je např. hledání množiny nejbližších sousedů daného bodu. Pro bod p_i jsou prvky této množiny body p_1, p_2, \dots, p_k , pro něž platí, že jejich Voronoiovy polygony $V(1), V(2), \dots, V(k)$ jsou sousedy polygonu $V(i)$. Stejně snadno lze prostřednictvím Voronoiova diagramu řešit úlohu konstrukce konvexní obálky množiny bodů. V tomto případě využijeme pravidla, že každý paprsek opouštějící Voronoioův diagram je osou půlicí právě jednu hranu patřící konvexní obálce, jak ukazuje poslední obrázek 9 této kapitoly.



Obr. 9: Konstrukce konvexní obálky množiny bodů prostřednictvím Voronoiových paprsků

3.5 Přírůstková konstrukce Voronoiova diagramu

Přírůstkový typ algoritmu pro konstrukci Voronoiova diagramu začíná s jednoduchým Voronoiovým diagramem s dvěma či třema generátory (body vstupní množiny S) a ten pak krok za krokem modifikuje přidáváním v každém kroku po jednom novém generátoru.

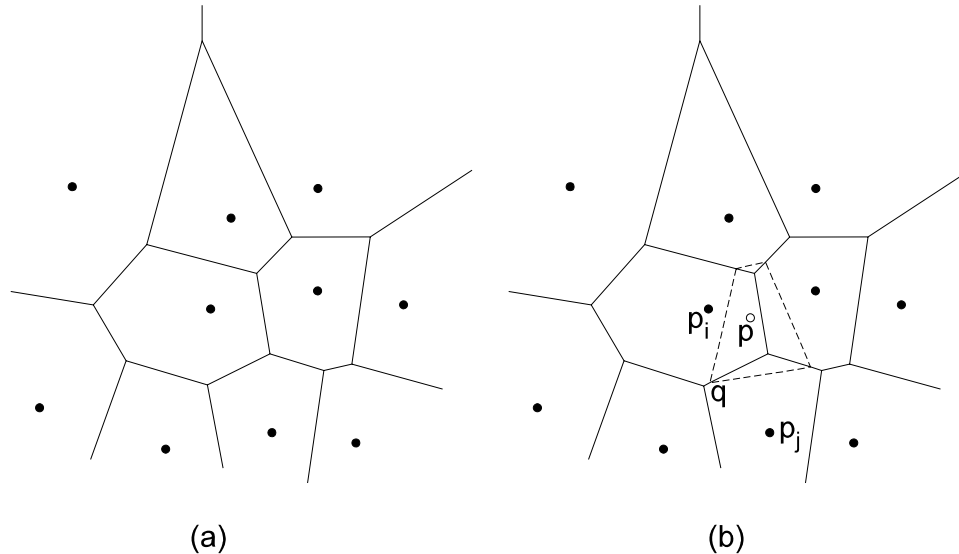
Základní princip algoritmu ilustruje obrázek 1, kde je na obr. 1(a) zobrazen daný Voronoiov diagram a na obr. 1(b) je označen prázdným kroužkem nový generátor p přidávaný do diagramu.

Voronoiov polygon tohoto nového generátoru p sestrojíme takto:

- (1) Lokalizujeme Voronoiov polygon $V(i)$, do kterého generátor p padl.
- (2) Sestrojíme osu půlící úsečku pp_i , a ta protne polygon $V(i)$ ve dvou bodech (body ležící v průniku vnitřní oblasti polygonu $V(i)$ a přímky procházející těmito dvěma průsečíky jsou stejně vzdálené od obou generátorů p i p_i , a navíc jsou k těmto dvěma generátorům blíže, než ke kterémukoli jinému v diagramu). Vezmeme jeden z těchto dvou průsečíků (označme q) a přejdeme jeho směrem do polygonu $V(j)$. Sestrojíme osu půlící úsečku pp_j , a ta protne polygon $V(j)$ ve dvou bodech. Jedním z nich je bod q .

Vydáme se směrem druhého průsečíku a tento proces opakujeme, až narazíme znovu na polygon $V(i)$, tj. Voronoiov polygon V generátoru p se uzavře.

- (3) Z diagramu vypustíme všechny vrcholy, hrany a části hran, které leží uvnitř polygonu V a aktualizovaný Voronoiov diagram doplníme o nové Voronoiovy vrcholy (vrcholy polygonu V) a nové Voronoiovy hrany (hrany tvořící obálku polygonu V - na obr. vyznačeny čárkovaně).



Obr. 1: (a) Daný Voronoiov diagram. (b) Přidání nového generátoru p .

Tuto proceduru, tj. kroky (1) až (3), opakujeme pro každý bod množiny S vkládaný do diagramu. Vhodnou úpravou, kterou se vyhneme řešení některých nepříjemných speciálních případů (rovnoběžné Voronoiovy paprsky) a která zároveň eliminuje počet Voronoiových paprsků na tři, je přidání tří pomocných generátorů takových, že všechny ostatní body množiny S budou ležet uvnitř trojúhelníka tvořeného těmito generátory.

Celková průměrná výpočtová složitost algoritmu je lineárně rostoucí s počtem generátorů vstupní množiny, tedy $O(N)$, není ovšem optimální, neboť nejhorší výpočtová složitost algoritmu je $O(N^2)$.

3.6 Příkladová konstrukce Voronoiova diagramu s důrazem na topologickou stabilitu

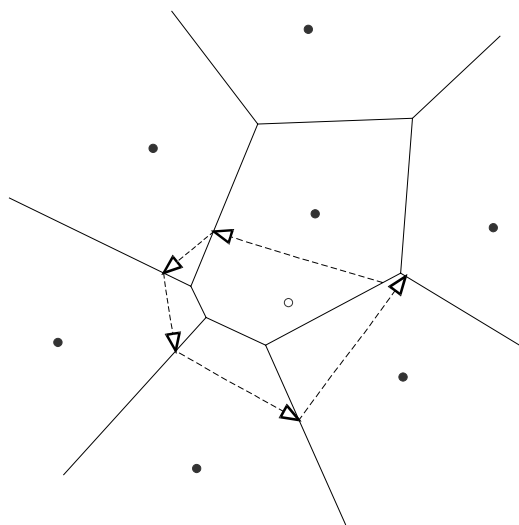
Většina algoritmů je navržena za předpokladu, že během výpočtu nebude docházet k numerickým chybám. V praxi se těmito chybám při použití běžné aritmetiky s plovoucí desetinou tečkou zcela nevyhneme (je například velmi obtížné korektně posoudit, jestli bod leží uvnitř, vně nebo přímo na kružnici). Geometrické nepřesnosti se často promítanou na topologických

chybách a tak se může stát, že teoreticky správný algoritmus generuje chybné výsledky. Takto vzniká hluboká propast mezi teoreticky správnými algoritmy a v praxi správně fungujícími programy. V této kapitole si představíme teoreticky správný algoritmus, který dává korektní výsledky i v praxi a vyplňuje tedy tuto propast.

Základní myšlenka vychází z předpokladu, že numerických chyb během výpočtu stále přibývá, takže další rozhodnutí založená na numerických výpočtech jsou nespolehlivá. Za tohoto předpokladu je očividně nemožné Voronoiův diagram sestavit pokaždé správně. Proto pozměníme cíl našeho snažení a pokusíme se sestavit diagram, jehož některé charakteristiky budou z topologického hlediska stejné jako charakteristiky ideálního Voronoiova diagramu. Topologická správnost pro nás bude tedy důležitější, než numerické výsledky. To ale neznamená, že numerické výpočty nejsou důležité. Naši snahou bude vyladit cestu numerických výpočtů s ohledem na topologické vlastnosti.

Základním kamenem pro další úvahy je přírůstková metoda konstrukce Voronoiova diagramu popsaná v předcházející kapitole (proto ji nyní nebudeme popisovat, abychom se zbytečně neopakovali). Tato metoda je principiálně velmi jednoduchá a dá se říct, že je i poměrně dost odolná proti numerickým chybám (proto jsou také programy založené na přírůstkové metodě často užívány v mnoha aplikacích).

Nicméně i algoritmy založené na přírůstkové metodě, stejně jako mnohé jiné, si těžko poradí s některými degeneracemi během výpočtu. Příkladem je následující obrázek, na kterém se po proběhnutí procedury konstrukce Voronoiova polygonu přírůstkovou metodou díky numerické chybě tento polygon neuzavře.



Obr. 1: Narušení topologie způsobené numerickými chybami při přírůstkové metodě konstrukce Voronoiova diagramu.

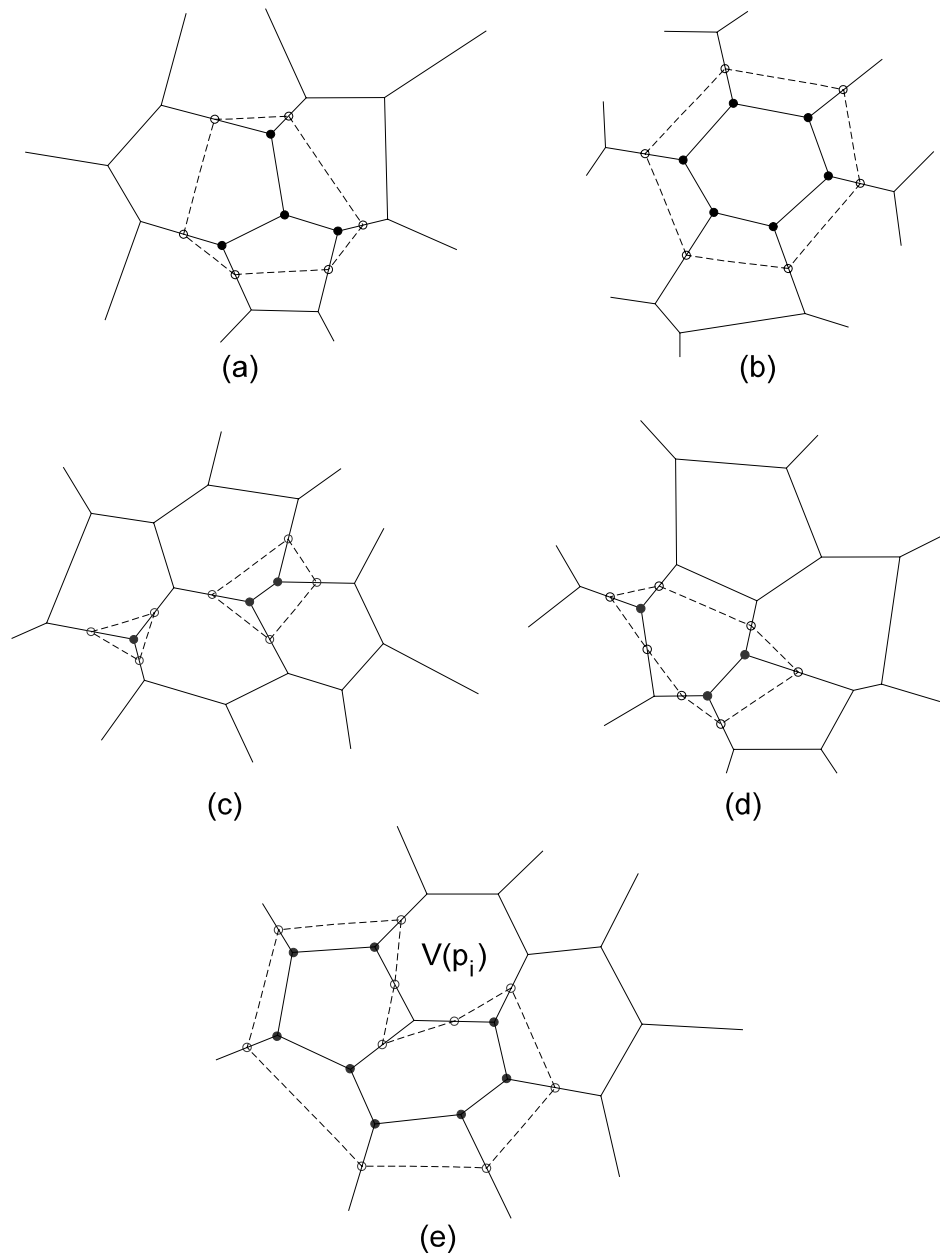
3.6.1 Přístup s důrazem na topologickou stabilitu

Voronoiův diagram je vlastně planární graf. Označme jako G_i graf sdružený s Voronoiovým diagramem sestrojeným na i generátorech p_1, p_2, \dots, p_i . Z topologického hlediska můžeme přidání nového generátoru p_l do Voronoiova diagramu sestrojeném na $l - 1$ generátorech p_1, p_2, \dots, p_{l-1} považovat za přechod od G_{l-1} k G_l . Tento přechod řeší následující procedura [SUGI92]:

procedure A

- A1. Vyber podmnožinu T množiny vrcholů grafu G_{l-1} .
- A2. Pro každou hranu spojující vrchol z T s vrcholem nepatřícím do T generuj nový vrchol, který tuto hranu rozdělí na hrany dvě.
- A3. Sestroj nové hrany spojující vrcholy generované krokem A2 tak, že tvoří cyklus uzavírající vrcholy z T .
- A4. Odstraň z T všechny vrcholy a s nimi spjaté hrany (vnitřní oblast cyklu považuj za Voronoiův polygon nového generátoru p_l) a označ výsledný graf za G_l .

Příklad chování této procedury je ukázán na obrázku 2, sledujte např. část (a). Plnými čarami je znázorněn graf G_{l-1} a čtyři plné kroužky reprezentují vrcholy vybrané krokem A1. Potom je krokem A2 generováno šest vrcholů znázorněných prázdnými kroužky, mezi kterými jsou krokem A3 generovány nové hrany (čárkovaná čára) tvořící „nový“ Voronoiův diagram. Podgraf ohraničený tímto polygonem je potom krokem A4 zrušen.

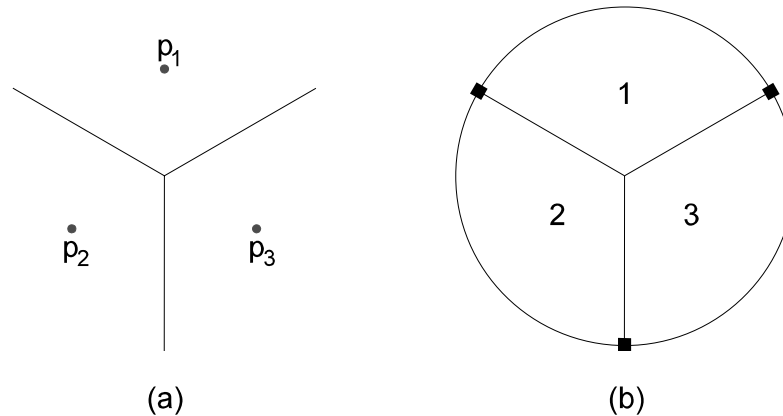


Obr. 2: Topologický aspekt přidání nového generátoru.

Nyní se budeme zabývat volbou množiny T , na které záleží korektnost celé procedury konstrukce Voronoiova diagramu. V prvním kroku sestrojíme pomocný trojúhelník tvořený třemi pomocnými generátory takový, že všechny body množiny generátorů leží uvnitř tohoto trojúhelníka. Dále přechíslijeme generátory tak, že p_1, p_2 a p_3 jsou pomocné generátory a p_4, p_5, \dots, p_n je posloupnost původních generátorů, kde $n = \text{původní počet} + 3$. Nyní se pokusíme sestřit Voronoiův diagram na množině bodů $S = \{p_1, p_2, \dots, p_n\}$.

Voronoiův diagram pomocných generátorů p_1, p_2 a p_3 je tvořen třemi polopřímkami (paprsky), viz. následující obrázek 3(a). Pro reprezentaci

topologické struktury tohoto diagramu použijeme planární graf G_3 , na obr. 3(b), ve kterém zavedeme dostatečně velkou uzavřenou křivku, jejíž průsečíky s Voroniovými paprsky (malé čtverečky) definují koncové body těchto paprsků.



Obr. 3: (a) Voroniov diagram pro tři generátory a (b) jeho graf.

Konstrukci začneme s G_3 , k němuž bod po bodu přidáváme generátory p_4, p_5, \dots, p_n . Díky tomu, že pomocné generátory p_1, p_2 a p_3 jsou jedinými vrcholy konvexní obálky množiny generátorů, je Voroniov polygon právě přidaného generátoru vždy uzavřený, tj. nikdy se neregenerují nové Voroniov paprsky (připomeňme, že Voroniov polygon $V(i)$ je neuzavřený pouze tehdy, patří-li vrchol p_i konvexní obálce vstupní množiny).

Nyní označme jako $G_{l-1}(T)$ podgraf grafu G_{l-1} , kde T je podmnožina množiny vrcholů grafu G_{l-1} . Tento podgraf $G_{l-1}(T)$ tedy obsahuje vrcholy z T a hrany spojující vždy dva tyto vrcholy. Podgraf $G_{l-1}(T)$ je vlastně také planárním grafem. Řekneme, že cyklus C grafu G_{l-1} je *primární*, platí-li, že C neohraničuje žádný vrchol či hranu (primární cyklus je vlastně Voroniov polygon). Průnikem $C \cap G_{l-1}(T)$ je podgraf grafu G_{l-1} sestavený z vrcholů a hran patřících jak cyklu C , tak i grafu $G_{l-1}(T)$.

Pro korektní funkci procedury A musí množina T splňovat následující podmínky:

- (C1) Množina T není prázdná.
- (C2) Graf $G_{l-1}(T)$ je strom.
- (C3) Pro každý primární cyklus C grafu G_{l-1} platí, že průnik $C \cap G_{l-1}(T)$ je spojitý.

Důkaz:

- (C1) Důkaz tohoto bodu je zřejmý, neboť každý nový generátor p_i musí mít svůj vlastní Voroniov polygon.

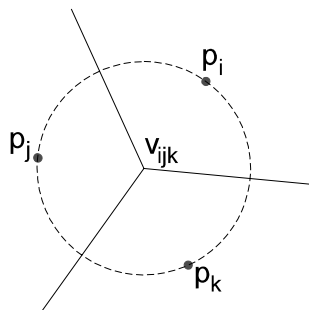
- (C2) Kdyby $G_{l-1}(T)$ obsahoval cyklus, pak by byl Voronoiův polygon původního generátoru vyjmut celý (obr. (e)), což není možné. Kdyby byl $G_{l-1}(T)$ nespojitý, pak by i nový Voronoiův polygon byl nespojitý, (obr. (c)), nebo by se původní Voronoiův polygon rozdělil (obr. (d)), což není možné. Proto $G_{l-1}(T)$ musí být strom.
- (C3) Je-li průnik $C \cap G_{l-1}(T)$ nespojitý (obr. (e), na kterém je C cyklus tvořící obálku polygonu $V(p_i)$), pak Voronoiovy polygony $V(p_i)$ a $V(p_l)$ sdílejí dvě nebo více hran, což je nemožné (z definice Voronoiova polygonu).

3.6.2 Použití numerických hodnot jako informace nižší priority

V této podkapitole budeme řešit otázku výběru prvků množiny T a teprve za tímto účelem použijeme numerických výpočtů.

Při konstrukci korektního Voronoiova diagramu, by se podmnožina generátorů T z procedury A skládala z vrcholů Voronoiova diagramu (sestrojeného na množině generátorů p_1, p_2, \dots, p_{l-1}) ležících uvnitř Voronoiova polygonu $V(p_l)$. Nedojde-li zde k numerické chybě, jsou tyto body nalezeny následujícím způsobem.

Nechť (x_i, y_i) jsou souřadnice generátoru p_i ($i = 1, 2, \dots, n$) a (x, y) jsou souřadnice libovolného bodu p v pravotočivém Kartézském systému souřadnic. Nechť dále (v souladu s dosavadními konvencemi) v_{ijk} označuje Voronoiův vrchol sdílený třemi Voronoiovými polygony $V(p_i)$, $V(p_j)$ a $V(p_k)$ v tomto pořadí, které odpovídá směru proti chodu hodinových ručiček (následující obr. 4).



Obr. 4: Voronoiův vrchol a „jeho“ tři generátory.

Pak v_{ijk} leží uvnitř Voronoiova polygonu nového generátoru p_l pouze v případě, že p_l leží uvnitř kružnice procházející vrcholy p_i, p_j a p_k .

Definujme $H(p_i, p_j, p_k, p)$ vztahem (1) [SUGI92]:

$$H(p_i, p_j, p_k, p) = \begin{vmatrix} 1 & x_i & y_i & (x_i^2 + y_i^2)/2 \\ 1 & x_j & y_j & (x_j^2 + y_j^2)/2 \\ 1 & x_k & y_k & (x_k^2 + y_k^2)/2 \\ 1 & x & y & (x^2 + y^2)/2 \end{vmatrix}$$

Pomocí tohoto vztahu můžeme určit polohu bodu p vzhledem ke kružnici procházející vrcholy p_i, p_j a p_k takto:

Bod p leží uvnitř této kružnice, platí-li nerovnost $H(p_i, p_j, p_k, p) < 0$ a naopak vně této kružnice leží, platí-li nerovnost $H(p_i, p_j, p_k, p) > 0$. (Rovnice $H(p_i, p_j, p_k, p) = 0$ s proměnnou p reprezentuje kružnici procházející trojicí bodů p_i, p_j a p_k .) Odtud by za předpokladu numerické správnosti výpočtů platilo, že v_{ijk} patří do množiny T v případě $H(p_i, p_j, p_k, p) < 0$ a naopak tento vrchol nepatří do množiny T když $H(p_i, p_j, p_k, p) > 0$. V případě, kdy $H(p_i, p_j, p_k, p) = 0$, je potřeba speciálního přístupu.

My však předpokládáme, že během výpočtu mohlo dojít k numerickým chybám a znaménko determinantu vždy nemusí být určeno správně. Proto při výběru prvků množiny T klademe větší důraz na podmínky (C1) až (C2). Numerických hodnot $H(p_i, p_j, p_k, p)$ použijeme pouze pro výběr nejslibnější množiny, množiny T , za předpokladu, že podmínky (C1), (C2) a (C3) jsou uspokojeny. Takovým způsobem pracuje následující procedura, která má za úkol najít při vkládání nového generátoru p_l podmnožinu T .

procedure B

- I. Najdi generátor p_i , který je nejbližší k bodu p_l , tj. najdi polygon $V(p_i)$, ve kterém se p_l nachází. Mezi Voronoiovými vrcholy v_{ijk} , které leží na hranici $V(p_i)$, vyber ten, pro který vychází $H(p_i, p_j, p_k, p)$ nejmenší. Množina T pak bude jednoprvková množina obsahující tento vrchol.
- II. Do chvíle, kdy již množinu T nebude možné rozšířit o další generátor, opakuj následující krok B.2.1.

Každý Voronoiov vrchol v_{ijk} spojený Voronoiovou hranou s prvkem množiny T přidej do T v případě, že $H(p_i, p_j, p_k, p) < 0$ a výsledná množina T uspokojuje podmínky (C2) a (C3).

To znamená, že nedošlo-li k degeneraci či k numerické chybě, procedura B vybere jako prvky množiny T Voronoiovy vrcholy, které by byly smazány během přidávání nového generátoru p_l . I když dojde k chybám v numerických výpočtech, množina T sestavená procedurou B uspokojuje podmínky (C1) až (C3), neboť alespoň jeden prvek je vybrán krokem B1 a platnost podmínek (C2) a (C3) je zajištěna, dokud je množina T krokem B2 rozšiřována.

Procedura B je volána v prvním kroku (A1) procedury A. To znamená, že pro danou množinu generátorů $S = \{p_1, p_2, \dots, p_n\}$ započneme konstrukci s grafem G_3 , z kterého pomocí procedury A (a v ní jako první krok použité

procedury B) sestrojíme grafy G_4, G_5, \dots, G_n . Bohužel velké numerické chyby mohou způsobit, že na výstupu je našim algoritmem vygenerován nějaký graf G_n , který „topologicky souhlasí“ v tom smyslu, že G_n je planárním grafem dělícím rovinu na n oblastí, přičemž žádné dvě z těchto oblastí nemají dvě nebo více hran společných.

Podmínky (C2) a (C3) mohou být ověřovány následujícím způsobem. Voronoiovy vrcholy označíme vždy jednou ze tří značek: „in“, „out“ a „?“ (nerozhodnuto). Značka „in“ označuje vrcholy přidané do množiny T , značka „out“ značí vrcholy, pro které bylo rozhodnuto, že do T přidány nebudou a konečně značku „?“ nesou zbývající vrcholy. Dále použijeme dvě značky „incident“ a „nonincident“ pro Voronoiovy polygony. Jako „incident“ budou označeny polygony, jejichž obálky obsahují vrchol z množiny T , a jako „nonincident“ ostatní polygony. V kroku B2.1 je pak vrchol v_{ijk} vybrán z Voronoiových vrcholů, které jsou označeny „?“ a navíc jsou sousedy vrcholu z množiny T . Pak $G_{l-1}(T \cup \{v_{ijk}\})$ je strom pouze v případě, že platí:

(C4) Voronoioův vrchol v_{ijk} nesousedí s dvěma nebo více vrcholy označenými značkou „in“.

Stejně tak podgraf $C \cap G_{l-1}(T \cup \{v_{ijk}\})$, pro každý primární cyklus C grafu G_{l-1} , je spojitý pouze v případě, že platí:

(C5) Pro každý Voronoioův polygon označený „incident“, tj. obsahující v obálce vrchol v_{ijk} , platí, že tento vrchol v_{ijk} sousedí s jiným vrcholem této obálky označeným značkou „in“.

Platnost podmínek (C2) a (C3) může tedy být posuzováno testováním značek Voronoiových polygonů příslušejících k v_{ijk} a Voronoiových vrcholů sousedících s v_{ijk} .

V inicializační části algoritmu přiřadíme natrvalo třem vrcholům grafu G_3 (malé čtverečky na pomocné křivce - viz. obr. výše) značku „out“. To proto, že Voronoioův polygon nového generátoru je vždy ohraničený, takže tyto tři vrcholy nemohou patřit množině T . Při každém spuštění procedury B je všem ostatním Voronoioovým vrcholům přiřazena značka „?“ a všechny Voronoiovy polygony jsou označeny jako „nonincident“. Pokaždé, když je Voronoioův vrchol přidán do T , jeho označení je změněno z „?“ na „in“ a s ním incidentní Voronoiovy polygony označené „nonincident“ jsou přejmenovány na „incident“. Podobně, když je rozhodnuto, že Voronoioův bod nebude přidán do množiny T , je jeho značka změněna z „?“ na „out“. Voronoiovy vrcholy a polygony, jejichž značky jsou během procedury B změněny, jsou uchovávány a po ukončení procedury B jsou tyto značky vymazány (tj. „in“ a „out“ se přepíše na „?“ a „incident“ se přepíše na „nonincident“). Krok B2 tedy může být proveden v čase úměrném velikosti podstruktury, která je rušena při vkládání nového generátoru.

3.6.3 Zdokonalení numerických výpočtů

Jak bylo řečeno výše, v této metodě je kladen větší důraz na topologickou stabilitu, než na numerické výpočty, což ovšem neznamená, že by numerické výpočty byly nedůležité. Nejdůležitější částí numerických výpočtů je určení $H(p_i, p_j, p_k, p_l)$. Proto se budeme snažit najít co nejspolehlivější způsob tohoto výpočtu.

Často užívanou metodou pro výpočet determinatu matice je Gaussova eliminační metoda, která bude páteří i našeho algoritmu. V našem případě počítáme determinanty mnoha matic lišících se od sebe jen velmi málo ($H(p_i, p_j, p_k, p_l)$ je vyčíslen pro pevné i, j a k a měnící se l). To znamená, že by bylo velmi výhodné využít ve výpočtu parciálních výsledků z předešlého výpočtu. Z těchto dvou hledisek tedy vyplývají dvě možné metody výpočtu $H(p_i, p_j, p_k, p_l)$ [SUGI92]:

Nejdříve upravíme $H(p_i, p_j, p_k, p_l)$ na (vztah (2)):

$$H(p_i, p_j, p_k, p_l) = \begin{vmatrix} 1 & x_i & y_i & (x_i^2 + y_i^2)/2 \\ 1 & x_j & y_j & (x_j^2 + y_j^2)/2 \\ 1 & x_k & y_k & (x_k^2 + y_k^2)/2 \\ 0 & x_l - x_m & y_l - y_m & ((x_l + x_m)(x_l - x_m) + (y_l + y_m)(y_l - y_m))/2 \end{vmatrix} =$$

$$= H_{ijk}^2 (x_l - x_m) - H_{ijk}^3 (y_l - y_m) + \frac{1}{2} H_{ijk}^4 ((x_l + x_m)(x_l - x_m) + (y_l + y_m)(y_l - y_m)),$$

kde m kterýkoli index z i, j nebo k a H_{ijk}^s , kde $s = 2, 3, 4$, je determinant podmatice 3×3 získané z původní matice vynecháním čtvrtého řádku a s -tého sloupce. Tedy:

Vztah (3):

$$H_{ijk}^2 = \begin{vmatrix} 1 & y_i & (x_i^2 + y_i^2)/2 \\ 1 & y_j & (x_j^2 + y_j^2)/2 \\ 1 & y_k & (x_k^2 + y_k^2)/2 \end{vmatrix} =$$

$$= \begin{vmatrix} y_j - y_i & ((x_j + x_i)(x_j - x_i) + (y_j + y_i)(y_j - y_i))/2 \\ y_k - y_i & ((x_k + x_i)(x_k - x_i) + (y_k + y_i)(y_k - y_i))/2 \end{vmatrix}$$

Vztah (4):

$$H_{ijk}^3 = \begin{vmatrix} 1 & x_i & (x_i^2 + y_i^2)/2 \\ 1 & x_j & (x_j^2 + y_j^2)/2 \\ 1 & x_k & (x_k^2 + y_k^2)/2 \end{vmatrix} = \\ = \begin{vmatrix} x_j - x_i & ((x_j + x_i)(x_j - x_i) + (y_j + y_i)(y_j - y_i))/2 \\ x_k - x_i & ((x_k + x_i)(x_k - x_i) + (y_k + y_i)(y_k - y_i))/2 \end{vmatrix}$$

Vztah (5):

$$H_{ijk}^4 = \begin{vmatrix} 1 & x_i & y_i \\ 1 & x_j & y_j \\ 1 & x_k & y_k \end{vmatrix} = \\ = \begin{vmatrix} x_j - x_i & (y_j - y_i) \\ x_k - x_i & (y_k - y_i) \end{vmatrix}$$

Metoda 1:

Pokaždé, když generujeme nový Voronoiov vrchol v_{ijk} , počítáme H_{ijk}^2 , H_{ijk}^3 a H_{ijk}^4 pomocí vztahů (3), (4) nebo (5) (či alternativních vztahů, ve kterých dominantní roli indexu i převezme index j nebo k) a výsledky uchováváme. Ty pak použijeme pro výpočet determinantu $H(p_i, p_j, p_k, p_l)$ dosazením do (2).

Souřadnice Voronoiova vrcholu v_{ijk} jsou reprezentovány dvojicí (6):

$$\left(\frac{H_{ijk}^2}{H_{ijk}^4}, \frac{-H_{ijk}^3}{H_{ijk}^4} \right)$$

Trojice $(H_{ijk}^2, -H_{ijk}^3, H_{ijk}^4)$ je tedy trojice homogeních souřadnic vrcholu v_{ijk} .

Mezivýsledky H_{ijk}^2 , H_{ijk}^3 a H_{ijk}^4 tedy můžeme opakovaně využít pro výpočty souřadnic Voronoiových vrcholů.

Determinant $H(p_i, p_j, p_k, p_l)$ dále upravme vztahem (7):

$$H(p_i, p_j, p_k, p_l) = J_{ijk}^2 (x_l - x_k) - J_{ijk}^3 (y_l - y_k) + \frac{1}{2} J_{ijk}^4 ((x_l - x_k)^2 + (y_l - y_k)^2),$$

kde:

(vztah (8))

$$J_{ijk}^2 = \begin{vmatrix} y_i - y_k & ((x_i - x_k)^2 + (y_i - y_k)^2)/2 \\ y_j - y_k & ((x_j - x_k)^2 + (y_j - y_k)^2)/2 \end{vmatrix}$$

(vztah (9))

$$J_{ijk}^3 = \begin{vmatrix} x_i - x_k & \left((x_i - x_k)^2 + (y_i - y_k)^2 \right) / 2 \\ x_j - x_k & \left((x_j - x_k)^2 + (y_j - y_k)^2 \right) / 2 \end{vmatrix}$$

(vztah (10))

$$J_{ijk}^4 = \begin{vmatrix} x_i - x_k & y_i - y_k \\ x_j - x_k & y_j - y_k \end{vmatrix}$$

Tyto vztahy jsou základem ke druhé metodě výpočtu $H(p_i, p_j, p_k, p_l)$.

Metoda 2:

Pokaždé, když generujeme nový Voronoioův vrchol v_{ijk} , počítáme J_{ijk}^2 , J_{ijk}^3 a J_{ijk}^4 pomocí vztahů (8), (9) nebo (10) a výsledky uchováváme. Ty pak použijeme pro výpočet determinantu $H(p_i, p_j, p_k, p_l)$ dosazením do (7).

Souřadnice Voronoiova vrcholu v_{ijk} jsou reprezentovány dvojicí (11):

$$\left(\frac{-J_{ijk}^2}{J_{ijk}^4} + x_k, \frac{J_{ijk}^3}{J_{ijk}^4} + y_k \right).$$

Trojice $(-J_{ijk}^2, J_{ijk}^3, J_{ijk}^4)$ je tedy trojice homogéních souřadnic vrcholu v_{ijk} vzhledem k původním souřadnicím (x_k, y_k) .

V druhé metodě má generátor p_k speciální význam a nastává zde nejednoznačnost při výběru p_k jako jednoho ze tří možných generátorů. Tu odstraníme tímto pravidlem volby p_k : velikost úhlu při vrcholu p_k v trojúhelníku $p_i p_j p_k$ je nejbližší $\pi/2$, neboť $|J_{ijk}^4/2|$ odpovídá ploše tohoto trojúhelníka a tato plocha může být určen nejpřesněji právě pomocí p_k .

Přírůstkový typ algoritmu mění strukturu Voronoiova diagramu lokálně, což znamená, že hodnoty $H(p_i, p_j, p_k, p_l)$ jsou počítány pro Voronoiovy vrcholy ležící nejbližší novému generátoru p_l . To však znamená, že tyto čtyři generátory p_i, p_j, p_k a p_l jsou navzájem svými nejbližšími sousedy. Proto je tato druhá metoda zatížena menším počtem numerických chyb než metoda první, neboť všechny výpočty jsou prováděny užitím původního p_k (originální p_k se dosadí do (11)).

Obě metody jsou velmi podobné Gaussově eliminační metodě. Liší se od Gaussovy metody pouze v posledním kroku, kdy ukládáme parciální výsledky, abychom je mohli znovu použít v následujícím výpočtu. Obvyklá jednoduchá verze Gaussovy eliminace potřebuje 13 násobení, tři dělení a 18 sčítání. Naproti tomu první metoda potřebuje 15 násobení a 24 sčítání a druhá metoda také 15 násobení, ale již jen 14 sčítání. Numerické chyby jsou obecně větší při sčítání, než při násobení a dělení. To znamená, že druhá metoda je spolehlivější než první i než obecná Gaussova metoda. První metoda využívá více sčítání než jednoduchá Gaussova, ale to je způsobeno použitím výrazu

$(a + b)(a - b)$ místo původního $a^2 - b^2$. (Použijeme-li opět výraz $a^2 - b^2$, bude první metoda používat stejný počet sčítání jako obecná Gaussova eliminační metoda.) Výraz $(a + b)(a - b)$ je však vyhodnocován všeobecně s menší numerickou chybou než výraz $a^2 - b^2$, proto předpokládáme, že i první metoda pracuje s menší numerickou chybovostí než klasická Gaussova metoda. Obecná Gaussova eliminační metoda navíc nedovoluje několikanásobné využití mezivýsledků výpočtu.

3.6.4 Závěrem

Jak už bylo řečeno výše, tyto dvě metody přírůstkové konstrukce Voronoiova diagramu kladou důraz na topologickou stabilitu struktury více než na výsledky numerických výpočtů. Přitom průměrná výpočtová složitost algoritmu se od původního klasického přírůstkového algoritmu nemění a zůstává lineární. Navíc struktura programu využívajícího tyto dvě nové metody je jednodušší než při použití klasické techniky, neboť odpadá nutnost speciálních řešení případů degenerací (připomeňme, že v aritmetice v „konečné“ přesnosti nemůžeme rozhodnout, zda k degeneraci došlo či nikoli, tj. klasické algoritmy předpokládají, že degenerace neexistují).

Navíc, a to je vlastně i hlavní záměr, autoři těchto metod (Kokichi Sugihara a Masao Iri, „Topology-oriented incremental method for constructing Voronoi diagrams robust against numerical errors“, Tokyo 1992) uvádí, že díky vylepšení způsobu numerických výpočtů je první metoda schopna generovat „korektní“ Voronoioův diagram pro 2×10^5 vrcholů (pro 3×10^5 chybje) a druhá metoda je úspěšná dokonce pro množiny s počtem prvků kolem 1 000 000 a to v single-precision aritmetice.

3.7 Konstrukce Voronoiova a Delaunayova diagramu pomocí topologicky orientovaného algoritmu rozděl-a-panuj

V této kapitole si ukážeme úpravu konvenčního algoritmu pro konstrukci Voronoiova diagramu založenou na metodě rozděl-a-panuj na algoritmus kladoucí důraz na topologickou stabilitu diagramu. Tj. stejně jako v minulé kapitole, v každém kroku algoritmu je považována za důležitější topologická otázka a výsledky numerických výpočtů jsou užity pouze nejsou-li v rozporu s topologií. Tento algoritmus produkuje „korektní“ diagram nezávisle na tom, jak moc malou přesnost v aritmetice použijeme. Korektní zde znamená konvergující k ideálnímu diagramu. Navíc výpočtová složitost tohoto upraveného algoritmu je opět stejná jako složitost originálního algoritmu rozděl-a-panuj.

Pro překlenutí propasti mezi teoreticky správnými algoritmy a v praxi správně fungujícími programy způsobenou numerickými chybami se

poslední dobou používají metody rozdělené podle způsobu přístupu k problému do čtyř základních skupin:

- I. *Využívající toleranci:*
Tolerance je malé, kladné a pevně dané číslo, řekněme ε . V případě, že dva geometrické elementy jsou od sebe vzdálené méně než ε , řekneme, že se tyto prvky nacházejí na stejné pozici. Tato metoda je velmi často používaná, přestože může generovat nesmyslné situace (např. když se tři nebo více elementů nachází velmi blízko k sobě).
- II. *Přístup používající vysoce přesnou aritmetiku:*
Pomocí velmi přesné aritmetiky (racionální aritmetika) je opravdu možné předejít topologickým odchylkám způsobeným numerickými chybami ve výpočtu, avšak za cenu velmi náročných výpočtů (jak časově, tak i paměťově), což tuto metodu vytlačuje z množiny metod prakticky použitelných.
- III. *Přístup s využitím analýzy chyb:*
Výsledky výpočtů jsou klasifikovány jako spolehlivé či nespolehlivé podle předem zjištěné chybovosti a využity jsou algoritmem pouze ty spolehlivé. Tato analýza je však velmi složitá a i samotný proces určování chybového intervalu je velmi komplikovaný, proto se ani tato metoda v praxi příliš neprosazuje.
- IV. *Topologicky orientovaný přístup:*
Důraz je kladen především na topologickou stabilitu a pouze v případě, že nehrozí narušení této stability je využito výsledků numerických výpočtů. Tento přístup není náročný na numerické výpočty. Mimoto nevyžaduje náročnou chybovou analýzu a s ní spojené složité větvení procesu výpočtu. Topologicky orientovaný přístup se tedy zdá být momentálně nejvhodnějším přístupem pro konstrukci algoritmu odolného proti numerickým chybám.

Z minulé kapitoly víme, že průměrná výpočtová složitost topologicky orientovaného přírůstkového algoritmu pro konstrukci Voronoiova diagramu je $O(N)$. Ta však není optimální, neboť v nehorším případě je složitost $O(N^2)$.

Výpočtová složitost algoritmu rozděl-a-panuj je (a je optimální) je $O(N \log N)$ (viz. kapitola „Konstrukce Voronoiova diagramu metodou rozděl-a-panuj“). V předchozích kapitolách uvažujeme průměrnou výpočtovou složitost tohoto algoritmu $O(N \log N)$. Avšak pánové Katajainen a Koppinen roku 1988 ve své práci „Constructing Delaunay triangulations by merging buckets in quadtree order“ (Fundamenta informaticae, vol. XI, 1988) ukázali, že algoritmus rozděl-a-panuj může běžet v průměrném čase $O(N)$, ovšem za předpokladu čtvrtinového dělení oblastí (konvenční metoda rozděl-a-panuj dělí rovinu rekursivně na dvě poloviny vertikální přímkou, zatímco Katajainenova metoda přidává ještě horizontální dělení v polovině, tj. dělení na čtvrtiny).

Topologicky orientované aplikace založené na algoritmu rozděl-a-panuj nejsou jednoduché, neboť, na rozdíl od přírůstkového algoritmu, strategie rozděl-a-panuj je již svojí podstatou paralelní a udržení topologické stability v paralelních procesech je poměrně nový a stále se vyvíjející problém. V této kapitole budou uvedeny „topologické podmínky“, kterými můžeme udržet konzistenci dat generovaných a modifikovaných během zpracování.

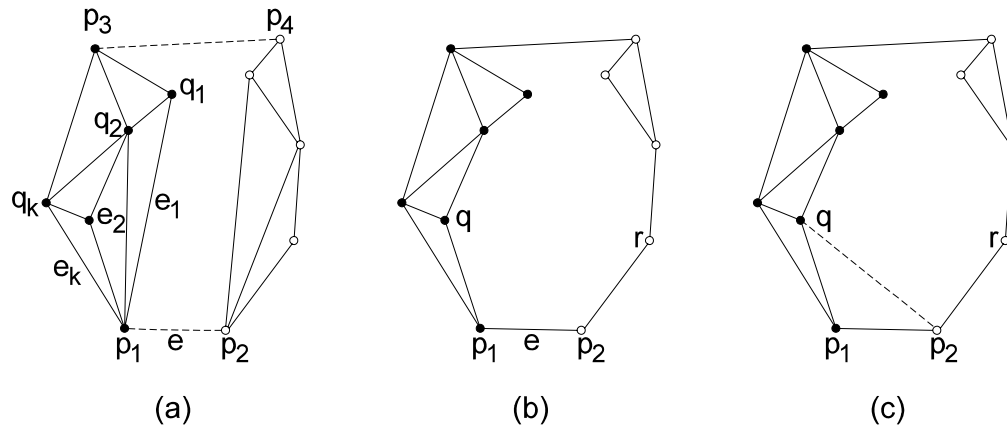
Nejdříve si stručně připomeňme označení zavedená v předcházejících kapitolách. Tedy S je množina N bodů v rovině. Tyto body p_i ($p_i \in S$) nazýváme generátory. Voronoiov diagram sestrojený na množině S označujeme jako $\text{Vor}(S)$. Ten se skládá z Voronoiových hran, které spojují vždy dva Voronoiovy vrcholy v_i . Spojením všech dvojic generátorů p_i a p_j ležících v sousedních Voronoiových polygonech (ty sdílejí právě jednu Voronoiovu hranu, osu půlící úsečku $p_i p_j$) vznikne jiný planární diagram, duální k Voronoiovu diagramu, Delaunayova triangulace či Delaunayův diagram. Delaunayův diagram sestrojený na množině generátorů S , označme jej jako $\text{Del}(S)$, se skládá z Delaunayových polygonů (trojúhelníků) a Delaunayových hran (Delaunayovými vrcholy jsou generátory p_i). Z duality obou grafů vyplývá, že každý Voronoiov vrchol z $\text{Vor}(S)$ je duální k právě jednomu Delaunayovu polygonu z $\text{Del}(S)$ a opačně. Transformace $\text{Del}(S)$ na $\text{Vor}(S)$ a naopak $\text{Vor}(S)$ na $\text{Del}(S)$ je proveditelná v čase $O(N)$.

Předpokládejme opět, že žádné čtyři body p_i, p_j, p_k a p_l patřící množině S neleží na jedné kružnici (*protidegenerační předpoklad*). Tím se vyhneme komplikovanému a zdlouhavému řešení pro různé typy degenerací.

3.7.1 Obvyklý algoritmus rozděl-a-panuj pro konstrukci Delaunayova diagramu

Množina S je, stejně jako v kapitole „Konstrukce Voronoiova diagramu metodou rozděl-a-panuj“, rozdělena na dvě podmnožiny S_1 a S_2 tak, že rozdíl jejich velikostí je nejvýše jedna a rozdělení může být realizováno přímkou. Na chvíli předpokládejme, že body patřící množině S_1 mají menší x -ovou souřadnici než body množiny S_2 . Množině S_1 (resp. S_2) pak budeme říkat *levá* (resp. *pravá*) *množina generátorů*. Obě množiny S_1 i S_2 jsou rekurzivně děleny stejným způsobem, dokud výsledná podmnožina vzniklá dělením není dostatečně malá (tj. přibližně tři až pět bodů), aby mohla být jednoduše ztriangularizována. Výsledné triangulace množin S_1 a S_2 spojíme a dostaneme tak Delaunayův diagram pro $S_1 \cup S_2$.

Nyní si popíšeme způsob spojování triangulací. Předpokládejme existenci již sestrojených triangulací $\text{Del}(S_1)$ a $\text{Del}(S_2)$, viz. obrázek 1(a).



Obr. 1: Spojení pravé a levé triangulace:
 (a) Konstrukce horní a dolní společné tečny.
 (b) Smazání nadbytečných hran.
 (c) Konstrukce nové Delaunayovy hrany

Na všech třech obrázcích jsou body množiny S_1 znázorněny plnými kroužky, zatímco body množiny S_2 prázdnými kroužky. Nejdříve najdeme $p_1 \in S_1$ a $p_2 \in S_2$ takové, že všechny ostatní body z S jsou zdola ohraničeny přímkou p_1p_2 a sestrojíme hranu p_1p_2 . Říkáme, že hrana p_1p_2 je *dolní společnou tečnou hranou*. Podobně najdeme $p_3 \in S_1$ a $p_4 \in S_2$ takové, že všechny ostatní body z S leží shora ohraničeny přímkou p_3p_4 a sestrojíme hranu p_3p_4 . Říkáme, že hrana p_3p_4 je *horní společnou tečnou hranou*. Tyto dvě nové hrany patří mezi Delaunayovy hrany v $\text{Del}(S_1 \cup S_2)$.

Nyní odstartujeme od dolní společné tečné hrany proces odstraňování nadbytečných hran z $\text{Del}(S_1)$ a $\text{Del}(S_2)$ a proces vkládání nových Delaunayových trojúhelníků do $\text{Del}(S_1 \cup S_2)$ jeden po druhém, dokud nenarazíme na horní společnou tečnou hranu. Nový Delaunayův trojúhelník je sestaven vložením hrany spojující generátor z množiny S_1 s generátorem z množiny S_2 . Této hraně budeme říkat *traverzová hrana*.

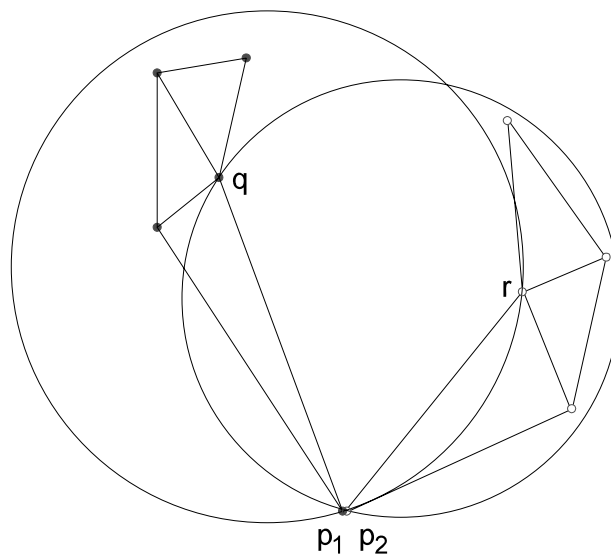
Označme dolní společnou tečnou hranu jako e . Konstrukce Delaunayova trojúhelníka příslušejícího hraně e se provede následujícím způsobem. Necht' p_1 je levý koncový bod hrany e (viz. obrázek (a)). Necht' jsou dále e_1, e_2, \dots, e_k hrany v $\text{Del}(S_1)$ vycházející z vrcholu p_1 seřazené proti směru chodu hodinových ručiček a q_1, q_2, \dots, q_k jsou druhé koncové body těchto hran. Pro $i = 1, 2, \dots, k - 1$ nyní testujeme, zda se uvnitř kružnice procházející vrcholy p_1, q_i a q_{i+1} nachází nějaký generátor z množiny S_2 . Je-li takový generátor nalezen, smažeme e_i z triangulace $\text{Del}(S_1)$. (Z $\text{Del}(S_1)$ na obrázku (a) budou smazány hrany e_1 a e_2 .) Stejným způsobem testujeme hrany $\text{Del}(S_2)$ vycházející z vrcholu p_2 (seřazené po směru chodu hodinových ručiček) a smažeme nevyhovující (neuspokojující kritérium prázdné kružnice) hrany. Tak dostaneme výsledný diagram, viz. obrázek (b).

Nyní předpokládejme, že hrana p_1q je první hranou po hraně e (v pořadí proti směru chodu hodinových ručiček) vycházející z vrcholu p_1 a podobně

hrana p_2r je první hranou po hraně e (v pořadí ve směru chodu hodinových ručiček) vycházející z vrcholu p_2 . Testujeme, jestli vrchol r leží uvnitř kružnice procházející body p_1, p_2 a q . V případě, že tomu tak není (tj. vrchol q leží uvnitř kružnice procházející body p_1, p_2 a r), sestrojíme traverzovou hranu p_2q (viz. obrázek 1(c)). V opačném případě sestrojíme traverzovou hranu p_1r .

Takto jsme tedy sestrojily novou traverzovou hranu. Tuto hranu označíme jako e a proces opakujeme, dokud právě vytvořený Delaunayův trojúhelník neobsahuje, jako jednu ze svých hran horní společnou tečnou hranu.

Tento algoritmus funguje, jsou-li numerické výpočty přesné. V praxi by však tento algoritmus následkem nekonzistentních situací způsobených numerickými odchylkami chyboval. Příklad takové situace je znázorněn na následujícím obrázku 2.



Obr. 2: Nekonzistentní situace způsobená numerickými chybami

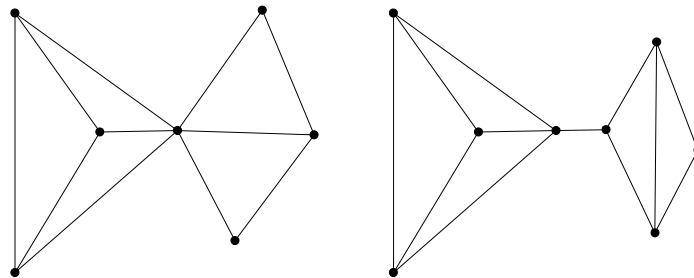
Na tomto obrázku se nacházíme ve fázi, kdy byla nalezena dolní společná tečná hrana a jsou již zrušeny všechny nadbytečné hrany z $\text{Del}(S_1)$ i $\text{Del}(S_2)$. Nyní však nastává problém při generování nových traverzových hran. Bod q totiž leží uvnitř kružnice procházející body p_1p_2r a současně bod r totiž leží uvnitř kružnice procházející body qp_1p_2 a proto ani jedna z hran p_1r a p_2q nepřichází v úvahu. V tuto chvíli se konvenční algoritmus rozděl-a-panuj ukazuje jako nedostačující (v aritmetice konečné přesnosti).

3.7.2 Algoritmus cestou kombinatorické geometrie

V této podkapitole se budeme zabývat popisem páteře algoritmu za použití pouze kombinatorických termínů. Z pohledu kombinatoriky je tedy Delaunayův diagram planárním grafem G . Předpokládejme, že počet prvků množiny S je $N \geq 3$. Graf G pak uspokojuje následující podmínky:

- (C1) Graf G je jednoduchý (tj. neobsahuje rovnoběžné hrany a žádná hrana nemůže začínat i končit v jednom vrcholu, tj. netvoří vlastní smyčku).
- (C2) Graf G je spojitý.
- (C3) V každém vrcholu grafu G se scházejí nejméně dvě hrany.
- (C4) Každá ohraničená oblast, kromě vnější oblasti, je ohraničena právě třemi hranami.
- (C5) Obálka vnější oblasti tvoří jednoduchý cyklus (tj. budeme-li touto obálkou procházet hranou po hraně, ocitneme se v každé hraně právě jednou, dokud znovu nenarazíme na první hranu pochodu).

Podmínka (C4) platí, neboť je dána protidegeneračním předpokladem. Podmínka (C5) je také zřejmá, neboť citovanou obálkou je vlastně konvexní obálka množiny generátorů (vrcholům tvořícím tuto obálku říkáme *krajní vrcholy* triangulace). To znamená, že například grafy na následujícím obrázku 3 jsou nepřípustné.



Obr. 3: Tyto dva grafy nemohou být Delaunayovou triangulací.

Pro $N = 3, 4$, nebo 5 není obtížné sestavit graf G vyhovující podmínkám (C1) až (C5). Tento graf však nemusí odpovídat „korektnímu“ Delaunayovu diagramu, neboť numerické výpočty mohly být zatíženy chybami. Můžeme však zaručit přinejmenším platnost podmínek (C1) až (C5), neboť všechny tyto podmínky jsou již svojí podstatou testovatelné bez použití numerických výpočtů.

Nyní předpokládejme, že planární grafy G_1 a G_2 vyhovující nárokům (C1) až (C5) odpovídají diagramům $\text{Del}(S_1)$ a $\text{Del}(S_2)$. Naším cílem je spojením G_1 a G_2 získat nový graf G odpovídající triangulaci $\text{Del}(S_1 \cup S_2)$. Tento úkol se pokusíme vyřešit využitím geometrické kombinatoriky

následujícím algoritmem. Algoritmus se bude opírat o obrázek „Spojení pravé a levé triangulace“.

Algoritmus 1 (kombinatoricky zaměřená kostra procesu spojování)

Vstup:

Levá a pravá množina generátorů (S_1 a S_2) a dva planární grafy G_1 a G_2 sestrojené na množinách vrcholů S_1 a S_2 vyhovující podmínkám (C1) až (C5).

Výstup:

Graf G sestrojený na množině vrcholů $S_1 \cup S_2$ a vyhovující podmínkám (C1) až (C5).

Tělo:

1. Sestroj dvě hrany p_1p_2 a p_3p_4 , kde $p_1, p_3 \in G_1$ a $p_2, p_4 \in G_2$, takové, že hrana p_1p_2 tvoří dolní společnou tečnou hranu a p_3p_4 tvoří horní společnou tečnou hranu.
2. Proveď přiřazení $e \leftarrow p_1p_2$.
3. Necht' p_1 je levý koncový bod hrany e . Necht' jsou dále e_1, e_2, \dots, e_k hrany v $\text{Del}(S_1)$ vycházející z vrcholu p_1 seřazené proti směru chodu hodinových ručiček a q_1, q_2, \dots, q_k jsou druhé koncové body těchto hran. Pro $i = 1, 2, \dots, k - 1$ nyní testuj, zda se uvnitř kružnice procházející vrcholy p_1, q_i a q_{i+1} nachází nějaký generátor z množiny S_2 . Je-li takový generátor nalezen, smaž e_i z triangulace $\text{Del}(S_1)$ (nemusí být smazána žádná hrana).
4. Necht' p_2 je pravý koncový bod hrany e . Necht' jsou dále e_1, e_2, \dots, e_k hrany v $\text{Del}(S_2)$ vycházející z vrcholu p_2 seřazené po směru chodu hodinových ručiček a q_1, q_2, \dots, q_k jsou druhé koncové body těchto hran. Pro $i = 1, 2, \dots, k - 1$ nyní testuj, zda se uvnitř kružnice procházející vrcholy p_2, q_i a q_{i+1} nachází nějaký generátor z množiny S_1 . Je-li takový generátor nalezen, smaž e_i z triangulace $\text{Del}(S_2)$ (nemusí být smazána žádná hrana).
5. Tvoří-li oblast shora sousedící s hranou e trojúhelník a obsahuje-li tento trojúhelník současně hranu e i hranu p_3p_4 , ukonči proceduru, neboť konstrukce grafu G je dokončena.
6. Sestroj traverzovou hranu e' sousedící s hranou e tak, že hrany e', e a ještě nějaká další tvoří nový trojúhelník doléhající na hranu e .
7. Proveď přiřazení $e \leftarrow e'$ a vrať se na krok 3.

Na první pohled se zdá, že tento algoritmus je čistě záležitostí geometrické kombinatoriky a že se tedy nemusíme obávat numerických chyb. Kroky 1, 3, 4 a 6 jsou však v této otázce nejednoznačné, neboť obsahují dvě nebo i více možností dalšího postupu, mezi nimiž se rozhodujeme s využitím výsledků numerických výpočtů. A právě numerické chyby vzniklé během těchto výpočtů mohou způsobit generování nekonzistentních situací. Abychom

se těmito situacím vyhnuli, zavedeme tzv. *topologické podmínky*. Tyto podmínky jsou popsány v termínech kombinatoriky a mohou zaručit, že algoritmus 1 proběhne aniž by generoval nekonzistentní situace, a že výsledný graf G vyhovuje podmínkám (C1) až (C5).

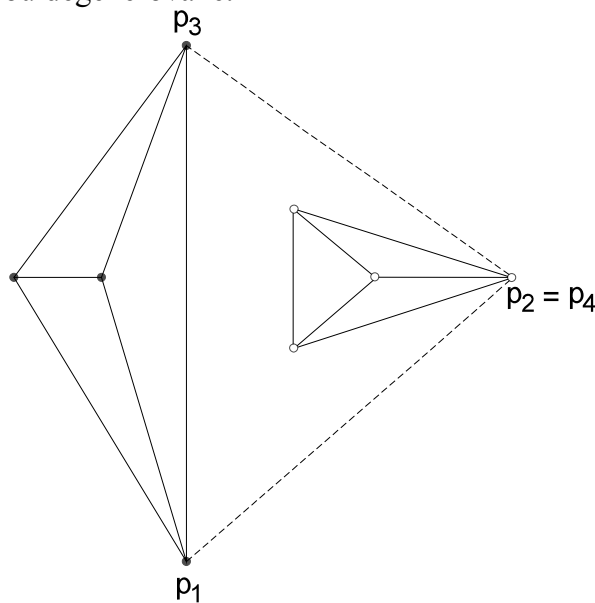
Nyní si ještě mírně rozšíříme náš terminologický slovníček. Hrana e , které je při inicializaci přiřazena dolní společná tečná hrana p_1p_2 , a později (v kroku 7) nová traverzová hrana, se nazývá *základní hrana*. Dále předpokládejme, že se nacházíme na konci čtvrtého kroku. Hrana ústící do levého (resp. pravého) koncového bodu základní hrany a je další v pořadí proti směru chodu hodinových ručiček (resp. po směru chodu hodinových ručiček) po základní hraně, se nazývá *levá sousední hrana* (resp. *pravá sousední hrana*) základní hrany.

První uvedená podmínka se bude týkat prvního kroku algoritmu 1.

Podmínka 1:

Pro vrcholy vybrané v kroku 1 musí platit, že buď $p_1 \neq p_3$ nebo $p_2 \neq p_4$.

K jednomu z případů, tj. buď $p_1 = p_3$ nebo $p_2 = p_4$, dojít může (viz. následující obrázek 4), nikoli však k oběma najednou, neboť S_1 a S_2 mají nejméně tři vrcholy a ty nejsou degenerované.



Obr. 4: Horní a dolní společné tečné hrany sdílejí vrchol p_2 resp. p_4 .

Zdá se, že Podmínka 1 je jediná možná podmínka použitelná v prvním kroku. Předpokládejme, že jsme vybraly libovolné vrcholy p_1, p_2, p_3 a p_4 za předpokladu, že Podmínka 1 není porušena. Pak se může stát, že pozici špatně určíme hrany p_1p_2 a p_3p_4 jako dolní a horní společné tečné hrany pro nové (špatně) pozice bodů v S_1 a S_2 . Takový zmatek na vstupu může mít stejné důsledky jako numerické chyby při správně určených vrcholech p_1, p_2, p_3 a p_4 .

Proto by bylo velmi obtížné nahradit Podmínku 1 tvrdší podmínkou. Totéž platí i pro následující Podmínky, které postupně uvedeme.

V krocích tři a čtyři chceme z grafů G_1 a G_2 vyjmout hrany, které nepatří do $\text{Del}(S_1 \cup S_2)$ (hrany, které v $\text{Del}(S_1 \cup S_2)$ nevyhovují kritériu prázdné Delaunayovy kružnice). Za tímto účelem zavedeme následující podmínku.

Podmínka 2:

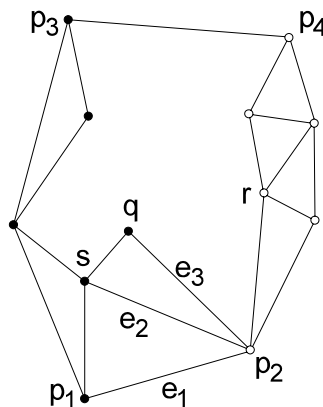
Hrana nesmí být v krocích 3 nebo 4 smazána, jestliže by tím byla způsobena nespojitost grafu.

Kdyby se graf stal nespojitým, nemohli bychom jej již algoritmem 1 doplnit do opět spojitého. Výsledný graf by potom nemohl být Delaunayovým diagramem. Vysvětleme si tuto podmínku na obrázku „Spojení pravé a levé triangulace“ - část (c). Sestrojili jsme tedy traverzové hrany p_1p_2 a p_2q a nacházíme se v kroku 4, přičemž základní hranou je hrana p_2q . Podmínka 2 pak říká, že hrana p_2r (viz. obr.) nesmí být zrušena.

Podmínka 3:

Hrana nesmí být v krocích 3 nebo 4 smazána, jestliže oběma svými vrcholy sousedí s traverzovými hranami.

Předpokládejme, že hrany e_1 , e_2 a e_3 jsou traverzovými hranami sestavenými algoritmem 1 (viz. následující obrázek 5) a základní hranou je nyní hrana e_3 .



Obr. 5: K Podmínce 3: Hrana sq nesmí být zrušena.

(Čtenář může poznamenat, že tato ilustrace má daleko ke korektní spojovací proceduře Delaunayova diagramu. Připomeňme však, že předpokládáme algoritmus pracující v nepřesné aritmetice a tak naši pozornost soustředíme na samotnou strukturu grafu, přičemž pozice bodů v této ilustraci může být

nepřesná.) Podmínka 3 tedy říká, že hrana qs by neměla být smazána, neboť její smazání by nevedlo ke korektnímu Delaunayovu diagramu.

Podmínka 4:

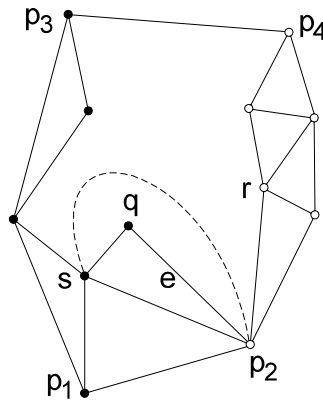
Hrana p_3p_4 sestrojená v kroku 1 nesmí být v kroku 3 nebo 4 smazána.

Tato podmínka je nezbytně nutná, neboť jak dolní společná tečná hrana tak i horní společná tečná hrana p_3p_4 jsou hranami patřící do $\text{Del}(S_1 \cup S_2)$. Dále se budeme věnovat kroku 6, který potřebuje dvě podmínky.

Podmínka 5:

Krok 6 nesmí generovat paralelní hrany.

Podívejme se na následující obrázek 6.



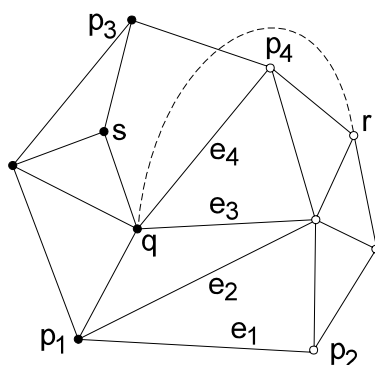
Obr. 6: K Podmínce 5: Hrana sp_2 nemůže být generována.

Nacházíme se v kroku 6 a základní hranou e je hrana qp_2 . Při generování následující traverzové hrany se nabízejí dvě možnosti, hrany p_2s a qr . Hrana p_2s ovšem nepřipadá v úvahu, neboť jejím zvolením by došlo k degradaci sítě, neboť vrcholy p_2 a s by byly spojeny dvěma paralelními hranami, což není možné. Proto jedinou možnou volbou je traverzální hrana qr , což zaručí Podmínka 5.

Podmínka 6:

Algoritmus v kroku 6 nesmí generovat hranu, měla-li by tato hrana porušit planaritu grafu.

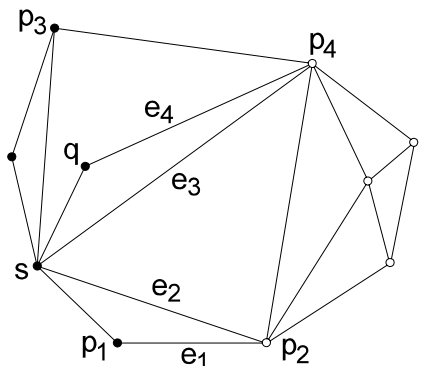
Podmínka 6 je částečně vysvětlena následujícím obrázkem 7.



Obr. 7: K Podmínce 6: Hrana qr by porušila planaritu grafu.

Předpokládejme, že jsou sestrojeny traverzální hrany e_1, e_2, e_3 a e_4 , a my se nacházíme v kroku 6, přičemž základní hranou je hrana e_4 . Pravá sousední hrana je horní společná tečná hrana p_3p_4 a proto jako novou traverzovou hranu nemůžeme sestrojít hranu qr , neboť křížením této hrany s p_3p_4 by byla porušena planarita grafu. Jediným možným řešením je tedy konstrukce traverzové hrany p_4s .

Podmínky 5 a 6 se však mohou dostat do rozporu. Příklad takové situace je ukázán na dalším obrázku 8.



Obr. 8: Konflikt mezi podmínkami 5 a 6.

Předpokládejme opět, že jsou sestrojeny traverzální hrany e_1, e_2, e_3 a e_4 , a my se nacházíme v kroku 6, přičemž základní hranou je hrana e_4 . Jelikož nemůžeme generovat paralelní hrany (Podmínka 5), nemůžeme sestrojít traverzovou hranu sp_4 . Pravá sousední hrana je horní společná tečná hrana p_3p_4 a proto jako novou traverzovou hranu nemůžeme sestrojít ani hranu qr (Podmínka 6).

Takové konflikty jsou navozeny situacemi, kdy jsou numerické výpočty nespolehlivé. Způsob, jak tyto konflikty vyřešit je takový, že zvlášť ztriangularizujeme zbývající oblast a ukončíme spojovací proceduru. Tuto speciální proceduru nazvěme např. *proti-konfliktní triangularizace*.

Poznamenejme snad ještě, že žádná proti-konfliktní triangularizace nenarušuje podmínky (C1) až (C5).

✦**Tvrzení 1:** Předpokládejme, že Algoritmus 1 proběhl takovým způsobem, že podmínky 1, 2, ..., 6 nebyly narušeny a že proti-konfliktní triangularizace byla nasazena v případech, kdy se dostali do konfliktu podmínky 5 a 6. Algoritmus 1 je pak proveden až do konce a graf G na výstupu uspokojuje podmínky (C1) až (C5).

Důkaz: Krok 1 může být proveden bez narušení podmínky 1, neboť jak G_1 tak i G_2 obsahují tři nebo více vrcholů. Kroky 3 a 4 mohou být provedeny bez narušení podmínk 2 a 3, neboť jednou z možných akcí provedenou těmito kroky je i „prázdná akce“ (nic se nesmaže). Krok 6 může být proveden, neboť i když dojde ke kolizi, můžeme problém dořešit pomocí žádná proti-konfliktní triangularizace. Kroky 2, 5 a 7 jsou zřejmé a nepotřebují komentář. Tak tedy, všechny kroky algoritmu mohou být provedeny aniž by byly generovány nekonzistentní situace.

Výsledný graf G uspokojuje podmínku (C1), neboť G_1 a G_2 jsou jednoduché a nikdy negenerujeme paralelní hrany či vlastní smyčky (Podmínka 5). Graf G uspokojuje také podmínku (C2), neboť G_1 a G_2 jsou spojitě, v kroku 1 jsou transformovány do jednoho spojitého grafu a výsledný i graf na výstupu zůstává spojitý (Podmínka 2). Graf G uspokojuje i podmínky (C3) a (C4), neboť Algoritmus 1 je ukončen až ve chvíli, kdy se všechny jeho oblasti (vyjma vnější oblasti vzhledem ke konvexní obálce) stanou trojúhelníky. A konečně ještě poslední důkaz, že graf G uspokojuje i podmínku (C5). Necht' $\{p_3, q_1, q_2, \dots, q_k, p_1\}$ je proti směru chodu hodinových ručiček seřazená posloupnost krajních vrcholů od p_3 do p_1 grafu G_1 a necht' $\{p_2, r_1, r_2, \dots, r_l, p_4\}$ je proti směru chodu hodinových ručiček seřazená posloupnost krajních vrcholů od p_2 do p_4 grafu G_2 . Pak posloupnost $\{p_3, q_1, q_2, \dots, q_k, p_1, p_2, r_1, r_2, \dots, r_l, p_4, p_3\}$ tvoří, během celého algoritmu, obálku vnější (jediné neuzavřené) oblasti, a to díky podmínkám 1 až 6. Proto výsledný graf G uspokojuje i podmínku (C5).

3.7.3 Použití numerických výpočtů

Tvrzení 1 zaručuje, že Algoritmus 1 vždy proběhne bez toho, aby vznikaly nekonzistentní situace, a graf na výstupu je topologicky konzistentní v tom významu, že uspokojuje podmínky (C1) až (C5). Zatím však nejsou vyřešeny nejasnosti kolem výběru způsobů zpracování. Naším primárním cílem je konstrukce Delaunayovy triangulace a k nalezení nejslibnějšího způsobu zpracování, který povede k Delaunayovu diagramu, použijeme numerických výpočtů.

Nejdůležitějším numerickým výpočtem používaným v krocích 3, 4 a 6, je test prázdnoty Delaunayovy kružnice. (Takový test je uveden již v kapitole „Přírůstková konstrukce Voronoiova diagramu s důrazem na topologickou stabilitu“. V této kapitole si jej krátce připomeneme a ukážeme jeho implementaci v Algoritmu 1.)

Předpokládejme tedy, že souřadnice vrcholu p_i jsou (x_i, y_i) . Pak:

$$H(p_i, p_j, p_k, p_l) = \begin{vmatrix} 1 & x_i & y_i & x_i^2 + y_i^2 \\ 1 & x_j & y_j & x_j^2 + y_j^2 \\ 1 & x_k & y_k & x_k^2 + y_k^2 \\ 1 & x_l & y_l & x_l^2 + y_l^2 \end{vmatrix}$$

Dále předpokládejme, že používáme pravotočivý souřadnicový systém, a že p_i, p_j, p_k je proti směru chodu hodinových ručiček seřazená posloupnost vrcholů trojúhelníka. Pak platí, že bod p_l leží uvnitř kružnice opsané trojúhelníku $p_i p_j p_k$ jen a pouze v případě, že $H(p_i, p_j, p_k, p_l) < 0$ (připomeňme, že rovnice $H(p_i, p_j, p_k, p) = 0$ s proměnnou p reprezentuje kružnici procházející trojicí bodů p_i, p_j a p_k).

Předpokládejme, že hrana $p_i p_j$ je základní hranou ($p_i \in S_1$ a $p_j \in S_2$) a její levý soused je hrana $p_i p_k$. Pak je v kroku 3 hrana $p_i p_k$ zrušena jen a pouze v případě, že:

- (i) zrušení této hrany nenaruší podmínku 2, 3 nebo 4,
- (ii) $H(p_i, p_j, p_k, p_l) < 0$ pro alespoň jeden vrchol z grafu G_2 .

Eliminování hran v kroku 4 je provedeno podobným způsobem.

Dále předpokládejme, že hrana $p_i p_j$ je základní hranou ($p_i \in S_1$ a $p_j \in S_2$), její levým sousedem je hrana $p_i p_k$ a její pravým sousedem je hrana $p_i p_l$. Překáží-li ve zvolení jedné z hran $p_i p_l$ a $p_i p_k$ kandidujících na novou traverzovou hranu Podmínka 5 nebo 6, zvolíme tu druhou. Jsou-li vyhovující obě tyto hrany, vybereme hranu $p_i p_l$, platí-li, že $H(p_i, p_j, p_k, p_l) < 0$, v opačném případě jako novou traverzovou hranu určíme $p_i p_k$.

3.7.4 Závěrem

V této kapitole byl prezentován numericky odolný algoritmus pro konstrukci Voronoiova diagramu topologicky orientovanou metodou rozděl-a-panuj, jehož výpočtová složitost je $O(N \log N)$ a je optimální. Díky topologickým podmínkám uvedeným v této kapitole může algoritmus předcházet nekonzistentním situacím způsobeným numerickými chybami. Nyní si shrneme vlastnosti Algoritmu:

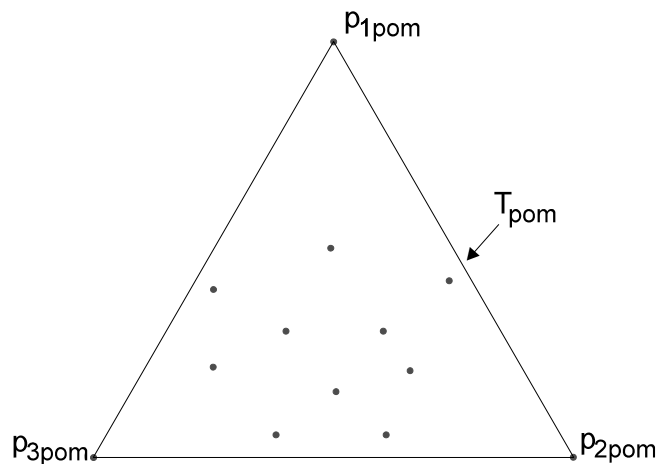
- (i) Algoritmus vždy na výstupu generuje topologicky konzistentní graf.
- (ii) Algoritmus je korektní v tom smyslu, že na výstupu generovaný graf konverguje k ideálnímu diagramu sestrojenému v přesné aritmetice.

- (iii) Algoritmus je jednoduchý v tom smyslu, že nevyžaduje větvené zpracování pro různé degenerace.
- (iv) Algoritmus je optimální v tom smyslu, že jeho průměrná i nejhorší výpočtová složitost je $O(N \log N)$. (Průměrná výpočtová složitost Katajainenovy metody je dokonce $O(N)$.)

3.8 Delaunayova přírůstková triangularizace

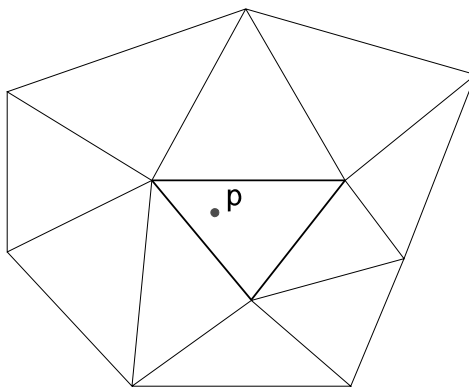
Jak už bylo řečeno v předešlých kapitolách, triangulace je nazvána Delaunayova, splňuje-li podmínku tzv. prázdných opsaných kružnic: to znamená, že uvnitř každé kružnice opsané libovolnému trojúhelníku triangulace neleží žádný jiný bod vstupní množiny. Delaunayova triangulace je optimální z několika hledisek. Maximalizuje, například, nejmenší úhly trojúhelníků a minimalizuje největší kružnice opsané trojúhelníkům všech možných triangulací na vstupní množině bodů. Proto je Delaunayova triangulace důležitým nástrojem pro kvalitní generování sítí na konečných množinách vstupních prvků. Poznamenejme však, že tato metoda neumožňuje při konstrukci trojúhelníkové sítě respektovat a udržovat množinu „pevných“ hran zadaných spolu se vstupní množinou bodů (viz. kapitola „Vázaná triangularizace“).

Delaunayova přírůstková triangularizace je založena na přidávání bodu po bodu do triangulace a lokální optimalizace trojúhelníkové sítě v jeho okolí. V inicializační části algoritmu se v rovině umístí tři pomocné body tvořící vrcholy pomocného trojúhelníka T_{pom} obklopujícího celou vstupní množinu (označme ji opět známým S), tj. žádný bod z S neleží vně T_{pom} , viz obrázek 1.



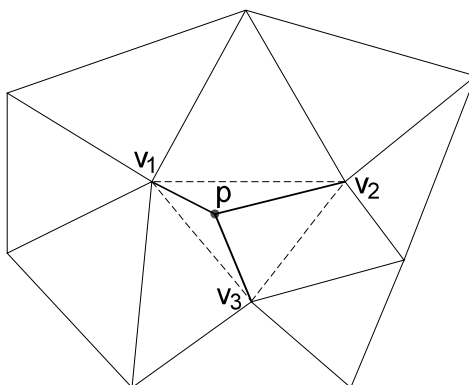
Obr. 1: Sestrojení pomocného trojúhelníka.

Tento trojúhelník vložíme do triangulace. Od této chvíle přidáváme po jednom bodu z množiny S a tento bod (označme jej p) vždy padne do některého trojúhelníka (označme jej T) triangulace (viz. obr. 2(a)).



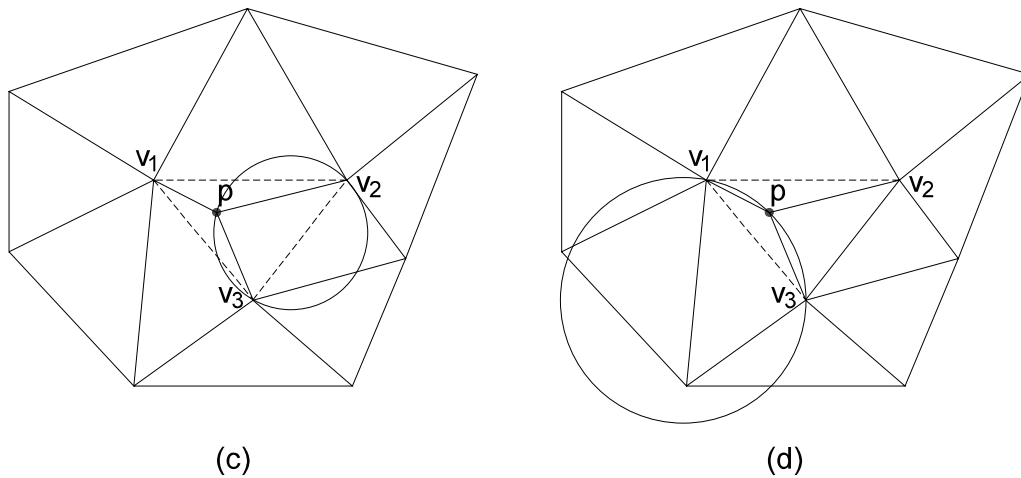
Obr. 2(a): Bod p vždy padne do některého trojúhelníka triangulace.

Vrcholy trojúhelníka T označme v_1 , v_2 a v_3 . Pak hrany pv_1 , pv_2 a pv_3 budou jistě patřit triangulaci (viz. obr. 2(b)).



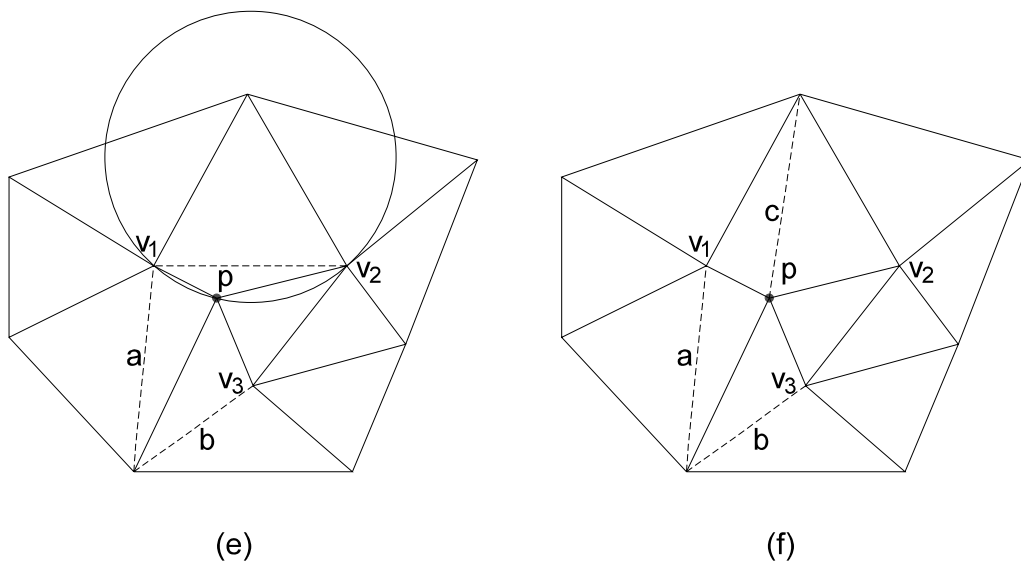
Obr. 2(b): Hrany pv_1 , pv_2 a pv_3 jistě patří triangulaci.

Nyní musíme rozhodnout, zda hrany v_1v_2 , v_2v_3 a v_3v_1 zůstanou v triangulaci či nikoli. Toto rozhodnutí bude závislé na hlavním kritériu Delaunayovy triangulace, tj. kružnice opsané trojúhelníkům pv_1v_2 , pv_2v_3 a pv_3v_1 nesmí obsahovat jiný bod množiny S (již přidáný do triangulace). Ukáže-li se tedy jedna z těchto tří hran nevyhovující, musíme provést v jejím okolí retriangularizaci splňující kritéria lokálního optima. Na obrázku 2(c) je test v pořádku a hrana v_2v_3 je ponechána v triangulaci, zatímco obrázek 2(d) ukazuje pozitivní výsledek testu (kružnice obsahuje jiný vrchol) hrany v_3v_1 .



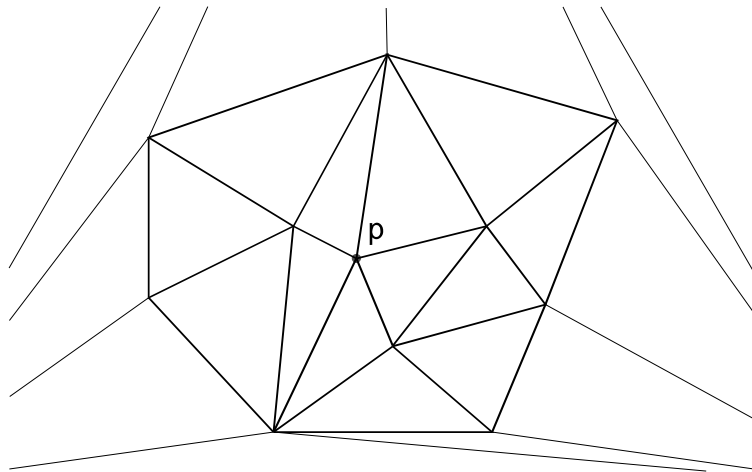
Obr. 2: (c) Negativní výsledek testu hrany v_2v_3 , (d) Pozitivní výsledek testu hrany v_3v_1 .

Musíme tedy provést retriangularizaci oblastí, jejichž jedna z hran je tvořena vrcholy ležícími na a uvnitř testované kružnice. Jako nové vyhovující hrany se nabízejí hrany a a b na obrázku 2(e), a hrana c po testu v_1v_2 , na obrázku 2(f).



Obr. 2: (e) Pozitivní test v_1v_2 , (f) Nové hrany a , b a c .

Přidáním hran a , b a c je dokončen proces vložení vrcholu p na obr. 2(g), a cyklus pokračuje vkládáním dalších bodů množiny S až do jejich vyčerpání.



Obr. 2(g): Po ukončení procesu vkládání bodu p .

Algoritmus tedy sestává z několika základních kroků:

procedure Incremental_Delaunay

- I. Inicializace: Generuj pomocný trojúhelník T_{pom} obsahující všechny body množiny S , a vlož ho do triangulace.
- II. Hlavní cyklus:
 - Vyber z S bod p a lokalizuj v triangulaci trojúhelník T , tvořený vrcholy v_1, v_2 a v_3 , ve kterém p leží.
 - Přidej do triangulace hrany pv_1, pv_2 a pv_3 .
 - Testuj hrany v_1v_2, v_2v_3 a v_3v_1 , tj. testuj, zda kružnice opsané trojúhelníkům pv_1v_2, pv_2v_3 a pv_3v_1 neobsahují jiný vrchol triangulace.
 - V případě pozitivního výsledku testu (kružnice obsahuje jiný vrchol) proved' retriangularizaci oblasti složené z trojúhelníků, jejichž jedna z hran je tvořena vrcholy ležícími na a uvnitř testované kružnice.
- III. Zruš v triangulaci pomocný trojúhelník T_{pom} a hrany spojené s jeho vrcholy.

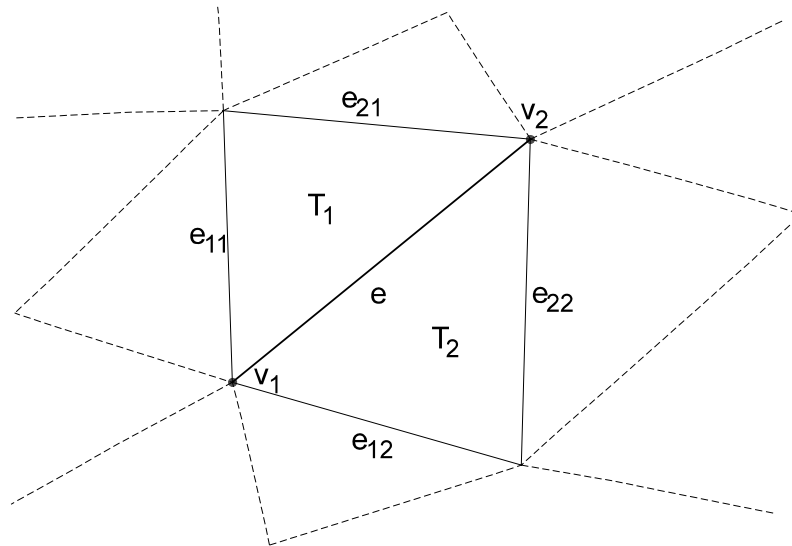
Úspěšnost a výpočetní složitost algoritmu závisí také na volbě datové struktury. Jednou z možných datových struktur se v tomto případě jeví „okřídlené hrany“.

3.8.1 Datová struktura „okřídlené hrany“

Položka této datové struktury [SKA92] obsahuje informaci nejen o jedné hraně, ale i o navazujících hranách, či o plochách, které tato hrana odděluje. Je možné uchovat i informaci o orientaci hran, či ploch.

V našem případě je situace zjednodušena tím, že reprezentovanými plochami jsou trojúhelníky. Datová struktura se tedy bude opírat o tři

seznamy: vrcholů (v_i), hran (e_j) a trojúhelníků (T_k). V dalším popisu budeme vycházet z obrázku 3.



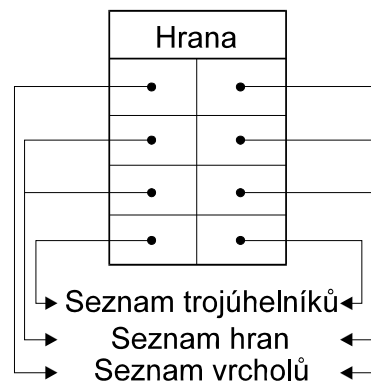
Obr. 3: Reprezentace „okřídlenými hranami“

Seznam vrcholů bude obsahovat všech N vrcholů v_i triangulace a jejich souřadnice.

Seznam hran bude obsahovat pro každou hranu e , orientovanou ve směru v_1v_2 , dva odkazy do seznamu vrcholů na samé koncové body, dva odkazy do seznamu trojúhelníků na „levý“ trojúhelník T_1 a „pravý“ trojúhelník T_2 a konečně čtyři odkazy do seznamu hran na předcházející hranu první po směru chodu hodinových ručiček e_{11} , předcházející hranu první proti směru chodu hodinových ručiček e_{12} , následující hranu první proti směru chodu hodinových ručiček e_{21} a následující hranu první po směru chodu hodinových ručiček e_{22} .

Seznam trojúhelníků bude pro každý trojúhelník triangulace obsahovat tři odkazy do seznamu hran na hrany tvořící jeho stěny.

Hrana e	
$\wedge v_1$	$\wedge v_2$
$\wedge e_{11}$	$\wedge e_{21}$
$\wedge e_{12}$	$\wedge e_{22}$
$\wedge T_1$	$\wedge T_2$



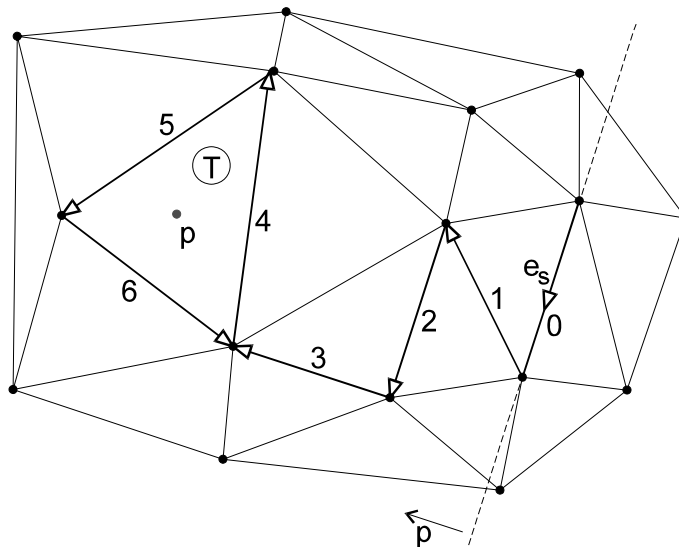
Obr. 4: Položka reprezentující hranu a její ukazatelé do seznamů.

Na základě znalosti použité datové struktury se nyní můžeme věnovat problému lokalizace trojúhelníku T , do kterého padl vkládaný bod p , tj. krok (2)(i) algoritmu, neboť právě na volbě použité struktury je závislá výpočtová složitost tohoto kroku a ta se bude velkou měrou podílet i na celkové výpočtové složitosti algoritmu. Jednou z použitelných metod pojmenujeme jako metodu „cesty po křídlech“.

Vyberme tedy z množiny S bod p a jednu z hran (libovolnou) triangulace označme e_s (startovní hrana). Pro popis cyklu samotné cesty načrtněme jednoduchý algoritmus.

procedure cesta

- krok (1) Inicializace: Za aktuální hranu dosad' e_s . Počáteční orientaci urči např. ze seznamu hran ve směru v_1v_2 .
- krok (2) Urči polorovinu danou přímkou v_1v_2 (levá či pravá vzhledem k orientaci hrany), ve které lží bod p .
- krok (3) Za aktuální hranu dosad' vzhledem k orientaci hrany následující ve směru poloroviny, v níž leží bod p . Opakuj od kroku (2), dokud nenarazíš na již jednou prošlou hranu.
- krok (4) Vrať nalezený trojúhelník (poslední tři hrany v cestě jsou jeho stranami).



Obr. 5: „Cesta po křídlech“ ze startovní hrany e_s až k trojúhelníku T . Pořadí hran v cestě je označeno čísly 0 až 6. Hrana 4 je jediná hrana, která se v cestě opakuje.

Výpočtová složitost lokalizace je $O(N)$. Tu je možno snížit až na $O(\log N)$, ale za cenu komplikovanější a paměťově náročnější datové struktury, či jiné techniky (problém lokalizace bodu v PSLG je řešitelný např. metodou

horizontálních pásů, řetězcovou metodou, viz. kapitola „Řetězcová metoda lokalizace bodu v PSLG“, metodou zjemňování triangulace, viz. kapitola „Lokalizace bodu metodou zjemňování triangulace“, a jinými). Celková výpočtová složitost našeho algoritmu je tedy řádu $O(N^2)$. Nicméně, v praxi je složitost lokalizace přeci jen nižší než $O(N)$. Toho může být dosaženo tím, že startovní hranu e_s neurčíme jako pevnou, či ji jinak nevolíme zcela náhodně, ale určíme jí na konci každého vyhledávacího cyklu jako v pořadí poslední hranu cesty po křídlech (což je samo o sobě i velmi jednoduchý způsob). V případě, že jsou vkládané body množiny S seříděny (např. jsou do souboru ukládány s rostoucí souřadnicí apod.), tj. momentálně vkládaný bod p leží velmi blízko bodu vloženému do triangulace v minulém cyklu, počet operací nutných pro lokalizaci bodu p klesá až na $O(N^{1/2})$. Dalším zmiňovaným faktem je, že počet vrcholů v triangulaci se po jednom zvyšuje ze čtyř až do $N + 3$ (tři vrcholy pomocného trojúhelníka) a N je vlastně počet vrcholů již patřících triangulaci a nikoli počet prvků množiny S .

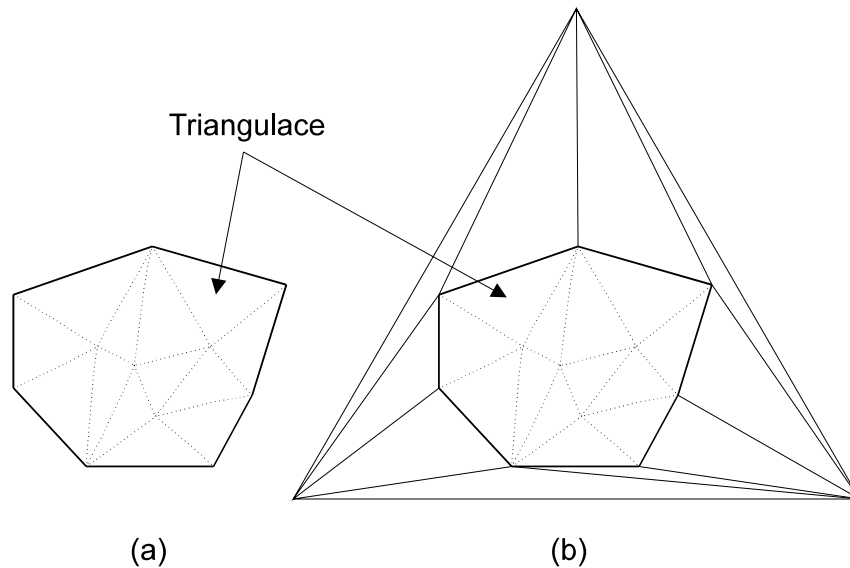
Použitelnou datovou strukturou je také DCEL (viz. kapitola DCEL), která potřebuje o dva ukazatele na položku méně, za předpokladu použití komplikovanějšího algoritmu lokalizace trojúhelníka T (viz. procedura VRCHOL(j) v kapitole DCEL).

Jedna z náročnějších (především paměťově) technik, která přináší snížení výpočtové složitosti lokalizačního algoritmu, je *metoda zjemňování triangulace*, kterou si nyní popíšeme.

3.8.2 Lokalizace bodu metodou zjemňování triangulace

Nejdříve si popíšeme tuto metodu obecně (Kirkpatrick, 1983) [PREP85] a pak si řekneme, jak tuto metodu aplikovat na náš problém, což bude ostatně zřejmé.

Předpokládejme, že PSLG o N vrcholech je triangulací. Tato triangulace, jak už jsme si řekli výše, má nejvýše $3N - 6$ hran. Opišme tuto triangulaci pomocným trojúhelníkem tak, že celý daný PSLG leží uvnitř tohoto trojúhelníka (viz. obrázek 6), jehož smysl bude brzy zřejmý.



Obr. 6: (a) Daná triangulace obsahuje nevyše $3N - 6$ hran.
(b) Daná triangulace opsaná pomocným trojúhelníkem obsahuje právě $3N - 6$ hran.

Doplněním dané triangulace o tento trojúhelník a o hrany spojující vrcholy pomocného trojúhelníka s vrcholy konvexní obálky dané triangulace vznikne nová trojúhelníková síť obsahující právě $3N - 6$ hran.

Je dána triangulace G o N vrcholech. Pak a je posloupnost triangulací $S_1, S_2, \dots, S_{h(N)}$, kde $S_1 = G$ a S_i dostaneme z S_{i-1} následovně:

- krok (i) Vyjmi z triangulace množinu vrcholů, a s nimi spjatých hran, takových, že žádné dva nejsou sousední a žádný netvoří vrchol konvexní obálky triangulace. (Způsobem výběru se budeme zabývat později.)
- krok (ii) Proveď retriangularizaci polygonů vzniklých vyjmutím vrcholů a jejich hran.

Poslední triangulace $S_{h(N)}$ tedy neobsahuje žádný vnitřní vrchol, tj. skládá se pouze z jednoho trojúhelníka.

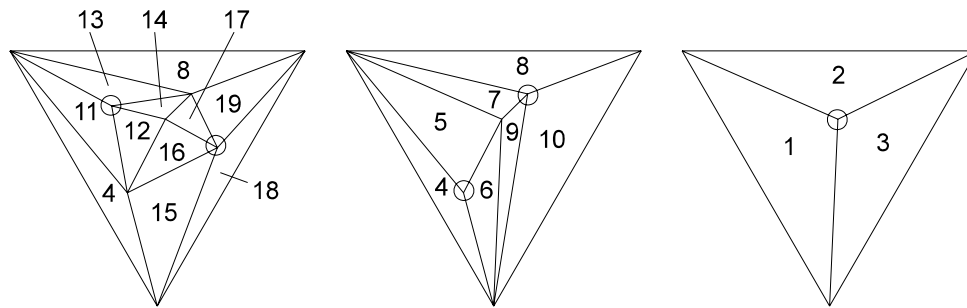
Poznamenejme, že jelikož v kroku (i) vyjímáme pouze interní vrcholy, všechny triangulace v posloupnosti mají totožnou konvexní obálku. Trojúhelník patřící stávající triangulaci S_i označme R_j , tj. $R_j \in S_i$, je-li R_j sestrojen krokem (ii) během konstrukce S_i .

Nyní vytvořme vyhledávací strom T , jehož uzly budou reprezentovat jednotlivé trojúhelníky triangulací. Uzel trojúhelníka R_k bude během konstrukce S_i z S_{i-1} spojen s uzlem trojúhelníka R_j v případě, že:

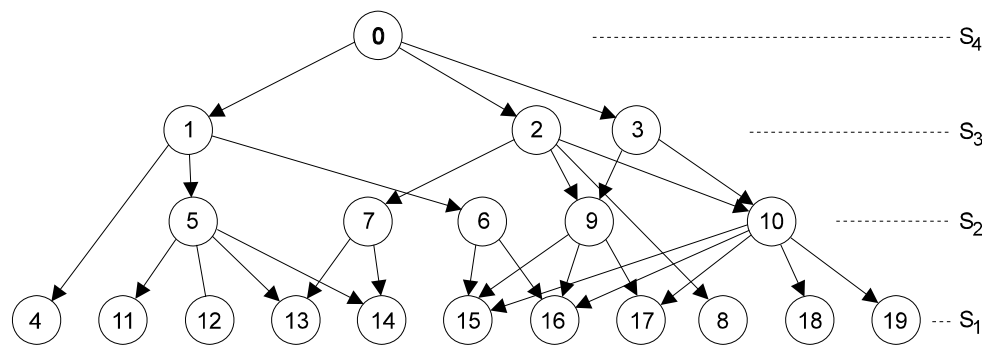
1. R_j je krokem (i) eliminován z S_{i-1} ;
2. R_k je sestrojen v S_i krokem (ii);

3. $R_j \cap R_k \neq \emptyset$.

Zřejmě, trojúhelníky triangulace jako jediní S_1 nemají žádné syny (jsou listy stromu T). Následujícími obrázky ilustrujeme proces eliminace vrcholů a konstrukce stromu T .



Obr. 7: Postupná eliminace vrcholů triangulace.



Obr. 8: Odpovídající vyhledávací strom T .

Nyní je zřejmé, jak bude probíhat lokalizace bodu. Nejdříve lokalizujeme tázaný bod v triangulaci $S_{h(N)}$. Pak procházíme stromem T , dokud nenarazíme na triangulaci S_1 , tj. na jeden z listů stromu. Konstrukce cesty se provede takto: dosáhneme-li nějaký uzel stromu (bod p leží v trojúhelníku s tímto uzlem spjatým), provedeme test jeho následovníků na p . Jelikož právě jeden z těchto následovníků v S_i obsahuje bod p , je jednoznačně určen další krok cesty tímto směrem. Vlastně zde dochází k lokalizaci p v jednom z trojúhelníků triangulace, který je v dalším kroku rozdělen na menší trojúhelníky, z čehož vychází název samotné metody zjemňování triangulace.

Přesněji, označme všechny následníky uzlu v v stromu T jako $NEXT(v)$ a trojúhelník odpovídající uzlu v jako $TRIANGLE(v)$. Lokalizační algoritmus pak bude vypadat takto:

```

procedure Lokalizace_bodu
begin
    if ( $p \notin TRIANGLE(kořen)$ ) then

```

```
    print "p leží vně ohraničené oblasti"  
else begin  
    v := kořen;  
    while (NEXT(v) ≠ NIL) do  
        for každý u ∈ NEXT(v) do  
            if (p ∈ TRIANGLE(u)) then v := u;  
        print v;  
    end  
end.
```

Podstatný vliv na výkonnost metody má způsob výběru množiny vrcholů, které budou vyjmuty z triangulace při konstrukci S_i z S_{i-1} . Předpokládejme, že jsme schopni tuto množinu vybrat tak, že označíme-li počet vrcholů triangulace S_i jako N_i , platí:

1. $N_i = \alpha_i N_{i-1}$, kde $\alpha_i \leq \alpha < 1$, pro $i = 2, \dots, h(N)$.
2. Každý tojůhelník $R_j \in S_i$ protíná nejvýše H trojúhelníků v S_{i-1} a naopak.

Z prvního bodu vyplývá, že $h(N) \leq \lceil \log_{1/\alpha} N \rceil = O(\log N)$. Z obou bodů dohromady plyne paměťová náročnost stromu T o velikosti $O(N)$. Tento prostor je použit pro uložení uzlů stromu a ukazatelů na jejich následníky. Jako důsledek Eulerova teorému

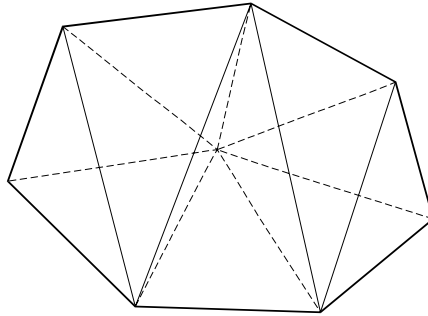
platí, že S_i obsahuje $F_i < 2N_i$ trojúhelníků. Počet uzlů stromu T , reprezentujících trojúhelníky v S_i , je nejvýše F_i (v odpovídající vrstvě se objeví pouze trojúhelníky patřící momentálně do S_i), odkud počet uzlů, které se objeví ve stromu T , je

$$2(N_1 + N_2 + \dots + N_{h(N)}) \leq 2N_1(1 + \alpha + \alpha^2 + \dots + \alpha^{h(N)-1}) < 2N / (1 - \alpha),$$

přičemž poznamenejme, že $N_1 = N$. Z hlediska paměťového prostoru potřebného pro uložení ukazatelů: z druhého bodu vyplývá, že každý uzel obsahuje nejvýše H ukazatelů, odkud dostáváme, že strom T obsahuje méně než $2HN / (1 - \alpha)$ ukazatelů.

Nyní se budeme zabývat samotným kritériem pro výběr množiny vyjímáných vrcholů. Toto kritérium zní: „Vyjmi z triangulace množinu vrcholů takových, že žádné dva nejsou sousední a žádný netvoří vrchol konvexní obálky triangulace a přitom jsou stupně menšího než K (K je celé číslo)“. Pořadí vyjímáných vrcholů je libovolné.

Nyní je druhý bod jasný. Odstraněním vrcholu stupně menšího než K dostaneme polygon s počtem hran také menším než K a každý z nahrazených trojúhelníků protíná nejvýše $K - 2 \approx H$ nových trojúhelníků (viz. obrázek 9).



Obr. 9: Vypuštění vrcholu stupně sedm a následná retiangulace polygonu o sedmi hranách.

Pro ověření prvního bodu je třeba použít některé z vlastností planárních grafů:

- i. Z Eulerovy formule o počtu hran triangulace sestavené na množině N bodů v rovině, speciálně triangulace, jejíž konvexní obálku tvoří trojúhelník, platí (e značí počet těchto hran):

$$e = 3N - 6.$$

- ii. Kromě případu, kdy naše triangulace neobsahuje žádné vnitřní body (tvoří ji pouze jeden trojúhelník a řešení je tedy triviální), je každý ze tří vrcholů konvexní obálky nejméně třetího stupně. Jelikož zde máme $3N - 6$ hran a každá přispívá k celkovému stupni vrcholu dvěma, je tento celkový stupeň $< 6N$. Z toho je vidět, že alespoň $N/2$ vrcholů je stupně menšího než 12. Proto necht' $K = 12$. Necht' v je počet vrcholů vybraných k vyjmutí. Jelikož každý vybraný vrchol může eliminovat nejvýše $K - 1 = 11$ sousedních vrcholů a tři vrcholy konvexní obálky jsou nevyjmutelné, dostáváme:

$$v \geq \lfloor 1/11(N/2 - 3) \rfloor.$$

Pak vychází, že $\alpha \cong 1 - 1/22 < 0.955 < 1$.

☞ Výsledek: Lokalizace bodu může tedy tímto algoritmem proběhnout v čase $O(\log N)$, při paměťové náročnosti $O(N)$ po předzpracování o složitosti $O(N \log N)$.

Vrátíme-li se nyní k problému Delaunayovy přírůstkové triangulace, vidíme určité rysy společné pro oba algoritmy. Metoda lokalizace bodu zjemňováním triangulace je založena na dekompozici dané triangulace až na její trojúhelníkovou obálku. Metoda Delaunayovy přírůstkové triangulace zase krok za krokem triangularizuje vnitřek také trojúhelníkové oblasti. Spojení těchto dvou technik (tj. Delaunayova

přírůstková triangularizace využívající pro lokalizaci bodu Kirkpatrickovu metodu zjemňování triangulace) je realizováno tak, že vyhledávací strom T potřebný pro lokalizační proceduru je vytvářen a aktualizován postupně, během konstrukce triangulace.

Bohužel, při velkém počtu zpracovávaných bodů je, díky nutnosti uchování trojúhelníků stromu (tj. trojúhelníků jednotlivých triangulací S_i), potřebný paměťový prostor velmi vysoký a za zmiňku stojí i fakt, že kromě případu, kdy přidáním bodu p do trojúhelníka triangulace dojde k jejímu rozšíření pouze o hrany spojující p s vrcholy tohoto trojúhelníka (v tomto případě vytvoříme pouze tři nové následníky, kteří budou zároveň listy stromu, uzlu reprezentujícímu tento trojúhelník), tj. v případech, kdy je nutno retriangularizovat i okolí tohoto trojúhelníka, se zkomplikuje procedura aktualizace stromu (je nutno aktualizovat i sousedící trojúhelníky). A navíc, tento „zdola“ konstruovaný strom nebude sestaven tak ideálně, jako původní metodou zjemňování triangulace a vyhledávání v něm proto bude o něco horší než $O(\log N)$.

I když se toto spojení nakonec pro použití v praxi ukázalo méně vhodné, je zde metoda zjemňování triangulace uvedena, neboť je se samotným problémem triangularizace úzce spjata.

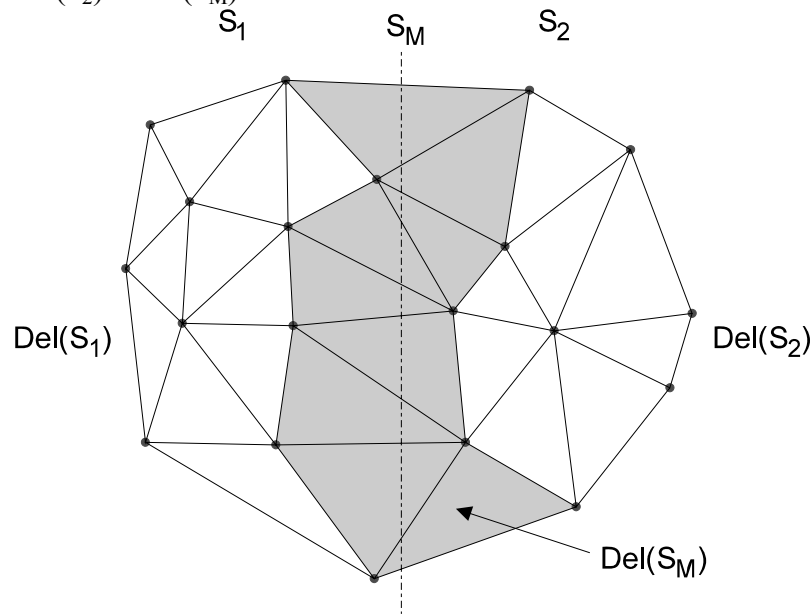
3.9 Triangularizační algoritmus DeWall (Delaunay Wall)

3.9.1 Realizace

Triangularizační algoritmus DeWall (Delaunayova zeď) je založen na metodě rozděl-a-panuj, kterou ovšem využívá poněkud jiným způsobem, než klasický algoritmus pro konstrukci Delaunayovy triangulace (či Voronoiova diagramu). V klasickém triangularizačním algoritmu založeném na metodě rozděl-a-panuj je nejsložitějším problémem fáze spojování (spojení dvou triangulací $Del(S_1)$ a $Del(S_2)$, kde S_1 je levá polovina množiny S a S_2 je pravá polovina množiny S - viz. kapitola „Konstrukce Voronoiova diagramu metodou rozděl-a-panuj“). Řešení tohoto problému vyžaduje mnoho lokálních úprav (retriangularizací) spojovaných triangulací. Tyto úpravy jsou proveditelné v E^2 , ovšem ve vyšedimenzionálních prostorech jsou prakticky neřešitelné. Tento problém řeší algoritmus DeWall úpravou konvenční metody rozděl-a-panuj, takže je nejen dobře pracující v rovině, ale může se stát odrazovým můstkem do E^d , kde $d > 2$.

Základní myšlenka je jednoduchá. Místo spojování partiálních triangulací, jako je tomu u klasické rozděl-a-panuj metody, triangularizuje DeWall nejdříve prostor rozdělení původní množiny generátorů S , čímž vzniká jakýsi pás trojúhelníků (triangulační zeď) rozdělující S na dvě poloviny a pak teprve počítá triangulace těchto podmnožin S_1 a S_2 .

Nechť tedy S je množina N generátorů, na které bude sestrojena Delaunayova triangulace $\text{Del}(S)$. Proces dělení rozdělí množinu S na množiny S_1 a S_2 , které mají přibližně stejný počet prvků (proces dělení probíhá jako u klasické metody, jen s tím rozdílem, že se střídá dělení vertikální a horizontální). Pak je provedena triangularizace generátorů, označme tuto množinu S_M , ležících podél dělicí přímky tak, že vznikne pás trojúhelníků, $\text{Del}(S_M)$, vyhovujících kritériu prázdné Delaunayovy kružnice a každý z těchto trojúhelníků má nenulový průnik s dělicí přímkou. Pak se rekurzivně triangularizují množiny S_1 a S_2 . Výsledná triangulace $\text{Del}(S)$ vznikne spojením $\text{Del}(S_1)$, $\text{Del}(S_2)$ a $\text{Del}(S_M)$.



Obr. 1: DeWall triangularizace: $\text{Del}(S_M)$ je tzv. Delaunayova zeď.

Kostra algoritmu bude tedy vypadat takto:

procedure DeWall

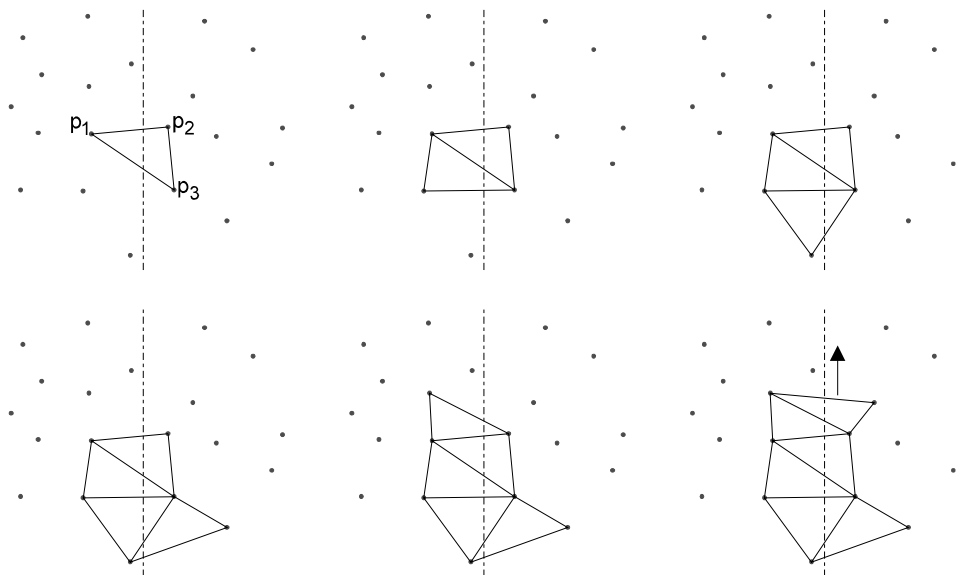
- I. Pokud triangulace množiny S není dokončena, rozděl S na přibližně stejně mohutné množiny S_1 a S_2 přímkou α .
- II. Sestroj Delaunayovu zeď $\text{Del}(S_M)$.
- III. Pokud triangulace množiny S_1 není dokončena, aplikuj na množinu S_1 rekurzivně proceduru DeWall.
- IV. Pokud triangulace množiny S_2 není dokončena, aplikuj na množinu S_2 rekurzivně proceduru DeWall.
- V. Vrať $\text{Del}(S_1) \cup \text{Del}(S_M) \cup \text{Del}(S_2)$.

Jak je vidět z procedury DeWall, jádrem metody je krok (2), tedy sestavení Delaunayovy zdi $\text{Del}(S_M)$. Ke konstrukci této zdi použijeme jednoduchý způsob přírůstkové triangularizace založený na udržování platnosti kritéria prázdné Delaunayovy kružnice. Konstrukci začneme

v jednom z trojúhelníků $\text{Del}(S)$, který protíná dělicí přímka. To znamená, že první trojúhelník na startu algoritmu je nutno vygenerovat. To provedeme tímto způsobem:

1. Najdeme bod $p_1 \in S_1$ ležící blízko k dělicí přímce.
2. Najdeme bod $p_2 \in S_2$ (leží na opačné straně dělicí přímky) s nejmenší Euklidovskou vzdáleností od bodu p_1 .
3. Najdeme bod $p_3 \in S$ takový, že uvnitř kružnice procházející vrcholy p_1, p_2 a p_3 neleží žádný bod množiny S .
4. Trojúhelník $p_1p_2p_3$ je hledaný startovní trojúhelník.

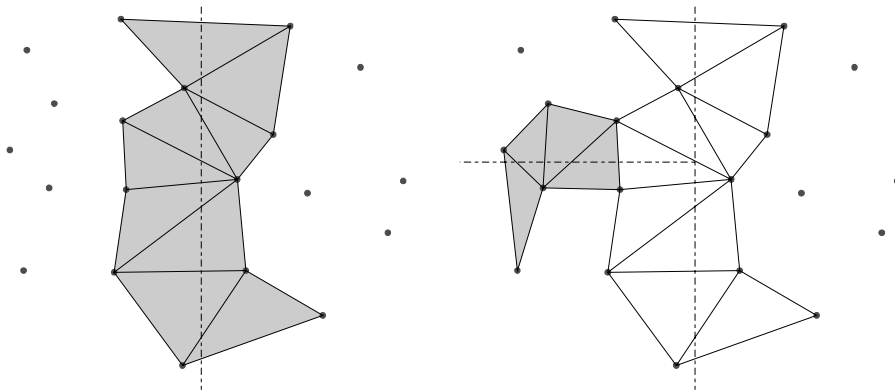
V konstrukci Delaunayovy zdi pak pokračujeme tímto způsobem (viz. následující obrázek 2).



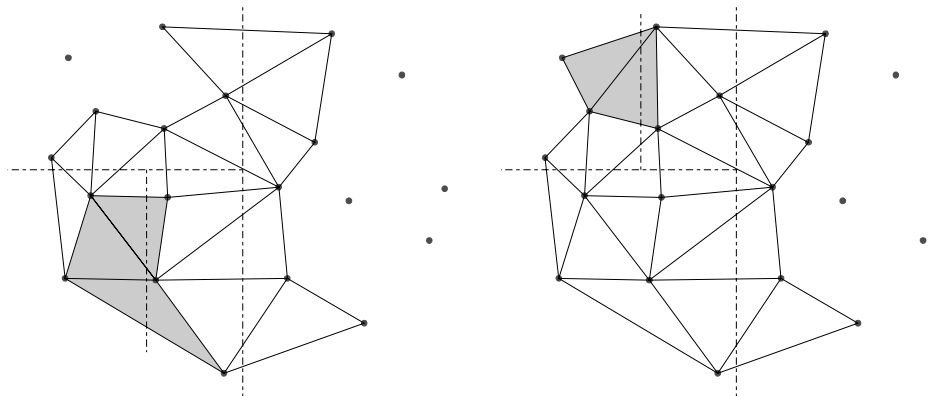
Obr. 2: Konstrukce první Delaunayovy zdi $\text{Del}(S_M)$.

Dělicí přímka protíná (vyhneme-li se opět úvahám o speciálních případech) dvě hrany trojúhelníka. V konstrukci pokračujeme směrem jedné z těchto hran tak, že hledáme takový bod množiny S , který bude pro danou hranu vyhovovat kritériu prázdné Delaunayovy kružnice. Tak pokračujeme dokud nenarazíme na hranu, která náleží konvexní obálce množiny S . Pak se vrátíme ke startovnímu trojúhelníku a postup opakujeme v opačném směru. Takto tedy sestrojíme první Delaunayovu stěnu. Konstrukce následujících stěn je jednodušší, neboť startovní trojúhelník je již daný (je to trojúhelník z předchozí Delaunayovy zdi, ve kterém se protínají na sebe kolmé dělicí přímky předchozí a aktuální množiny S) a konstrukce probíhá pouze jedním směrem (vlevo při dělení S_1 a vpravo při dělení S_2).

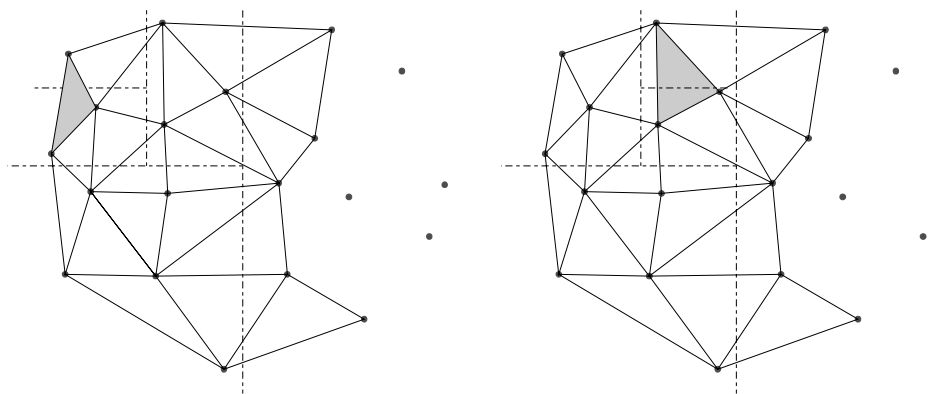
Samotnou proceduru DeWall ilustruje následující sekvence obrázků.



Obr. 3: Procedura DeWall: konstrukce první Delaunayovy zdi a první rekurzivní volání procedury.



Obr. 4: Procedura DeWall: na levém obr. již nedojde k rekurzivnímu volání procedury, neboť triangularizace stávající množiny S je již provedena (při konstrukci Delaunayovy zdi již v prvním kroku nenalezneme žádný bod, který by vyhovoval kritériu prázdné kružnice opsané a zároveň by nepatřil do již hotové triangulace).



Obr. 5: Procedura DeWall: dokončení triangularizace levé části množiny generátorů (tj. množiny S_1 po prvním dělení). Konstrukce pravé části je analogická.

3.9.2 Závěrem:

Změna, kterou se tento algoritmus liší od algoritmu pro konstrukci Delaunayovy triangularizace či Voronoiova diagramu založeného na klasické metodě rozděl-a-panuj, nepřináší změnu ve výpočetní složitosti a ta tedy zůstává $O(N \log N)$. Díky ní se však algoritmus DeWall vyhýbá složité slučovací fázi, která brání v použití metody rozděl-a-panuj ve vícedimenzionálním prostoru, než je rovina. Proto je možné přejít od planárního DeWall algoritmu k algoritmům určeným pro konstrukci trojúhelníkové sítě na množinách vstupních generátorů rozmístěných v E^d , kde $d > 2$. Druhou výhodou, kterou tato metoda přináší oproti ostatním klasickým algoritmům díky nepřítomnosti fáze spojování parciálních triangulací, je možnost snadnější paralelizace algoritmu [CIG92].

3.10 Delaunayova triangularizace využívající pravidelnou mříž

Nové rysy tohoto algoritmu [FANG93]:

- Je to algoritmus založený na přímé konstrukční metodě s menší režií, než mají algoritmy založené na metodě rozděl-a-panuj (správa zásobníku, dělení množiny generátorů, spojování partikulárních výsledků). Nevyžaduje také třídění vstupní množiny.
- Pro konstrukci trojúhelníků a především lokalizaci bodů využívá pravidelnou mřížku.
- Trojúhelníky generuje kruhovým „odkrajováním“ z množiny generátorů. Tím je zajištěna kompletnost a korektnost triangulace. Výstupní datová struktura je generována spolu s konstrukcí nových trojúhelníků.
- Používá seznam hran.
- Je odolný proti koincencím a kocirkularitám.
- Výpočtová složitost algoritmu je lineární funkcí.

Následující část se bude zabývat otázkou předzpracování. Pak se budeme věnovat interní datové struktuře založené na pravidelné mřížce, samotným triangularizačním procesem a využitím algoritmu k výpočtu konvexní obálky.

3.10.1 Předzpracování

Jako obvykle, vstupní množinou S je množina N generátorů v rovině. Snažíme se vytvořit takovou datovou strukturu pro tyto generátory, která nám umožní rychlou lokalizaci bodu, tj. máme hranu ab a chceme rychle najít bod

c takový, že trojice abc tvoří Delaunayův trojúhelník. A právě za tímto účelem sestrojíme pravidelnou planární mříž.

Prvním krokem této konstrukce je sestrojení *min-max boxu* obsahující všechny body množiny generátorů. Koincidenční toleranci (tolerance přesné polohy bodu při numerických výpočtech) označme TOL . Tato tolerance zvětší následujícím způsobem *min-max box* (kvůli řešení situací, kdy bod bude ležet na hranách mřížky).

$$\begin{aligned}x_{\min} &= x_{\min} - TOL \\x_{\max} &= x_{\max} + TOL \\y_{\min} &= y_{\min} - TOL \\y_{\max} &= y_{\max} + TOL\end{aligned}$$

Velikost mřížky je pak:

$$velikost = \sqrt{\frac{(x_{\max} - x_{\min})(y_{\max} - y_{\min})}{N}}$$

Počet buněk mřížky ve směru osy x (resp. osy y) je dán x -ovým rozlišením (resp. y -ovým rozlišením) mřížky:

$$\begin{aligned}x_{res} &= \text{int}\left(\frac{x_{\max} - x_{\min}}{velikost}\right) + 1 \\y_{res} &= \text{int}\left(\frac{y_{\max} - y_{\min}}{velikost}\right) + 1\end{aligned}$$

kde $\text{int}()$ je operátor konvertující double na integer.

Když je mřížka vygenerována, přiřadíme každému bodu množiny S právě jednu buňku mřížky. Předpokládejme, že bod p_i má souřadnice (x_i, y_i) , pak pro $i = 0, 1, 2, \dots, N$ proved'

(1)

$$\begin{aligned}grid_x &= \frac{x_i - x_{\min}}{velikost} \\grid_y &= \frac{y_i - y_{\min}}{velikost} \\i_cell &= \text{int}(grid_x) \\j_cell &= \text{int}(grid_y)\end{aligned}$$

kde $grid_x$ a $grid_y$ jsou souřadnice normalizované vzhledem k původním souřadnicím (x_{\min}, y_{\min}) a i_cell a j_cell jsou indexy buňky.

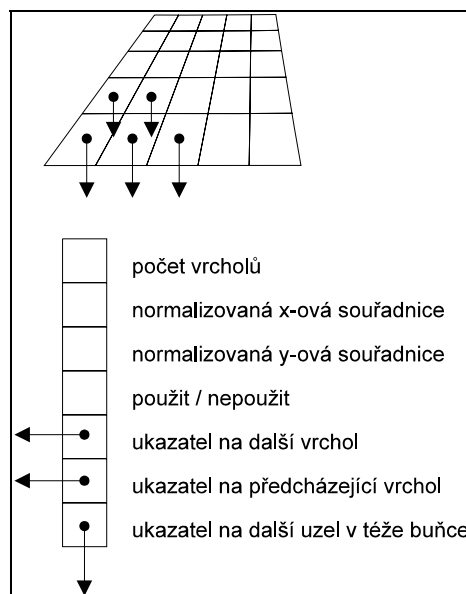
(2) Je-li buňka o indexech (i_cell, j_cell) prázdná, vlož do ní bod p_i .

- (3) Obsahuje-li již buňka o indexech (i_cell, j_cell) nějaký bod, testuj jej s novým bodem na koincidenci (má-li stejnou polohu v mezích TOL). Je-li nový bod s některým bodem buňky koincidentní, ignoruj ho, jinak nový generátor do buňky přidej.

Takto je každému bodu množiny S (vyjma koincidentním bodům, které jsou ignorovány) přiřazena právě jedna buňka mřížky. Leží-li nějaký generátor na hraně mřížky, vyber buňku sousedící s inkriminovanou hranou vpravo (pro vertikální hranu) a nahoře (pro horizontální hranu). Pro generátor ležící na horním okraji (resp. pravém okraji) mřížky je vybrána buňka dole (resp. vlevo) od tohoto okraje.

3.10.2 Datová struktura

Datová struktura užívaná tímto algoritmem je matice ukazatelů, kde každý pointer ukazuje do seznamu bodů ležících ve stejné buňce.



```
struct cellnode
{
    int vertex_number;
    double x, y;
    int used;
    struct cellnode *previous_point;
    struct cellnode *next_point;
    struct cellnode *next_node;
}
```

Obr. 1: Datová struktura využívající pravidelnou mřížku.

Proměnná *vertex_number* tedy udává číslo generátoru, *x* a *y* jsou normalizované souřadnice bodu, proměnná *used* je praporek informující, zda je bod užít v triangulaci či nikoli, *previous_point* a *next_point* tvoří řetězec bodů užítých v triangulaci a *next_node* ukazuje na další bod ležící ve stejné buňce.

3.10.3 Triangularizační proces

Samotný triangularizační proces se skládá ze tří základních kroků, kterým se postupně budeme věnovat. Jsou to:

1. Nalezení startovního generátoru a první hrany.
2. Konstrukce trojúhelníků.
3. Spojení těchto trojúhelníků do sítě.

3.10.3.1 Krok 1: Nalezení startovního generátoru a první hrany

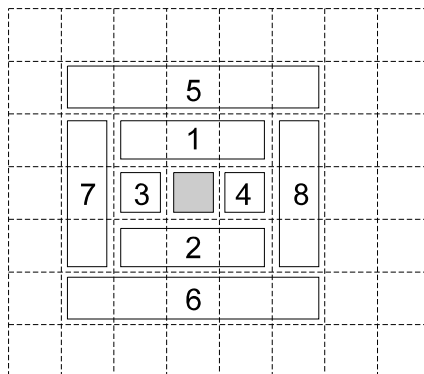
Otázka nyní zní: „Kde, či přesněji jakým bodem a jakou hranou, by triangularizační proces měl začít?“. Tento algoritmus může odstartovat odkudkoli, z hlediska výpočtové složitosti je však nejvýhodnější odstartovat co nejbližší ke středu množiny generátorů. Algoritmus a ilustrační obrázek (obr. 2) následují:

- (1) Najdi prostřední buňku s indexy:

$$(m, n) = (x_{res}/2, y_{res}/2)$$

- (2) V případě, že tato buňka není prázdná, vyber libovolný její bod (vrchní položku seznamu generátorů této buňky).
- (3) Je-li tato buňka prázdná, pokračuj v hledání v sousedních buňkách.

Výběr sousední buňky v kroku (3) je možno provést více způsoby. Jedním z nich je jednoduchý postup, ve kterém nejdříve prohledáme tři horní sousední buňky, pak tři dolní, levého souseda, pravého souseda a není-li ani nyní vyhledávací procedura úspěšná, pokračujeme horní pětící, dolní pětící, levou trojicí, pravou trojicí atd., viz. obrázek.



Obr. 2: Hledání startovního bodu.

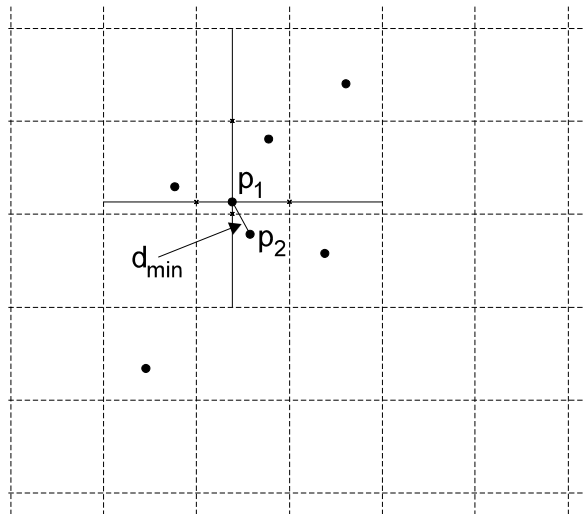
Jedinou nepříjemností, kterou může tento vyhledávací algoritmus způsobit, je když určí jako startovní bod generátor ležící blízko hranice *min-max boxu*.

Tento problém se však dá snadno vyřešit.

Máme-li startovní bod, začneme hledat nejbližší sousední bod, abychom mohli sestrojít první hranu triangulace:

- I. Nastavíme nejmenší vzdálenost d_{\min} na nějaké velké číslo, např. délku diagonály *min-max boxu*.
- II. Obsahuje-li nalezená buňka kromě startovního bodu i jiné generátory, najdeme ten z nich nejbližší k startovnímu bodu (označme jej p_1) a označme jako d jejich vzdálenost. Je-li d menší než d_{\min} , nastavíme d_{\min} jako d .
- III. Procházíme řádky a sloupce okolo buňky stejným způsobem, jako v minulé proceduře (tři horní, pak tři dolní sousední buňky, levého souseda, pravého souseda, atd.).
- IV. Pro každou řadu a sloupec provedeme následující:
 - Spust' z p_1 kolmici na stranu buňky k p_1 nejbližší.
 - Je-li vzdálenost mezi p_1 a průsečíkem kolmice se stranou buňky menší než d_{\min} , prohledej řadu či sloupec ve směru kolmice. Není-li tomu tak, označ tento směr (horní řada, dolní řada, levý sloupec, nebo pravý sloupec) jako neúspěšný.
 - Jsou-li v daném směru nalezeny nějaké body, vypočítáme jejich vzdálenost od p_1 a je-li tato vzdálenost menší než d_{\min} , aktualizujeme jej.
- V. Vyhledávání ukončíme, jsou-li všechny směry označeny jako neúspěšné, což znamená, že všechny body ležící v neprohledaných (označených) směrech jsou k p_1 vzdálenější, než ten se vzdáleností d_{\min} .

Nalezený bod, označme jej jako p_2 , leží tedy k bodu p_1 nejbliže a hrana p_1p_2 bude hledanou startovní hranou.



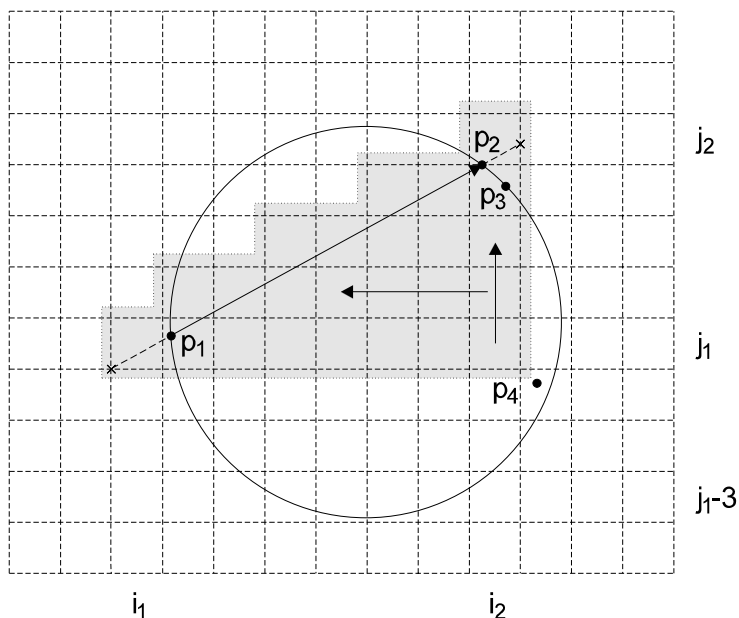
Obr. 3: Hledání první hrany p_1p_2 .

3.10.3.2 Krok 2: Konstrukce trojúhelníků

Máme tedy nalezen startovní bod p_1 a hranu p_1p_2 triangulace. K nalezení třetího bodu p_3 takového, že trojúhelník $p_1p_2p_3$ vyhovuje Delaunayovu kritériu kružnice opsané, použijeme následující algoritmus:

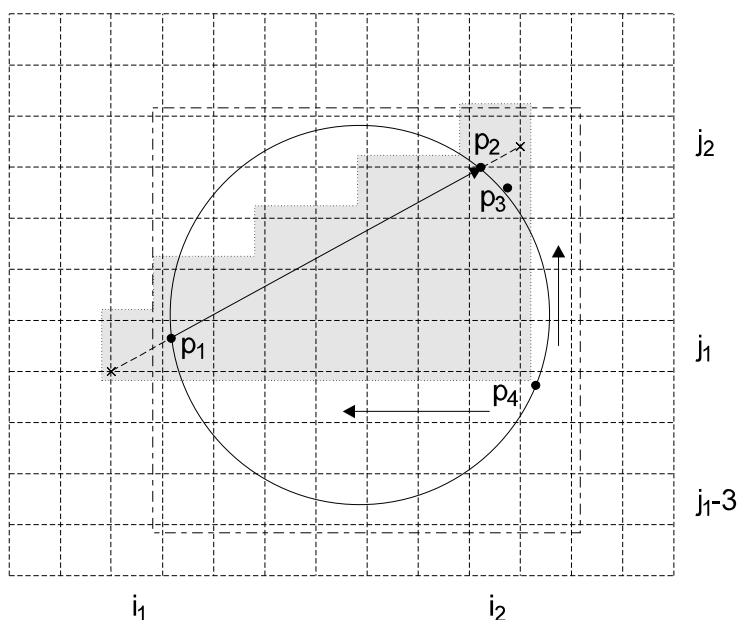
- I. Hledej na pravé straně úsečky p_1p_2 (předpokládáme orientaci z p_1 do p_2).
- II. Najdi buňky s indexy (i_1, j_1) a (i_2, j_2) takové, že jsou buď buňkami koncových bodů p_1p_2 , nebo jejich bezprostředními sousedy a to tímto způsobem. Vypočítej průsečík spodního okraje buňky obsahující bod p_1 s přímkou p_1p_2 . Tento průsečík definuje buňku (i_1, j_1) . Podobně vypočítej průsečík pravého okraje buňky obsahující bod p_2 s přímkou p_1p_2 . Tento průsečík definuje buňku (i_2, j_2) . Tento proces ilustruje obrázek 4.
- III. Vytvoř trojúhelníkovou oblast sestavenou z buněk tak, že buňky (i_1, j_1) , (i_2, j_1) a (i_2, j_2) tvoří vrcholy tohoto trojúhelníka (viz. vyplněná oblast na obr. 4). Jako diagonální stranu algoritmus vybírá všechnu buňky, které jsou pťrotnuty přímkou p_1p_2 . (Výpočet průsečíků je díky použití mřížky jednoduché, neboť stačí vypočítat sklon přímky, první průsečík a pak už jen postupovat mřížkou pouze pomocí operace sčítání k dalšímu průsečíku.)
- IV. Je-li vytvořena tato trojúhelníková oblast, začneme testovat každou buňku uvnitř oblasti. Prohledávání může být opět provedeno několika způsoby. Algoritmus může například prohledávat každý sloupec startující v buňce (i_2, j_1) a postupovat doleva, dokud nenarazí na buňku (i_1, j_1) (viz. obrázek 4). Další možností je prohledat buňku (i_2, j_1) a pak postupovat po diagonálách, tj. buňkami (i_2, j_1+1) , (i_2-1, j_1) , pak (i_2, j_1+2) , (i_2-1, j_1+1) ,

$(i_2 - 2, j_1)$, atd. Tak algoritmus vyhledává nejdříve body tvořící trojúhelník s co největšími úhly. Pro tento algoritmus je použit velmi jednoduchý způsob vyhledávání, neboť v praxi je počet prohledávaných buněk menší než deset.



Obr. 4: Konstrukce delaunayova trojúhelníka.

- (5) Jsou-li nějaké body nalezeny, vyber ten z nich, který dává největší úhel (tj. nejmenší kosinus). Vezmi kružnici procházející body p_1, p_2 a naším bodem s nejmenším kosinem a sestroj její *min-max box* (jeho strany jsou limitovány hranami mřížky). Není-li žádný bod nalezen, prohledej řádky od $(i_1 - k, j_1 - k)$ do $(i_2 + k, j_1 - k)$, a sloupce od $(i_2 + k, j_1 - k)$ do $(i_2 + k, j_2 + k)$, pro $k = 1, 2, \dots$, dokud nové body nalezeny nejsou.
- (6) Poté, co je sestrojen první *min-max box* (kružnice), pokračuje hledání řádky a sloupce uvnitř tohoto *min-max boxu* a postupujeme stejně jako v bodě (5), tj. vyber bod s nejmenším kosinem, sestroj kružnici procházející tímto bodem a generátory p_1, p_2 a aktualizuj *min-max box* (této kružnice). Tento proces ilustruje obrázek 5.
- (7) Prohledávání ukonči v případě, že uvnitř *min-max boxu* stávající kružnice už nejsou žádné nenavštívené řádky a sloupce.



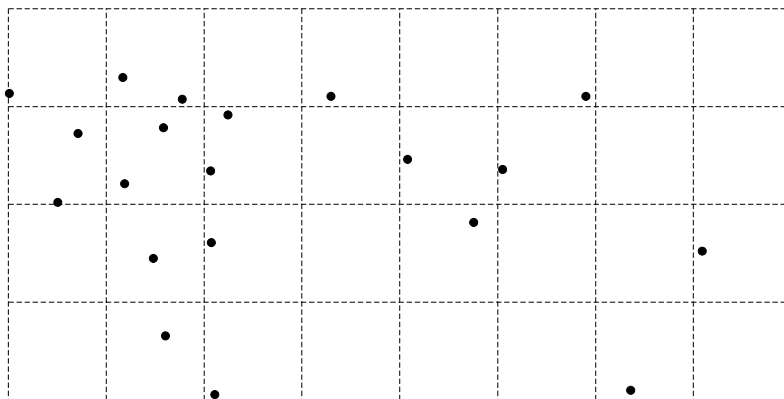
Obr. 5: Prohledávání buňek uvnitř obdélníku opsaného Delaunayově kružnici.

Hlavní částí algoritmu je dynamická aktualizace obdélníku opsaného Delaunayově kružnici.

Je-li hrana vertikální či horizontální, pro vyhledávání se místo trojúhelníkové oblasti použije oblast obdélníková.

3.10.3.3 Krok 3: Spojování sestrojených trojúhelníků

Jak spojit sestrojené trojúhelníky do sítě, aby vznikla úplná a korektní triangulace? Tak zní otázka, kterou nyní musíme vyřešit. Účinnou cestou je najít první trojúhelník a pak k němu přidávat další a další tak, aby nevznikly žádné díry ani mosty. Triangularizační strategie prezentovaného algoritmu přidává trojúhelníky kruhově. Všechny kroky tohoto algoritmu budou v následující části kapitoly vysvětleny na konkrétním příkladu, počínaje obrázkem 6.

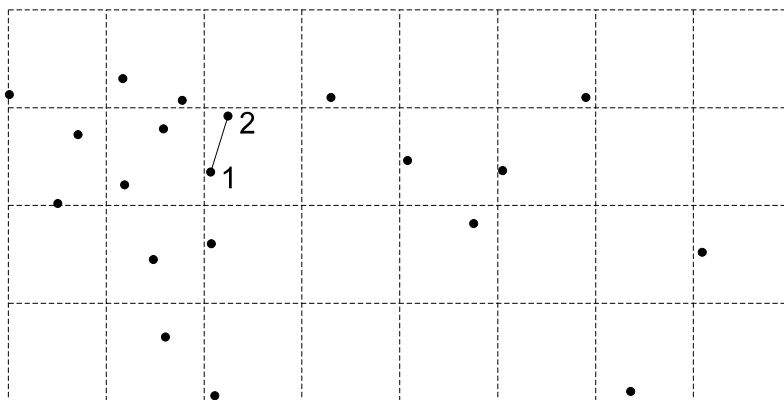


Obr. 6: Testovací množina generátorů a vygenerovaná mřížka.

Nalezení první hrany p_1p_2 je ilustrováno obrázkem 7. Algoritmus tuto hranu rozdělí na dvě „polohrany“ p_1p_2 a p_2p_1 . Ty pak vloží do seznamu hran, který je tvořen kruhovou frontou. Tuto frontu algoritmus používá po celý triangulační proces pro správu stávající hrany, kterou použije k nalezení dalšího trojúhelníka. Obsah fronty po inicializaci je tedy

$$p_2p_1, p_1p_2,$$

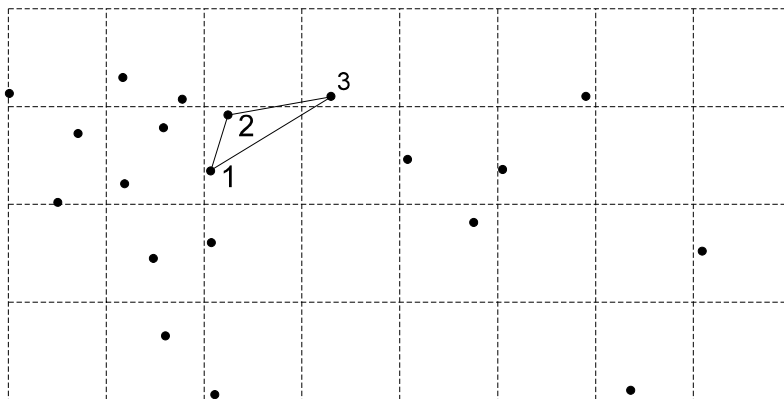
kde stávající aktivní hrana je ve frontě vždy poslední. (V dalším bude platit, že začátek fronty je první položka a konec je poslední položka.)



Obr. 7: Generování první hrany.

Pomocí stávající aktivní hrany p_1p_2 algoritmus najde bod p_3 a sestrojí dvě nové polohrany p_3p_2 a p_1p_3 (viz. obrázek 8). Seznam nových polohran je vždy takto uspořádaný: První polohrana ukazuje z nalezeného bodu na konec aktivní polohrany a druhá polohrana ukazuje ze začátku aktivní polohrany na nalezený bod. Jelikož polohrana p_1p_2 již není užívána, algoritmus ji vymaže z fronty a naopak na konec fronty přidá nové polohrany, takže máme:

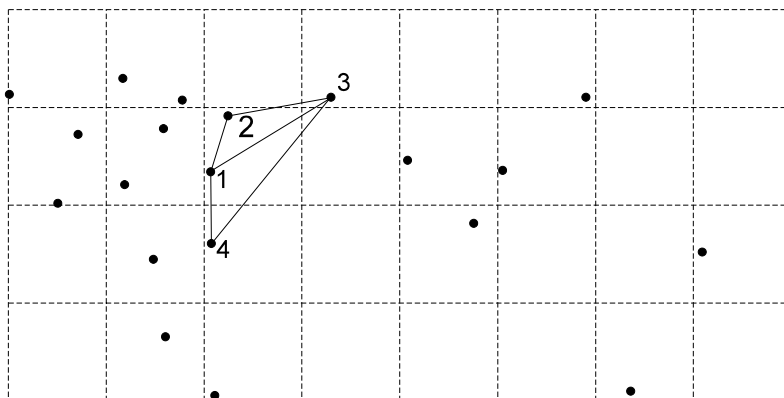
$$p_2p_1, p_3p_2, p_1p_3.$$



Obr. 8: Sestrojení prvního trojúhelníka.

Aktivní hranou je nyní p_1p_3 . Algoritmus tuto hranu nyní použije k nalezení bodu p_4 a polohran p_4p_3 a p_1p_4 (viz. obrázek 9). Po vymazání aktivní hrany a přidání nových polohran máme:

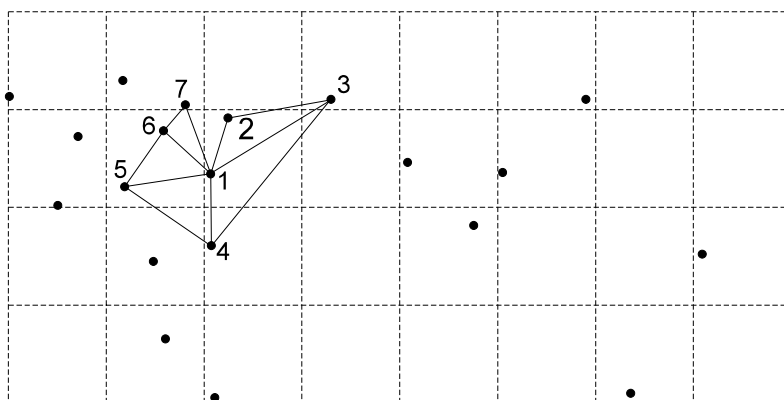
$$p_2p_1, p_3p_2, p_4p_3, p_1p_4.$$



Obr. 9: Sestrojení druhého trojúhelníka pomocí nových polohran.

Pokračováním v tomto procesu, tj. použití aktivní polohrany k vytvoření dvou nových polohran a aktualizace seznamu hran, se dostaneme až k situaci ilustrované obrázkem 10. Seznam hran zde je:

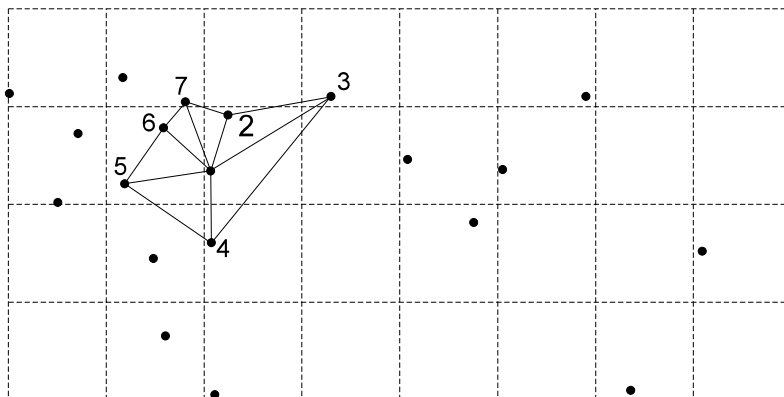
$$p_2p_1, p_3p_2, p_4p_3, p_5p_4, p_6p_5, p_7p_6, p_1p_7.$$



Obr. 10: Situace před uzavíráním triangulace.

Pomocí aktivní hrany p_1p_7 algoritmus najde bod p_2 a sestrojí dvě nové polohrany p_2p_7 a p_1p_2 . Polohrana p_1p_2 je dvojčetem polohrany p_2p_1 na začátku seznamu hran. Jelikož již byly obě polohrany zpracovány, mohou být v seznamu hran eliminovány. Dostaneme tak nový seznam hran (viz. obrázek 11):

$p_3p_2, p_4p_3, p_5p_4, p_6p_5, p_7p_6, p_2p_7$.



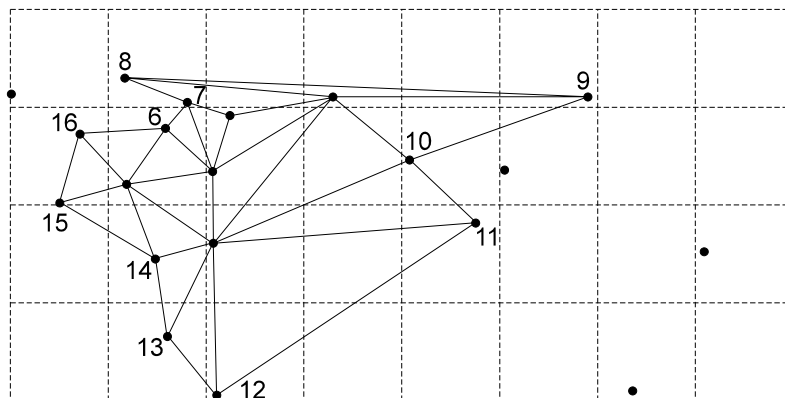
Obr. 11: Generátor 1 je vymazán.

Situaci, ve které byla nalezená polohrana dvojčetem polohrany na začátku seznamu hran, nazvěme *pravý dotek*. *Levý dotek* je pak situace, kdy je nalezená polohrana duální k polohraně předposlední v seznamu hran (později bude ukázáno formou příkladu). Jedna z těchto dvou situací nastane, najde-li algoritmus již jednou použitý bod. Proto pokaždé, když nalezne algoritmus nový bod, označí jej jako použitý. Najde-li pak algoritmus opakovaně již jednou použitý bod, otestuje jej na *dotekové situace*. Jinak tento test není nutný.

Obrázek 11 demonstruje, že bod p_1 již není potřebný a proto může být eliminován (na obr. znázorněno zmizením čísla 1). Tento krok je důležitý, neboť snižuje čas potřebný ke konstrukci nového trojúhelníka (viz. obrázek 4).

Obrázek 12 pak ukazuje další zajímavou etapu triangulačního procesu. Seznam hran je následující:

$p_7p_6, p_8p_7, p_9p_8, p_{10}p_9, p_{11}p_{10}, p_{12}p_{11}, p_{13}p_{12}, p_{14}p_{13}, p_{15}p_{14}, p_{16}p_{15}, p_6p_{16}$.



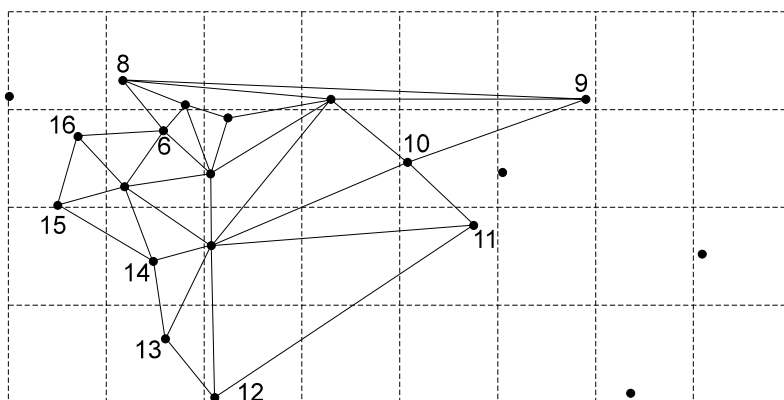
Obr. 12: Ukázka situace, ve které vznikla dutina.

Použitím poslední hrany seznamu algoritmus najde bod p_8 a generuje polohrany p_8p_{16} a p_6p_8 . Jelikož p_8 je již použitý bod, algoritmus provede test na *dotekové situace*. Nenalezne však žádný dotek ani s přední a ani s předposlední polohranou v seznamu. Tuto situaci vyřešíme tak, že přesuneme vrchní polohranu seznamu na poslední místo. Tento přesun je nutný a předejdeme jím tvorbě děr a mostů. Seznam hran je tedy:

$p_8p_7, p_9p_8, p_{10}p_9, p_{11}p_{10}, p_{12}p_{11}, p_{13}p_{12}, p_{14}p_{13}, p_{15}p_{14}, p_{16}p_{15}, p_6p_{16}, p_7p_6$.

Nyní je aktivní hranou polohrana p_7p_6 a s ní algoritmus najde bod p_8 a sestrojí dvě nové polohrany p_8p_6 a p_7p_8 . Jelikož p_8 je již použit, algoritmus jej opět testuje na *dotekové situace*. Zjistí *pravý dotek* s vrchní hranou seznamu p_8p_7 a proto smaže bod p_7 z množiny vrcholů a polohrany p_7p_6 a p_8p_7 ze seznamu hran. Nový seznam hran je po této úpravě (viz. obrázek 13):

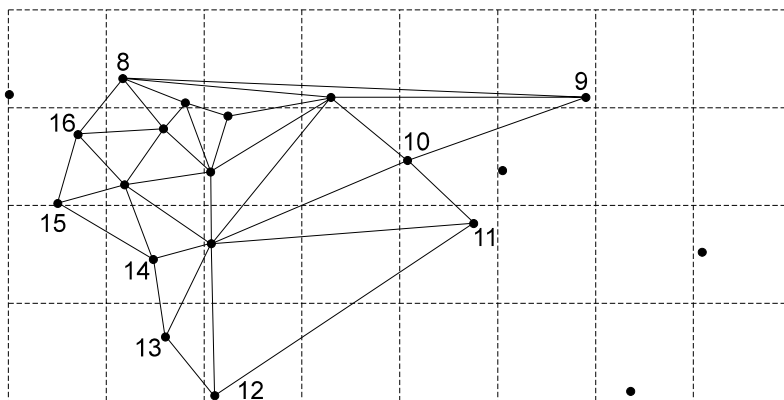
$p_9p_8, p_{10}p_9, p_{11}p_{10}, p_{12}p_{11}, p_{13}p_{12}, p_{14}p_{13}, p_{15}p_{14}, p_{16}p_{15}, p_6p_{16}, p_8p_6$.



Obr. 13: Je detekován *pravý dotek* poté, co byla přeskočena hrana p_6p_{16} .

Aktivní hranou je polohrana p_8p_6 a s ní algoritmus najde bod p_{16} a sestrojí dvě nové polohrany $p_{16}p_6$ a p_8p_{16} . Jelikož p_{16} je již použit, algoritmus jej opět testuje na *dotekové situace*. Je detekován *levý dotek* s hranou p_6p_{16} . Algoritmus odstraní tuto hranu spolu poslední hranou seznamu (p_8p_6) a na její místo přidá nově sestrojenou hranu p_8p_{16} . Nový seznam hran odpovídá obrázku 14 a vypadá takto:

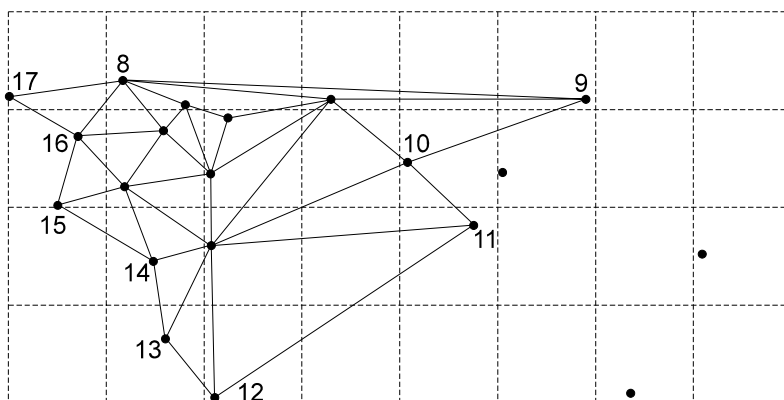
$p_9p_8, p_{10}p_9, p_{11}p_{10}, p_{12}p_{11}, p_{13}p_{12}, p_{14}p_{13}, p_{15}p_{14}, p_{16}p_{15}, p_8p_{16}$.



Obr. 14: Příklad *levého doteku*.

Pomocí p_8p_{16} algoritmus najde bod p_{17} . Ten nemá nastaven příznak „použit“ a tak není nutné testovat dotekové situace. Ze seznamu hran se vyjme p_8p_{16} a přidají se nové polohrany $p_{17}p_{16}$ a p_8p_{17} na poslední pozici v seznamu. Nový seznam hran odpovídá obrázku 15 a vypadá takto:

$p_9p_8, p_{10}p_9, p_{11}p_{10}, p_{12}p_{11}, p_{13}p_{12}, p_{14}p_{13}, p_{15}p_{14}, p_{16}p_{15}, p_{17}p_{16}, p_8p_{17}$.



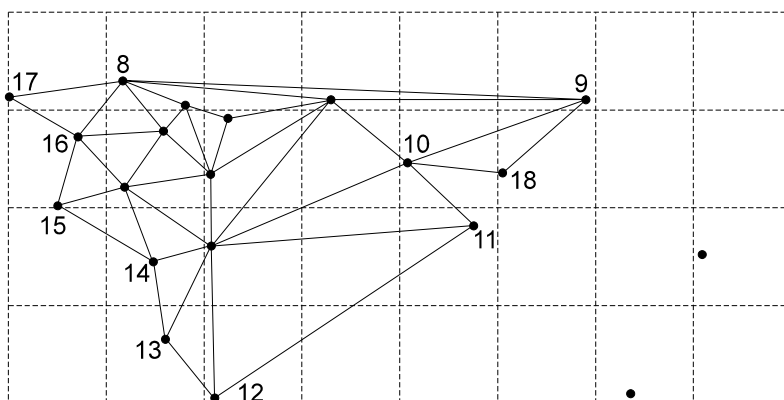
Obr. 15: Je nalezena hrana patřící obálce.

Pomocí poslední hrany p_8p_{17} algoritmus nenajde žádný nový vrchol. Narazili jsme na hranu patřící obálce, tj. na *hranu obálky*. Algoritmus tuto hranu vymaže ze seznamu a přesune hranu ze začátku seznamu na konec. Nový seznam hran vypadá následovně:

$$p_{10}p_9, p_{11}p_{10}, p_{12}p_{11}, p_{13}p_{12}, p_{14}p_{13}, p_{15}p_{14}, p_{16}p_{15}, p_{17}p_{16}, p_9p_8.$$

Hrana p_9p_8 je také hranou obálky a je tedy smazána ze seznamu. Algoritmus opět přesune hranu ze začátku seznamu na jeho konec. Nyní je aktivní hranou hrana $p_{10}p_9$. Najdeme dosud nepoužitý bod p_{18} a generujeme nové polohrany $p_{18}p_{10}$ a p_9p_{18} . Nový seznam hran odpovídá obrázku 16 a vypadá takto:

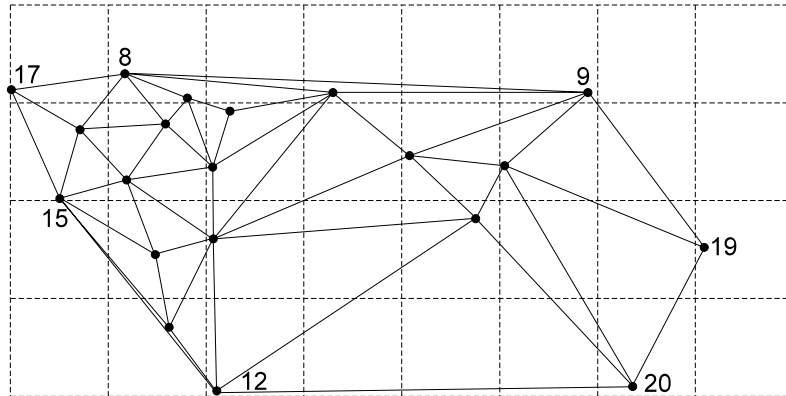
$$p_{11}p_{10}, p_{12}p_{11}, p_{13}p_{12}, p_{14}p_{13}, p_{15}p_{14}, p_{16}p_{15}, p_{17}p_{16}, p_{18}p_9, p_{10}p_{18}.$$



Obr. 16: Generování nového trojúhelníka po detekci dosud nepoužitého bodu p_{18} .

V této chvíli jsou tedy popsány všechny situace, ke kterým během triangularizačního procesu může dojít. Výše uvedenými postupy tedy algoritmus pokračuje v triangularizaci až do chvíle, kdy je seznam hran prázdný. V tom případě je triangulace dokončena. Obrázek 17 ilustruje okamžik, kdy algoritmus dospěl ke svému konci. Všimněte si, že jsou smazány

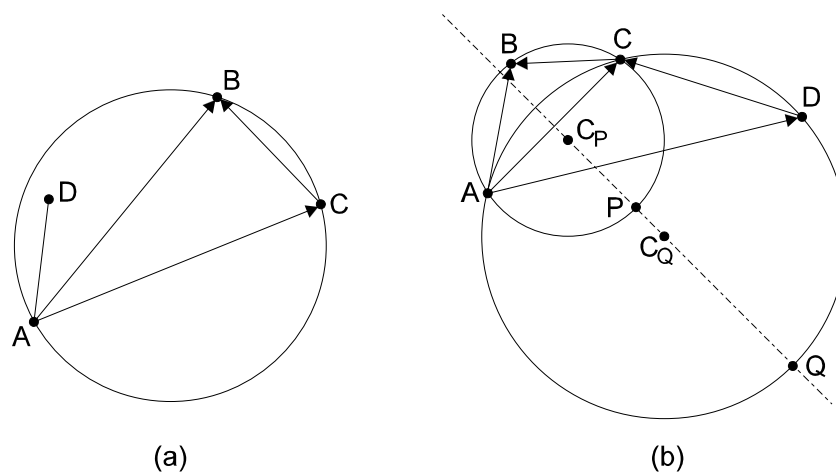
všechny body, vyjma generátorů patřících obálce. Časově nejnáročnější je proces vyhledávání. Ten se však zrychluje právě díky mazání bodů.



Obr. 17: Dokončená Delaunayova triangulace metodou využívající pravidelnou mřížku.

3.10.3.4 Delaunayovo kritérium

Nyní je nutné dokázat, že triangulace na výstupu tohoto algoritmu je korektní Delaunayova triangulace, tj. že pro každý sestrojený trojúhelník platí, že uvnitř kružnice jemu opsané se nenachází žádný jiný generátor. Jelikož algoritmus ve své vyhledávací části pracuje pouze s oblastí na pravé straně hrany, stačí, když dokážeme, že uvnitř této opsané kružnice neleží žádný bod z levé strany hrany. (V oblasti po pravé straně takový bod neexistuje, neboť algoritmus vybere vždy generátor s nejmenším kosinem, tj. největším úhlem). Důkaz se skládá ze dvou kroků. Důkaz je ilustrován obrázkem 18.



Obr. 18: K důkazu platnosti Delaunayova kritéria.

Předpokládejme, že první hranou je hrana AB (obr. 18(a)). Je-li B nejbližším bodem k bodu A , pak pro každý bod C tvořící první trojúhelník ABC nalezený algoritmem platí, že hrana AC je delší, nebo stejně dlouhá jako AB . Proto úhel ACB je menší než 90° . Proto oblouk z A , který nemůže obsahovat bod C je menší než půlkruh. Objeví-li se uvnitř této kružnice, na levé straně hrany AB bod D , pak vzdálenost AD musí být menší než AB , což je ovšem v rozporu s předpokladem, že B je nejbližším bodem k bodu A .

Dále předpokládejme (viz. obrázek 18(b)) existenci hrany AC trojúhelníka ABC sestrojeného výše. Je nutno dokázat, že žádný bod D ležící po pravé straně hrany AC a vně kružnice opsané trojúhelníku ABC nemůže spolu s hranou AC tvořit trojúhelník ACD takový, že kružnice jemu opsaná obsahuje nějaký jiný bod ležící po levé straně hrany AC . (Uvnitř kružnice opsané trojúhelníku ABC se nemůže nacházet žádný bod, je-jí tento trojúhelník Delaunayovým trojúhelníkem.) Leží-li tedy bod D vně kružnice ABC , pak můžeme sestrojít kružnici opisující trojúhelník ACD a zároveň procházející bodem Q , který leží na ose hrany AC za bodem P , kde P je průsečík této osy s kružnicí opsanou trojúhelníku ABC . Pak průnik kružnice AQC s vnitřní oblastí kružnice ABC , je oblouk, který celý leží vlevo od hrany AC . Kruhová úseč definovaná tímto obloukem a hranou AC leží celá uvnitř trojúhelníku ABC . Protože se však ve vnitřní oblasti trojúhelníku ABC nenachází žádný jiný bod, nemůže se žádný bod nacházet ani v této úseči. To znamená, že kružnice opsaná trojúhelníku ACD neobsahuje žádný bod ležící na levé straně hrany AC .

Aplikujeme-li tento postup na hrany AD , AE , ..., vidíme, že vygenerovaná triangulace je opravdu Delaunayovou triangulací.

3.10.4 Algoritmus

Nyní se podíváme na kostru samotného algoritmu v pseudo-C.
Funkce testující *dotekové situace*:

```
check_touch_case(Ps, P, Pe)
{
    if (Ps == P.next) return(RIGHT_TOUCH);
    if (Pe == P.previous) return(LEFT_TOUCH);
    return(NO_TOUCH);
}
```

Tělo algoritmu:

```
Konstrukce mřížky.
Nalezení prvního bodu  $p_1$  (v algoritmu P1) a první hrany  $p_1p_2$ .
Inicializace seznamu hran vložení  $p_1p_2$  a  $p_2p_1$ .
P1.previous = P1.next = P2;
P2.previous = P2.next = P1;
```



```
while (seznam hran není prázdní)
{
    aktivní hranou je hrana (Ps, Pe);
    nalezen ← najdi_třetí_bod(..., P,...);
    /*třetím nalezeným bodem je bod P*/
    if (nalezen)
    {
        if (třetí bod byl již použit)
        {
            touch ← check_touch_case(Ps, P, Pe);
            switch (touch)
            {
                case RIGHT_TOUCH:
                    vymaž vrchní polohranu ze seznamu hran;
                    vymaž aktivní polohranu z konce seznamu;
                    připoj na konec seznamu první ze dvou nově
                    sestrojených polohran;
                    Pe.previous = P;
                    P.next = Pe;
                    break;
                case LEFT_TOUCH:
                    vymaž aktivní polohranu z konce seznamu;
                    vymaž předposlední polohranu (hned před aktivní
                    hranou) seznamu hran;
                    připoj na konec seznamu druhou ze dvou nově
                    sestrojených polohran;
                    Ps.next = P;
                    P.previous = Ps;
                    break;
                case NO_TOUCH:
                    přesuň vrchní polohranu seznamu na poslední místo;
            }
        } else /*třetí bod ještě nebyl použit*/
        {
            nastav třetí bod na „použit“;
            vymaž poslední polohranu ze seznamu;
            připoj na konec seznamu první ze dvou nově sestrojených
            polohran;
            připoj na konec seznamu druhou ze dvou nově
            sestrojených polohran;
            P.previous = Ps;
            P.next = Pe;
            Ps.next = P;
            Pe.previous = P;
        }
    } else /*žádný bod nebyl nalezen*/
    {
        vymaž poslední polohranu ze seznamu;
        přesuň vrchní polohranu seznamu na poslední místo;
    }
}
```

3.10.5 Konstrukce konvexní obálky

Tento algoritmu nabízí dvě snadné cesty pro konstrukci konvexní obálky. První je založena na seznamu hran, druhá na využití datové struktury mřížky.

Metoda založená na seznamu hran ukládá během triangularizačního procesu hrany patřící obálce před tím, než jsou smazány. Například pro naši předešlou ukázkou jsou uloženy tyto hrany v tomto pořadí:

$p_8p_{17}, p_9p_8, p_{12}p_{20}, p_{15}p_{12}, p_{17}p_{15}, p_{19}p_9, p_{20}p_{19}$.

Metoda založená na datové struktuře mřížky je stejně jednoduchá. Na obrázku 17 vidíme, že všechny body, které nebyly smazány, jsou rovněž body konvexní obálky. Navíc ke každému bodu máme dva ukazatele - předcházející a další. Můžeme tedy začít jedním z těchto zbývajících vrcholů a postupovat pomocí ukazatelů buď vpřed (ukazatel další) nebo zpět (ukazatel předcházející) k ostatním vrcholům a projít tak popořadě celou konvexní obálkou. Tak tedy může být na výstupu uzavřený polygon, jehož strany jsou seřazeny buď proti nebo po směru chodu hodinových ručiček.

3.10.6 Degenerace

V této podkapitole se budeme zabývat dvěma možnými degeneracemi: kolinearitou a koincencí bodů vstupní množiny. Obsahuje-li tedy vstupní množina generátorů jen body ležící na přímce, je to problém, neboť v takovém případě algoritmus nenajde na pravé či na levé straně první hrany žádný bod. Jelikož seznam hran je zatím v tomto případě prázdný, program skončí, aniž by vypočítal hrany podél kolineárních bodů. Jednoduché řešení takového problému je test na takovou situaci a konstrukce hran uspořádaných podle datové struktury mřížky.

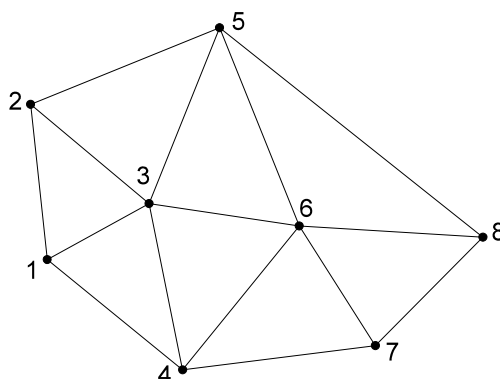
Neleží-li body vstupní množiny vesměs na přímce, není třeba speciálního přístupu, neboť algoritmus najde nejdříve body neležící na přímce a ty tvoří trojúhelníky, ze kterých postupuje triangulace dál a zahrne i kolineární body.

Koincidentní body (neboli body ležící na stejné pozici) nejsou žádným problémem, neboť jsou eliminovány již během konstrukce datové struktury mřížky.

3.10.7 Datová struktura „seznam vrcholů“

Během triangularizačního procesu musíme někde ukládat vypočítané trojúhelníky. Výstupní datová struktura může být vytvořena několika různými způsoby. Můžeme použít například strukturu „okřídlených hran“ (viz. kapitola

„Datová struktura „okřídlené hrany“). Dalším možným řešením je *seznam vrcholů*. Seznamem vrcholů můžeme nahradit seznam hran a řídit jím triangularizační proces. Nyní si tedy tuto datovou strukturu představíme a ukážeme si, jak s ní lze vygenerovat výstup ve stejném čase. Sledujme příklad na obrázku 19.



Obr. 19: Příkla trojúhelníkové sítě ilustrující datovou strukturu *seznam vrcholů*.

Algoritmus používá dva seznamy bodů. Jeden pro vnitřní vrcholy triangulace, druhý pro vrcholy patřící konvexní obálce. Každý vrchol je spjat se seznamem vrcholů, kteří jej obklopují. Příklad takové datové struktury popisující triangulaci na obrázku 19 je uveden na obrázku 20.

3	5	6	4	1	2
6	8	7	4	3	5
5	8	6	3	2	
8	7	6	5		
7	4	6	8		
4	1	3	6	7	
1	2	3	4		
2	5	3	1		

Obr. 20: Datová struktura reprezentující triangulaci na obr. 19.

Body v prvním sloupci pod dvojitou horizontální čarou jsou body patřící obálce, body nad touto čarou jsou interní body triangulace. Tato datová struktura nám může dát odpověď na mnoho otázek, například:

- Všechny trojúhelníky sdílející jeden vrchol. Pro vrchol 3 jsou to trojúhelníky (3, 5, 6), (3, 6, 4), (3, 4, 1), (3, 1, 2) a (3, 2, 5). Pro vrchol obálky 5 jsou to trojúhelníky (5, 8, 6), (5, 6, 3) a (5, 3, 2).
- Trojúhelníky sdílející jednu hranu. Např. pro hranu (3, 6) dostáváme trojúhelník (3, 6, 4) ze seznamu trojky a trojúhelník (6, 3, 5) ze seznamu šestky. Pro hranu (2, 1) dostaneme pouze trojúhelník (1, 2, 3), protože ze seznamu dvojky se dozvíme, že jde o hranu obálky.

- Všechny hrany sdílející jeden vrchol. Např. pro vrchol 7 dostáváme hrany (7, 4), (7, 6) a (7, 8).
- Všechny trojúhelníky sousedící s daným trojúhelníkem. Např. pro trojúhelník (3, 6, 4) hledáme trojúhelníky sousedící s hranami (3, 6), (6, 4) a (4, 3) a dostáváme trojúhelníky (6, 3, 5), (4, 6, 7) a (3, 4, 1).

Nyní se budeme zabývat otázkou, jak současně tvořit trojúhelníky a datovou strukturu seznam vrcholů a s touto otázkou se vrátíme ke kapitole „Krok 3: Spojování sestrojených trojúhelníků“.

Na obrázku 7 je znázorněna situace po nalezení první hrany. Algoritmus vloží dvě nové polohrany do seznamu vrcholů, viz. obrázek 21(a). Aktivní hranou je hrana (1, 2), tj. $F(1)L(1)$. Po nalezení bodu 3 (obr. 8), algoritmus aktualizuje seznam takto:

1. Připojí bod 3 do seznamu F .
2. Vloží bod 3 na druhou pozici seznamu L .
3. Vytvoří nový seznam, do něj vloží bod 3 a připojí $F(1)$ a $L(1)$.
4. Posune ukazatel seznamu L .

Nový seznam odpovídající obrázku 8 je tabulka 21(b). Aktivní hranou je hrana $F(1)L(1)$, tj. (1, 3) a novým nalezeným bodem je vrchol 4 (viz. obrázek 9). Následuje stejná procedura, kterou algoritmus aktualizuje seznam vrcholů do podoby tabulky 21(c).

První zajímavou situací je pravý dotek na obrázku 10. Odpovídající seznam vrcholů je znázorněn tabulkou 21(d). Aktivní hranou je zde hrana (1, 7). Algoritmus najde nový bod 2, detekuje pravý dotek (užitím funkce `check_touch_case`) a tímto způsobem aktualizuje seznam:

1. Uloží seznam F .
2. Připojí $L(1)$ na konec seznamu následujícího za seznamem F .
3. Vloží $F(2)$ na druhou pozici seznamu L .
4. Posune ukazatel F .

Tabulka 21(e) představuje seznam vrcholů odpovídající obrázku 11, po zpracování pravého doteku. Aktivní hranou je hrana (6, 16). Algoritmus najde nový bod 8 a takto zpracuje bezdotekovou situaci (case `NO_TOUCH`):

1. Zkopíruje seznam F na konec seznamu vrcholů.
2. Posune ukazatele F a L .

Obrázek 13 ilustruje levý dotek. Odpovídající seznam vrcholů je znázorněn tabulkou 21(g). Aktivní hranou je zde hrana (8, 6). Algoritmus najde nový bod 16, detekuje levý dotek a tímto způsobem aktualizuje seznam:

1. Uloží seznam L .
2. Připojí seznam L (poslední prvek) na konec seznamu F .
3. Posune ukazatel L o jednu pozici zpět.
4. Vloží $F(1)$ na druhou pozici seznamu L .

Seznam vrcholů odpovídající obrázku 14 je znázorněn tabulkou 21(h).

Na obrázku 15 je konečné stav triangularizačního procesu a tabulka 21(i) znázorňuje odpovídající datovou strukturu. Aktivní hranou je zde hrana (8, 17). Žádný nový bod již algoritmem není nalezen: hrana patří konvexní obálce. V tomto případě algoritmus jednoduše posune oba ukazatele F i L .

Na konci triangularizačního procesu máme tedy dva druhy dat:

1. Seznam vrcholů uložených po každém levém i pravém doteku.
2. Seznam zbývajících (nesmazaných) vrcholů kruhového seznamu, tj. vrcholů patřících konvexní obálce.

Tabulka 22(a) znázorňuje seznam uložených vrcholů (obr. 17). Tabulka 22(b) obsahuje body zbylé v kruhovém seznamu.

Tento druhý seznam (22(b)) tedy obsahuje všechny vrcholy náležející konvexní obálce triangulace. Posloupnost těchto vrcholů tvoří první sloupec tabulky. Spojením těchto dvou seznamů dostaneme úplnou datovou strukturu reprezentující generovanou triangulaci (triangulaci na obrázku 17 reprezentuje struktura znázorněná tabulkou 22).

F:	1	2		F:	1	2	3		F:	1	2	3	4		F:	1	2	3	4	5	6	7
L:	2	1			2	3	1			2	3	1				2	3	1				
(a)				L:	3	1	2			3	4	1	2			3	4	1	2			
				(b)						L:	4	1	3			4	5	1	3			
										(c)						5	6	1	4			
																6	7	1	5			
															L:	7	1	6				
															(d)							
F:	2	3	1	7		F:	6	7	1	5	16				F:	8	9	3	7	6		
	3	4	1	2			7	8	3	2	1	6				9	10	3	8			
	4	5	1	3			8	9	3	7						10	11	4	3	9		
	5	6	1	4			9	10	3	8						11	12	4	10			
	6	7	1	5			10	11	4	3	9					12	13	4	11			
L:	7	2	1	6			11	12	4	10						13	14	4	12			
(e)							12	13	4	11						14	15	5	4	13		
							13	14	4	12						15	16	5	14			
							14	15	5	4	13					16	6	5	15			
							15	16	5	14						L:	6	8	7	1	5	16
							L:	16	6	5	15				(g)							
							(f)															
F:	8	9	3	7	6	16			F:	8	9	3	7	6	16	17						
	9	10	3	8						9	10	3	8									
	10	11	4	3	9					10	11	4	3	9								
	11	12	4	10						11	12	4	10									
	12	13	4	11						12	13	4	11									
	13	14	4	12						13	14	4	12									
	14	15	5	4	13					14	15	5	4	13								
	15	16	5	14						15	16	5	14									
L:	16	8	6	5	15					16	17	8	6	5	15							
(h)										L:	17	8	16									
										(i)												

Obr. 21: Aktualizace seznamu vrcholů.

1	2	3	4	5	6	7					20	12	11	19			
2	3	1	7								12	15	13	4	11	20	
3	4	1	2	7	8	9	10				15	17	16	5	14	13	12
4	5	1	3	10	11	12	13	14			17	8	16	15			
5	6	1	4	14	15	16					8	9	3	7	6	16	17
7	8	3	2	1	6						9	19	18	10	3	8	
6	8	7	1	5	16						19	20	11	18	9		
10	11	4	3	9	18						(b)						
11	12	4	10	18	19	20											
14	15	5	4	13													
13	15	14	4	12													
16	17	8	6	5	15												
18	19	11	10	9													
(a)																	

Obr. 22: (a) Uložený seznam vrcholů. (b) Seznam vrcholů patřících konvexní obálce.

1	2	3	4	5	6	7			
2	3	1	7						
3	4	1	2	7	8	9	10		
4	5	1	3	10	11	12	13	14	
5	6	1	4	14	15	16			
7	8	3	2	1	6				
6	8	7	1	5	16				
10	11	4	3	9	18				
11	12	4	10	18	19	20			
14	15	5	4	13					
13	15	14	4	12					
16	17	8	6	5	15				
18	19	11	10	9					
20	12	11	19						
12	15	13	4	11	20				
15	17	16	5	14	13	12			
17	8	16	15						
8	9	3	7	6	16	17			
9	19	18	10	3	8				
19	20	11	18	9					

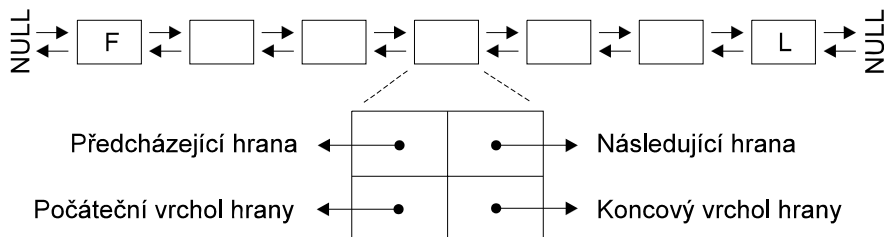
Obr. 23: Výsledný seznam vrcholů.

3.10.8 Implementace metody Delaunayovy triangulizace využívající pravidelnou mříž

Jak už bylo řečeno, tato velmi zajímavá metoda by měla vykazovat lineární výpočtovou složitost, a to jako jediná z metod výše popsanych. To

jsem se rozhodl ukázat implementaci této metody a završit tak pozvolný přechod od teorie k praxi, v jehož duchu je sestavena celá tato práce.

Při programování jsem použil Borland C++ v. 3.1. Program běží na platformě DOS. První program založený na této metodě používá pro řízení triangulace datovou strukturu *seznamu hran* stejným způsobem, jaký je popsán výše.



Obr. 24: Datová struktura SEZNAM HRAN

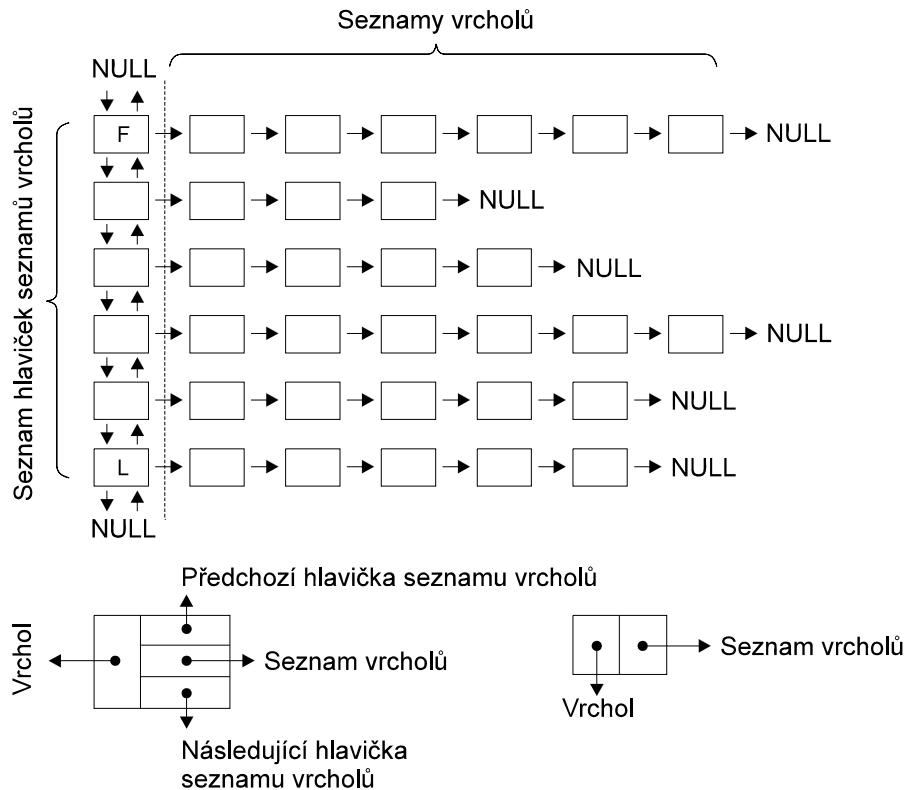
Datová struktura SEZNAM HRAN k obr. 24:

```
typedef struct edge_list {
    struct cellnode *vertex_s;
    struct cellnode *vertex_e;
    struct edge_list *previous_edge;
    struct edge_list *next_edge;
} EDGE_LIST;
```

V programu jsem použil variantu seznamu ohraničeného první (*L*) a poslední (*F*) hranou, kde poslední hrana je právě aktivní hranou, pro kterou je triangularizačním procesem vyhledáván třetí vrchol. Nepatrně lepší variantou je použití kruhového seznamu. Ten zjednoduší správu *seznamu hran* v případě, že není pro aktivní hranu nalezen žádný třetí bod (hrana konvexní obálky) a v případě, že nalezený třetí bod již patří konstruované trojúhelníkové síti a nesousedí s body aktivní hrany (není detekována žádná doteková situace) tím, že pro přesun první hrany na konec seznamu stačí pouze posunout ukazatele *F* a *L* na následující hranu. Nalézáním nových bodů se hrany do seznamu přidávají a během triangularizačního procesu jsou zpracované hrany ukládány, takže není třeba udržovat celý seznam všech hran v paměti. Uložené hrany jsou pak výstupem programu, který končí v případě, že je *seznam hran* prázdný.

Druhou možnou datovou strukturou pro řízení triangularizačního procesu je *seznam vrcholů*, o kterém je zmíněno výše. *Seznam vrcholů* je poněkud komplikovanější datovou strukturou než *seznam hran*, ale dává mnohem komplexnější informace o zkonstruované trojúhelníkové síti při nezměněné výpočtové složitosti a proto jsem ho použil při sestavování druhého programu (soubor „DTUG.EXE“ je přiložen na disketě a zdrojové

texty jsou uvedeny v příloze). Datová struktura *seznamu vrcholů* je znázorněna na obr. 25.



Obr. 25: Datová struktura *seznam vrcholů* použitá v programu.

Datová struktura *seznam vrcholů* k obr. 25:

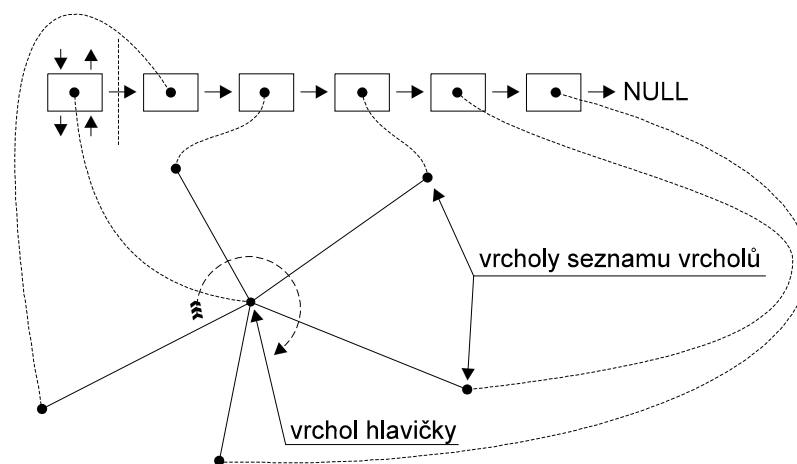
```
typedef struct vertex_list {
    struct cellnode far *vertex;
    struct vertex_list far *next_vertex;
} VERTEX_LIST;
```

Datová struktura *seznam hlaviček seznamů vrcholů* k obr. 25:

```
typedef struct header_vertex_list {
    struct cellnode far *vertex;
    struct header_vertex_list far *previous_header_vertex;
    struct header_vertex_list far *next_header_vertex;
    struct vertex_list far *next_vertex;
} HEADER_VERTEX_LIST;
```

Stejně jako *seznam hran* lze i *seznam hlaviček seznamů vrcholů* zřetěžit do kruhového seznamu a tak zjednodušit výše uvedené postupy při aktualizaci

obsahu struktury. V případě *seznamu vrcholů* je aktivní hrana přístupná přes ukazatele F a L , kde F ukazuje na hlavičku obsahující pointer na počáteční bod aktivní hrany a L ukazuje na hlavičku obsahující pointer na koncový bod aktivní hrany, tj. $F(1)L(1)$ je tázaná aktivní hrana. Při konstrukci nového trojúhelníka je každý nový bod (tvořící třetí vrchol k aktivní hraně) připojen k seznamu vrcholů příslušejícímu k jednomu z vrcholů aktivní hrany. V každém *seznamu vrcholů* jsou tedy uloženy vrcholy tvořící s vrcholem hlavičky hrany triangulace a seříděny ve směru chodu hodinových ručiček (viz. obrázek 26).



Obr. 26: Kruhové uspořádání vrcholů seznamů vrcholů.

Podobně jako u *seznamu hran* ani *seznam vrcholů* neudrží najednou ukazatele na všechny vrcholy triangulace a již dokončené lokální triangulace kolem vrcholu hlavičky jsou průběžně ukládány do výstupního souboru a uvolňují paměťový prostor.

Samotný algoritmus lze rozdělit do tří hlavních částí:

I. *Inicializace:*

V této části jsou určeny parametry mřížky, alokován paměťový prostor a sestrojena samotná mřížka. Do ní jsou pak vloženy body vstupní množiny generátorů.

V tomto podkroku je založena datová struktura *seznam vrcholů* a ta je inicializována prvním nalezeným vrcholem a první hranou triangulace.

II. *Detekce třetího bodu:*

Toto je nejcitlivější, nejnáchylnější a nenáročnější část algoritmu, která nejvíce ovlivňuje výpočtovou složitost algoritmu. Náplní této části je

nalézt pro aktivní hranu třetí vrchol tvořící spolu s aktivní hranou trojúhelník vyhovující Delaunayovu kritériu.

III. *Správa seznamu vrcholů a řízení triangulace:*

Tento blok má za úkol detekovat *dotekové situace* a na základě výsledků aktualizovat *seznam vrcholů* a tím řídit běh celého triangularizačního procesu. Zde je také generován podnět k ukončení procesu po akceptování situace, kdy v seznamu zůstaly pouze hrany (resp. vrcholy) patřící konvexní obálce trojúhelníkové sítě.

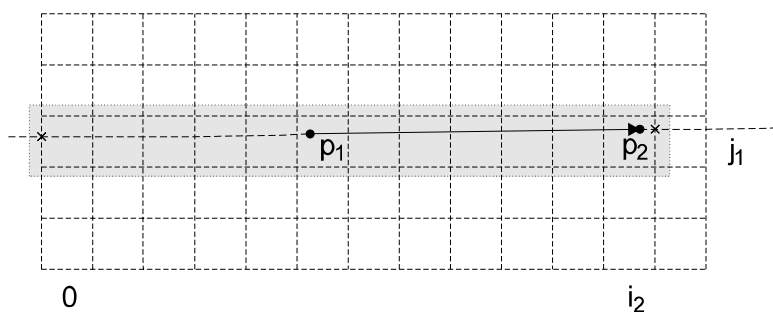
3.10.8.1 Inicializace

První (inicializační) část je, myslím, jasná již z předešlého výkladu a nebudeme se jí tudíž více zajímat. Naproti tomu druhé části, která pro mne byla největším oříškem, se budeme věnovat podrobněji na následujících řádkách.

3.10.8.2 Detekce třetího bodu

Funkce provádějící detekci třetího bodu se skládá ze dvou hlavních kroků. Prvním je nalezení prvního kandidáta. Není-li zde žádný bod nalezen, funkce vrátí nulovou hodnotu a aktivní hranu tak prohlásí za hranu konvexní obálky. Je-li nový generátor nalezen, postupuje do druhého kroku, ve kterém je testován na platnost Delaunayova kritéria. Bod vyhovující tomuto kritériu je pak funkcí vrácen jako třetí bod a je předán části spravující *seznam vrcholů*.

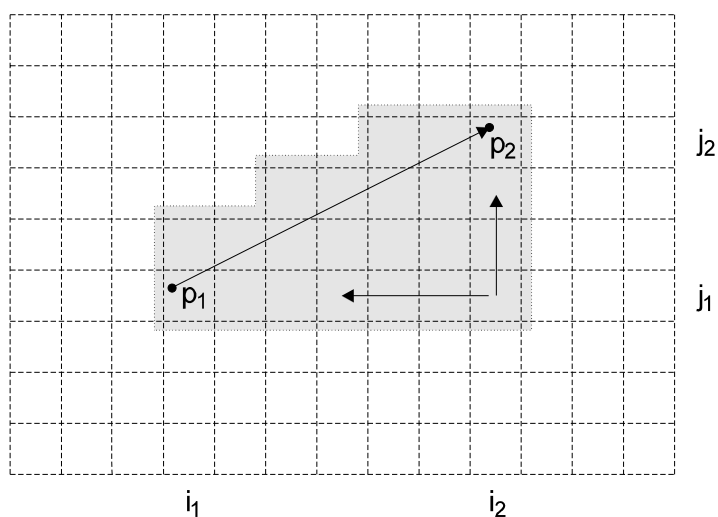
První krok tedy začíná konstrukcí a následným prohlížením příslušné trojúhelníkové oblasti (viz. obrázek 4). A právě při konstrukci této oblasti se objevil první problém. Vrcholy přepony tohoto trojúhelníka se totiž určují výpočtem její průsečíků s hranami mřížky a problémy nastávají v případech, kdy je aktivní hrana „téměř“ rovnoběžná s vertikálními, či horizontálními hranami mřížky. Tyto situace jsou řešitelné např. testem těchto situací a převedením problému na výpočet průsečíků přepony s okrajovými hranami mřížky (viz. obrázek 27).



Obr. 27: Konstrukce trojúhelníkové oblasti pomocí průsečíku s krajní hranou mřížky.

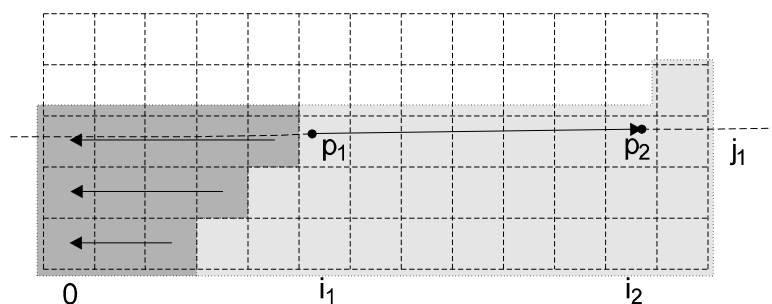
Během ladění a testování algoritmu jsem však zjistil, že největší vliv má na rychlost programu určení nevhodnější inkriminované oblasti a efektivnost jejího prohledávání. To znamená, že je nutno k aktivní hraně najít třetí bod testem co možná nejmenšího počtu buněk. Pravděpodobnost výskytu třetího bodu vyhovujícího Delaunayovu kritériu klesá s rostoucí vzdáleností od středu úsečky p_1p_2 . Jak je však vidět na obrázku 27, budeme prohledávat sloupce od buňky obsahující vrchol p_1 až do nultého sloupce a pravděpodobnost, že bod nalezený v těchto buňkách bude vyhovovat Delaunayovu kritériu (kromě oblastí blízkých se obálce) je velmi nízká. Situace se však ještě zhorší v případě, je-li zde takový bod opravdu nalezen. Ten totiž postupuje do dalšího testu opsaných kružnic a generuje veliké prohledávací oblasti a tím zpomaluje výpočet. Řešení, které jsem nakonec zvolil, je následující.

Při určování vrcholů trojúhelníkové oblasti není použit výpočet průsečíků a za vrcholové buňky jsou jednoduše určeny buňky obsahující krajní body aktivní hrany (viz. obrázek 28).



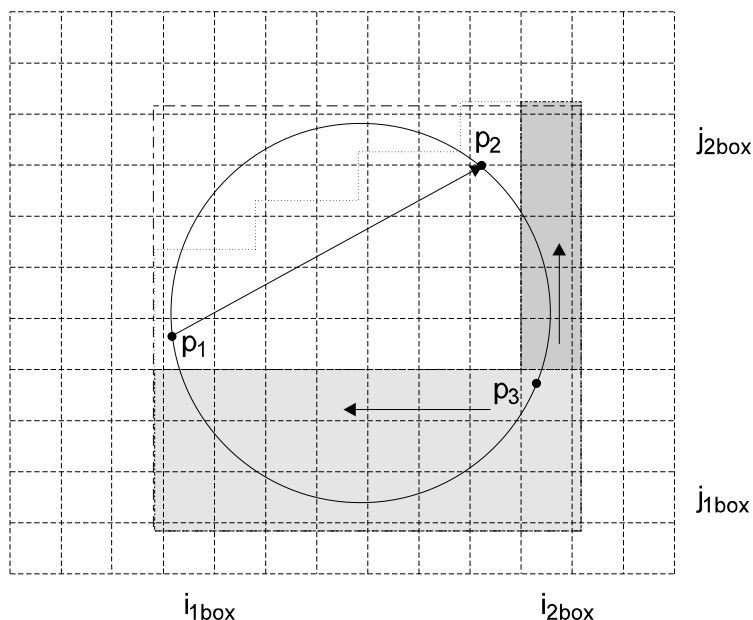
Obr. 28: Vrcholy trojúhelníkové oblasti jsou buňky obsahující krajní body aktivní hrany.

Způsob prohledávání pak zůstává stejný. V případě neúspěchu je i druhé kolo rozšířeného prohledávání (prohledávání řádků směrem dolů a sloupců směrem vpravo inkrementováním proměnné k) stejné. Je-li však i to neúspěšné, musíme prohledat ještě řádky a sloupce vynechané úpravou prvního průchodu. Pravděpodobnost, že se však třetí bod bude vyskytovat až v těchto „odlehklých“ buňkách je však malá a k tomuto třetímu testu většinou vůbec nedojde. Prohledávání se provádí opět posunováním řádků a sloupců směrem od aktivní hrany, ovšem tak, že testujeme oblast doplňkovou k oblasti prohledané v druhém průběhu. Tento způsob je ilustrován obrázkem 29, na našem „problematickém“ případě.



Obr. 29: Třetí průběh prohledávání po tom, co byly první dva neúspěšné. Světlejším odstínem je vyznačeno sjednocení vyhledávacích oblastí prvního a druhého průchodu, tmavějším pak nová doplňková oblast prohledávaná ve směru šipek od horního řádku směrem dolů.

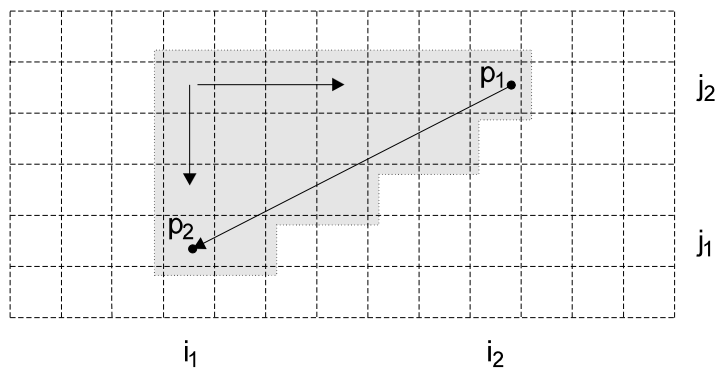
Při prvním průchodu jsou otestovány všechny buňky patřící oblasti a do kružnicového testu postoupí bod s největším úhlem (nejmenší kosinus). Druhý a třetí průběh je ukončen již při detekci prvního bodu a ten je předán dalšímu zpracování. Tím je, jak už jsme si řekli výše, kružnicový test na platnost Delaunayova kritéria. Ten proběhne ve dvou fázích, každá pro jeden obdélník (viz. obrázek 30).



Obr. 30: Dvě prohledávané obdéníkové oblasti.

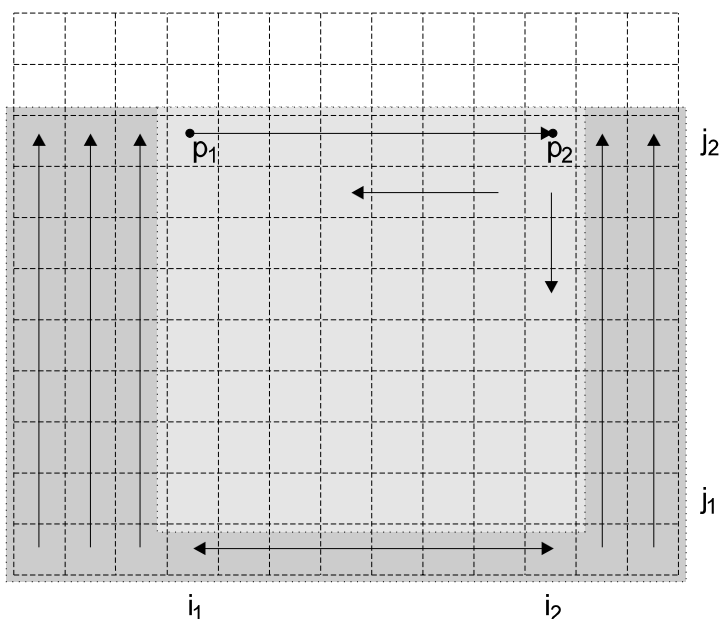
Testování je nutno provádět pro řádky směrem od j_1-1 k $j_{1\text{box}}$ a pro sloupce směrem od i_2+1 k $i_{2\text{box}}$. Testování je ukončeno a třetí bod je nalezen v případě, že v daném „min-max boxu“ již nenajdeme žádný jiný bod s větším úhlem (menším kosinem).

Tento postup je aplikován na osm speciálních případů - na osm směrů aktivní hrany, tj. pro případ hrany orientované ve směru vpravo nahoru (viz. obrázky výše), vpravo dolů, vlevo dolů, vlevo nahoru a pro horizontální či vertikální hrany orientované vpravo, dolů, vlevo a nahoru. Test orientace je proveden triviálním porovnáním souřadnic koncových bodů.



Obr. 31: Trojúhelníková oblast pro hranu orientovanou ve směru vlevo dolů.

Pro horizontální a vertikální hrany je základní test dvouступňový. V prvním průchodu testujeme buňky ve čtvercové oblasti (viz. obrázek 32).

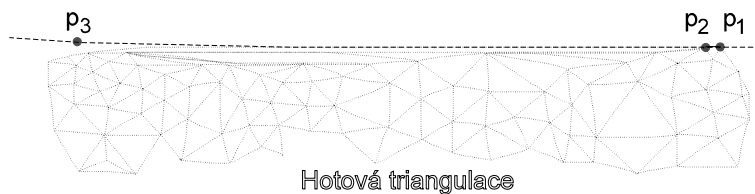


Obr. 32: V případě horizontální hrany je v prvním průchodu testována čtvercová oblast (světlejší odstín) a po neúspěchu druhá oblast (tmavší odstín).

První oblast je prohledávána směrem od aktivní přímky (tím se zamezí zbytečnému prohazování ukazatele na třetí bod) a je prohledávána celá. Při neúspěchu nastupuje druhý průchod. V něm se prohledává v každém kroku (v mezích hranic mřížky) vždy jedna řádka $j_1 - k$ pro sloupce $i_1 - k + 1$ až $i_2 + k - 1$ a dva sloupce $i_1 - k$ a $i_2 + k$ pro řádky $j_1 - k$ až j_2 , pro $k = 1, 2, \dots$. Do kružnicového testu postupuje hned první nalezený bod. Test na platnost Delaunayova kritéria pak probíhá ještě jednodušeji, než pro šikmé hrany, neboť je třeba testovat, jinak stejným způsobem, jen jednu obdélníkovou oblast.

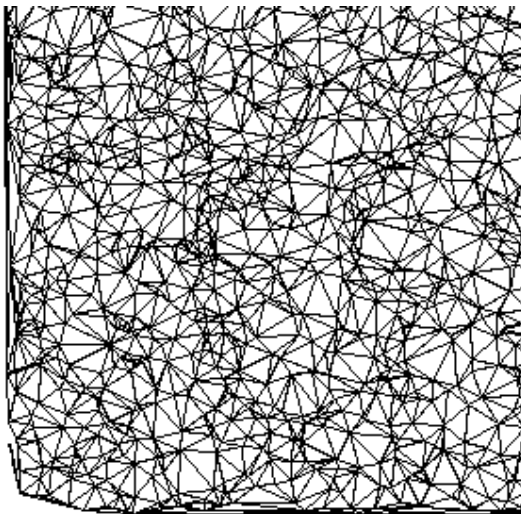
Druhým problémem tohoto algoritmu (a nejen tohoto), se ukázal být test pozice bodu vzhledem k hraně. S rostoucím počtem generátorů, jejichž souřadnice jsou normalizovány do prostoru mřížky, roste i pravděpodobnost vzniku chyby v topologii následkem nesprávných výsledků numerických výpočtů. Zde se v praxi objevuje ona propast mezi teoretickou metodou a hotovým programem, o které je zmíněno v kapitole „Přírůstková konstrukce Voronoiova diagramu s důrazem na topologickou stabilitu“. V programu jsem pro test pozice bodu vzhledem k aktivní hraně (přímce) použil testu znaménka výsledku po dosažení do parametrické rovnice. Algoritmus jsem testoval pro mohutnost vstupní množiny generátorů do 5500. Četnost této chyby byla nulová pro interní vrcholy triangulace, avšak pravděpodobnost jejího vzniku vzrostla zpracováváním hran podél konvexní obálky, kde může dojít k situaci

ilustrované obrázkem 33, kdy jsou krajní body aktivní hrany „velmi blízko“ a testovaný třetí bod je od aktivní hrany „velmi daleko“, avšak „velmi blízko“ přímce proložené aktivní hranou.



Obr. 33: Situace, kdy může být špatně testována pozice bodu p_3 vůči přímce p_1p_2 .

Nastane-li taková situace, mohou být vygenerována topologická chyba. V případě, že bod p_3 je (sledujte prosím obr. 33) detekován na levé straně přímky p_1p_2 , avšak ve skutečnosti leží na pravé straně, je hrana p_1p_2 neprávem prohlášena za hranu obálky a v síti se objeví „dutina“ (obr. 33). Program takovou chybu neumí opravit, detekuje ji však a podá o ni zprávu.



Obr. 33: Dutina v levém dolním rohu triangulace vznikla po chybě numerických výpočtů.

3.10.8.3 Správa seznamu vrcholů a řízení triangulace

Řízení triangularizačního procesu je prováděno stejně jako v případě *seznamu hran* na základě výsledků funkce `check_touch_case()`. Tu však musíme pro *seznam vrcholů* modifikovat. Aktivní hrana, kterou je poslední

položka *seznamu hran*, je nyní dána vrcholy uloženými v hlavičkách seznamů vrcholů F a L . Počáteční bod aktivní hrany P_s je tedy nahrazen ukazatelem na $F(1)$ a stejně tak koncový bod P_e je nahrazen ukazatelem na hlavičku seznamu vrcholů $L(1)$. Funkce pak vypadá takto:

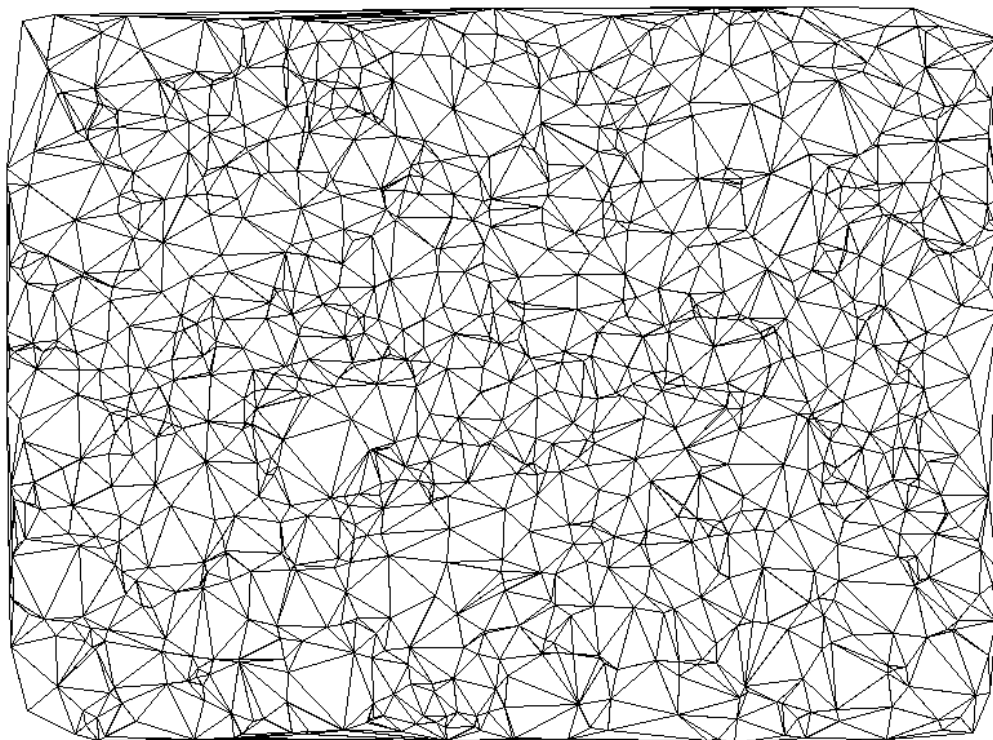
```
int check_touch_case(CELLNODE huge *p_3) {
    if (F->vertex == p_3->next_point)
        return 1; // Right touch
    if (L->vertex == p_3->previous_point)
        return 2; //Left touch
    return 0;    // No touch
}
```

Stejná úprava P_s na $F(1)$ a P_e na $L(1)$ je provedena ve funkcích aktualizujících *seznam vrcholů*, tj. ve funkcích `point_was_not_used()`, `right_touch()` a `left_touch()`. Algoritmus založený na datové struktuře *seznamu hran* je ukončen v případě, že *seznam hran* je prázdný. Algoritmus řízený *seznamem vrcholů* je ukončen v případě, že všechny vrcholy zbylé ve struktuře jsou vrcholy konvexní obálky. V případě, že taková situace nastane, není již pro žádnou aktivní hranu vrcholů seznamu nalezen třetí bod a seznam je pouze posouván (v případě že je kruhový, jsou posouvány pouze ukazatelé F a L).

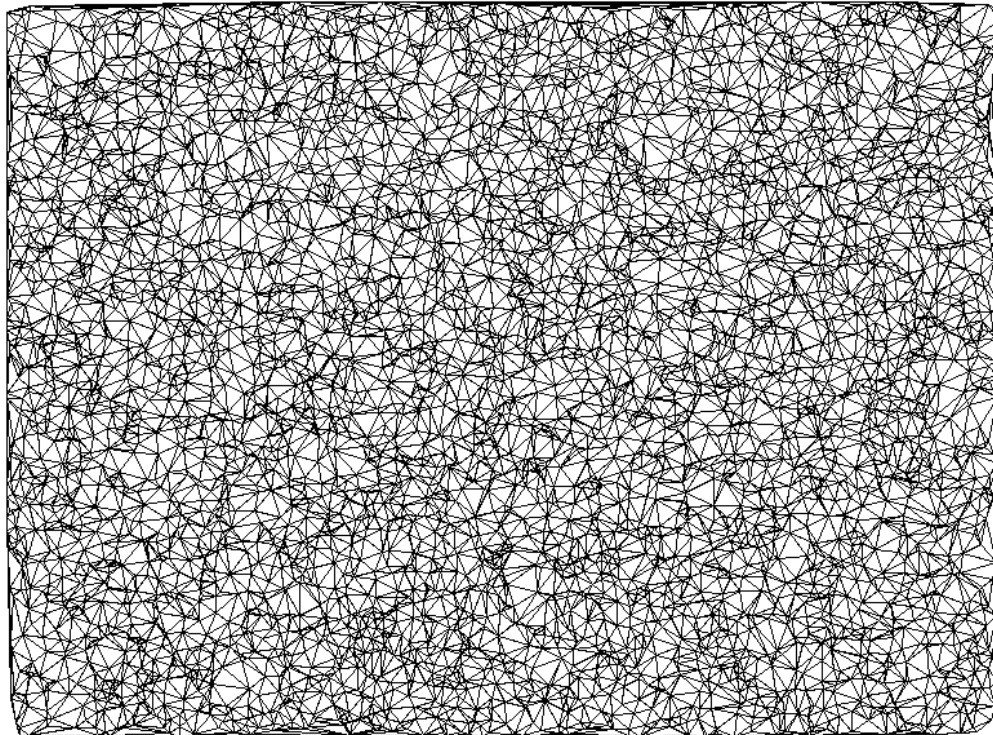
3.10.9 Výsledky testování

Rychlost algoritmu je závislá především na způsobu hledání třetího bodu pro konstrukci nového trojúhelníka. Cesta za zrychlením vede k hledání efektivnějšího způsobu prohledávání buněk daných oblastí. Cílem je tedy snížení průměrného počtu testů buněk mřížky nutných k nalezení či potvrzení neexistence třetího bodu.

Na obrázku 34 je ukázka triangulace množiny 1000 bodů, na obrázku 35 pak triangulace množiny 5500 bodů.



Obr. 34: Triangularizace množiny 1000 náhodně generovaných bodů.

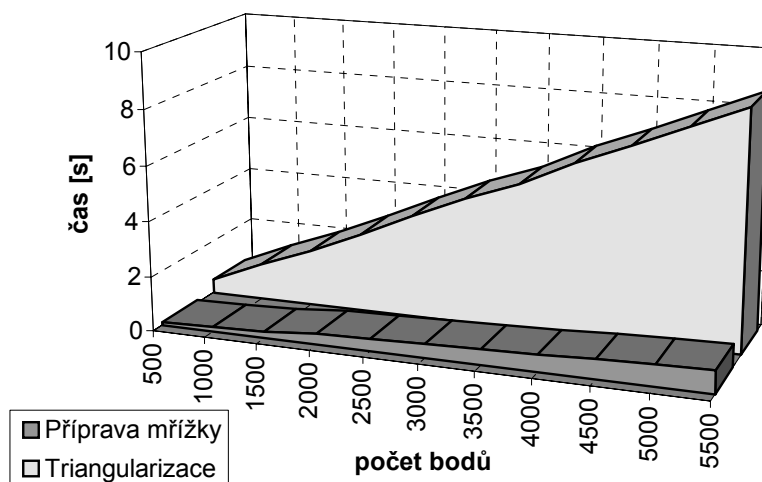


Obr. 35: Triangularizace množiny 5500 náhodně generovaných bodů.

Zkomplikování datové struktury a její správy se při přechodu od *seznamu hran* k *seznamu vrcholů* nepromítlo na zvýšení výpočtové složitosti. Algoritmus byl testován na počítači PC 486/DX2 66MHz s operačním systémem MS-DOS 6.22. Zvlášť byly testovány časy potřebné pro přípravu mřížky a pro samotnou triangularizaci. Výsledky testů jsou prezentovány tabulkou na obrázku 36 a grafem na obrázku 37.

Počet bodů	Příprava mřížky [s]	Triangulace [s]
100	0,054945	0,054945
200	0,054945	0,21978
300	0,054945	0,32967
400	0,054945	0,43956
500	0,10989	0,504396
1000	0,164835	1,318681
1500	0,21978	2,032967
2000	0,384615	2,857143
2500	0,43956	3,736264
3000	0,494505	4,56044
3500	0,549451	5,274725
4000	0,604396	6,208791

Obr. 36: Časy zpracování náhodně generovaných množin bodů.



Obr. 37: Graf závislosti času přípravy mřížky a triangularizačního procesu na počtu generátorů.

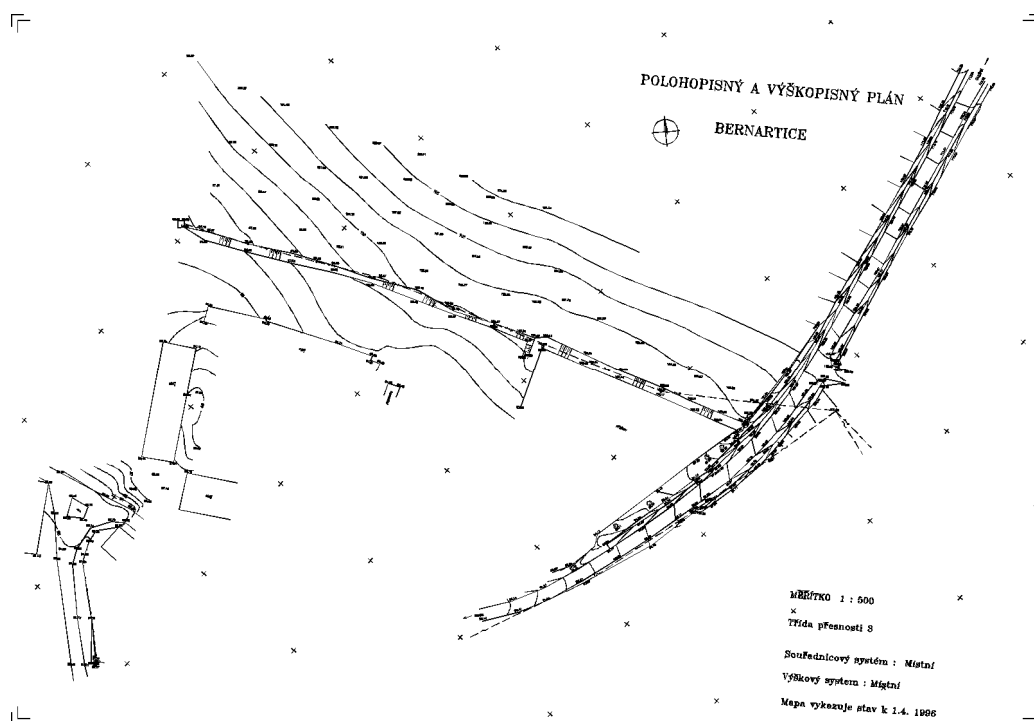
3.10.10 Závěrem

Algoritmus prezentovaný v této kapitole je založen na použití datové struktury využívající pravidelnou mříž sestavenou na vstupní množině generátorů a na kruhovém způsobu generování nových trojúhelníků triangulace. Díky této nové strategii jsou z datové struktury postupně eliminovány interní vrcholy triangulace a tím se snižuje čas potřebný pro lokalizaci generátorů při konstrukci nových trojúhelníků. I samotná lokalizace bodů je díky použití nové struktury velmi rychlá. Výpočtová složitost tohoto

algoritmu je lineární, což skutečně potvrdila její implementace. Struktura dat na výstupu nese ucelenou informaci o sestrojené trojúhelníkové síti a poskytuje možnost (pro *seznam vrcholů*), vedle snadného určení konvexní obálky bez dalších výpočtů, velmi rychle získat odpověď na velké množství dotazů.

4 Triangularizace v praxi

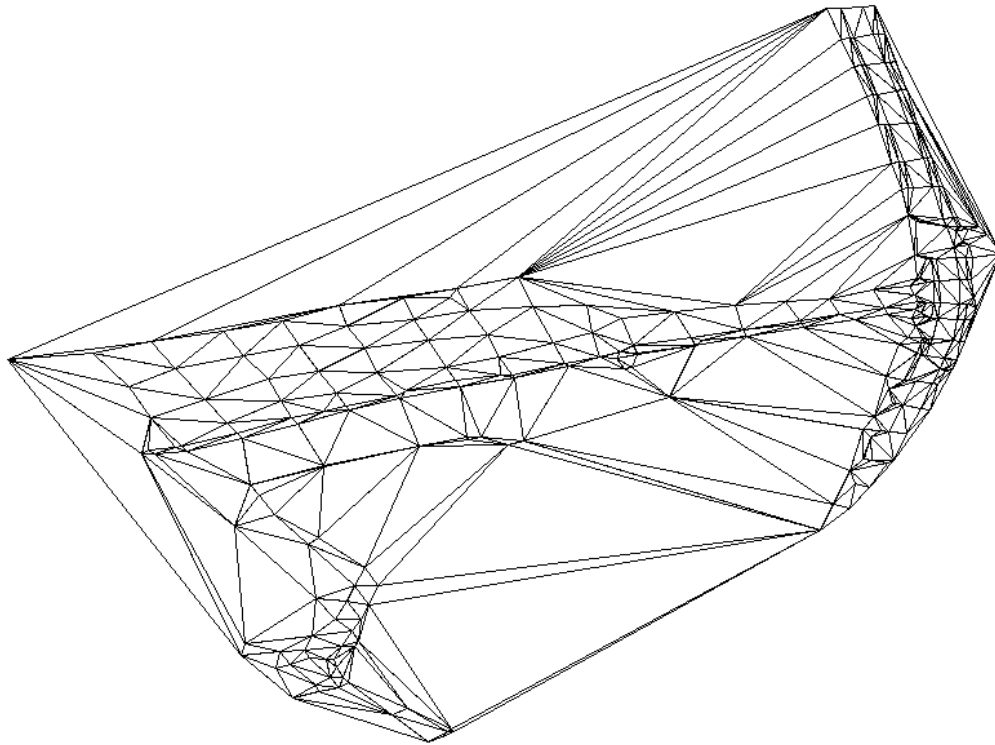
Trojúhelníková síť našla široké uplatnění v mnoha odvětvích lidské činnosti. Můžeme jmenovat např. přírodní vědy, matematika, fyzika, robotika, a mnoho dalších. V počítačové grafice se trojúhelníková síť používá např. jak pro aproximaci povrchu modelovaných těles a ploch, tak třeba i povrchu reálné krajiny. Tak jsme se přenesli do další oblasti lidského vědění, ve které má triangularizace své pevné místo, do geografie, geodézie a kartografie. Zaměřováním trigonometrické sítě je triangulací zjišťována poloha důležitých bodů na zemském tělese a výsledky jsou zpracovávány například ve formě map. Dá se říci, že prakticky celý zemský povrch naší planety je pokryt různě hustou či řídkou trigonometrickou sítí. Aniž si to uvědomujeme, při svých každodenních cestách ulicemi měst i přírodou mjíme mnoho triangulačních bodů, které jsou elementy této sítě. Pomocí těchto bodů jsou vyměřovány například nové silnice a budovy a byl jimi vyměřen i plán, který pro moji malou ukázkou laskavě zapůjčil Katastrální úřad. Je to polohopisný a výškopisný plán katastru Bernartice v měřítku 1:500 (viz. obrázek 1) a byl mi předán v souboru v jazyce HPGL. Protože byl tento soubor připraven pro tisk na velký formát, zanikají v mém zmenšeném obrázku detaily psané drobným písmem (např. výškopisné údaje).



Obr. 1: Polohopisný a výškopisný plán katastru Bernartice.

Druhým obdržným souborem byl seznam souřadnic triangulačních bodů na plánu. Z něj jsem použil pouze x -ové a y -ové souřadnice (z -ové souřadnice se využívají v navazujícím zpracování). Trojúhelníková síť sestavená mezi těmito body je pak využívána např. pro modelování terénu či pro výpočet a kresbu vrstevnic.

Pro konstrukci této trigonometrické sítě jsem použil právě poslední jmenovanou triangularizační metodu, metodu Delaunayovy triangulace s využitím pravidelné mřížky. Bohužel při převodu zobrazené triangulace do bitové mapy došlo k porušení poměru stran, přesto si můžete všimnout např. silnice vedoucí v pravé části výřezu triangulace. Výsledná triangulace je na obrázku 2.



Obr. 2: Triangulace sestavená Delaunayovou triangularizací využitím pravidelné mřížky.

Tato metoda je velmi účinná a svou lineární výpočtovou složitostí by byla velkým přínosem pro řešení problémů této oblasti, avšak její použití v této práci mělo pouze ilustrativní účel. Na první pohled je patrné, že triangularizační metoda používaná v kartografii většinou bude muset brát zřetel na pevné hrany, tj. na hrany, které jsou známy již před započítáním samotného triangularizačního procesu (viz. kapitola vázaná triangulace). Ty značívávají např. silnice, cesty, povrchové zlomy, koryta řek atd., a jednou ze základních vlastností naší metody je neschopnost zahrnout tyto pevné hrany do triangulace. Proto by bylo nutné tuto metodu s ohledem na tyto hrany modifikovat.

Závěr

Na začátku této práce jsme si připoměli základní pojmy z geometrie, které nás provázely ve všech následujících kapitolách. Hned v té první z nich jsme se seznámili s pojmem *optimální triangulace* a uvedli jsme si kritéria pro její dosažení. Definovali jsme *lokálně* a *globálně* optimální triangulaci a začali jsme používat termín *Delaunayova triangulace*. Seznámili jsme se se základním rozdělením triangularizačních algoritmů podle způsobu konstrukce triangulární sítě a začali jsme se jednomu po druhém věnovat pečlivěji. Nyní si přehledně shrneme do několika odstavců základní vlastnosti prezentovaných algoritmů a metod. Mezi algoritmy generující přímo Delaunayovu triangulaci zařadíme i algoritmy konstrující Voronoiov diagram, neboť tyto dva grafy jsou ekvivalentní (viz. kapitola „Voronoiův diagram metodou rozděli-a-panuj“).

Nenasytá triangularizace - základní verze

Typ algoritmu: Přírůstkový.

Použitá datová struktura: Zásobník.

Základní princip: Postupné přidávání předem setříděné množiny hran do triangulace od nejkratší k nejdelší.

Vlastnosti algoritmu: Velmi jednoduchý pro implementaci, je schopen zahrnout do triangularizačního procesu pevné hrany. Je paměťově náročný. Přejít do vyšší dimenze je teoreticky možný, prakticky neproveditelný pro vysokou výpočtovou složitost.

Výpočtová složitost: $O(N^3)$. Je do značné míry závislá na technice použité pro test křížení nových hran s hranami v hotové triangulaci.

Nenasytá triangularizace - modifikovaná verze (Gilbert, 1979)

Typ algoritmu: Přírůstkový.

Použitá datová struktura: Segmentový strom.

Základní princip: Postupné přidávání předem setříděné množiny hran do triangulace od nejkratší k nejdelší.

Vlastnosti algoritmu: Je schopen zahrnout do triangularizačního procesu pevné hrany. Je paměťově náročný. Použitá datová struktura je poměrně složitá.

Výpočtová složitost: $O(N^2 + N^2 \log N)$. Zlepšení oproti základní verzi přineslo použití datové struktury *segmentový strom* pro test křížení hran.

Vázaná triangularizace

Typ algoritmu: V základě přírůstkový.

Použitá datová struktura: Zásobník. Pro prohledávání řetězců při lokalizaci bodu je vhodné použít binární vyhledávací strom.

Základní princip: Dekompozice na jednodušší polygonální oblasti (monotónní polygony), které jsou již snadno triangularizovatelné, založená na řetězcové metodě lokalizace bodu v *PSLG*.

Vlastnosti algoritmu: Je vhodný především pro řešení úloh zatížených pevnými hranami. Proces správy řetězců a lokalizace bodu je poměrně složitý.

Výpočtová složitost: $O(N \log N)$.

Voronoiův diagram metodou rozděl-a-panuj

Typ algoritmu: Rozděl-a-panuj.

Použitá datová struktura: DCEL (Double-Connected-Edge-list).

Základní princip: Rozklad problému na podproblémy. Množina generátorů S je rekurzivně dělena vrtikální čarou na dvě přibližně stejně mohutné množiny S_1 a S_2 , pro které je sestaven Voronoiův diagram. Ve zpětném chodu jsou rekurzivně $\text{Vor}(S_1)$ a $\text{Vor}(S_2)$ spojovány konstrukcí rozdělovacího řetězce σ .

Vlastnosti algoritmu: Rychlý algoritmus plně využívající metody rozděl-a-panuj. Voronoiův diagram na výstupu podává komplexní informaci o zpracované množině generátorů a může být využit např. pro rychlou konstrukci konvexní obálky, či pro rychlé řešení otázky lokalizace bodu. Proces spojování parciálních diagramů je poměrně složitý. Díky tomu metodu nelze použít ve vyšších dimenzích. Algoritmus neumí do zpracování zahrnout pevné hrany.

Výpočtová složitost: $O(N \log N)$.

Přírůstková konstrukce Voronoiova diagramu

Typ algoritmu: Přírůstkový.

Použitá datová struktura: DCEL.

Základní princip: V každém kroku je lokalizován nový vkládaný generátor, pro který je vytvořen nový Voronoiův polygon a diagram je aktualizován. K lokalizaci je využíván hotový Voronoiův diagram.

Vlastnosti algoritmu: Je jednodušší než předešlá metoda. Výstupní data podávají ucelenou informaci o množině generátorů. Je možný přechod do vyšších dimenzí.

Výpočtová složitost: $O(N^2 \log N)$.

Přírůstková konstrukce Voronoiova diagramu s důrazem na topologickou stabilitu

Typ algoritmu: Přírůstkový.

Použitá datová struktura: DCEL.

Základní princip: Vychází z předešlé klasické přírůstkové metody. Před konstrukcí nového Voronoiova polygonu ovšem uzavře zpracovávnoú oblast polygonem sestrojeným mezi pomocnými body a v něm pak provede lokální restrukturalizaci diagramu.

Vlastnosti algoritmu: Metoda vychází z předešlé metody při nezměněné výpočtové složitosti. Samotná složitost řešení je ovšem vyšší. Přinese však oddálení problémů topologické nestability v důsledku numerických chyb a zvýšení počtu bodů množiny vstupních generátorů při stejné přesnosti.

Výpočtová složitost: $O(N^2 \log N)$.

Delaunayova triangularizace metodou rozděl-a-panuj

Typ algoritmu: Rozděl-a-panuj.

Základní princip: Rozklad problému na podproblémy. Množina generátorů S je rekurzivně dělena vrtikální čarou na dvě přibližně stejně mohutné množiny S_1 a S_2 , pro které je samostatně sestrojena Delaunayova triangulace. Ve zpětném chodu jsou rekurzivně $\text{Del}(S_1)$ a $\text{Del}(S_2)$ spojovány konstrukcí traverzových příček mezi levou a pravou triangulací $\text{Del}(S_1)$ a $\text{Del}(S_2)$.

Vlastnosti algoritmu: Rychlý algoritmus plně využívající metody rozděl-a-panuj. Díky procesu spojování parciálních triangulací je algoritmus složitější než klasický algoritmus založený na přírůstkové metodě. Metodu nelze použít ve vyšších dimenzích. Algoritmus neumí do zpracování zahrnout pevné hrany.

Výpočtová složitost: $O(N \log N)$.

Delaunayova triangularizace topologicky orientovanou metodou rozděl-a-panuj

Typ algoritmu: Rozděl-a-panuj.

Základní princip: Konstrukce vychází z prosté Delaunayovy triangularizace metodou rozděl-a-panuj, ve fázi spojování parciálních triangulací však ke konstrukci spojovacích příček používá speciální topologická kritéria.

Vlastnosti algoritmu: Poměrně nový stále se vyvíjející algoritmus spojující výkonnost metody rozděl-a-panuj s topologickou stabilitou modifikované přírůstkové metody. Složitostí fáze spojování parciálních triangulací při dodržení topologických kritérií je zaplácena odolnost algoritmu vůči

numerickým chybám. Metodu nelze použít ve vyšších dimenzích. Algoritmus neumí do zpracování zahrnout pevné hrany.

Výpočtová složitost: $O(N \log N)$.

Delaunayova přírůstková triangularizace

Typ algoritmu: Postupné vkládání.

Použitá datová struktura: Okřídlené hrany.

Základní princip: V prvním kroku je vygenerován pomocný trojúhelník obsahující celou množinu generátorů. V každém dalším kroku je pak lokalizován nový vkládaný generátor, pro který je proveden test na Delaunayovo kritérium kružnice opsané a v případě nutnosti je provedena lokální retriangularizace.

Vlastnosti algoritmu: Velmi jednoduchý. Paměťově náročný, neboť je třeba stále udržovat již vypočtenou triangulaci. Je možná úprava pro zpracování pevných hran. Je možný přechod do vyšších dimenzí.

Výpočtová složitost: $O(N^2)$.

Triangularizační algoritmus DeWall (Delaunay Wall)

Typ algoritmu: Rozděl-a-panuj.

Základní princip: Princip rekurzivního dělení vstupní množiny je upraven tak, že dělení probíhá střídavě horizontálně a vertikálně. Dělení tedy probíhá na čtvrtiny. Další postup je oproti klasické Delaunayově triangularizaci metodou rozděl-a-panuj opačný, tj. nejdříve mezi rozdělenými množinami vygeneruje úzký triangulační pás a pak teprve triangularizuje jím oddělené podmnožiny. Tím se vyhne náročné fázi spojování parciálních triangulací.

Vlastnosti algoritmu: Tím, že je vynechána spojovací fáze, může být algoritmus DeWall použit pro řešení problému ve vyšších dimenzích. Nepřítomnost slučovací fáze navíc algoritmus předurčuje pro paralelní zpracování, které je při použití klasických metod rozděl-a-panuj velmi obtížně realizovatelné [CIG92]. Algoritmus neumí do zpracování zahrnout pevné hrany.

Výpočtová složitost: $O(N \log N)$.

Delaunayova triangularizace využívající pravidelnou mříž

Typ algoritmu: Přírůstkový.

Použitá datová struktura: Seznam hran, či seznam vrcholů.

Základní princip: Využití pravidelné mřížky pro rychlou lokalizaci zpracovávaného generátoru a spolu se seznamem hran či seznamem vrcholů

pro novou „kruhovou“ strategii řízení triangularizačního procesu. Touto strategií jsou výsledky triangularizačního procesu ukládány již během výpočtu, interní vrcholy triangulace jsou tak uvolňovány z datové struktury a nezpomalují zbytečně proces.

Vlastnosti algoritmu: Velmi rychlý algoritmus s lineární složitostí. Struktura seznam vrcholů poskytuje na výstupu velmi ucelené informace o generované trigonometrické síti. V prezentované verzi neumožňuje zpracování pevných hran.

Výpočtová složitost: $O(N)$.

Jak je z tohoto srovnání vidět, každá metoda a každý algoritmus má své výhody a své nevýhody a její výběr je tedy nutné přizpůsobit specifickým požadavkům řešeného problému a situace.

Použitá literatura

- [BOW81] A. Bowyer, „Computing Dirichlet tessellations“
The Computer Journal, VOL. 24, NO. 2, 1981, str. 162-166
- [CIG92] P. Cignoni, C. Montani, R. Perego, R. Scopigno
„Parallel 3D Delaunay Triangulation“, November 1992
(E-mail: scop@icnucevm.cnuce.cnr.it), str. 1-22
- [FANG93] Tsung-Pao Fang, Les A. Piegl, „Delaunay Triangulation Using
a Uniform Grid“, IEEE Computer Graphics & Applications,
May 1993, str. 36-47
- [HOSCH] Hoscheck, Lasser, „Grundlagen der geometrischen
Datenverarbeitung“, B. G. Teubner Stuttgart,
kap. „Triangulierung von Punktmengen“, str. 320-327
- [SUGI92] Kokichi Sugihara, Masao Iri, „Construction of the Voronoi
Diagram for „One Million“ Generators in Single-Precision
Arithmetic“, PROCEEDINGS OF THE IEEE, VOL. 80,
NO. 9, September 1992, str. 1471-1484
- [OIS93] Yasuaki Oishi, Kokichi Sugihara, „Topology-Oriented
Divide-and-Conquer Algorithm for Voronoi Diagrams“,
METR 93-12, July 1993, str. 1-21
- [PAE95] Alan W. Paeth, „Graphics Gems V“, Academic press 1995,
kap. „Fast Polygon Triangulation Based on Seidel’s Algorithm“
Atul Narkhede, Dinesh Manocha, str. 394-397, kap.
„Incremental Delaunay Triangulation“, Dani Lischinski,
str. 47-59
- [PREP85] Franko P. Preparata, Michael Ian Shamos, „Computational
Geometry an introduction“, Springer-Verlag 1985
- [SKA92] Václav Skala, „Algoritmy Počítačové grafiky“,
Ediční středisko ZČU, Plzeň 1992