Výpočetní geometrie

Pavel Strachota

FJFI ČVUT v Praze

19. července 2023

Obsah

Obsah

Problémy výpočetní geometrie

problémy z oboru teoretické matematiky, numerické matematiky, počítačové grafiky, robotiky, kartografie...

- poloha bodu vůči polygonu, mnohostěnu (polyhedron)
- poloha bodu na mapě (vzhledem k rozdělení roviny)
- průsečíky úseček, průniky polygonů
- triangulace polygonu
- **nalezení konvexního obalu** ve 2D, 3D
  - jeden z nejstarších problémů
- Voroného diagramy
- Deloneho triangulace
- hledání nejkratší cesty
- detekce kolizí v simulacích, počítačových hrách atd.
- generování výpočetních sítí pro metodu konečných prvků, resp. objemů

Řešení problémů výpočetní geometrie

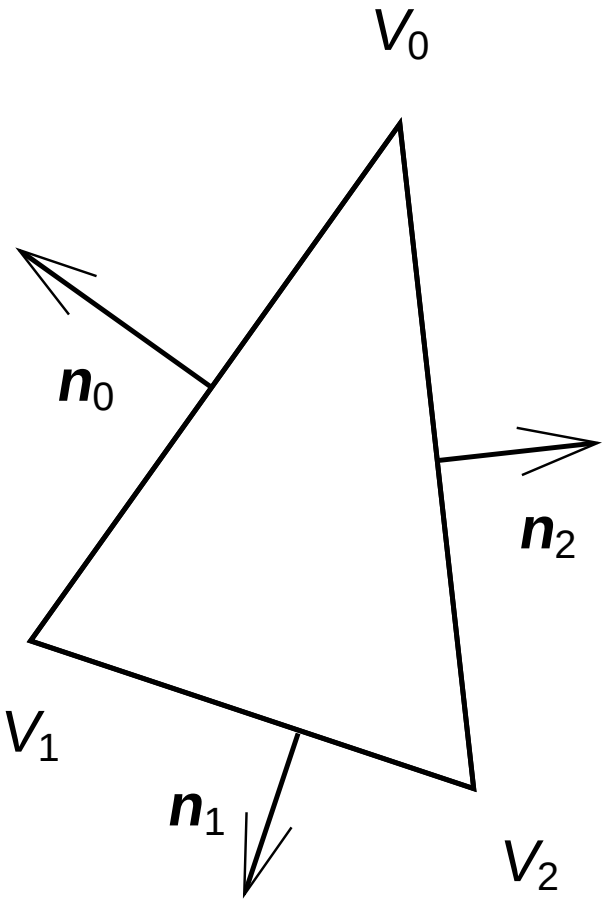
- **datové struktury**
  - BSP stromy - *Binary Space Partitioning tree*
  - quadtree, octree
- **složitost** algoritmů
  - časová
  - paměťová
- snaha vyvinout co **nejefektivnější** algoritmy
  - rozděl a panuj
  - znáhodněné algoritmy
- **robustní** algoritmy - odolné vůči
  - speciálním degenerovaným případům
  - problémům při implementaci v **aritmetice s konečnou přesností**

Obsah

Poloha bodu vůči polygonu

- určit, zda určitý bod P leží uvnitř nebo vně daného polygonu v rovině
- např. u **trojúhelníku**:
- vrcholy V_i, $i = 0,1,2$ **proti směru** hodinových ručiček
- hrany $\boldsymbol{e}_i = \left(e_i^x, e_i^y\right)^{\mathrm{T}} = V_{(i+1) \bmod 3} - V_i$, $i = 0,1,2$
- vnější normály ke hranám jsou pak $\boldsymbol{n_i} = \left(e_i^y, -e_i^x\right)^{\mathrm{T}}$
- P leží uvnitř $\triangle$, jestliže $\forall i = 0,1,2$

$$\boldsymbol{n}_i \cdot (P - V_i) < 0$$



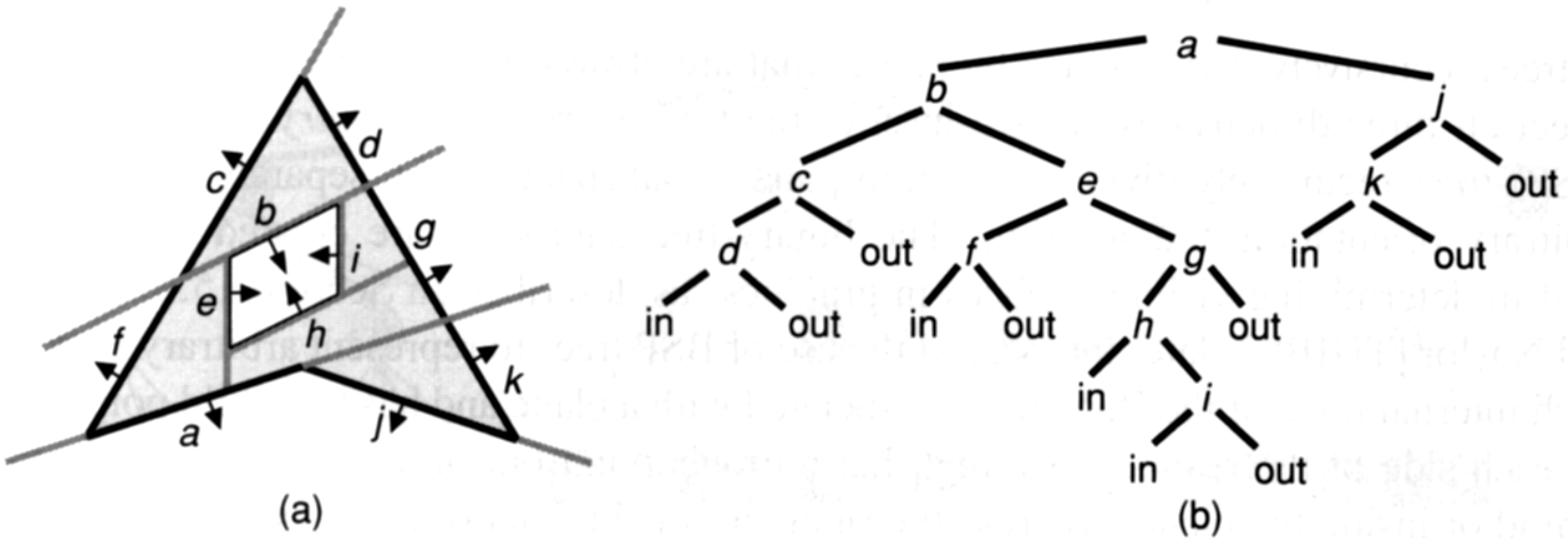
Poloha bodu vůči polygonu

BSP stromy

- **BSP** = *Binary Space Partitioning* - binární rozdělení prostoru
- rekurzivní dělení roviny na poloroviny oddělené libovolně umístěnou a orientovanou přímkou
- reprezentace libovolných polygonálních útvarů v rovině, resp. mnohostěnů v prostoru
- uzel BSP stromu - svázán s přímkou, jejíž normála směřuje ven z objektu (úmluva)
- 2 potomci:
  - **levý** potomek = rozdělení poloroviny **proti směru** normály k přímce, nebo NULL (lze implementovat jako list „*in*“ - identifikuje polorovinu, která leží uvnitř útvaru)
  - **pravý** potomek = rozdělení poloroviny **ve směru** normály k přímce, nebo NULL (lze implementovat jako list „*out*“ - identifikuje polorovinu, která leží vně útvaru)

Poloha bodu vůči polygonu

BSP stromy - příklad



Poloha bodu vůči polygonu

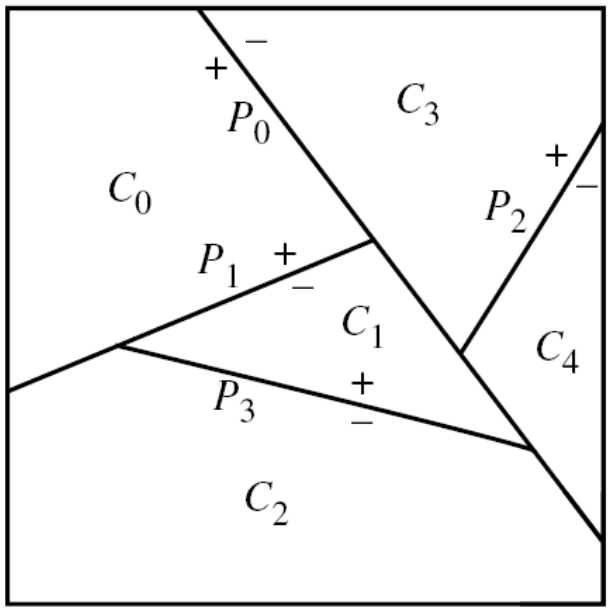
Vyšetření pomocí BSP stromu

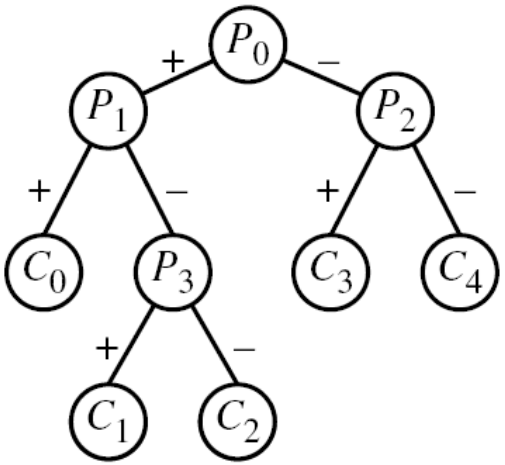
- začínáme v kořeni stromu
- přímka p daná bodem X a normálovým vektorem $\boldsymbol{n}$
- test, v jaké polorovině leží bod P:
  - $\boldsymbol{n} \cdot (P - X) < 0 \implies P$ leží v polorovině **proti** směru normály $\implies$ vyšetřujeme **levý** podstrom
  - $\boldsymbol{n} \cdot (P - X) > 0 \implies P$ leží v polorovině **po** směru normály $\implies$ vyšetřujeme **pravý** podstrom
- pokud přejdeme do oblasti „**in**“ (tj. při přesunu **vlevo** narazíme na NULL), bod P leží **v** polygonu
- pokud přejdeme do oblasti „**out**“ (tj. při přesunu **vpravo** narazíme na NULL), bod P leží **mimo** polygon

Poloha bodu v rovině

Vyšetření pomocí BSP stromu

- prostor rozdělen na oblasti (státy na mapě apod.)
- zobecnění - přímka dělí prostor na levou a pravou polorovinu (poloroviny $+$ a $-$)
- v listech stromu je místo „*in*“ a „*out*“ označení oblasti C_i
- procházení stromem nám řekne, v jaké oblasti leží bod P

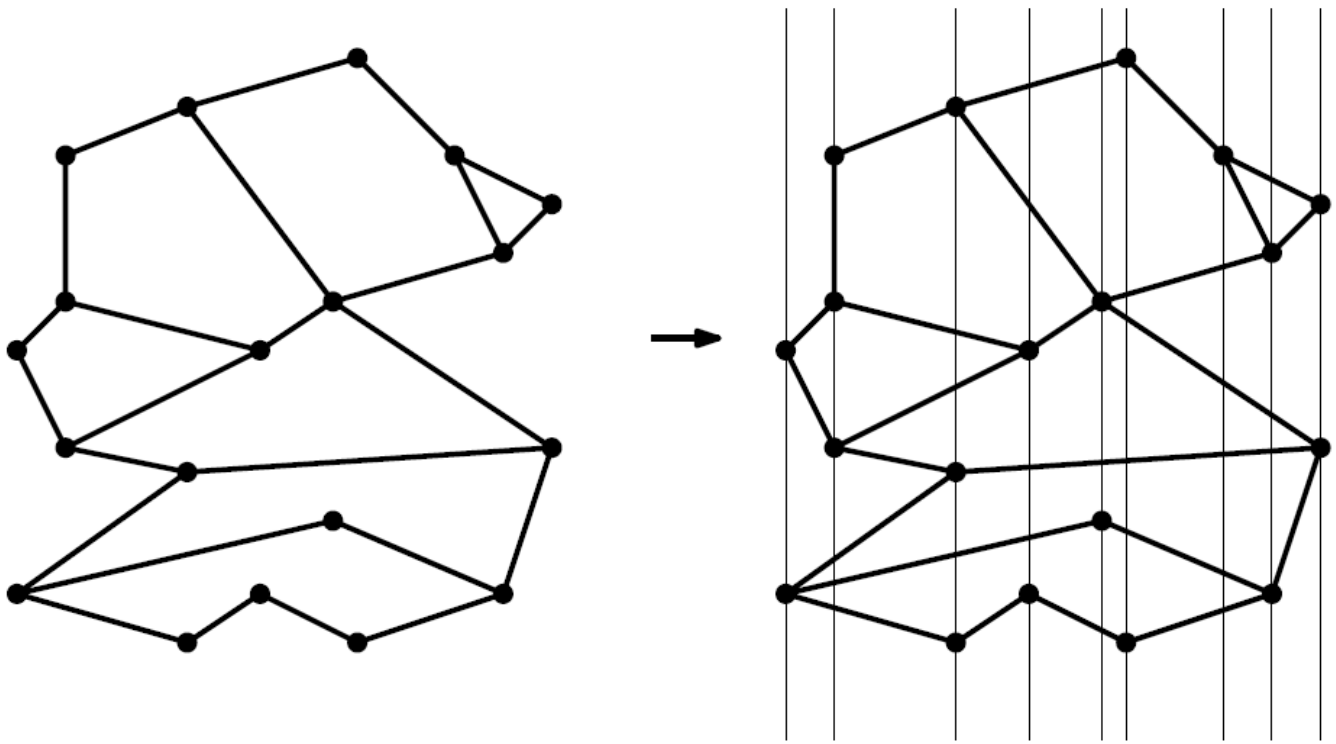




Poloha bodu v rovině

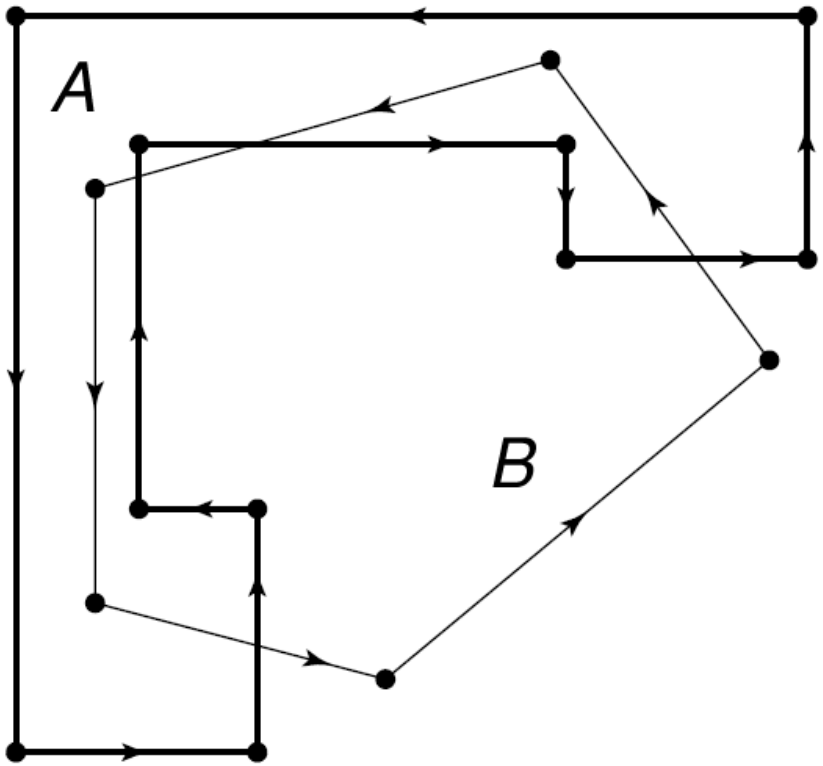
Obecný případ

- oblasti omezeny k sobě přiléhajícími polygony
- algoritmus pro vytvoření **mapy lichoběžníků** vytvoří datovou strukturu (opět strom), v níž lze následně snadno vyhledávat pozici bodu [deBerg, kap. 6]



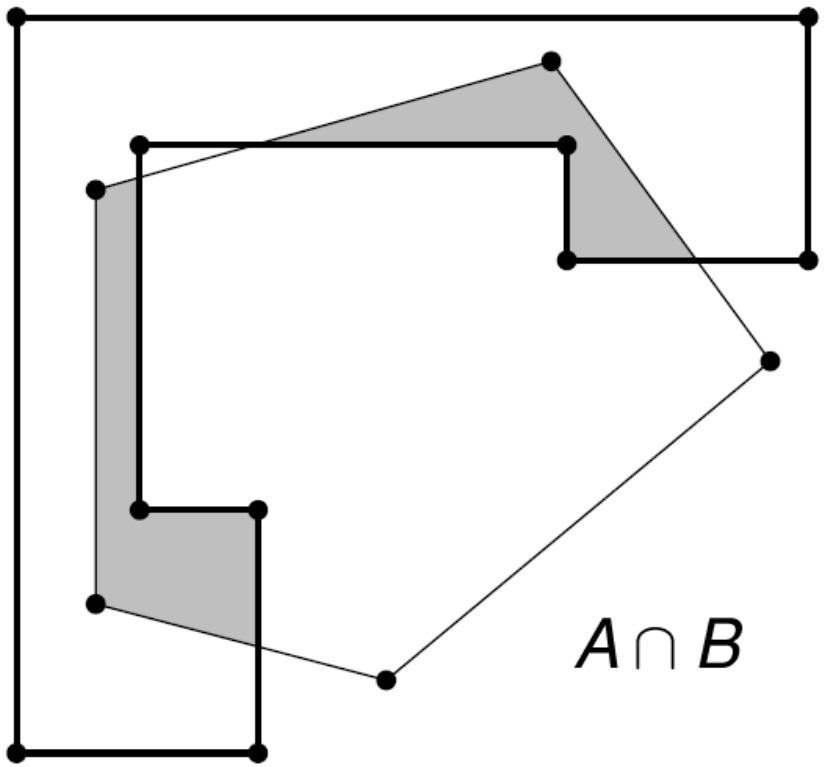
Booleovské operace s polygony

- průnik, sjednocení, rozdíl, negace
- snadno implementovatelné pomocí BSP stromů



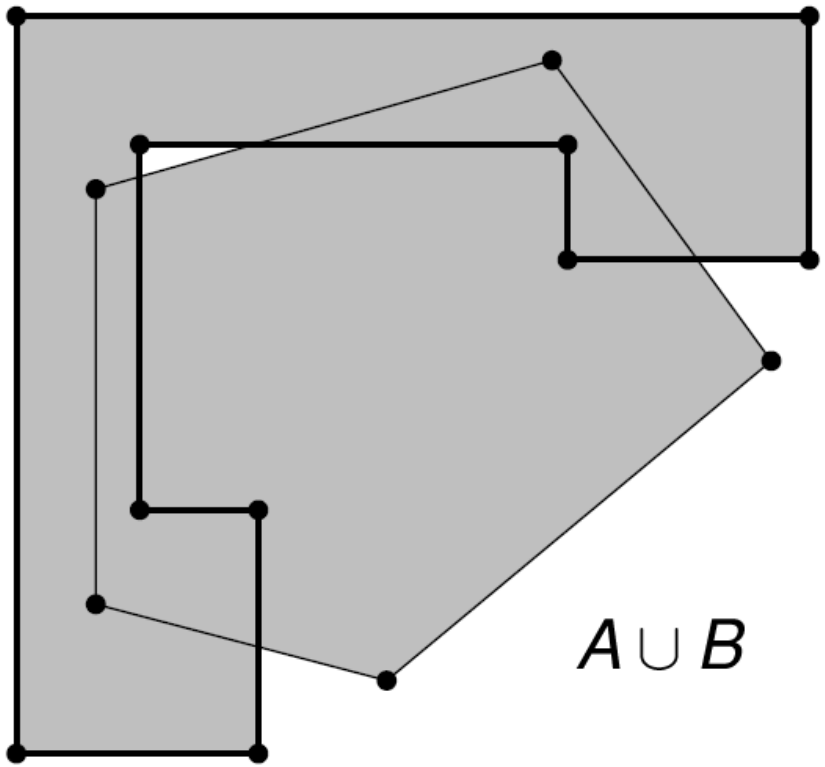
Booleovské operace s polygony

- průnik, sjednocení, rozdíl, negace
- snadno implementovatelné pomocí BSP stromů



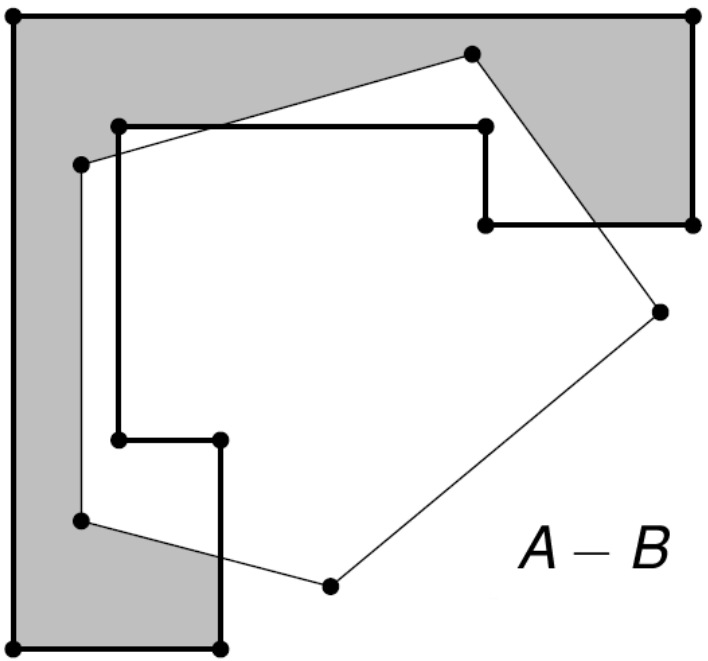
Booleovské operace s polygony

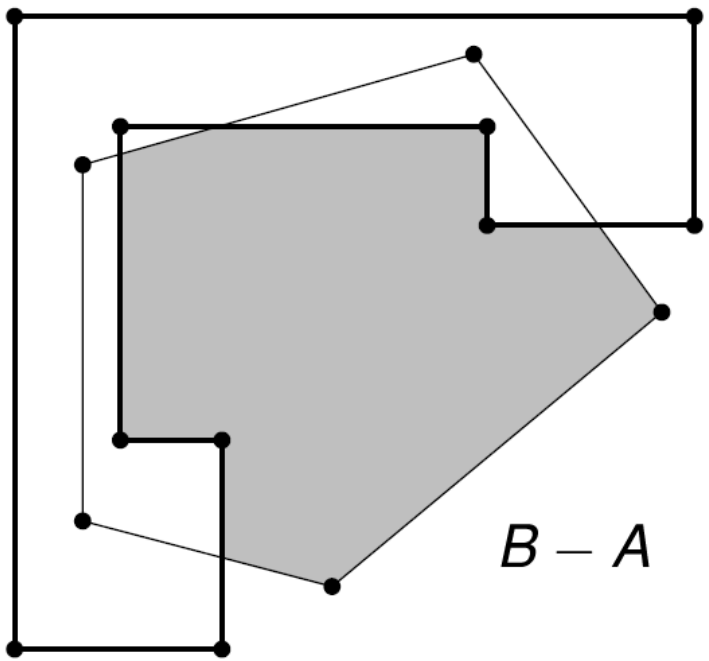
- průnik, sjednocení, rozdíl, negace
- snadno implementovatelné pomocí BSP stromů



Booleovské operace s polygony

- průnik, sjednocení, rozdíl, negace
- snadno implementovatelné pomocí BSP stromů





Booleovské operace s polygony

Implementace pomocí BSP

negace

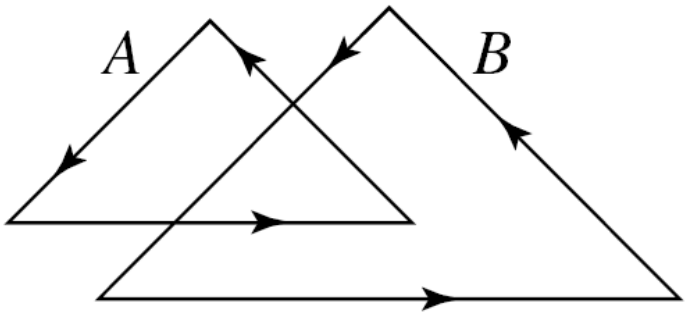
- polygon - seznam orientovaných hran
- orientace určuje směr normály - BSP reprezentace snadná
- negovaný polygon - seznam stejných hran orientovaných opačně $\Longrightarrow$ normály směřují na druhou stranu

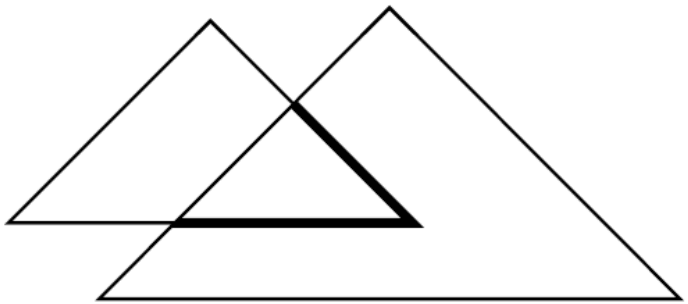
průnik

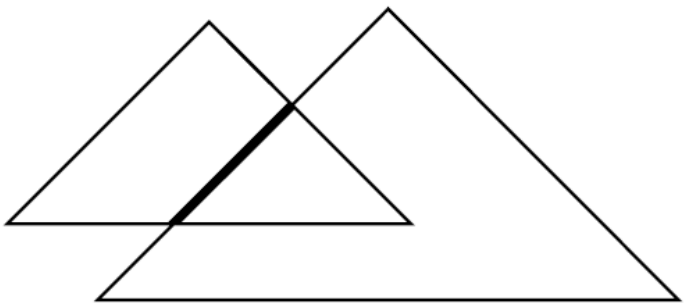
- ořezávání polygonu polygonem - viz přednáška „rastrové algoritmy“
- polygony A, B s m, resp. n hranami
- princip: $A \cap B =$
  průnik hran A s polygonem B + průnik hran B s polygonem A $\Longrightarrow$ celkem mn testů
- BSP reprezentace $\Longrightarrow$ omezení počtu testů

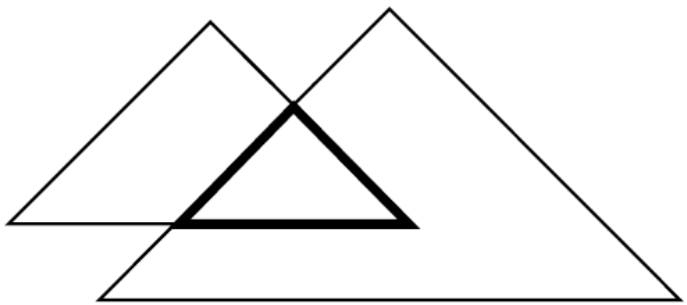
Booleovské operace s polygony

Implementace pomocí BSP









Booleovské operace s polygony

Implementace ostatních operací

- sjednocení

$$A \cup B = \neg(\neg A \cap \neg B)$$

- rozdíl

$$\begin{aligned} A - B &= A \cap \neg B \\ B - A &= B \cap \neg A \end{aligned}$$

Nalezení konvexního obalu

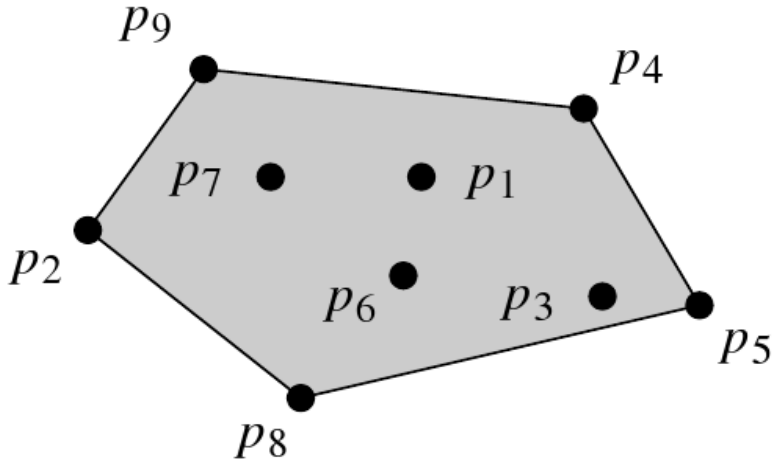
Obecná definice

Konvexní obal (*convex envelope*, *convex hull*) množiny A je nejmenší konvexní množina obsahující A.

- konvexní obal množiny bodů $A = \{p_1, p_2, \ldots p_n\}$:

$$\mathcal{CH}(A) = \left\{ \sum_{k=1}^{n} \alpha_k p_k \middle| \alpha_k \geq 0, \sum_{k=1}^{n} \alpha_k = 1 \right\}$$

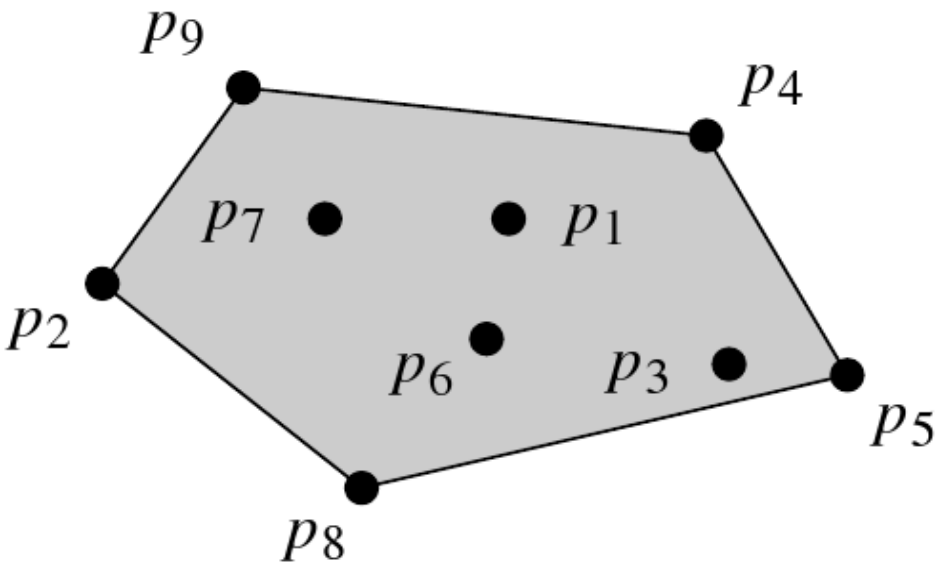
- algoritmy pro nalezení konvexního obalu
  - balení dárku
  - quickhull
  - inkrementální algoritmus
  - Grahamův algoritmus
  - algoritmus typu rozděl a panuj



Nalezení konvexního obalu

Hledání vrcholů konvexního obalu

- **hranice** konvexního obalu množiny bodů A v rovině je konvexní polygon $\mathcal{CP}(A)$
- výsledek nalezení konvexního obalu - posloupnost vrcholů tohoto polygonu (po směru hodinových ručiček)



$$A = \{p_1, p_2, \ldots, p_9\}$$
$$\mathcal{CP}(A) = (p_4, p_5, p_8, p_2, p_9)$$

Nalezení konvexního obalu

Hledání vrcholů konvexního obalu

- **hranice** konvexního obalu množiny bodů A v rovině je konvexní polygon $\mathcal{CP}(A)$
- výsledek nalezení konvexního obalu - posloupnost vrcholů tohoto polygonu (po směru hodinových ručiček)

Pozorování

Bod $p \in A$ je vrcholem $\mathcal{CP}(A) \iff p$ neleží **uvnitř** žádného trojúhelníku tvořeného jinými třemi body z A.

primitivní hledání vrcholů $\mathcal{CP}(A)$:

- pro každý bod $p \in A$ bod sestroj všechny trojúhelníky qrs ($q, r, s \in A$, $p \neq q \neq r \neq s$).
- zařaď p do $\mathcal{CP}(A)$ pokud vždy $p \notin \text{interior}(\triangle qrs)$

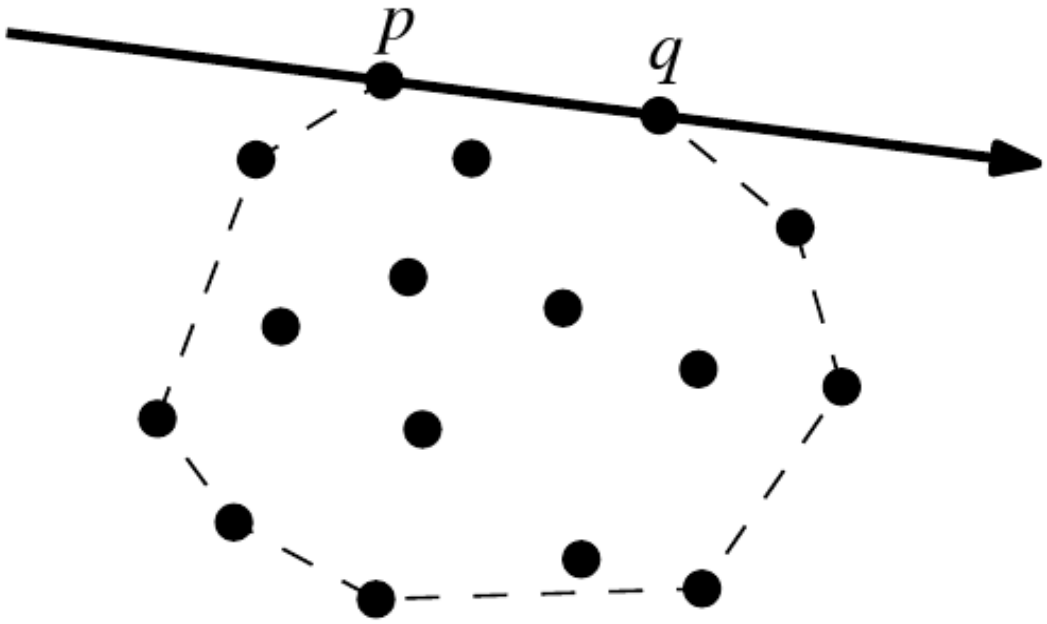
$\implies$ časová složitost $O\left(n^4\right)$ - nepoužitelné

Nalezení konvexního obalu

Hledání hran konvexního obalu

Pozorování

Úsečka $\overline{pq}$, $p, q \in A$ je hranou $\mathcal{CP}(A) \iff$ všechny body $r \in A$ leží na jedné straně přímky pq



Nalezení konvexního obalu

Hledání hran konvexního obalu

Pozorování

Úsečka $\overline{pq}$, $p, q \in A$ je hranou $\mathcal{CP}(A) \iff$ všechny body $r \in A$ leží na jedné straně přímky pq

primitivní hledání hran $\mathcal{CP}(A)$:

- pro každou přímku pq a každý bod $r \in A, r \neq p, q$ najdi polohu r vůči pq
  - opět na základě znaménka skalárního součinu $(r - p)$ s normálovým vektorem přímky
- hranu $\overline{pq}$ zařaď do $\mathcal{CP}(A)$, pokud je znaménko skal. součinu $\forall r$ stejné
- seřaď vrcholy po směru hodinových ručiček

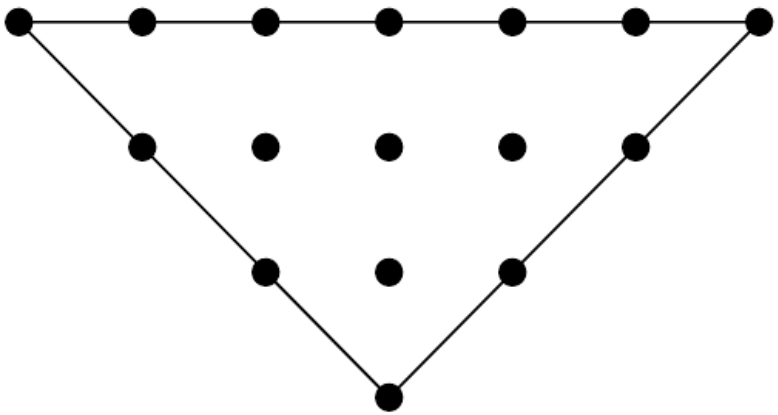
$\implies$ časová složitost $O\left(n^3\right)$

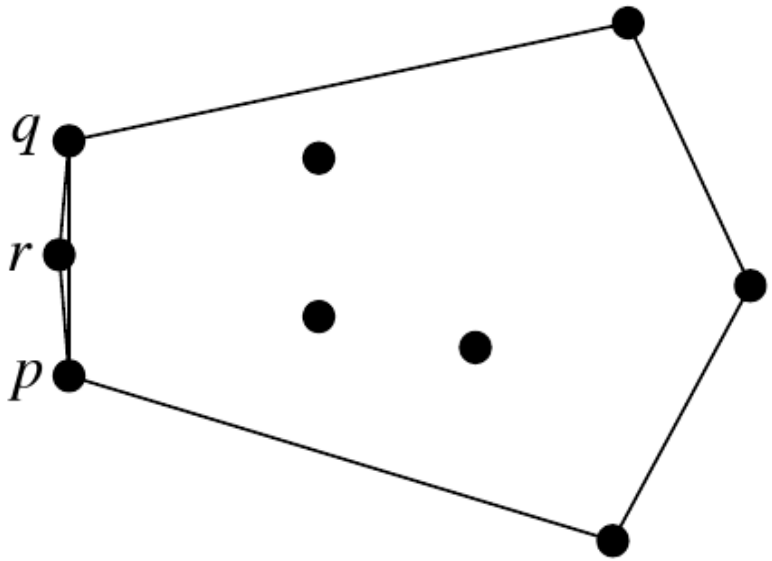
Nalezení konvexního obalu

Problémy při hledání hran konvexního obalu 1/2

takto navržený algoritmus **není robustní**:

- **degenerované** případy: bod r přímo nebo **téměř** na úsečce $\overline{pq}$
  - r přímo na $\overline{pq} \implies$ správně by neměl hrát roli v rozhodování o této hraně

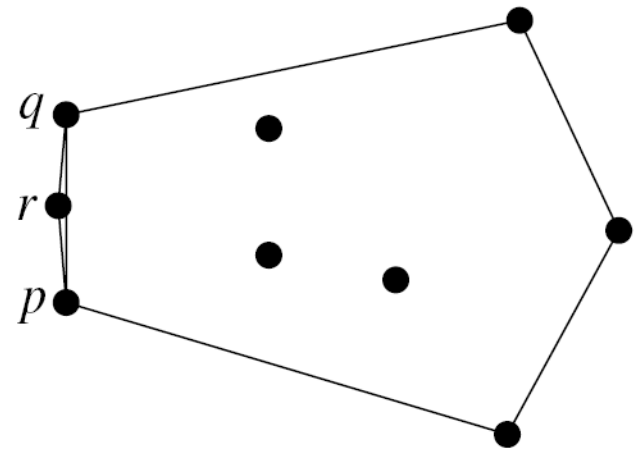


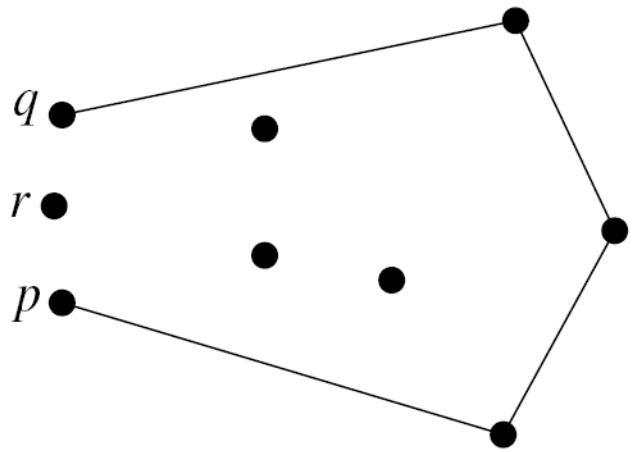


Nalezení konvexního obalu

Problémy při hledání hran konvexního obalu 2/2

- projeví se **konečná přesnost**
- např. může zdánlivě být zároveň
  - r vpravo od pq (při cestě z p do q máme bod r po pravé straně),
  - q vpravo od pr,
  - p vpravo od rq
- geometricky nelze, pouze efekt numerických chyb
  $\Longrightarrow$ všechny 3 hrany se zařadí do $\mathcal{CP}(A)$
- nebo naopak: všechny body „vlevo“
  $\Longrightarrow$ žádná z hran do $\mathcal{CP}(A)$
  $\Longrightarrow$ $\mathcal{CP}(A)$ nepopisuje uzavřený polygon

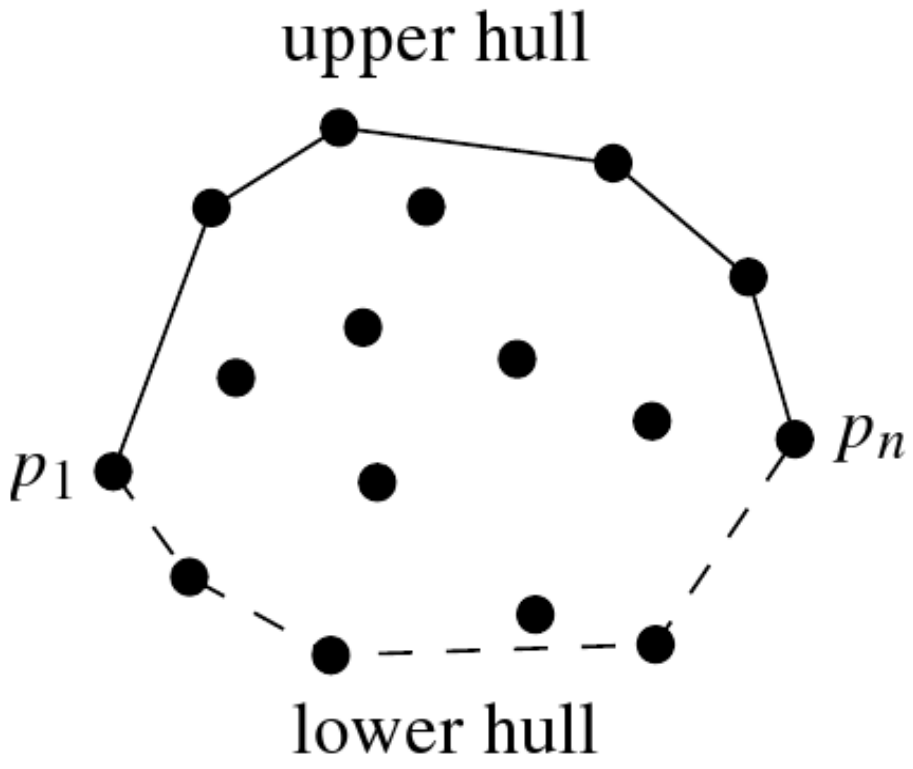




Nalezení konvexního obalu

Inkrementální algoritmus

- procházíme body z A podle souřadnice x (zleva doprava)
- bod s minimální souřadnicí x určitě patří do $\mathcal{CP}(A)$
- tvoříme **horní** část konvexního obalu (*upper hull*) - $\mathcal{CP}_u(A)$
- $\mathcal{CP}_u(A)$ se tvoří postupně - v i-tém kroku máme $\mathcal{CP}_u(\{p_0, p_1, \ldots p_i\})$
- **horní** část skutečného konvexního obalu vždy tvoří tzv. **pravý záhyb** (*right turn*), tj. při cestě od p_i k p_{i+1} zahýbáme doprava



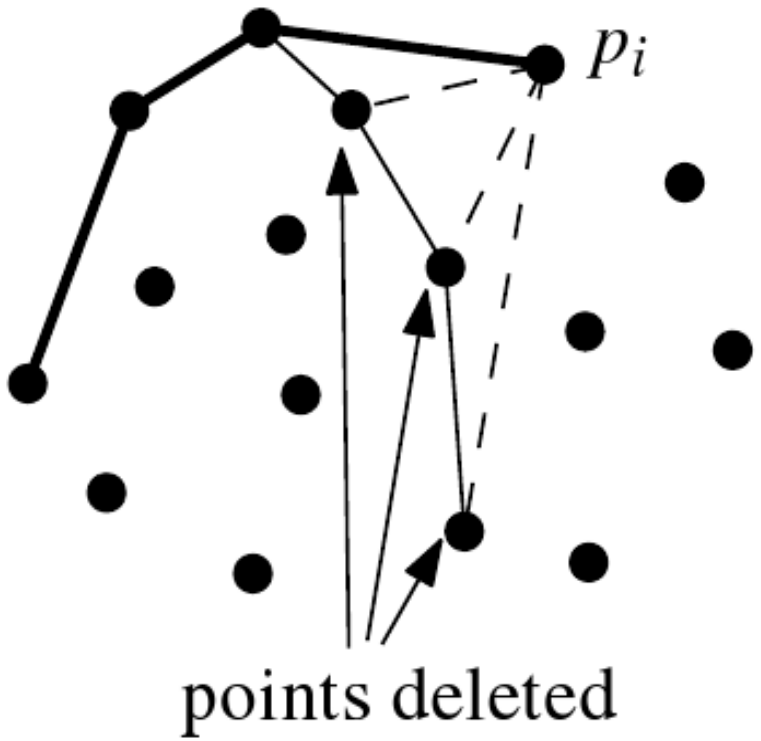
Nalezení konvexního obalu

Inkrementální algoritmus - pravý záhyb

iterace od $\mathcal{CP}_u(\{p_0, p_1, \ldots p_i\})$ k $\mathcal{CP}_u(\{p_0, p_1, \ldots, p_{i+1}\})$

---

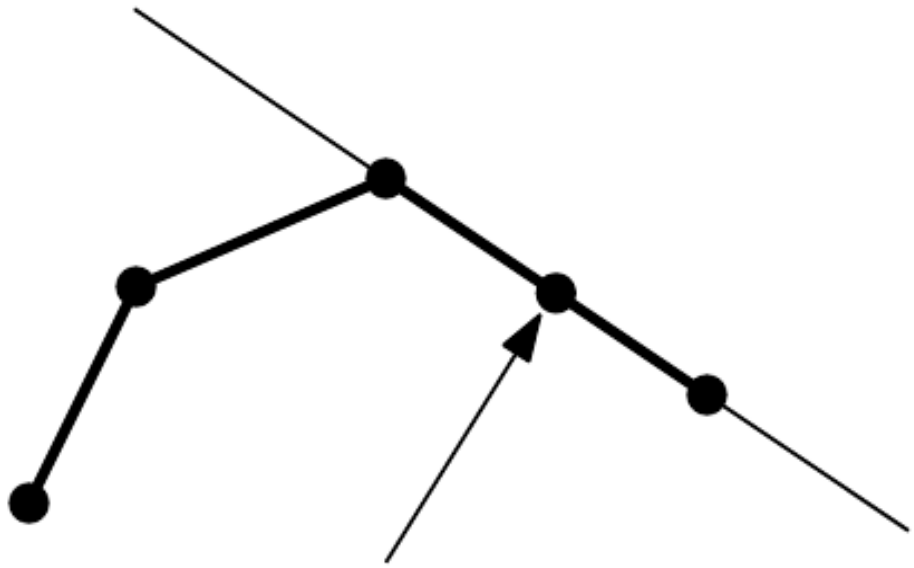
1. $M := \mathcal{CP}_u(\{p_0, p_1, \ldots p_i\})$
2. přidáme bod p_{i+1} do M
3. pokud v M jsou max. 2 body
   $\Longrightarrow$ M je $\mathcal{CP}_u(\{p_0, p_1, \ldots p_{i+1}\})$
4. zjistíme, zda **poslední 3 body** v M tvoří pravý záhyb
   - pokud **ANO**, M už je $\mathcal{CP}_u(\{p_0, p_1, \ldots p_{i+1}\})$
   - pokud **NE** (tvoří **levý** záhyb), odstraníme **prostřední bod** a jdeme na bod 3.



Nalezení konvexního obalu

Inkrementální algoritmus - detaily

- tři body na jedné úsečce netvoří ani pravý, ani levý záhyb, ale prostřední bod v $\mathcal{CP}_u(A)$ nepotřebujeme
  $\Longrightarrow$ test „**NE**tvoří pravý záhyb“ = „tvoří levý záhyb nebo jsou na přímce“
- neuvažovali jsme více bodů se stejnou souř. x
  $\Longrightarrow$ místo uspořádání podle x lze vzít **lexikografické** uspořádání podle x, y

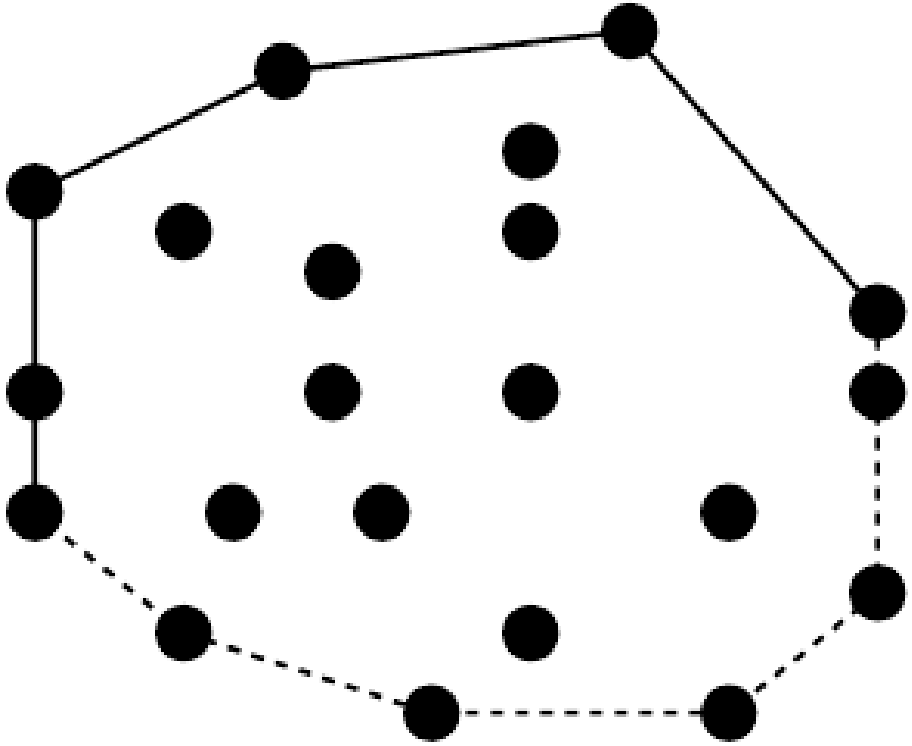


$$\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} > \begin{pmatrix} x_2 \\ y_2 \end{pmatrix} \iff (x_1 > x_2 \vee (x_1 = x_2 \wedge y_1 > y_2))$$

Nalezení konvexního obalu

Inkrementální algoritmus - detaily

- tři body na jedné úsečce netvoří ani pravý, ani levý záhyb, ale prostřední bod v $\mathcal{CP}_u(A)$ nepotřebujeme
  $\Longrightarrow$ test „**NE**tvoří pravý záhyb“ = „tvoří levý záhyb nebo jsou na přímce“
- neuvažovali jsme více bodů se stejnou souř. x
  $\Longrightarrow$ místo uspořádání podle x lze vzít **lexikografické** uspořádání podle x, y



$$\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} > \begin{pmatrix} x_2 \\ y_2 \end{pmatrix} \iff (x_1 > x_2 \vee (x_1 = x_2 \wedge y_1 > y_2))$$

Nalezení konvexního obalu

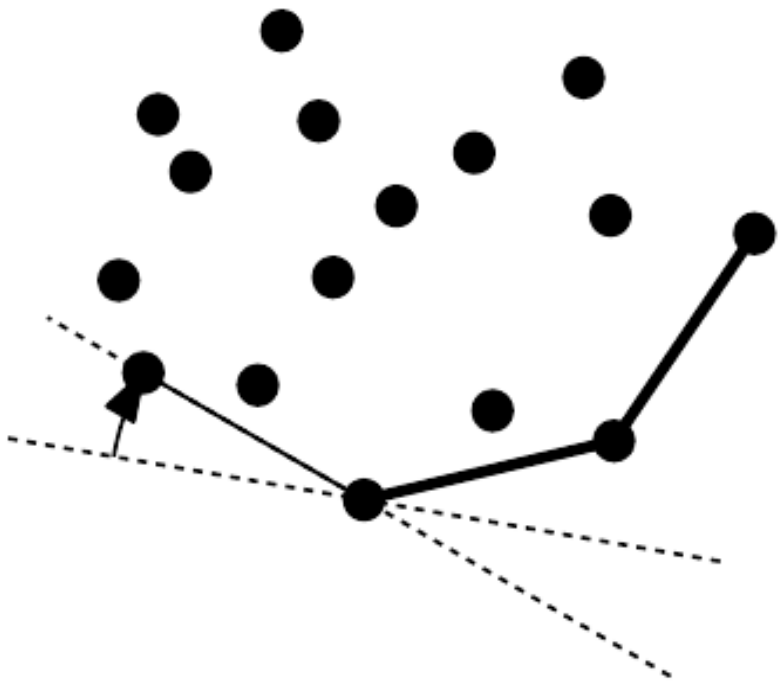
Inkrementální algoritmus - dokončení

- analogicky nalezneme dolní část konvexního obalu $\mathcal{CP}_l(A)$
- postupujeme stejně, ale zprava doleva
- odstraníme první a poslední bod z $\mathcal{CP}_l(A)$, neboť již jsou v $\mathcal{CP}_u(A)$
- chyby způsobené **numerickou nepřesností**:
  - nějaký bod navíc, resp. nějaký chybí
  - stále máme seznam uzlů, který lze interpretovat jako uzavřený polygon
  - odchylka od skutečného $\mathcal{CH}(A)$ je stejného řádu jako numerické chyby
    $\Longrightarrow$ algoritmus **je robustní**
- **časová složitost** je $O(n \log n)$
  - seřazení bodů lze provést v $O(n \log n)$
  - samotné nalezení $\mathcal{CP}_u(A)$ a $\mathcal{CP}_l(A)$ trvá jen $O(n)$: každý bod totiž přidáme i testujeme na odebrání právě jednou

Nalezení konvexního obalu

Algoritmus balení dárku

- začneme od bodu p_0 s maximální souřadnicí x
- rotujeme vertikální přímku procházející p_0 **po směru hodinových ručiček**, dokud nenarazíme na další bod p_1
- přepneme se do bodu p_1 a pokračujeme v rotování přímky
- skončíme, když narazíme opět na p_0
- složitost je $O(nh)$, kde h je počet vrcholů $\mathcal{CP}(A)$
  - „rotování přímky“ totiž spočívá v otestování všech zbývajících bodů z A a výběru toho s minimálním úhlem rotace



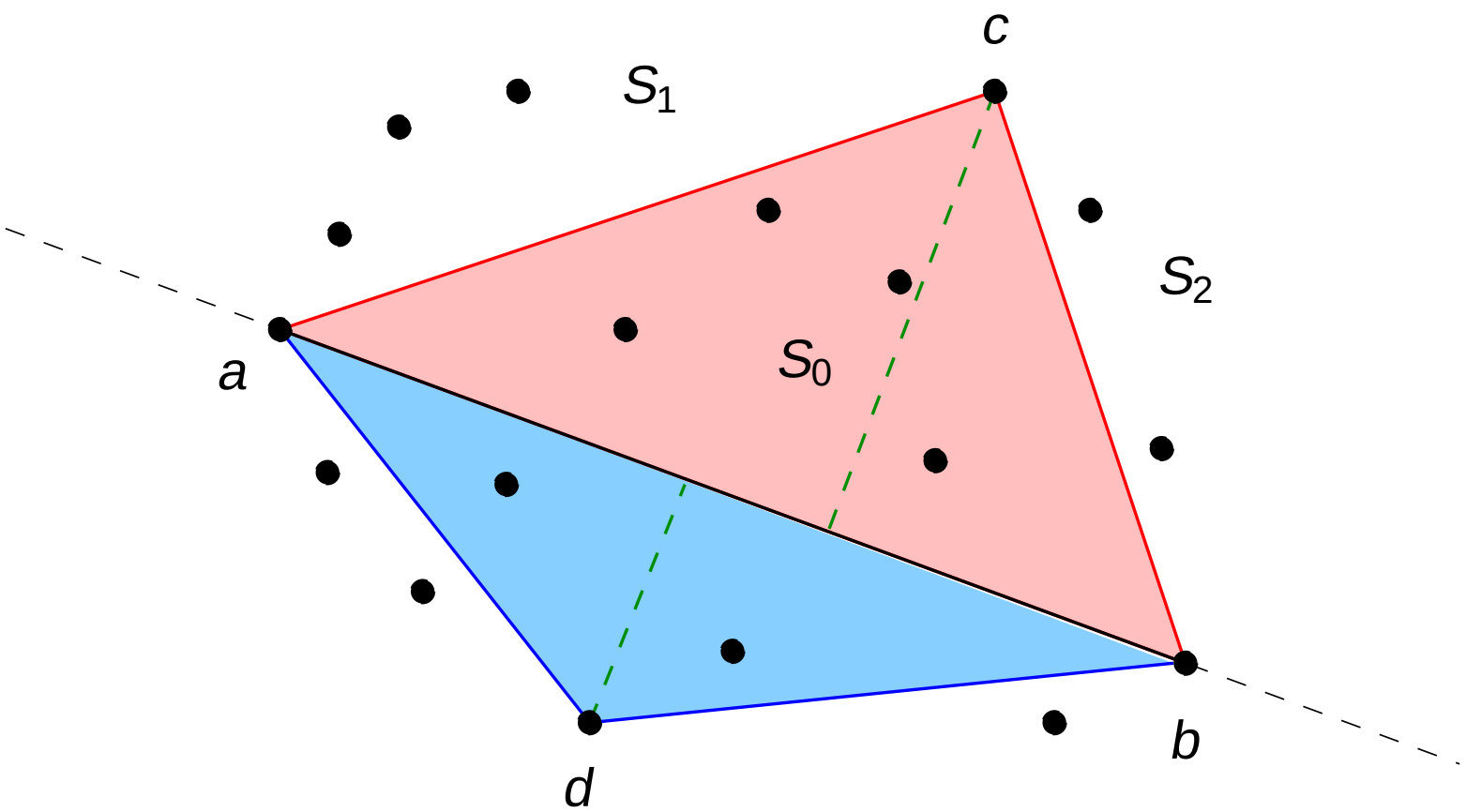
Nalezení konvexního obalu

Algoritmus Quickhull 1/4

1. nalezneme extrémní body na ose x (a nejvíce vlevo, b nejvíce vpravo)
   $\Longrightarrow\ a, b \in \mathcal{CP}(A)$
2. nalezneme body c, d nejdále od přímky ab, resp. ba **nalevo**
   $\Longrightarrow\ c, d \in \mathcal{CP}(A)$
3. odstraníme množinu S_0 všech bodů uvnitř $\triangle abc$ (resp. $\triangle abd$)
4. algoritmus rekurzivně aplikujeme na
   - přímku ac a množinu S_1 všech bodů nalevo od ac
   - přímku cb a množinu S_2 bodů nalevo od cb

Nalezení konvexního obalu

Algoritmus Quickhull 2/4



Nalezení konvexního obalu

Algoritmus Quickhull 3/4

pseudokód:

```
function QuickHull(point a, point b, set S) {
 if(S = ∅) return ∅;
 find point c farthest from line ab to the left;
 assemble sets S0, S1, S2;
 return QuickHull(a, c, S1) ∪ {c} ∪ QuickHull(c, b, S2)
}
function QuickHull_Main(set A) {
 find initial extreme points a, b;
 return {a} ∪ QuickHull(a, b, A) ∪ {b} ∪ QuickHull(b, a, A);
}
```

Nalezení konvexního obalu

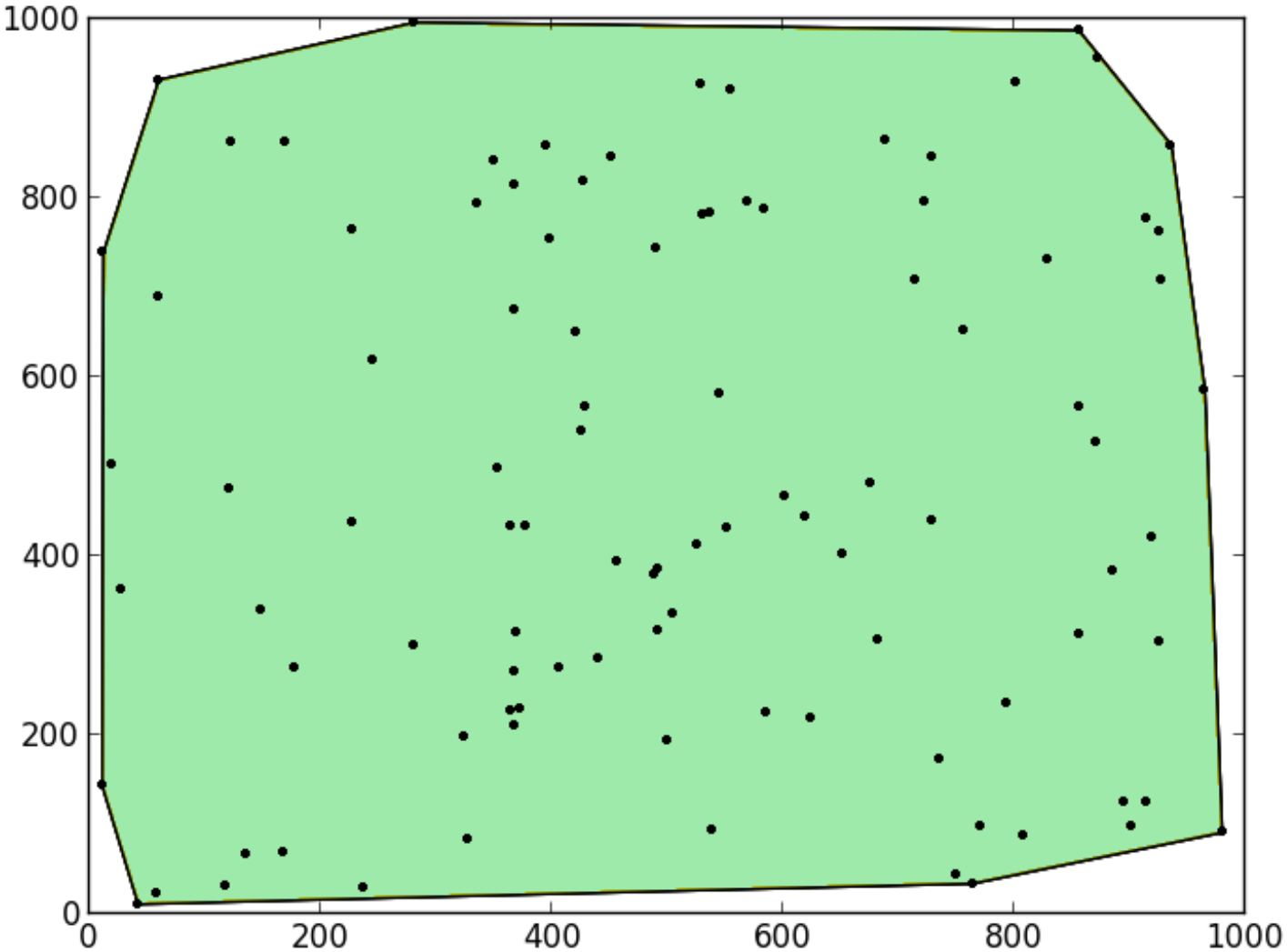
Algoritmus Quickhull 4/4

složitost algoritmu:

- v nejhorším případě $O(nh)$, kde h je počet vrcholů $\mathcal{CP}(A)$
  - trojúhelníků se konstruuje $h-2$, na polohu bodu vůči trojúhelníku testujeme max. n bodů
  - nastane tehdy, když většina bodů je již na hranici konvexního obalu
  - žádné body nezahazujeme
- v typickém případě $O(h \log n)$
  - při konstrukci každého $\triangle$ zahodíme do množiny S_0 přibližně polovinu bodů
  - nastane, když body jsou v rovině rozmístěny rovnoměrně

Srovnání reálné složitosti algoritmů

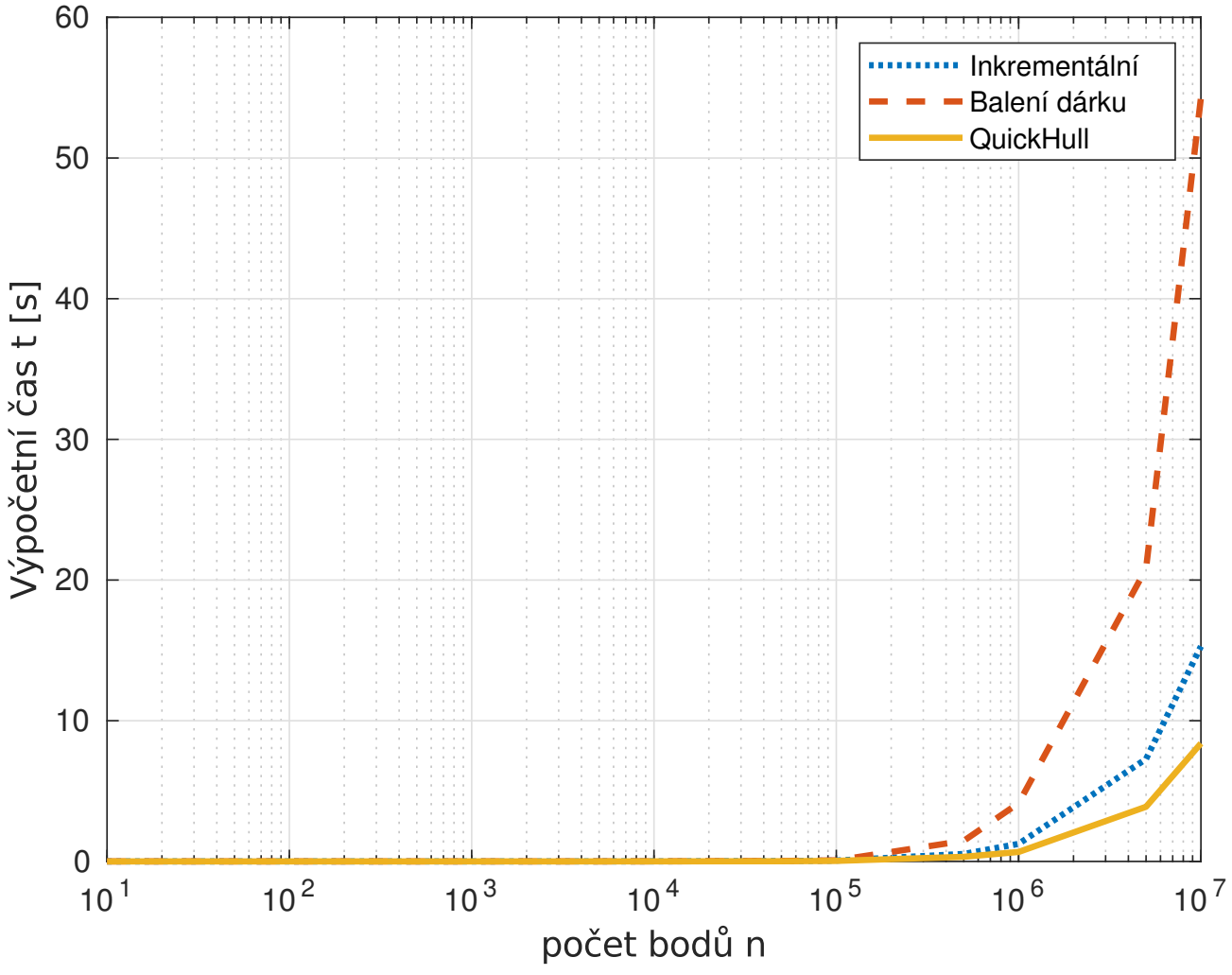
Zápočtová práce studenta POGR1 Davida Fridricha v AR 2012/13 (algoritmy v C++)



test: 100 bodů v rovině

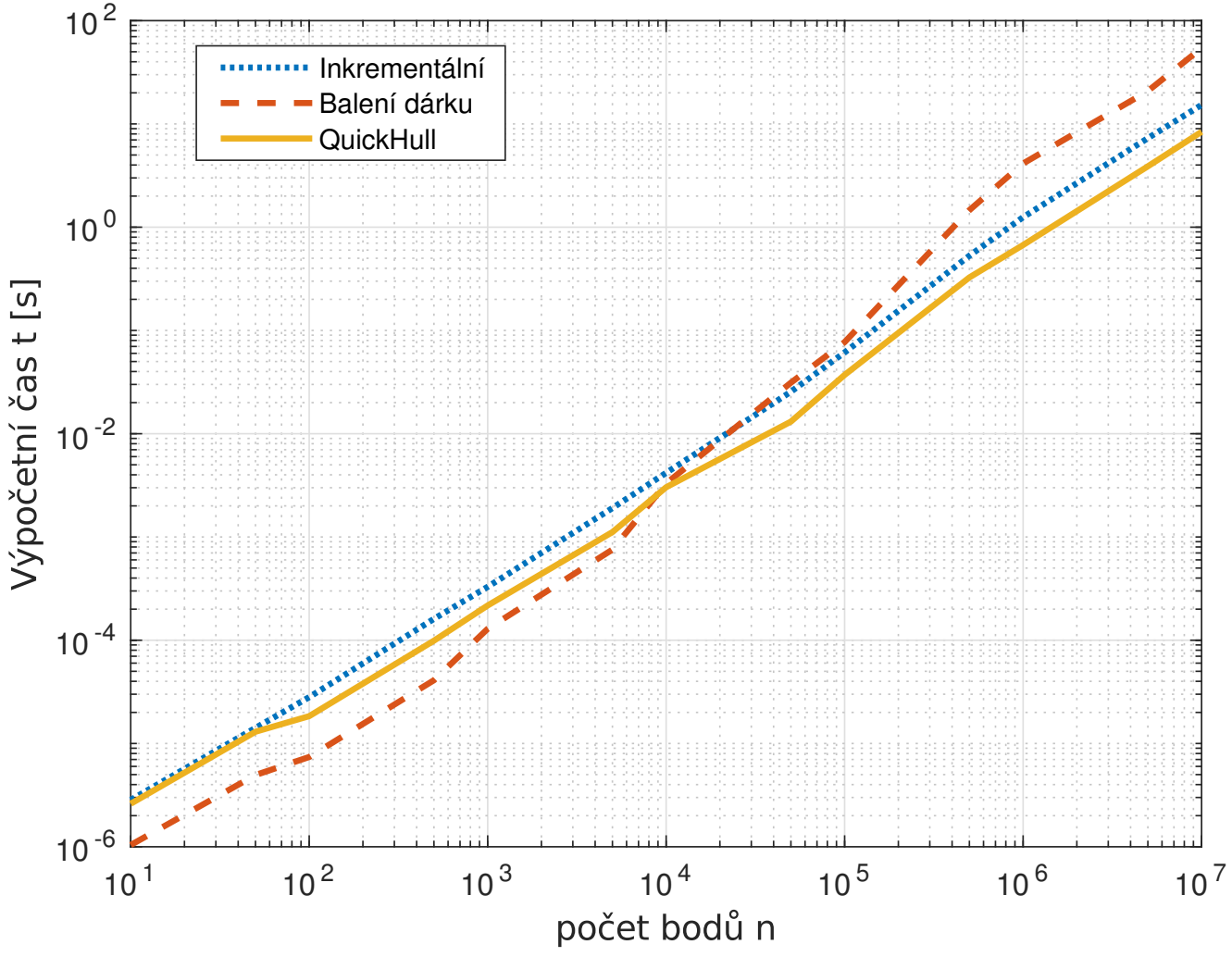
Srovnání reálné složitosti algoritmů

Zápočtová práce studenta POGR1 Davida Fridricha v AR 2012/13 (algoritmy v C++)



Srovnání reálné složitosti algoritmů

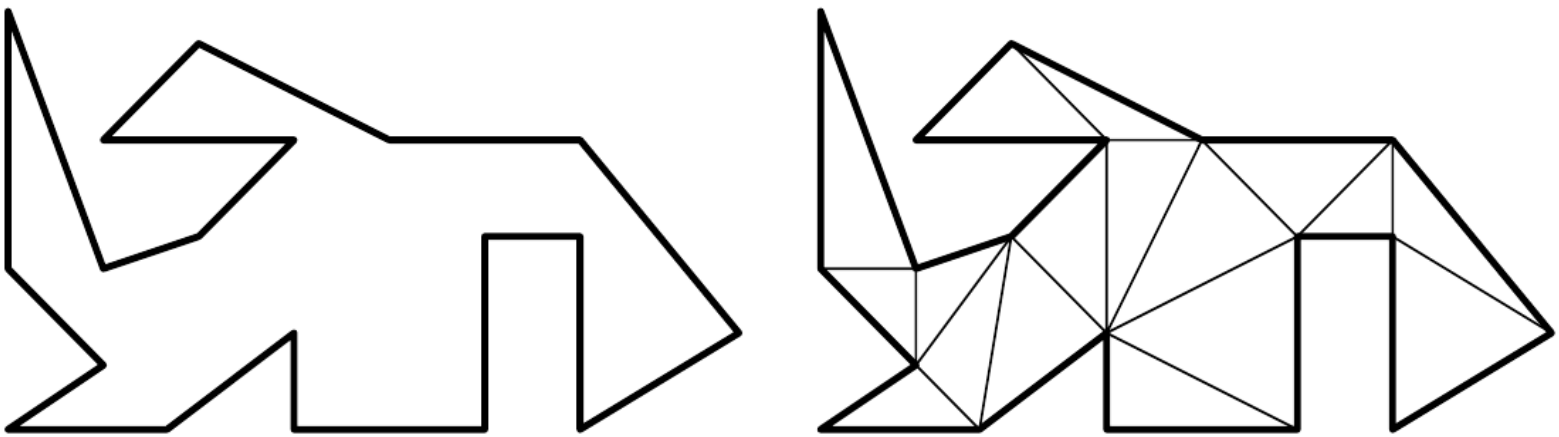
Zápočtová práce studenta POGR1 Davida Fridricha v AR 2012/13 (algoritmy v C++)



Triangulace polygonu

Pozorování

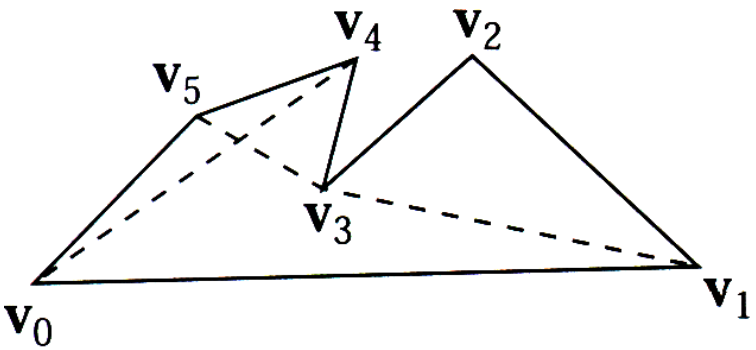
Každý jednoduchý polygon o n vrcholech má triangulaci. Každá jeho triangulace se skládá z $n-2$ trojúhelníků.

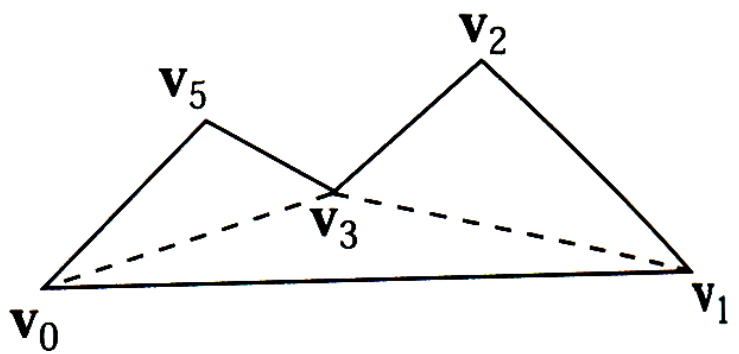


Triangulace (jednoduchého) polygonu

Ořezávání rohů

- *ear clipping* - snadný na implementaci, složitost $O\left(n^2\right)$
- polygon $P = (\boldsymbol{v}_0, \boldsymbol{v}_1, , \ldots, \boldsymbol{v}_{n-1})$; $\boldsymbol{v}_n = \boldsymbol{v}_0$
- procházíme vrcholy $\boldsymbol{v}_i, \boldsymbol{v}_{i+1}, \boldsymbol{v}_{i+2}$ (modulo n)
- jestliže úsečka $\boldsymbol{v}_i\boldsymbol{v}_{i+2}$ leží celá uvnitř polygonu, tak vrchol $\boldsymbol{v}_{i+1}$ tvoří roh
  - to nastane právě tehdy, když v $\triangle \boldsymbol{v}_i\boldsymbol{v}_{i+1}\boldsymbol{v}_{i+2}$ neleží žádný další vrchol polygonu
- vrchol $\boldsymbol{v}_{i+1}$ odstraníme z polygonu, vytvoříme $\triangle \boldsymbol{v}_i\boldsymbol{v}_{i+1}\boldsymbol{v}_{i+2}$





Obsah

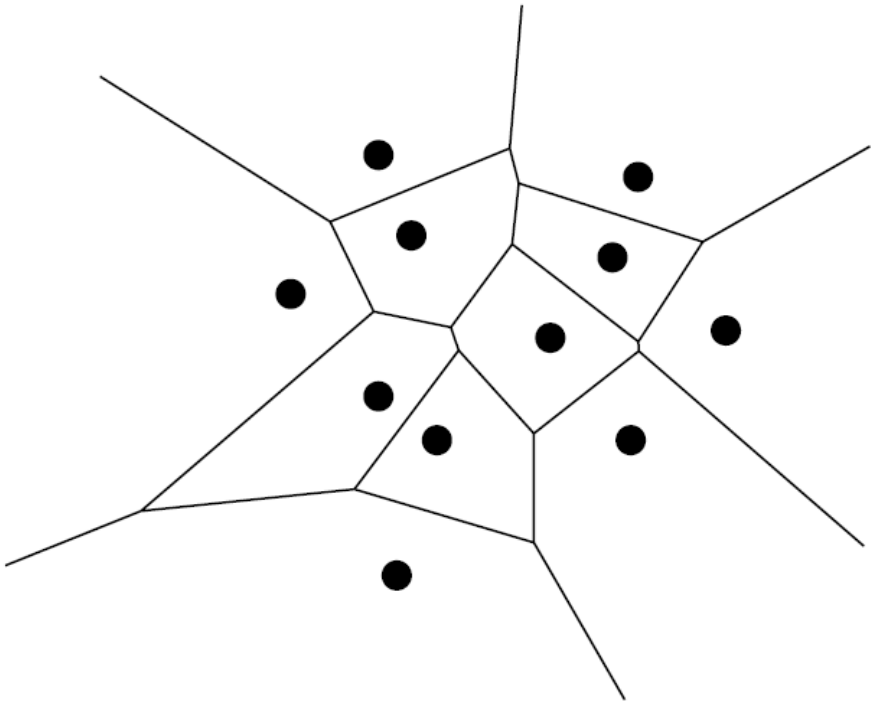
Voroného diagramy

- *Voronoi diagrams* (Gregorij Voronoj)
- množina n bodů $p_1, \ldots p_n$ v rovině
- rozdělení roviny do n oblastí $\Omega_1, \ldots, \Omega_n$
- $\forall x \in \Omega_i$ platí

$$|x - p_i| = \min_{j=1,\ldots,n} |x - p_j|$$

tj. Ω_i je množina bodů, které mají nejblíže k p_i
(v euklidovské normě)

- např. rozdělení oblastí pro zásobování z různých skladů apod.



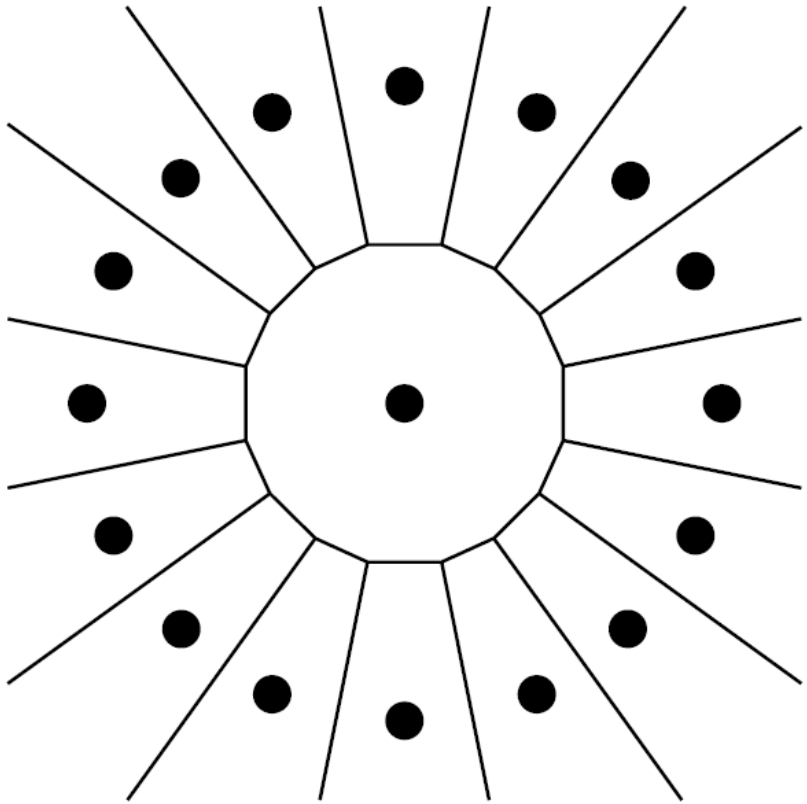
Voroného diagramy

- *Voronoi diagrams* (Gregorij Voronoj)
- množina n bodů $p_1, \ldots p_n$ v rovině
- rozdělení roviny do n oblastí $\Omega_1, \ldots, \Omega_n$
- $\forall x \in \Omega_i$ platí

$$|x - p_i| = \min_{j=1,\ldots,n} |x - p_j|$$

tj. Ω_i je množina bodů, které mají nejblíže k p_i
(v euklidovské normě)

- např. rozdělení oblastí pro zásobování z různých skladů apod.



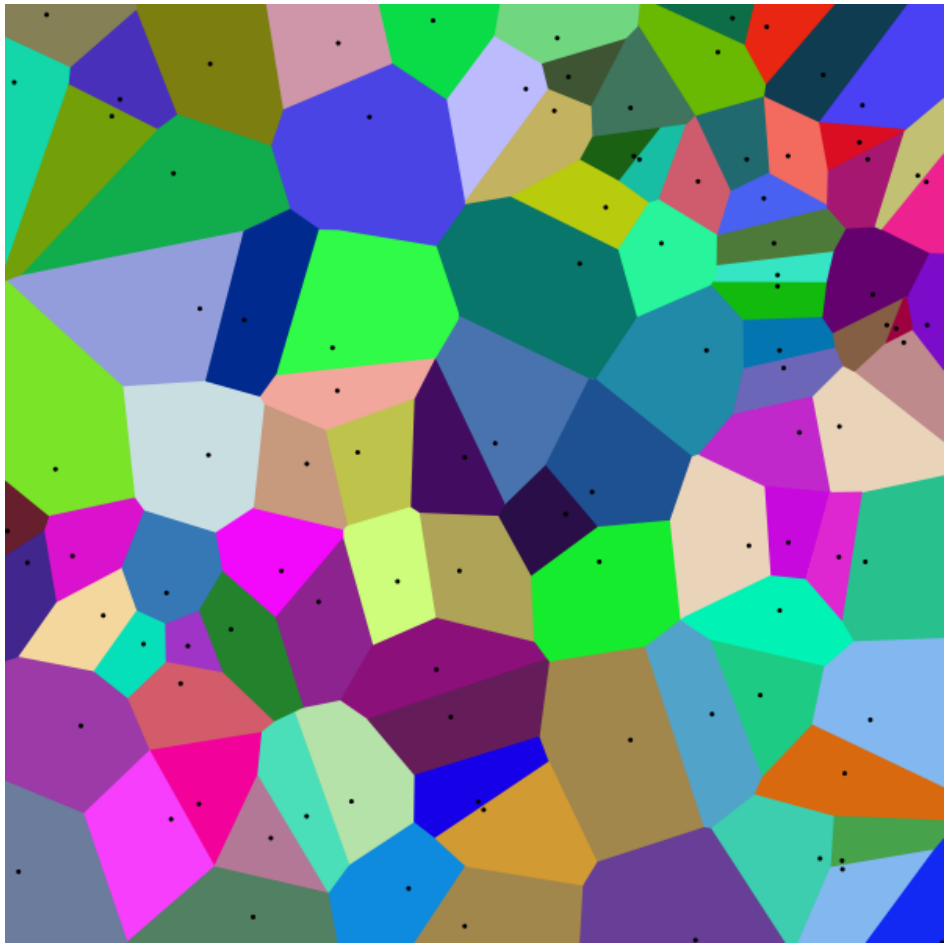
Voroného diagramy

- *Voronoi diagrams* (Gregorij Voronoj)
- množina n bodů $p_1, \ldots p_n$ v rovině
- rozdělení roviny do n oblastí $\Omega_1, \ldots, \Omega_n$
- $\forall x \in \Omega_i$ platí

$$|x - p_i| = \min_{j=1,\ldots,n} |x - p_j|$$

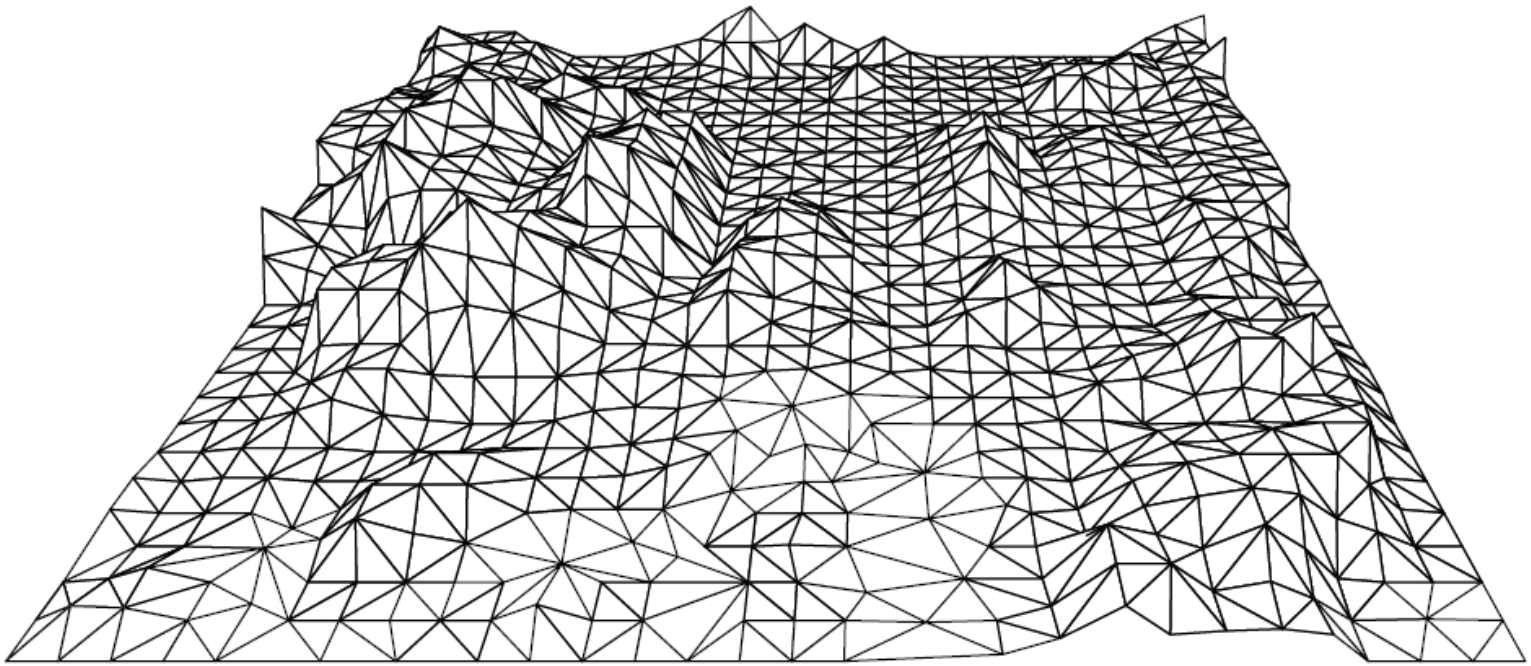
tj. Ω_i je množina bodů, které mají nejblíže k p_i
(v euklidovské normě)

- např. rozdělení oblastí pro zásobování z různých skladů apod.



Deloneho triangulace

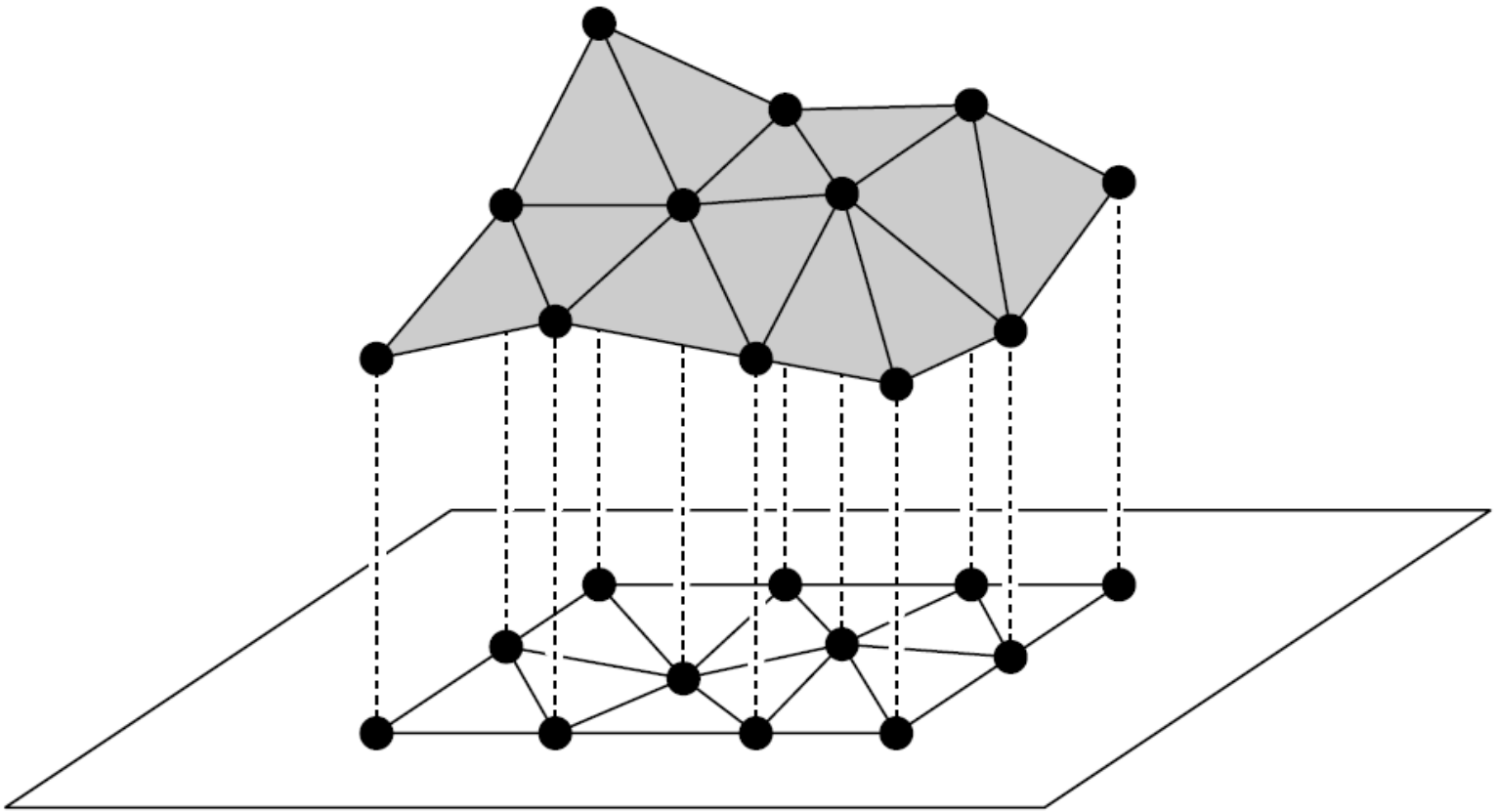
Motivace



Terén

Deloneho triangulace

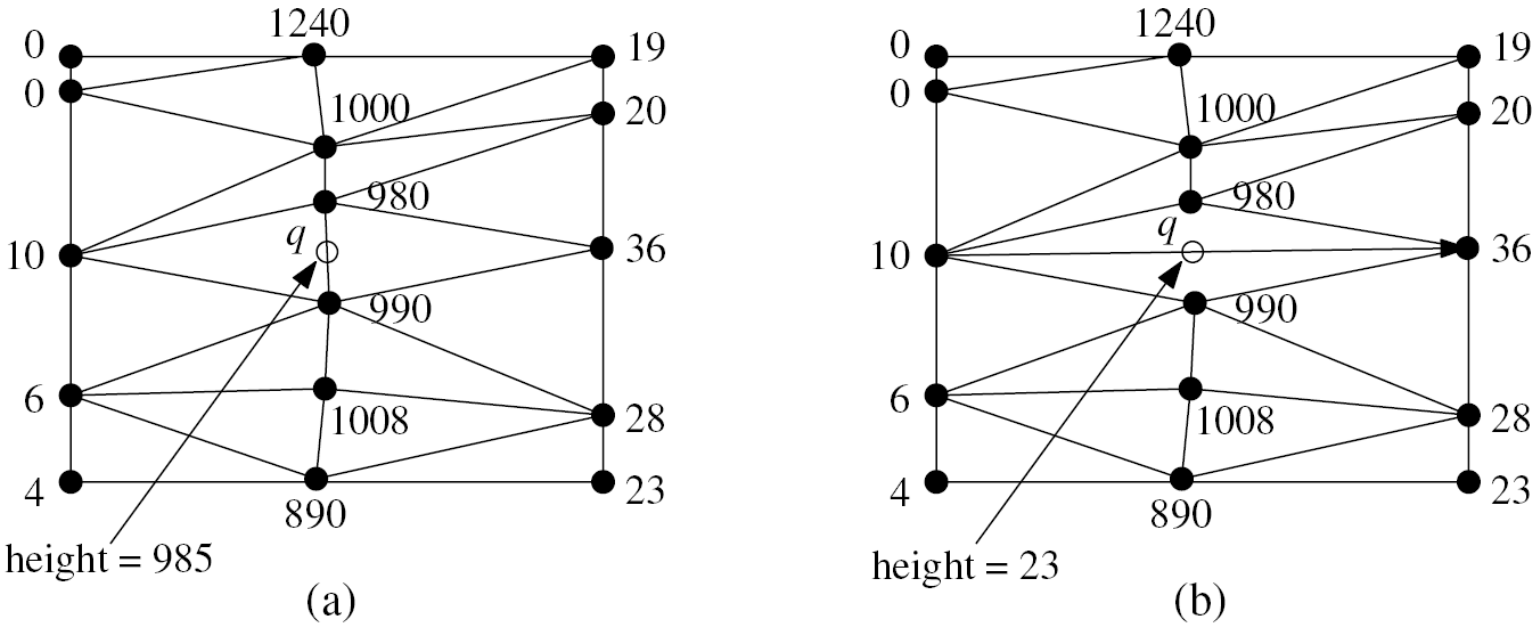
Motivace



Výšková mapa na trojúhelníkové síti

Deloneho triangulace

Motivace

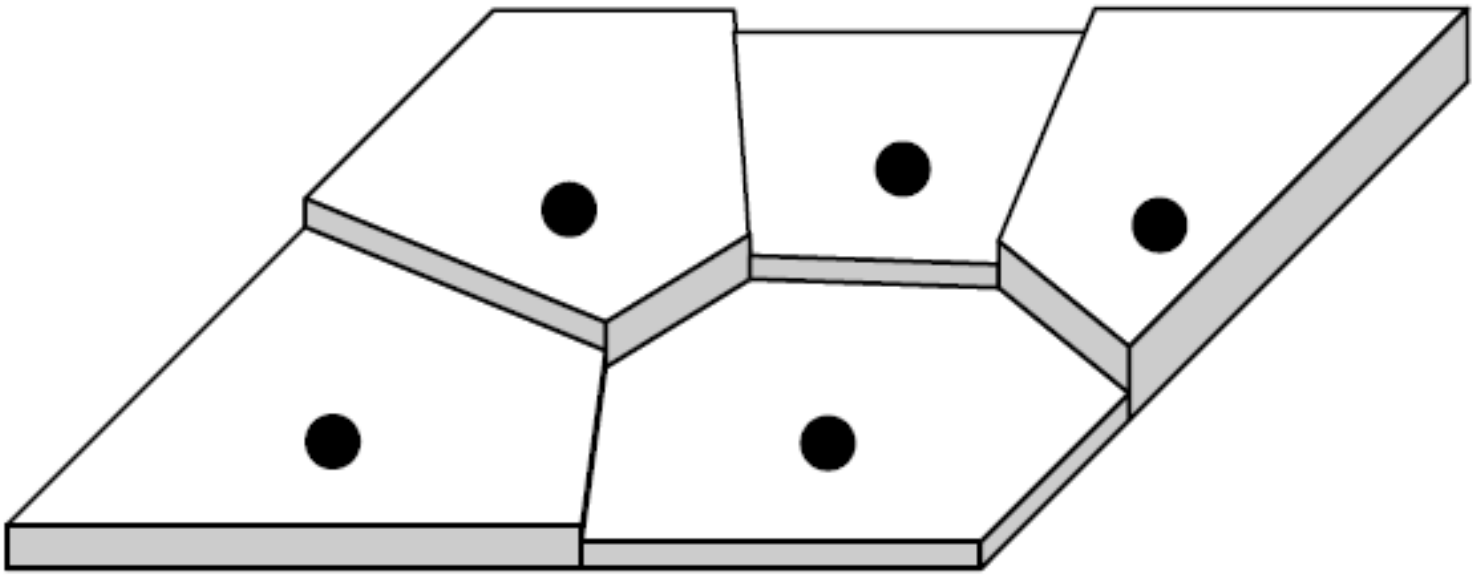


Interpolace výšky ze vzorků na síti v závislosti na triangulaci

- uprostřed je pohoří - proč je případ *a*) mnohem realističtější?

Deloneho triangulace

Motivace

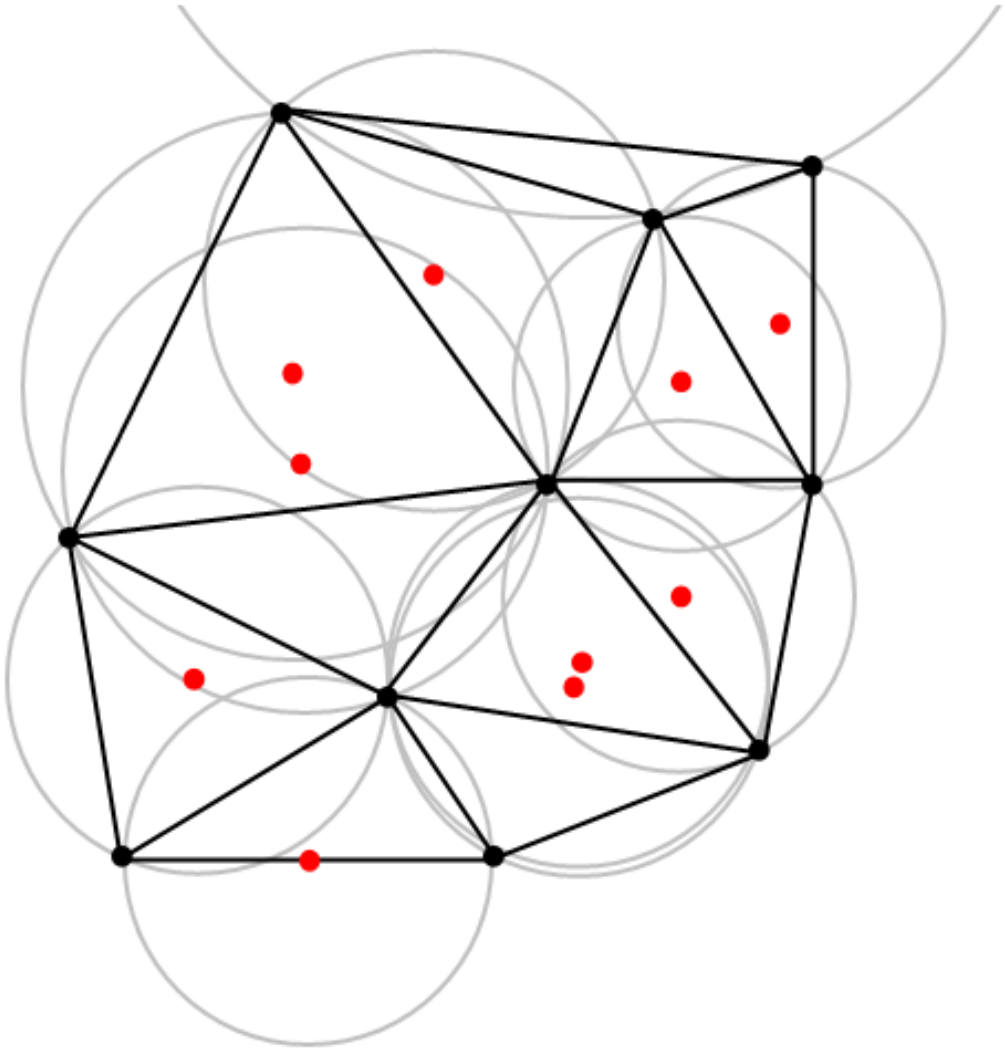


Výšková mapa - Voroného diagram

- interpolace výšky obecného bodu bude nejpřesnější, pokud použijeme co nejbližší vzorky

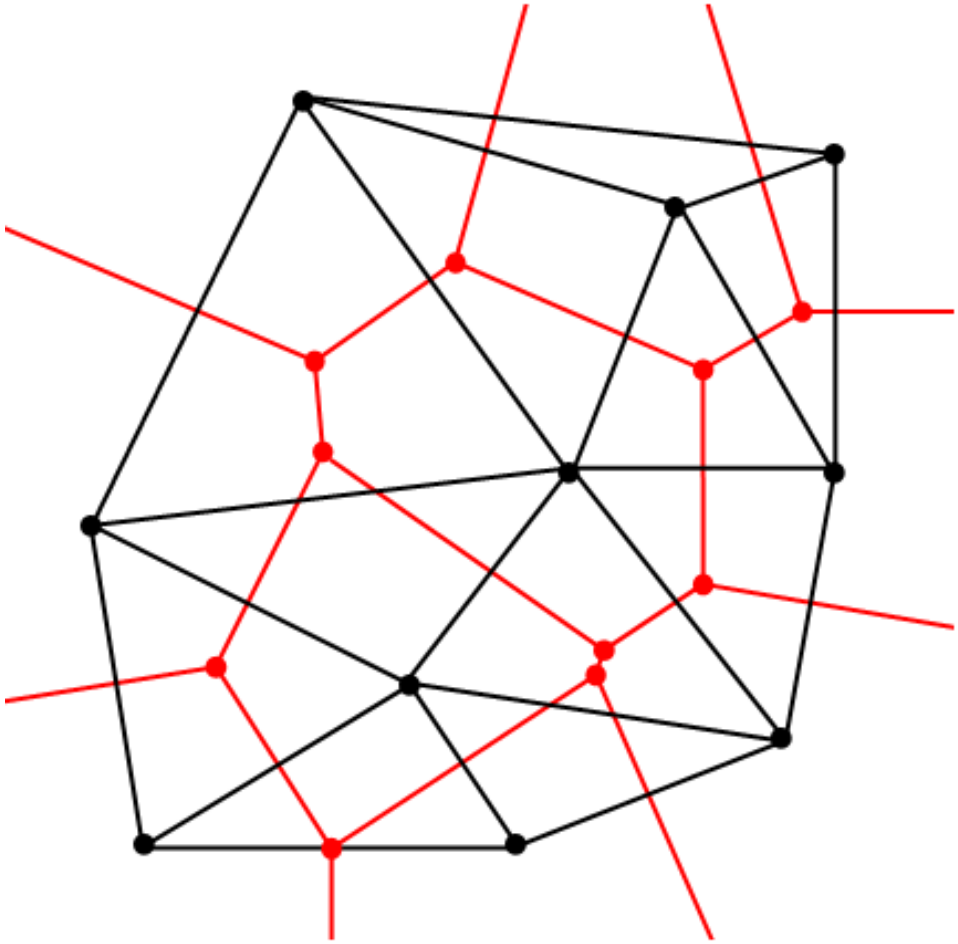
Deloneho triangulace

- *Delaunay triangulation* (Boris Delone, původně po otci de Launay)
- **triangulace konvexního obalu bodů v rovině**, kde v **kruhu opsaném** libovolnému trojúhelníku neleží další bod
- **maximalizuje minimální úhel** v celé síti
- spojnice středů kruhů vymezují hranice ve Voroného diagramu



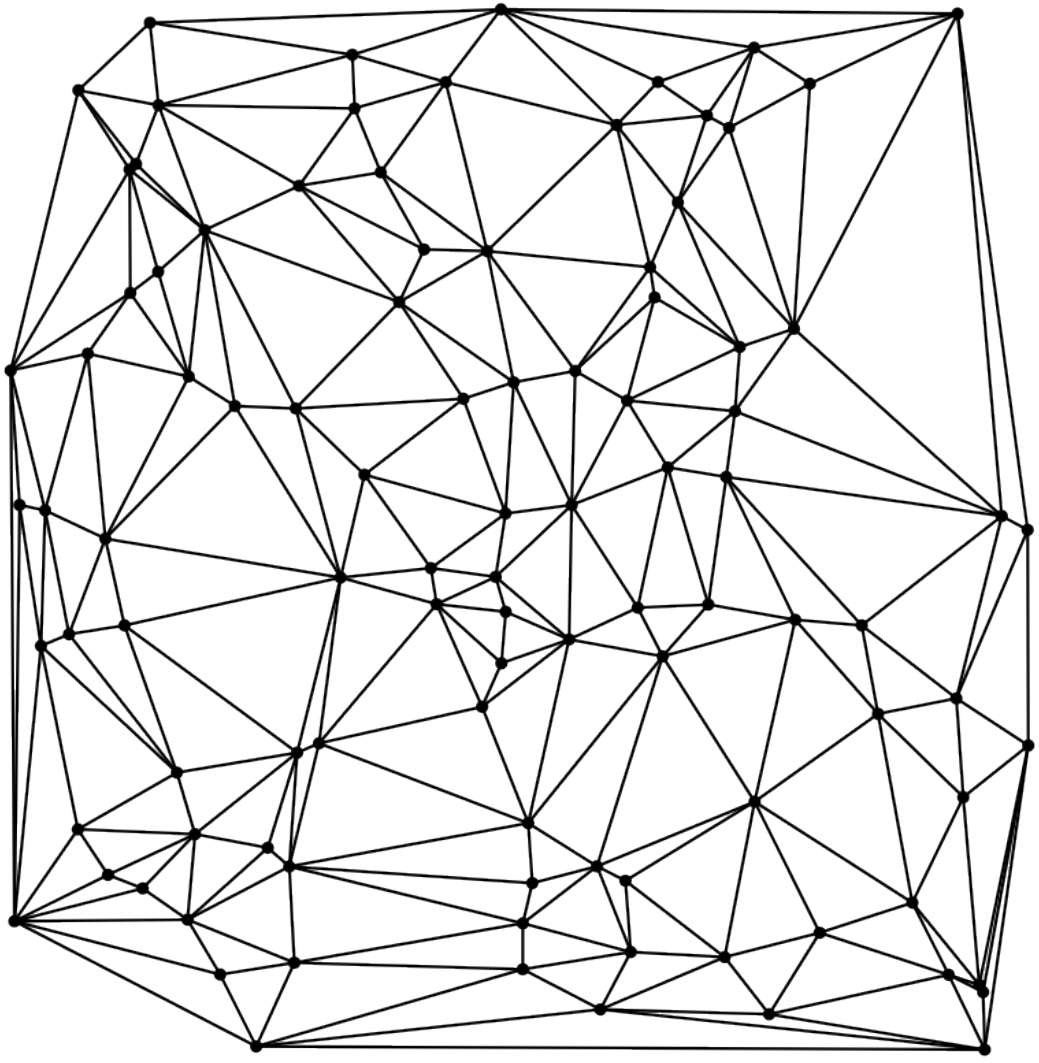
Deloneho triangulace

- *Delaunay triangulation* (Boris Delone, původně po otci de Launay)
- **triangulace konvexního obalu bodů v rovině**, kde v **kruhu opsaném** libovolnému trojúhelníku neleží další bod
- **maximalizuje minimální úhel** v celé síti
- spojnice středů kruhů vymezují hranice ve Voroného diagramu



Deloneho triangulace

- *Delaunay triangulation* (Boris Delone, původně po otci de Launay)
- **triangulace konvexního obalu bodů v rovině**, kde v **kruhu opsaném** libovolnému trojúhelníku neleží další bod
- **maximalizuje minimální úhel** v celé síti
- spojnice středů kruhů vymezují hranice ve Voroného diagramu



Deloneho triangulace 100 bodů v rovině

Generování sítě pro numerické řešení PDE

- numerické řešení úloh pro **parciální diferenciální rovnice** (PDE, *Partial Differential Equations*)
  - mechanika a dynamika kontinua (simulace deformací a namáhání pevných těles, proudění tekutin,...)
- **metoda konečných prvků** (FEM, *Finite Element Method*)
- **metoda konečných objemů** (FVM, *Finite Volume Method*)
- potřeba pokrýt výpočetní oblast Ω sítí polygonů (*computational mesh*, *grid*)
- oblast Ω
  - podmnožina roviny, resp. povrch objektu ve 3D $\Longrightarrow$ zpravidla trojúhelníky, čtyřúhelníky
  - podmnožina prostoru $\Longrightarrow$ kvádry, čtyřstěny atd.

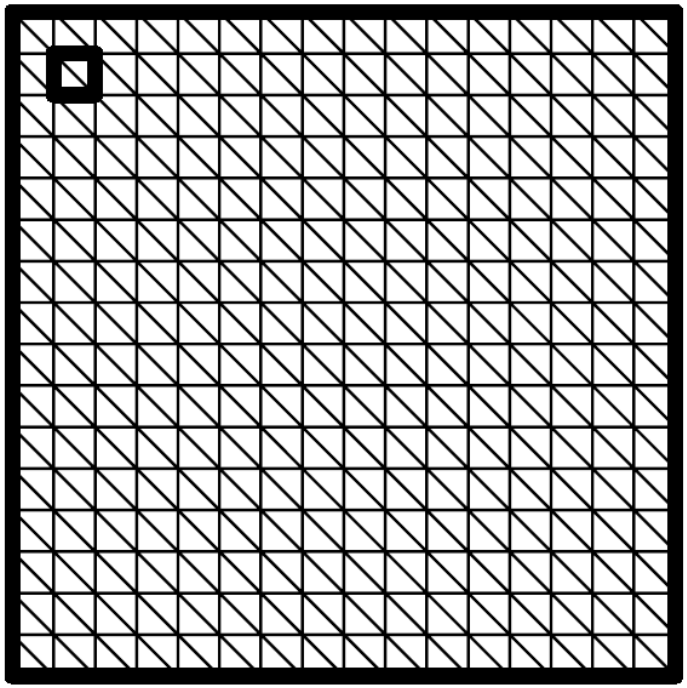
Generování sítě pro numerické řešení PDE

Typy sítě

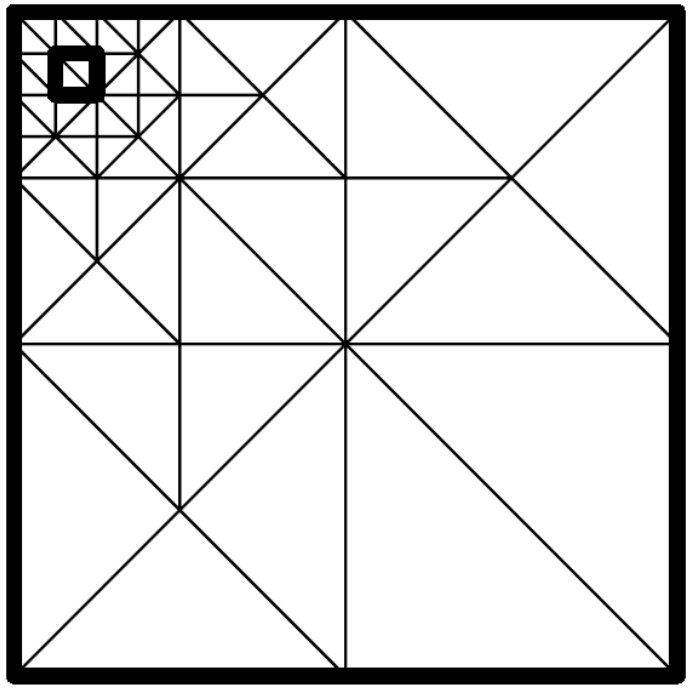
- **strukturovaná** (*structured*, *uniform*)
  - jednoduchý tvar oblasti
  - pravidelná mřížka vrcholů
  - souřadnice vrcholů dány jednoduchým vztahem
- **nestrukturovaná** (*unstructured*, *non-uniform*)
  - obecná síť vrcholů
  - souřadnice vrcholů uloženy v paměti nebo stromové struktuře (viz dále)

Generování sítě pro numerické řešení PDE

Typy sítě



strukturovaná síť



nestrukturovaná síť

- stejné rozlišení („jemnost sítě“) nemusí být potřeba všude

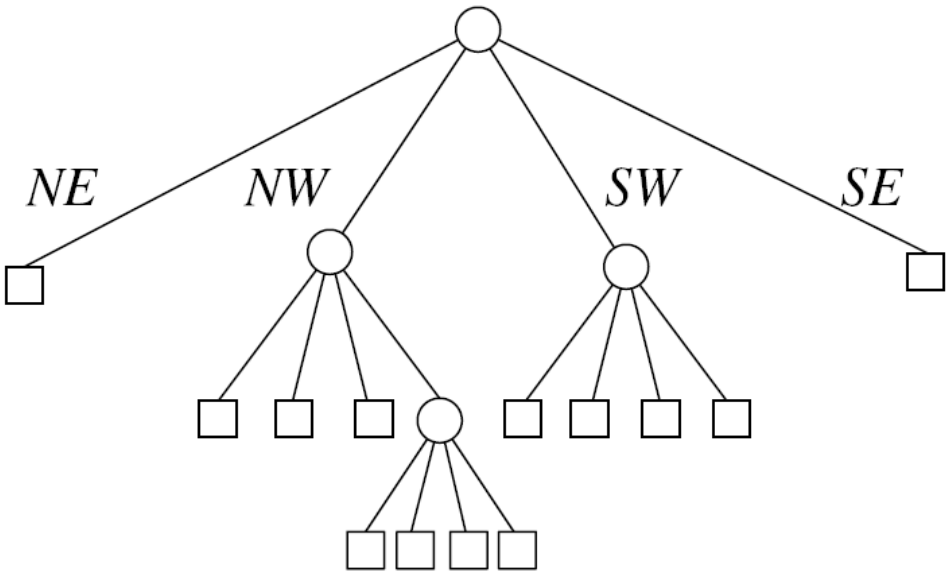
Generování sítě pro numerické řešení PDE

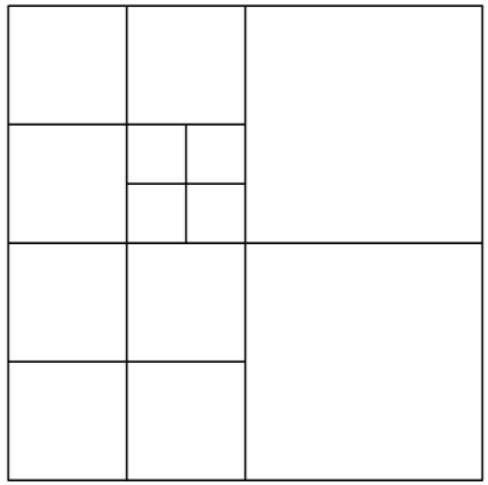
Nestrukturované sítě

- metody založené na **Deloneho triangulaci** nebo na datových strukturách **quadtree** ve 2D, resp. octree ve 3D
- adaptivní **zjemňování** (*refinement*), resp. zhrubování (*coarsening*)
- síť je jemnější
  - v geometricky složitých partiích
  - v místech, kde má **řešení** složitý průběh $\Longrightarrow$ změna sítě za běhu výpočtu
- umožňuje ušetřit paměť a čas tam, kde není potřeba velká přesnost (viz předchozí obrázek)

Generování sítě pro numerické řešení PDE

Nestrukturované sítě založené na quadtree

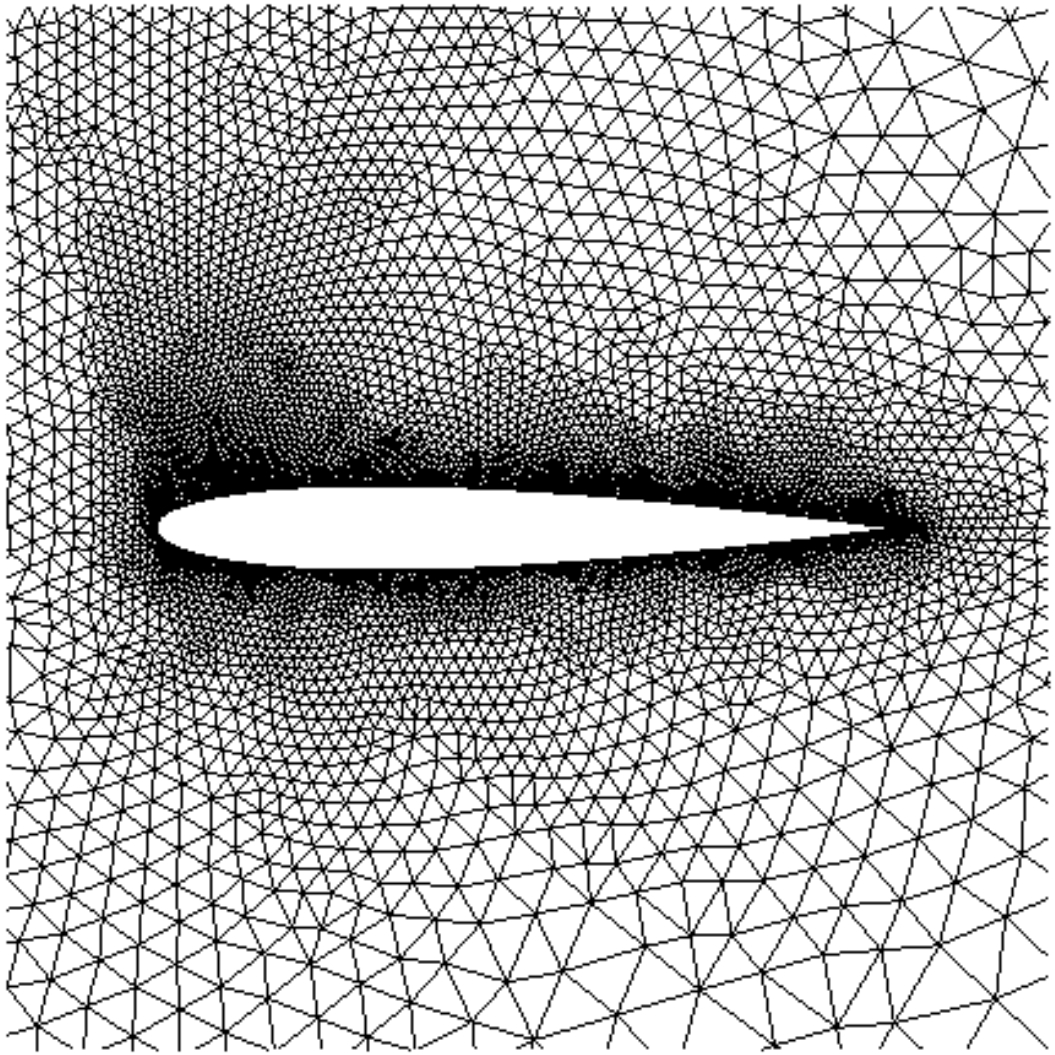




- 4 potomci uzlu určují rozdělení 4 kvadrantů oblasti
- v listech stromu, tj. „□“, jsou hodnoty veličin (proměnných)

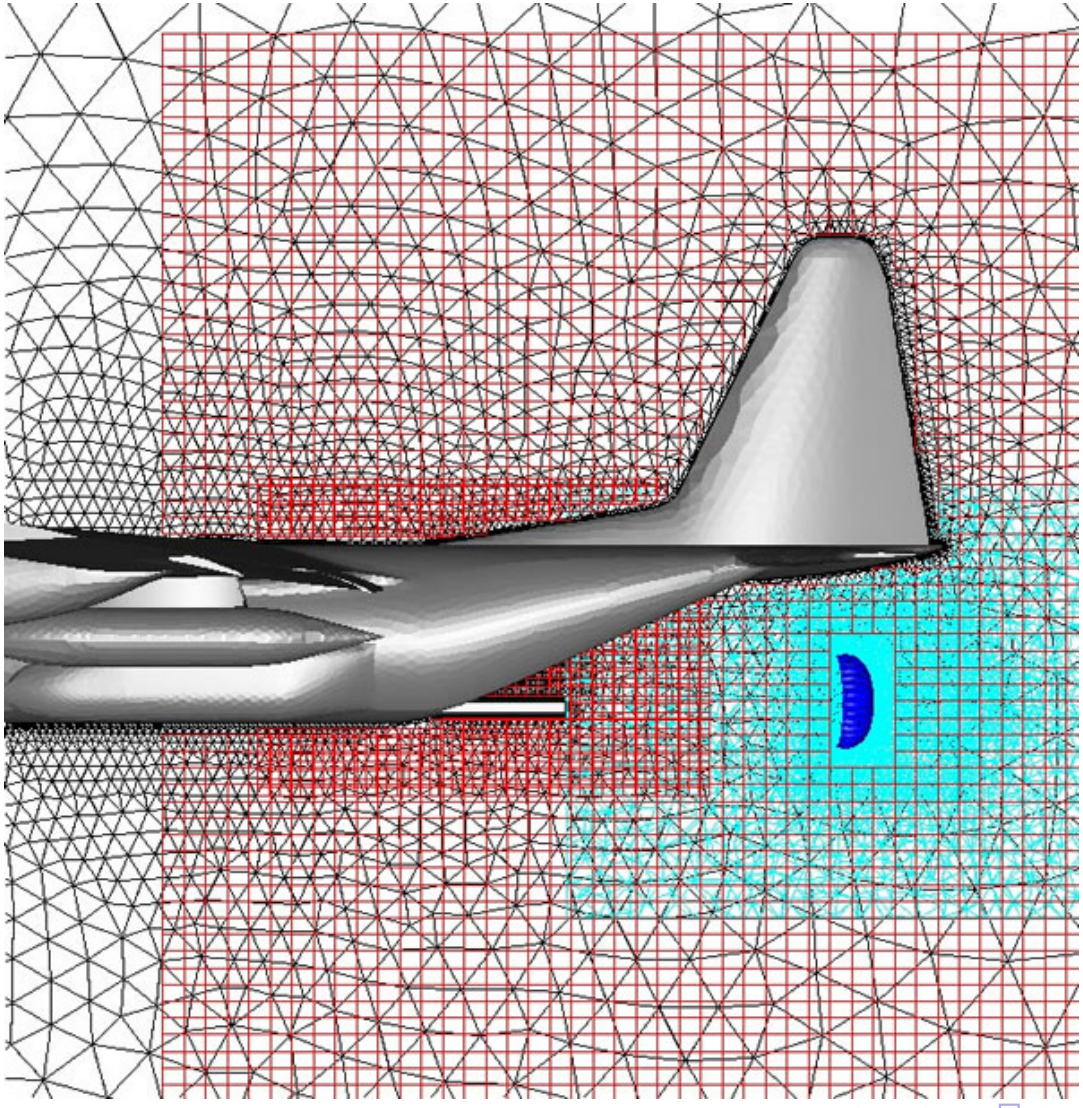
Generování sítě pro numerické řešení PDE

Příklady sítí



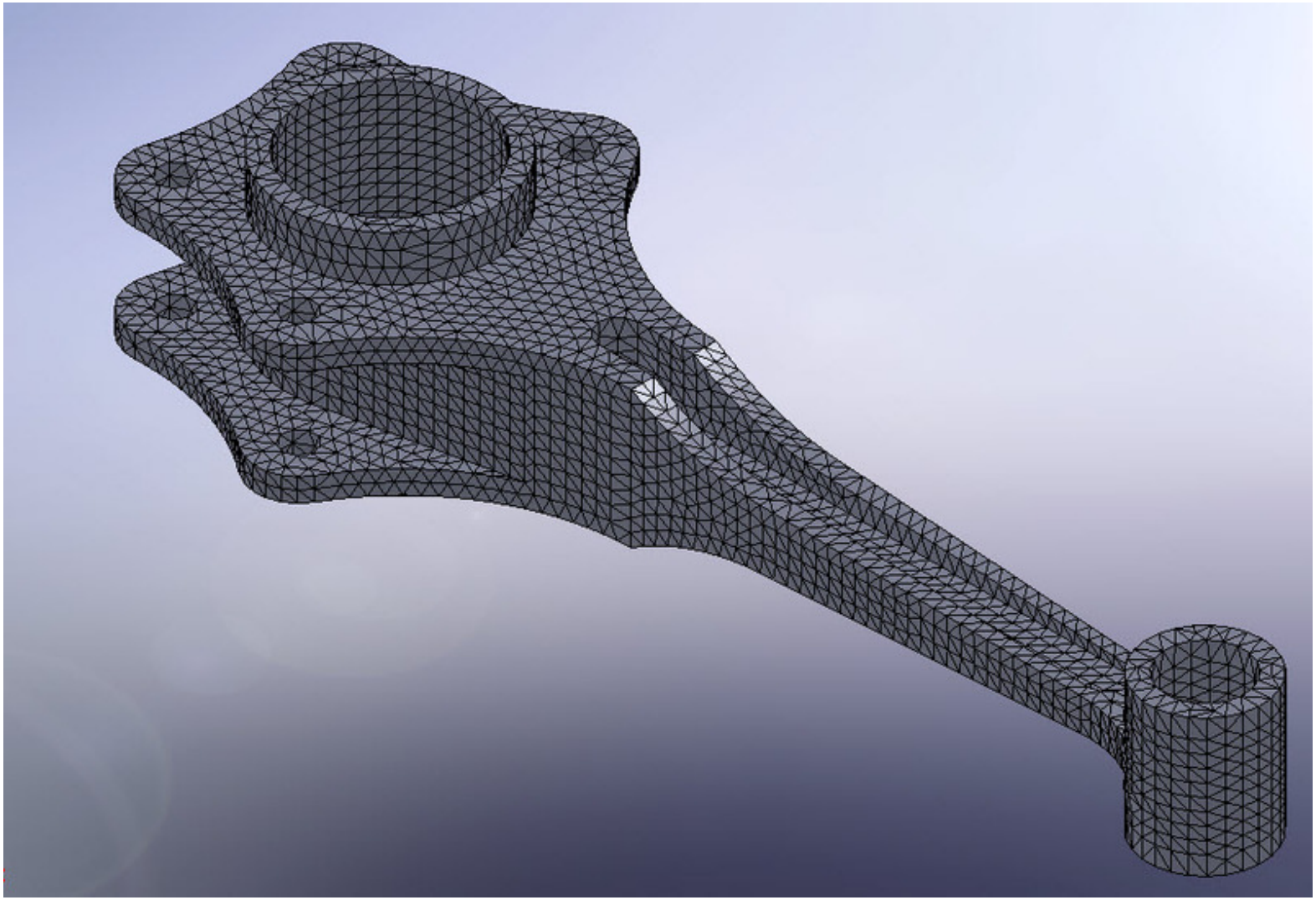
Generování sítě pro numerické řešení PDE

Příklady sítí



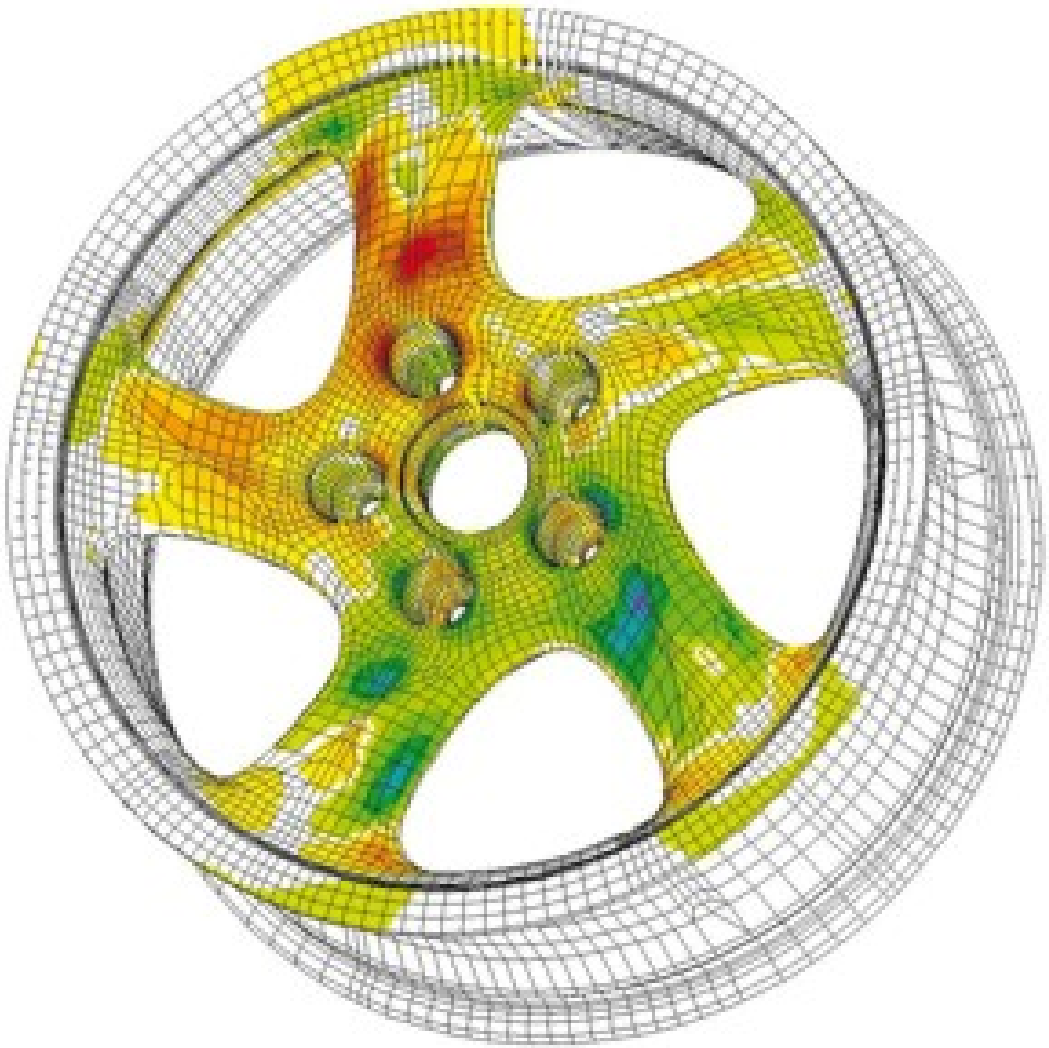
Generování sítě pro numerické řešení PDE

Příklady sítí



Generování sítě pro numerické řešení PDE

Příklady sítí



Literatura

M. de Berg, O. Cheong, M. van Kreveld, M. Overmars: *Computational Geometry - Algorithms and Applications*, 3rd. ed, Springer-Verlag, Berlin Heidelberg 2008.

P. J. Schneider, D. H. Eberly: *Geometric Tools for Computer Graphics*, Elsevier Science, 2003.