3D számítógépes geometria 2

Diszkrét geometriai algoritmusok

Várady Tamás, Salvi Péter / BME

November 8, 2018

Naív megoldás

- Adott két szakasz: (a, b) és (c, d) az L_{ab} ill. L_{cd} egyeneseken
- Paraméteres egyenletek:

$$p(s) = a + s(b - a) = a + sA$$
$$q(t) = c + t(d - c) = c + tC$$

- Ebből

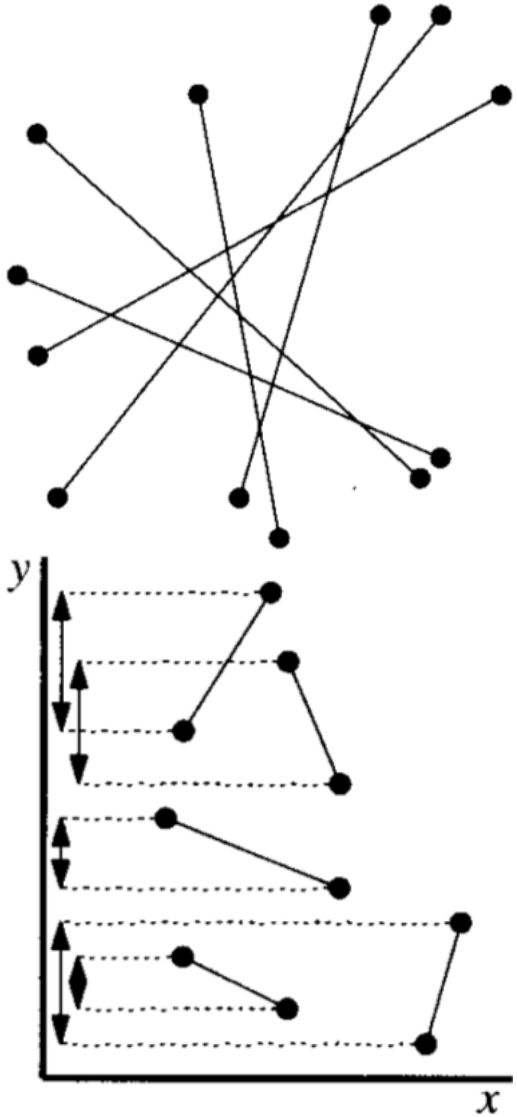
$$\begin{aligned} s &= [a_x(d_y - c_y) + c_x(a_y - d_y) + d_x(c_y - a_y)]/D \\ t &= [a_x(c_y - b_y) + b_x(a_y - c_y) + c_x(b_y - a_y)]/D \\ D &= C_x A_y - A_x C_y \end{aligned}$$

Pontosítások

- Ha s vagy t nem $[0,1]$-ben van, nincs metszés
- Párhuzamos szakaszok esetén 0-val osztanánk
  - Teszt: $\left|\frac{\langle A, C^{\perp}\rangle}{\|A\|\cdot\|C\|}\right| < \varepsilon \Rightarrow D^2 = \|A_y C_x - A_x C_y\|^2 < \varepsilon^2 \|A\|^2 \|C\|^2$
- Ilyenkor meg kell nézni, hogy lehet-e átfedés
  - Legyen $E = c - a$, kérdés: E párhuzamos-e A-val?
  - Teszt: $\left|\frac{\langle E, A^{\perp}\rangle}{\|E\|\cdot\|A\|}\right| < \varepsilon \Rightarrow \|E_x A_y - E_y A_x\|^2 < \varepsilon^2 \|E\|^2 \|A\|^2$
- Azonos egyenesek esetén kiszámoljuk az átfedést
  - Legyen $c = p(s_0)$ és $d = p(s_1)$
  - Ebből $s_0 = A \cdot E / \|A\|^2$, $s_1 = s_0 + A \cdot C / \|A\|^2$
  - Az átfedés képlete: $[0,1] \cap [\min(s_0, s_1), \max(s_0, s_1)]$

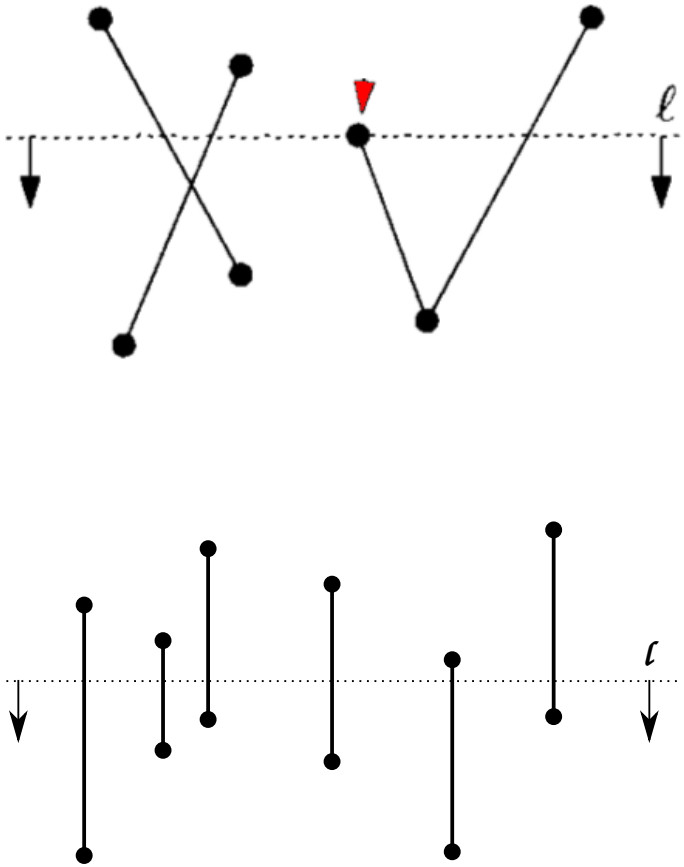
Feladat

- n szakasz összes metszéspontja?
- Triviális megoldás:
  - Ellenőrizzünk minden párt – $O(n^2)$
  - „Optimális” – legrosszabb esetben (n^2 metszés) minden algoritmus $\Omega(n^2)$
- Jobb megoldás:
  - Kimenet-érzékeny algoritmus
  - Kevés metszéspont $\rightarrow$ kevés számolás
- Ötlet:
  - Csak azokat ellenőrizzük, ahol átfedés van az y-tartományban
  - Vízszintes söprőegyenes



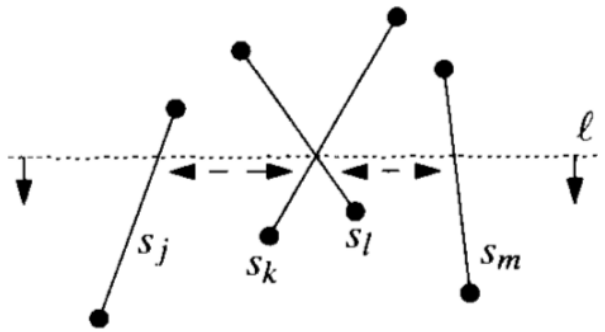
Síksöprés

- Állapot:
  - l-t metsző szakaszok halmaza
  - Szakaszok végpontjainál változik
  - $\Rightarrow$ „eseménypontok”
- Események:
  - Kezdőpont:
    $\Rightarrow$ Metszés-keresés
    a halmazban levő szakaszokkal
  - Végpont:
    $\Rightarrow$ Szakasz kivétele a halmazból
- Probléma:
  - Ez még nem kimenet-érzékeny
  - Pl. függőleges szakaszok amelyek metszik az x tengelyt



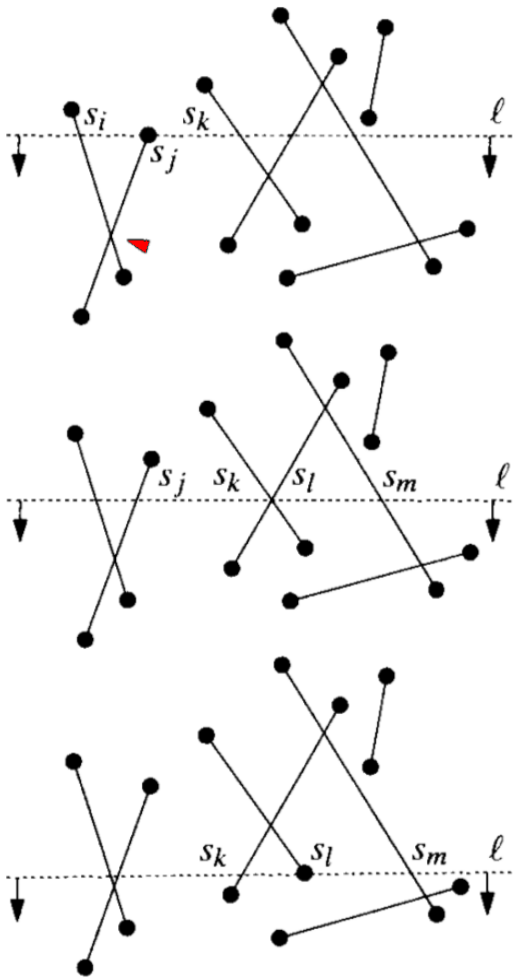
Síksöprés rendezéssel

- Ötlet:
  - Rendezzünk balról jobbra
  - Csak a szomszédos szakaszokat kell ellenőriznünk
- Állapot:
  - l-t metsző szakaszok, rendezve
  - Változik metszéspontoknál is
- Események:
  - Kezdőpont: metszés a szomszédokkal?
  - Metszéspont: helycsere; metszés az új szomszédokkal?
  - Végpont: kivétel a listából; metszés az új szomszédok közt?



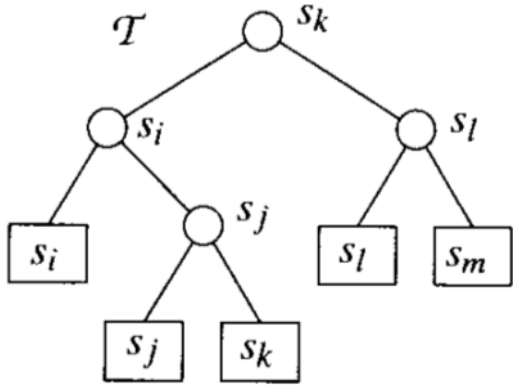
Részletek

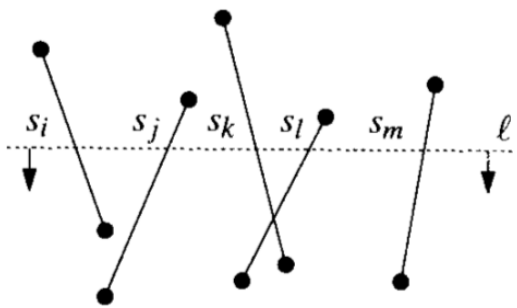
- Általános:
  - Mindig csak alsó metszés érdekes
  - Metszés = esemény, elrakjuk későbbre
  - Egy metszés többször megtalálható (!)
- Kezdőpont:
  - Beszúrjuk a listába
  - Szomszédokkal keresünk metszést
- Metszéspont:
  - A két szakasz helyet cserél a listában
  - Új szomszédokkal metszéskeresés
- Végpont:
  - Kivesszük a listából
  - A volt szomszédai közt metszéskeresés



Adatszerkezetek

- Eseménysor Q [bináris keresőfa]:
  - Következő esemény (≈pont)
    - Söprővonal alatti legfelső
    - Azonosak közt balról jobbra
    - Mintha ε dőlésszögű lenne a söprés
- Állapot T [bináris keresőfa]:
  - Szakasz beszúrása / törlése
  - Belső csomópontokban döntés: a szakasz melyik oldalon vagyunk?
- Speciális esetek (ezekre kell figyelni):
  - Vízszintes szakaszok
  - Egy pontban ≥ 3 szakasz
  - Végpont–kezdőpont „metszés”
  - Részben átfedő szakaszok





Algoritmus (Bentley–Ottmann, 1979)

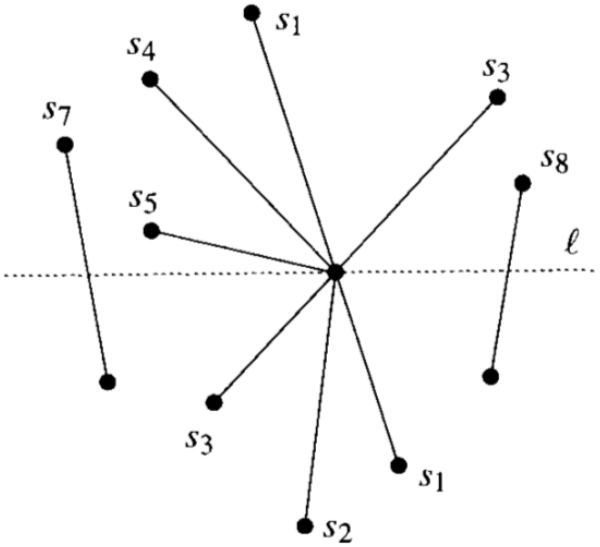
1. T üres; Q-ba betesszük a végpontokat
2. Amíg Q nem üres:
   - 2.1 Kivesszük Q-ból a következő eseményt (p)
   - 2.2 $U(p)$: amely szakaszok felső végpontja p
   - 2.3 $L(p)$ és $C(p)$: a szomszédos szakaszokból amelyek (rendre) alsó végpontként vagy belül tartalmazzák p-t
   - 2.4 Ha $L(p) \cup U(p) \cup C(p)$-ben > 1 szakasz van $\Rightarrow$ metszés kiírása
   - 2.5 Töröljük $L(p) \cup C(p)$-t T-ből
   - 2.6 Beszúrjuk $U(p) \cup C(p)$-t T-be (jó sorrend – vízszintes hátul!)
   - 2.7 Ha $U(p) \cup C(p) = \emptyset$

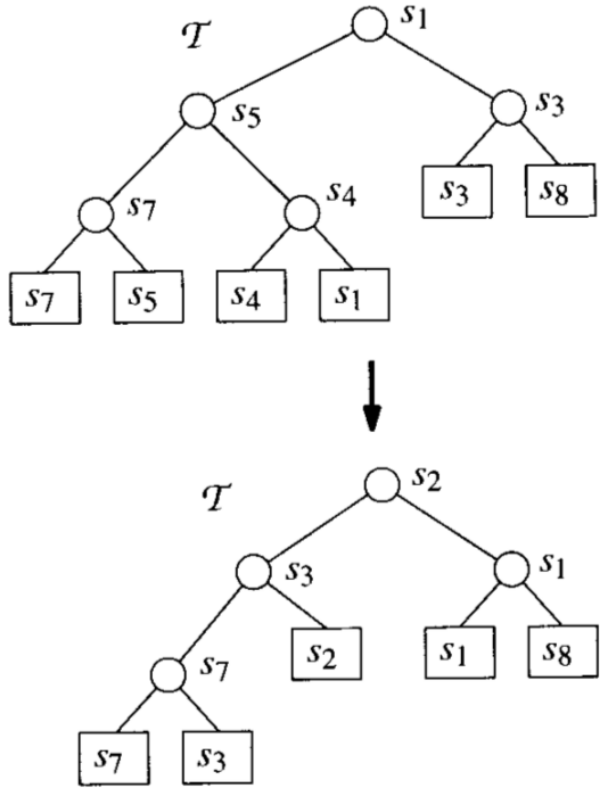
Akkor: s_l és s_r a p szomszédai $\Rightarrow$ `FindNewEvent`(s_l, s_r, p)

Különben: s', s'' a leg(bal/jobb)oldalibb szakasz $U(p) \cup C(p)$-ből T-ben
s_l, s_r ezek bal- ill. jobboldali szomszédai T-ben
$\Rightarrow$ `FindNewEvent`(s_l, s', p)
$\Rightarrow$ `FindNewEvent`(s'', s_r, p)

FindNewEvent(s_l, s_r, p)

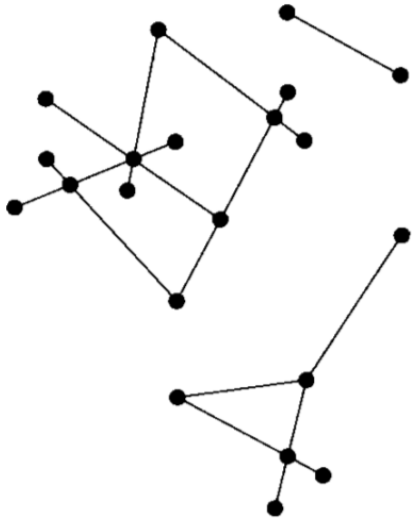
- Ha s_l és s_r metszi egymást a söprővonal alatt...
  - ... *vagy* rajta, de p-től jobbra
  - ... *és* a metszéspont még nincs Q-ban
  - ... *akkor* betesszük a metszéspont eseményt Q-ba

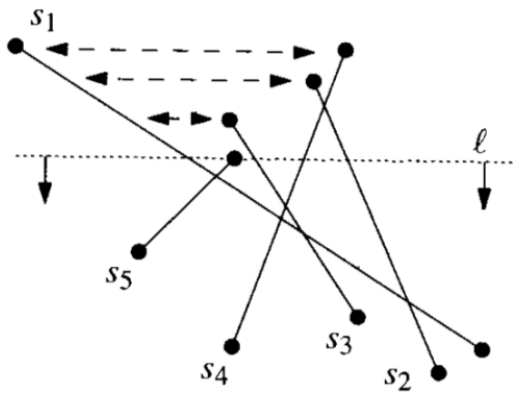




Komplexitás

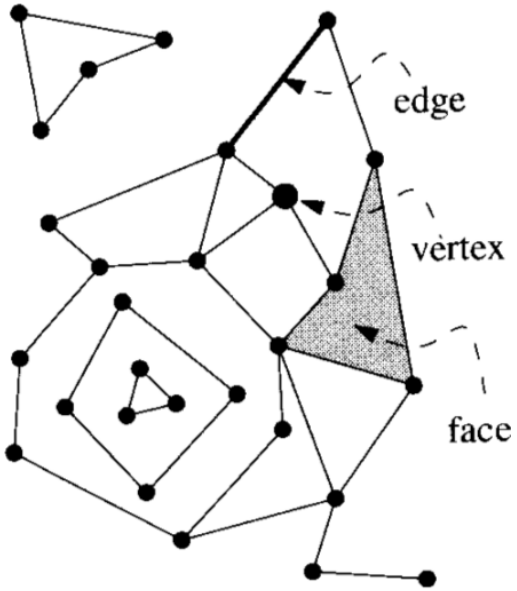
- Műveletigény: $O((n+k)\log n)$, ahol k a metszéspontok száma
- Memóriaigény:
  - T: $O(n)$, Q-ban lehet $O(n+k)$ pont
- Lineáris memóriaigényhez:
  - Töröljük a metszéspont-eseményeket, ha a szakaszok nem szomszédosak már
- Chazelle–Edelsbrunner (1988/1992):
  - $O(n\log n+k)$ műveletigény (optimális)
  - $O(n+k)$ memóriaigény
- Clarkson–Shor (1989): random optimális
- Balaban (1995): optimális algoritmus

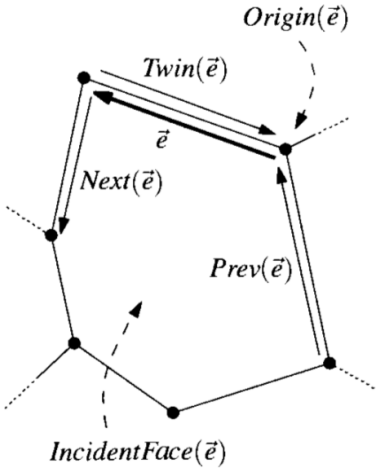


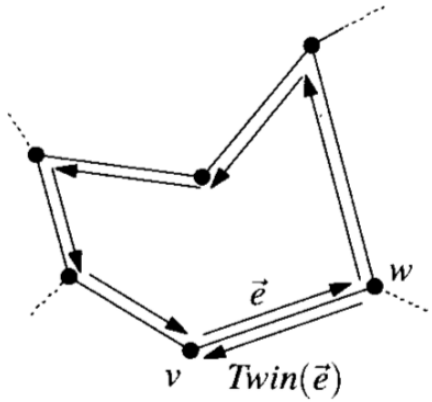


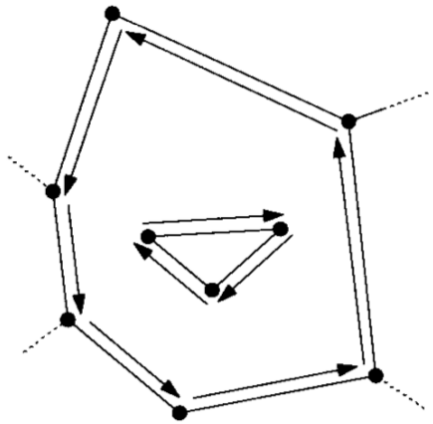
Fél-él adatstruktúra (ismétlés)

- Síkfelosztás:
  - Sokszögháló
  - Nem feltétlenül összefüggő
  - Lehetnek benne lyukak
- Topológiai lekérdezések
- Tárolt: lapok, fél-élek, csúcsok



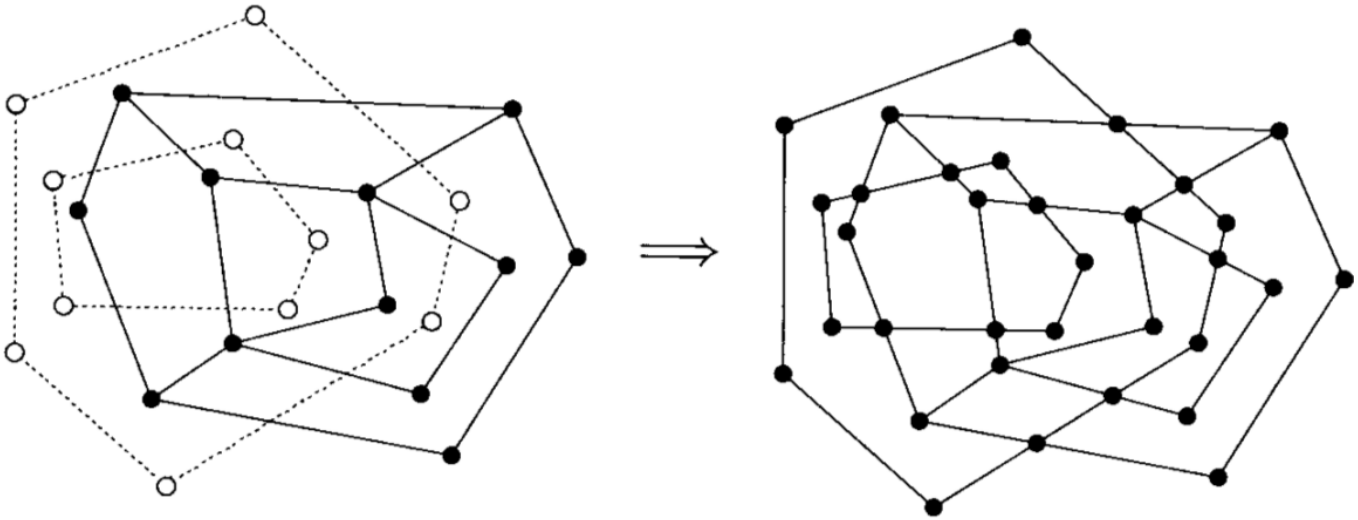






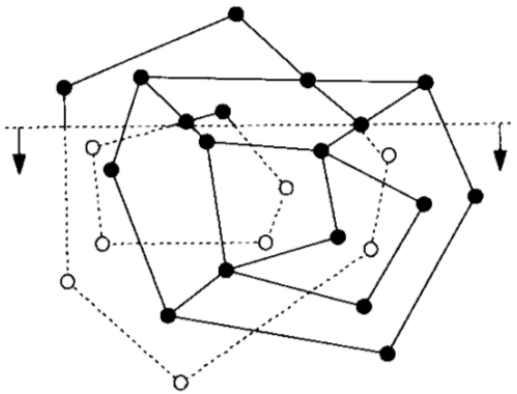
Feladat

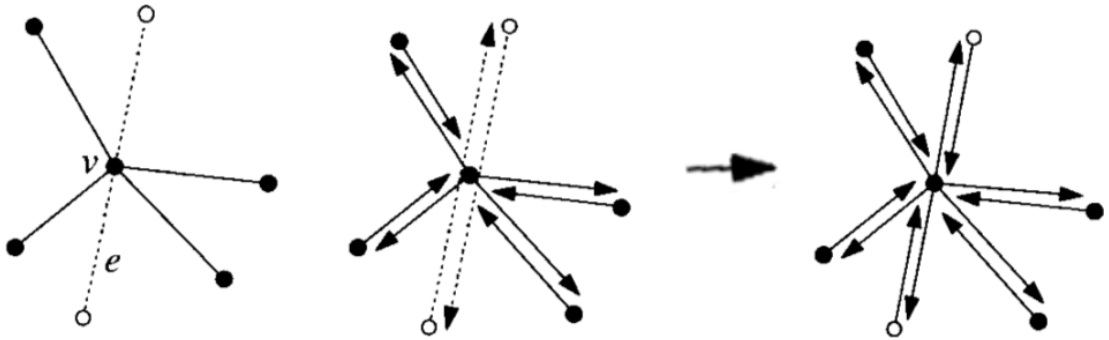
- Két fél-él struktúra S_1 és S_2 metszete
- Új csúcsok, (fél-)élek, lapok születnek
- Ha a lapokhoz volt valamilyen attribútum,
  az új lapokban mindkét ős attribútumának szerepelnie kell
- Pl. színezés stb.



Újrafelhasználás

- A fél-élek nagy része használható
- Csak az változik, amin új metszéspont van
- Ötlet:
  - „Összerakjuk” a két fél-él struktúrát
  - Kijavítjuk, hogy érvényes legyen
- Kiszámoljuk a metszéspontokat
  - Söprőegyeneses algoritmus
  - Q és T mellett tároljuk $D = S_1 + S_2$-t is
  - Hivatkozások a szakaszok és élek közt

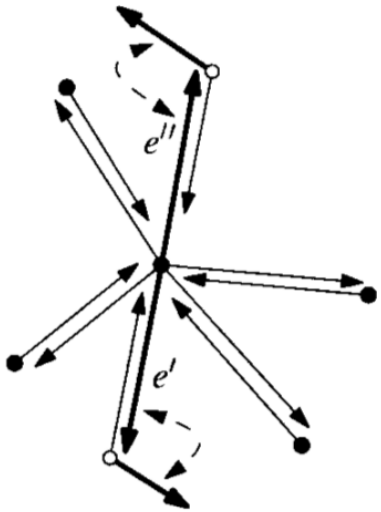


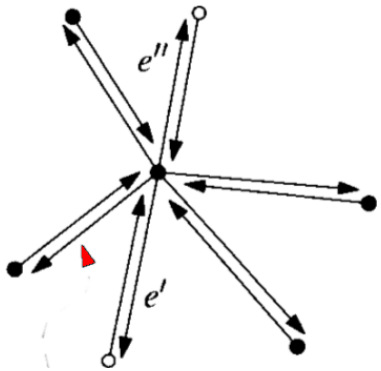


Példa részletesebben

- S_1-ből az e él átmegy S_2 v csúcsán
- e-ből e' és e'' $\Rightarrow$ 4 fél-él kell

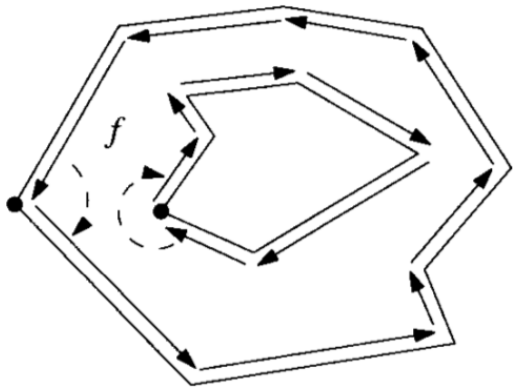
1. Két új fél él v kezdőponttal
2. A régi fél-élek *twin*jét ezekre állítjuk
   $\Rightarrow$ e' és e'' egy régi és egy új félélből állnak
3. Előző és következő fél-élek beállítása:
   - 3.1 Az újak a régi nem-*twin* rákövetkezőjét kapják (ezeknél az előzőre hivatkozás is módosul)
   - 3.2 v körül bonyolultabb a helyzet:
     meg kell állapítani a helyes körüljárást
     Pl. v-be menő e' fél-él rákövetkezője:
     a (CW) következő v-ből jövő fél-él
     $\Rightarrow$ $O(m)$, ahol m a v csúcs fokszáma





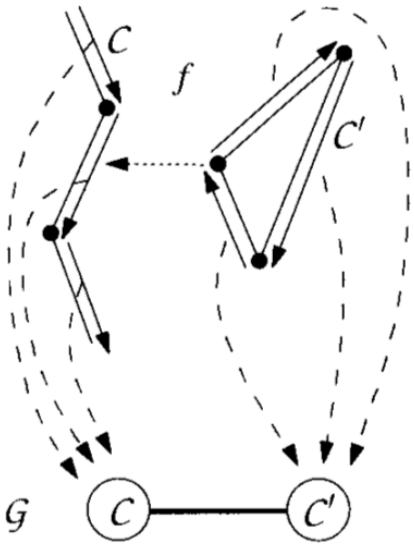
Lap-adatok javítása (1)

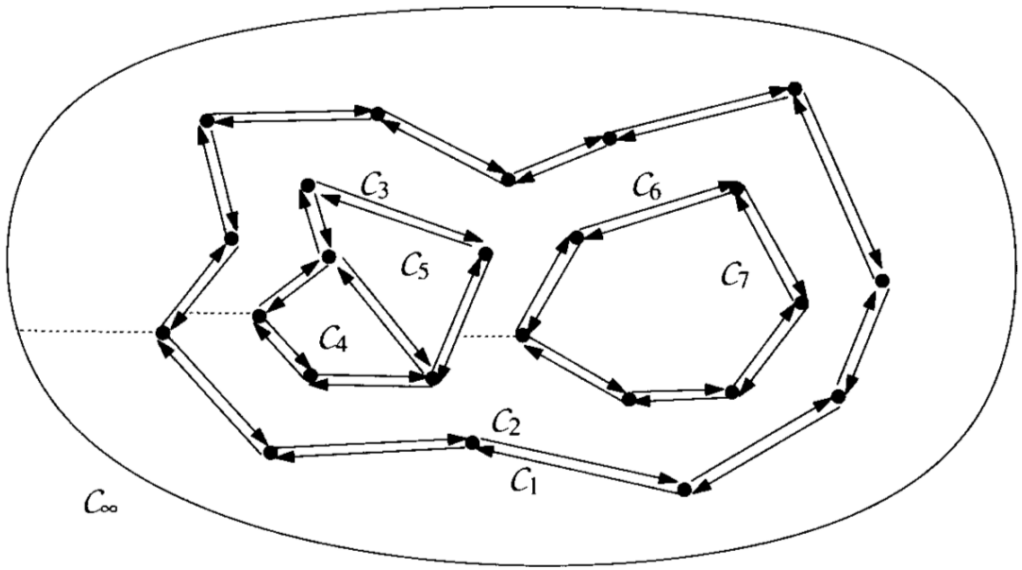
- Fél-él (és csúcs) adatok javítása után
- Lapokhoz:
  - Hivatkozás egy határoló fél-élre
  - Hivatkozás minden belső komponensből egy fél-élre
- Fél-élekhez:
  - Hivatkozás a határolt lapra
- Lapok száma = külső határok száma + 1
  - A határokat könnyű megszámolni
  - Külső, ha a legbaloldalibb pontnál (azon belül legalsónál) <180 fokot fordul
  - Különben lyukat határol
  - A +1 a külső (végtelen) lap miatt van

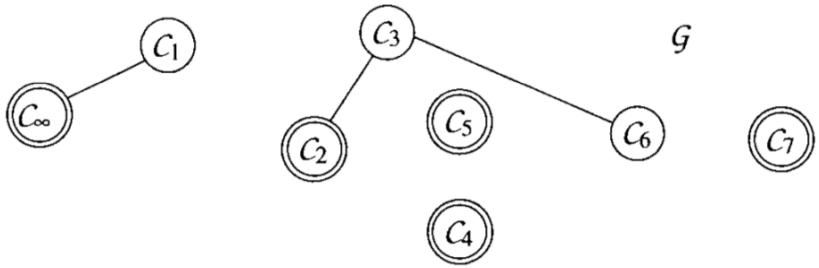


Lap-adatok javítása (2)

- Gráfot készíthetünk:
  Ha egy hurok baloldali pontjától balra van a másiknak fél-éle
- Ez épp az egy laphoz tartozó hurkokat adja



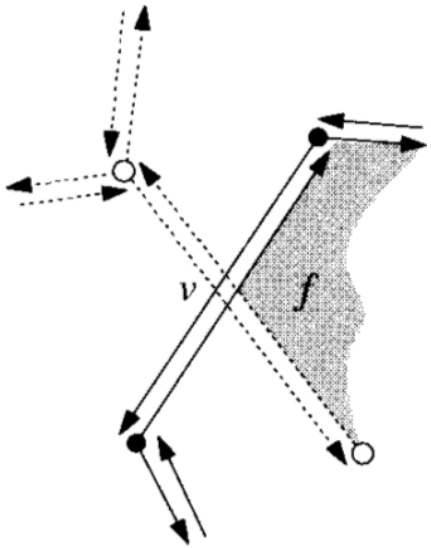


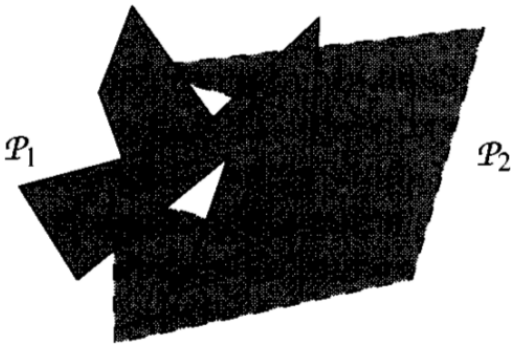


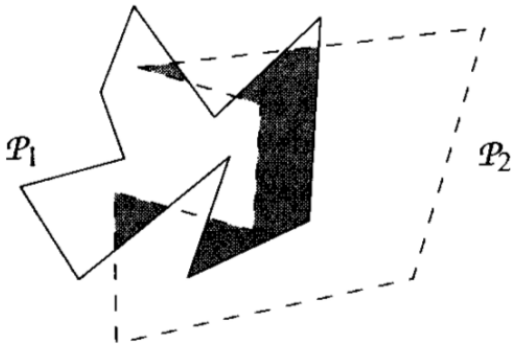
$\Leftarrow$ metszéskor ismerjük a szomszédokat!

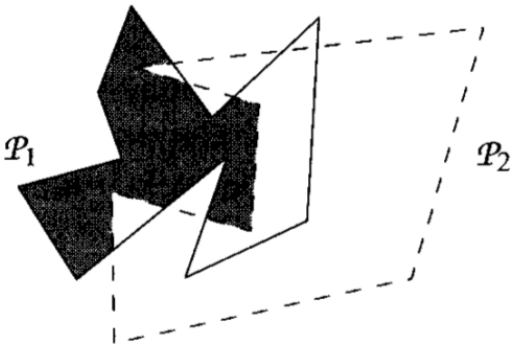
Attribútumok

- Legyen v a lap egy csúcsa
- Ha ez S_1 és S_2-beli élek metszete, egyszerű
- Ha eredetileg is valamelyiknek a csúcsa volt?
  - Ezekre egy újabb síksöpréssel kell meghatározni az attribútumokat...
- Alkalmazás: pl. Boolean-operációk



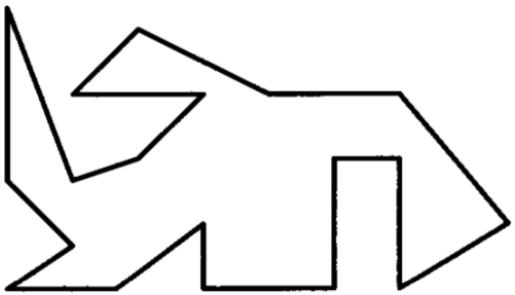


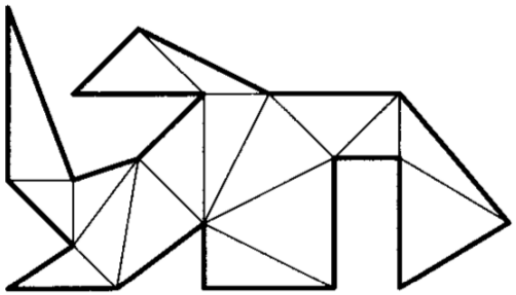




Feladat

- Hogyan osszunk fel egy P sokszöget háromszögekre?

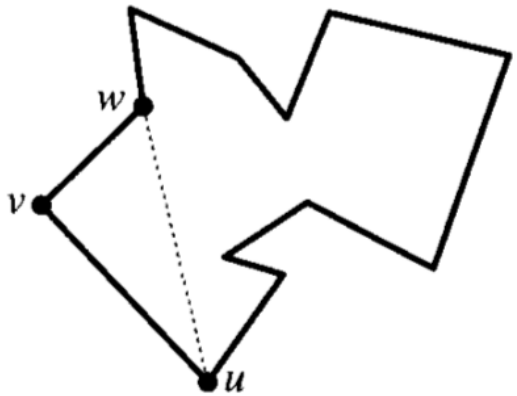


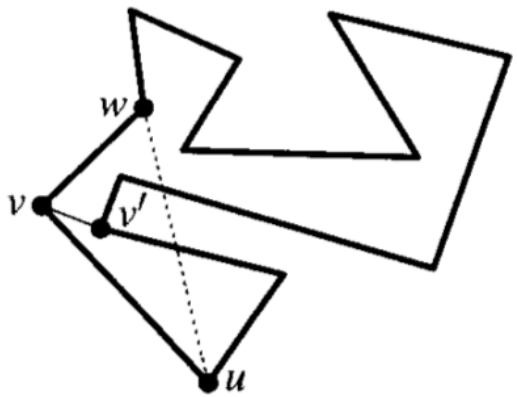


- Húzzuk be az összes lehetséges átlót, hogy:
  - Minden átló a sokszög belsejében halad
  - Az átlók nem metszik egymást
- n pont esetén $n-2$ háromszög
- Bizonyítás (teljes indukció, $n = 3$-ra triviális):
  - Egy átló behúzása $\Rightarrow$ két sokszög, rendre m_1 és m_2 csúccsal
  - Indukciós feltétel: $(m_1 - 2) + (m_2 - 2)$ háromszög
  - Az átló csúcsain kívül minden pont csak az egyik sokszögben
  - Tehát $m_1 + m_2 = n + 2$; ebből $(m_1 - 2) + (m_2 - 2) = n - 2$

Helyesség

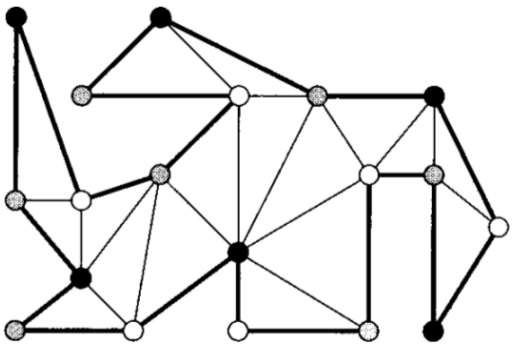
- Indukciós bizonyítás
  - $n = 3$-ra triviális
  - Tegyük fel, hogy $m < n$-re működik
- Kérdés: létezik-e (sokszögön belüli) átló?
  - Legyen v a legbaloldalibb csúcs (azonosaknál a legalsó)
  - u és w a két szomszéd
  - Ha az $\overline{uw}$ belül megy, készen vagyunk
  - Kül.: az $\triangle_{uvw}$-ben vagy $\overline{uw}$-n van csúcs
  - v' az $\overline{uw}$-től legtávolabbi ilyen csúcs
  - $\overline{vv'}$-t nem metszheti él, mert annak egy végpontja $\triangle_{uvw}$-ben lenne, de $\overline{uw}$-től távolabb, mint $v' \Rightarrow \overline{vv'}$ átló
- Az új rész-sokszögekre működik (indukció)

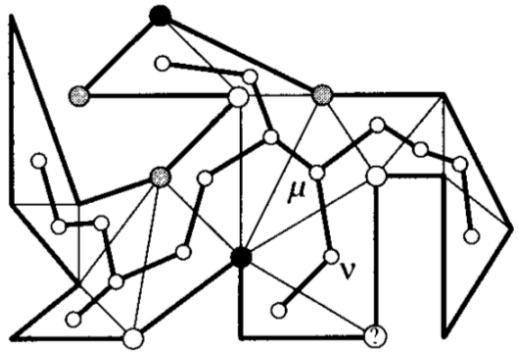




Galéria-probléma

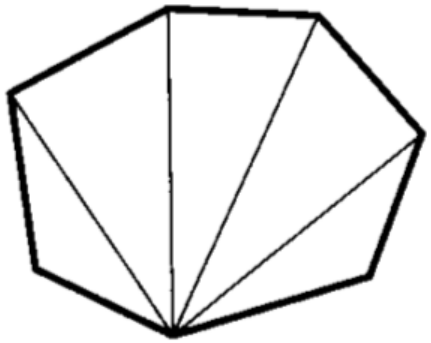
- Hova kell helyezni kamerákat, hogy belássák a galériát?
- Minden háromszögbe egy kamera $\Rightarrow n-2$ kamera
- Átlón levő kamera két háromszöget figyel $\Rightarrow$ kb. $n/2$ kamera
- Csúcsban levő kamerákkal még hatékonyabb
  - Úgy válasszunk csúcsokat, hogy minden $\triangle$ tartalmazzon egyet
  - Három színnel kiszínezzük a csúcsokat
  - Azt a színt választjuk, amelyikből a legkevesebb van $\Rightarrow \left\lfloor \frac{n}{3} \right\rfloor$
  - Mindig van színezés (ha nincs lyuk), mert a duális gráf fa $\Rightarrow$ mélységi bejárás; μ-ből ν-be lépéskor ν többi szomszédját még nem látogattuk, ν harmadik csúcsának színe adódik

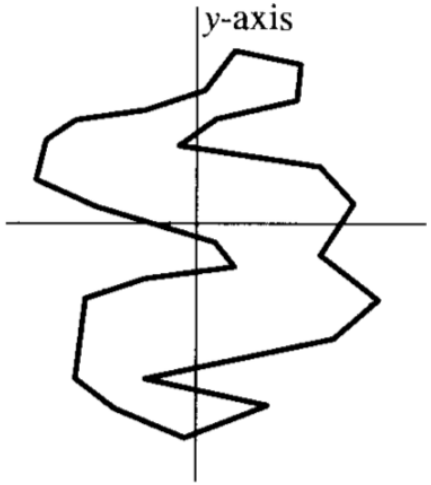




Hatékony algoritmus?

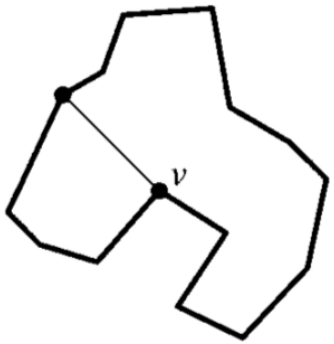
- Az előző (naív) algoritmus $O(n^2)$
- Konvex sokszög:
  - Egy pontból az összes átló – $O(n)$
- Ötlet:
  - Konvex sokszögekre bontás
  - Ugyanolyan nehéz, mint a háromszögelés :(
- P monoton l-re nézve, ha l-re merőleges egyenessel vett metszete összefüggő
  - y-monoton P-nél a legfelső csúcstól a legalsóig menve sosem lépünk felfele
- Ötlet:
  - y-tengelyre monoton sokszögekre bontás

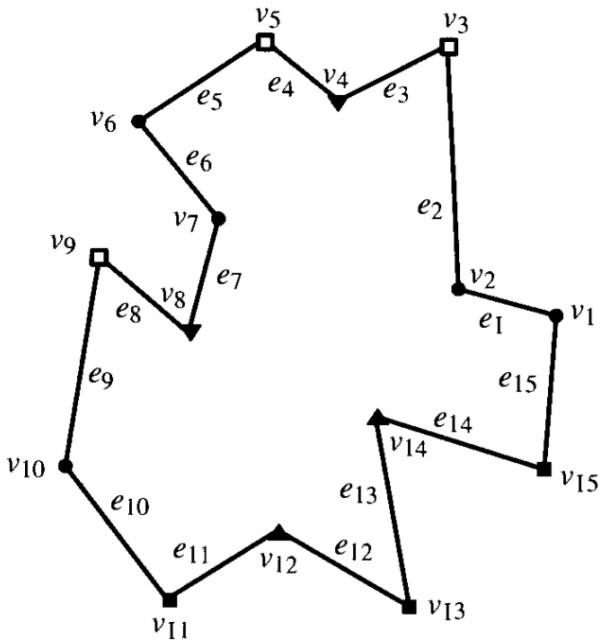




Csúcs-típusok

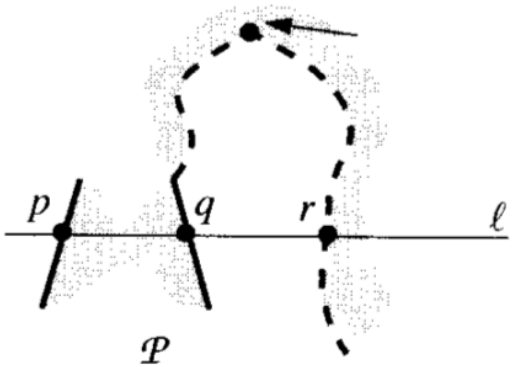
- Elindulunk a legfelső csúcsból lefele
- *Forduló csúcs*: fel–le irányváltás
- Ezektől kell megszabadulni átlók behúzásával
- Pl. ha v-ből mindkét él lefele megy és a sokszög belseje v felett van (azaz a belső szög $> \pi$):
  - v-ből induló felfele menő átló kell
- Négy típusú forduló csúcs:
  1. □ kezdő csúcs (3,5,9)
  2. ■ végcsúcs (11,13,15)
  3. ▲ kettéválasztó (split) csúcs (12,14)
  4. ▼ egyesítő (merge) csúcs (4,8)

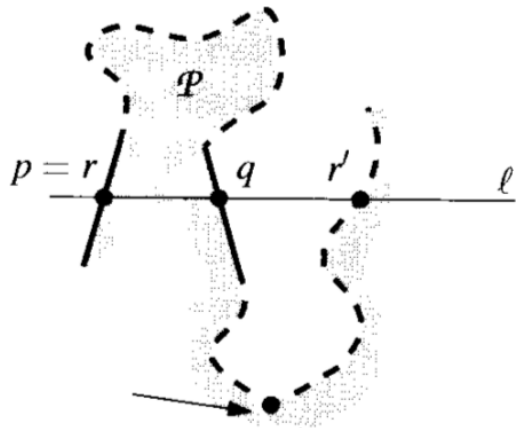




y-monotonitás

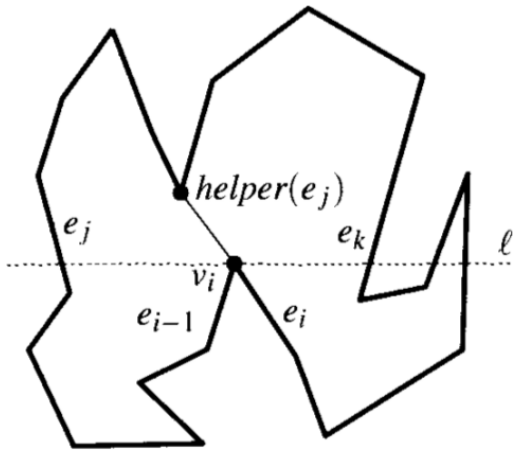
- Állítás:
  - Ha P-ben nincsenek *split* és *merge* csúcsok, akkor y-monoton
- Bizonyítás (indirekt):
  - Van egy l vízszintes egyenes, ami több darabra vágja P-t
  - Válasszuk l-t úgy, hogy a legbaloldalibb komponens szakasz
  - A szakasz kezdőpontja p, végpontja q
  - q-tól követjük a sokszöget a következő metszésig (r)
  - Ha $p = r$, akkor a másik irányban keresünk metszést (r')
  - A legmagasabb ill. legalacsonyabb csúcs split ill. merge lesz

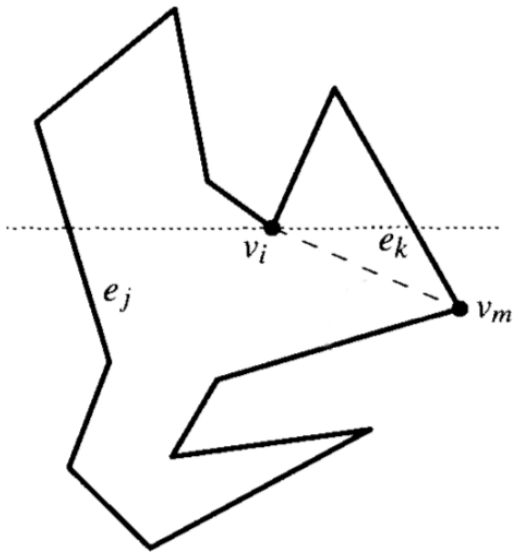




Söprőegyenes-módszer

- Események = csúcsok
  - Felülről lefele, azon belül balról jobbra
- Split csúcs (v_i) kezelése:
  - Söprőegyenesen szomszéd élek: e_j és e_k
  - e_j és e_k közötti legalacsonyabb v_i feletti csúcshoz kötjük
  - Ha nincs $\Rightarrow$ e_j vagy e_k felső csúcsához
  - `helper`(e_j): legalsó e_j-hez köthető csúcs
- Merge csúcs (v_i) kezelése:
  - e_j és e_k közötti legmagasabb v_i alatti csúcshoz kötnénk... de mi az?
  - v_m lesz v_i helyett az új `helper`(e_j)
  - Ha a `helper` változik, ellenőrizni kell, hogy a régi merge csúcs volt-e





Adatszerkezetek

- Események kezelése
  - Q prioritásos sor – következő lekérdezése $O(\log n)$
  - Statikus $\Rightarrow$ lehet sima lista, amit előre rendezünk – $O(n \log n)$
- Állapot tárolása
  - Szükségünk van a balra levő élek lekérdezésére
  - T bináris keresőfában a söprőegyenest metsző élek
  - Elég azokat tárolni, amelyektől jobbra van a sokszög belseje
  - Tároljuk a `helper`-t is hozzájuk
- Sokszög reprezentáció
  - Az átlóbehúzások rész-sokszögeket készítenek
  - Fél-él adatstruktúrában tároljuk
  - A T-ben tárolt élek erre mutatnak

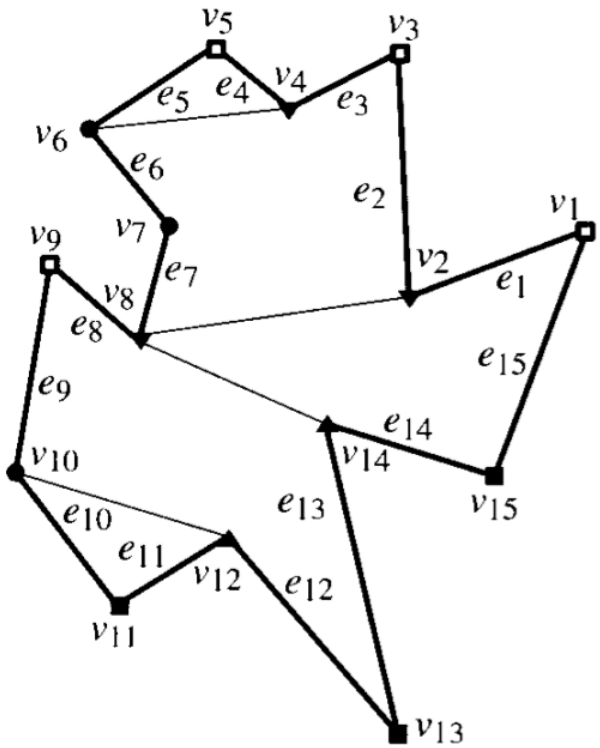
Algoritmus (Lee–Preparata, 1977)

- Bemenet: P egyszerű sokszög, mint D fél-él struktúra
- Kimenet: Monoton rész-sokszögekre osztás, D-ben tárolva

1. Csúcsok (események) Q-ba
2. T üres bináris keresőfa
3. Amíg Q nem üres:
   - 3.1 Kivesszük a következő elemet Q-ból (v_i)
   - 3.2 A v_i csúcs típusától függően kezeljük
     - `HandleStartVertex`(v_i)
     - `HandleEndVertex`(v_i)
     - `HandleSplitVertex`(v_i)
     - `HandleMergeVertex`(v_i)
     - `HandleRegularVertex`(v_i)

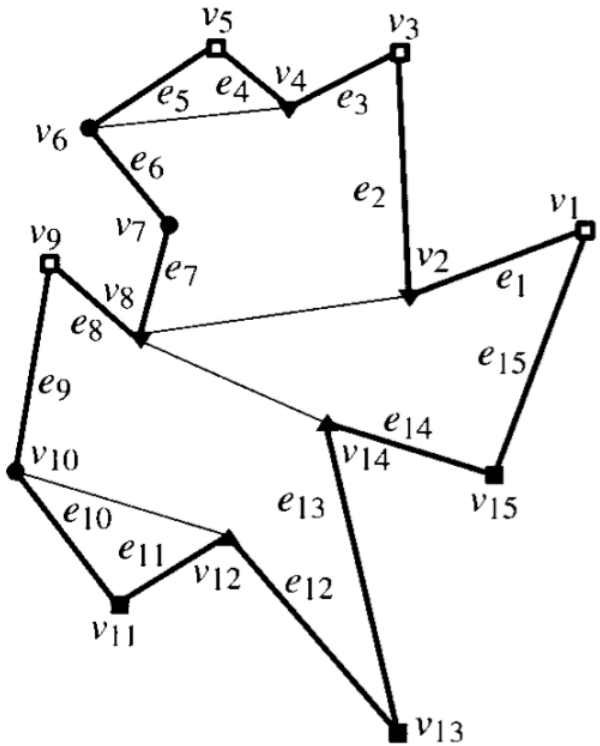
Start, End

- Merge csúcs emlékeztető:
  - Ha a `helper` változik $\Rightarrow$ ellenőrzés, hogy a régi merge csúcs volt-e
  - A `helper` változik...
    - Ha tőle jobbra új pont van, ami összeköthető vele
    - Ha az alsó csúcsához érkeztünk
- `HandleStartVertex`(v_i)
  - $T \leftarrow e_i$, `helper`$(e_i) = v_i$
- `HandleEndVertex`(v_i)
  - Ha `helper`(e_{i-1}) merge csúcs:
    - Átló v_i és `helper`(e_{i-1}) közt D-be
  - e_{i-1} törlése T-ből



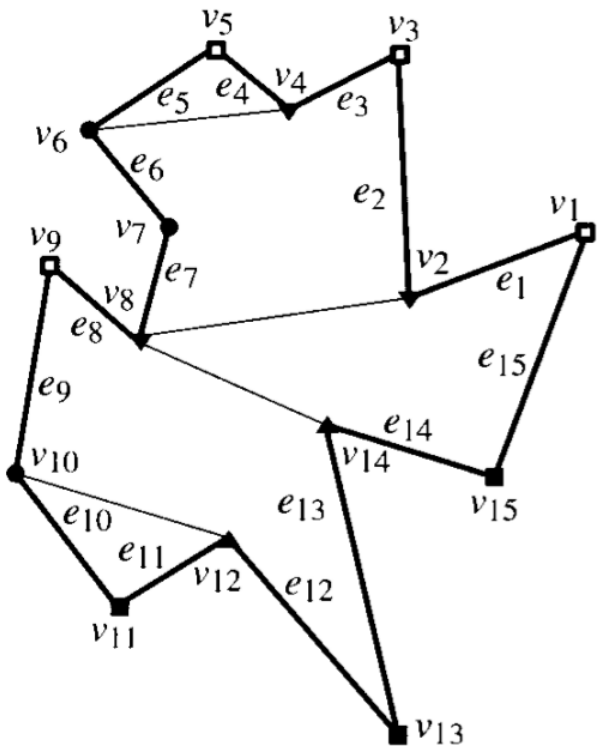
Split, Merge

- `HandleSplitVertex`(v_i)
  - Baloldali e_j keresése T-ben
  - Átló v_i és `helper`(e_j) közt D-be
  - `helper`(e_j) = v_i
  - $T \leftarrow e_i$, `helper`(e_i) = v_i
- `HandleMergeVertex`(v_i)
  - Ha `helper`(e_{i-1}) merge csúcs:
    - Átló v_i és `helper`(e_{i-1}) közt D-be
  - e_{i-1} törlése T-ből
  - Baloldali e_j keresése T-ben
  - Ha `helper`(e_j) merge csúcs:
    - Átló v_i és `helper`(e_j) közt D-be
  - `helper`(e_j) = v_i



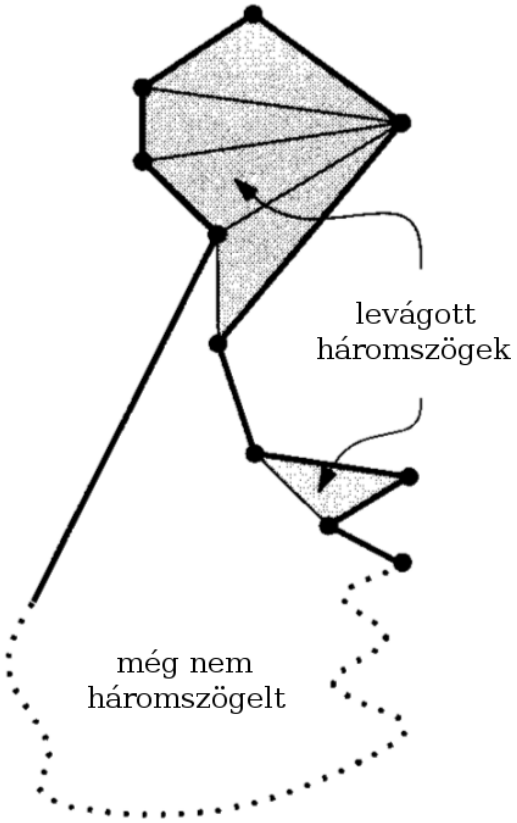
Regular

- `HandleRegularVertex`(v_i)
  - Ha v_i-től jobbra van P belseje:
    - Ha `helper`(e_{i-1}) merge csúcs:
      Átló v_i és `helper`(e_{i-1}) közt D-be
    - e_{i-1} törlése T-ből
    - $T \leftarrow e_i$, `helper`(e_i) $= v_i$
  - Ha v_i-től balra van P belseje:
    - Baloldali e_j keresése T-ben
    - Ha `helper`(e_j) merge csúcs:
      Átló v_i és `helper`(e_j) közt D-be
    - `helper`(e_j) $= v_i$
- Műveletigény: $O(n \log n)$
- Memóriaigény: $O(n)$



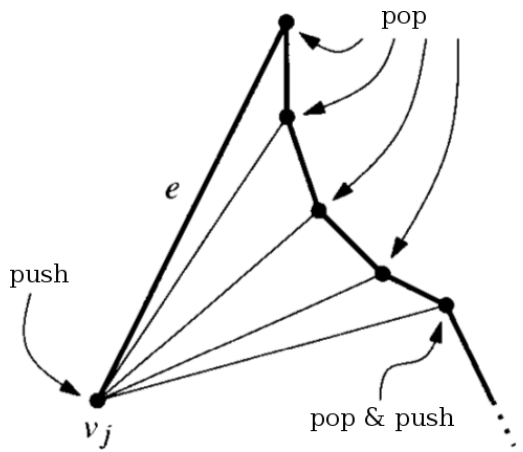
Hatékony háromszögelés

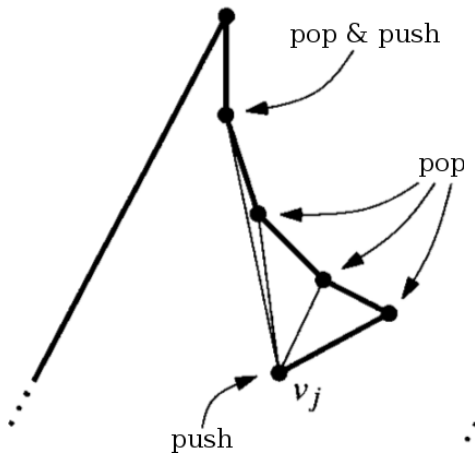
- Monoton sokszögekre $O(n)$
  - Mohó algoritmus
- Tfh. P szigorúan y-monoton ($\emptyset$ vízszintes)
- Felülről lefele megyünk megint
- Egy S veremben a látogatott csúcsok, amelyekhez még lehet húzni átlókat
- Egy csúcsból minél több S-belihez átló
- Még nem háromszögelt rész tölcsér alakú
  - Egyik oldalt egy él
  - Másik oldalt csak *visszaforduló* csúcsok, amelyek belső szöge $\geq \pi$ („reflex" csúcs)
  - Csak a legfelső csúcs konvex

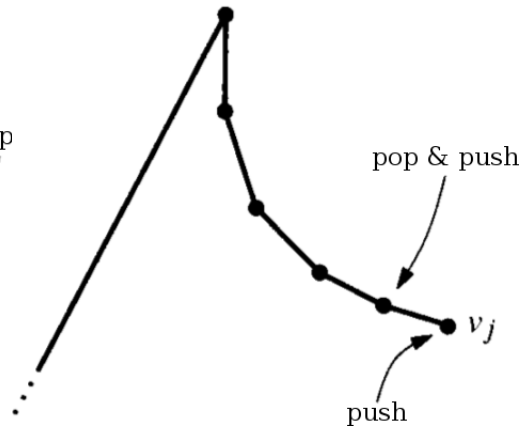


Átlók részletesebben

- Következő kezelendő csúcs: v_j
- Ha v_j a reflexekkel szemben van:
  - Minden csúcshoz húzható átló, kivéve a legrégebbit
  - A legalsó átló csúcsai maradnak S-ben
- Ha v_j a reflexek mellett van:
  - Átlókat húzunk S-be, amíg csak tudunk
    - S tetejét átugorva
  - v_j összeköthető v_k-val, ha az előző pop-olt csúccsal való $\triangle$ nem fordul ki





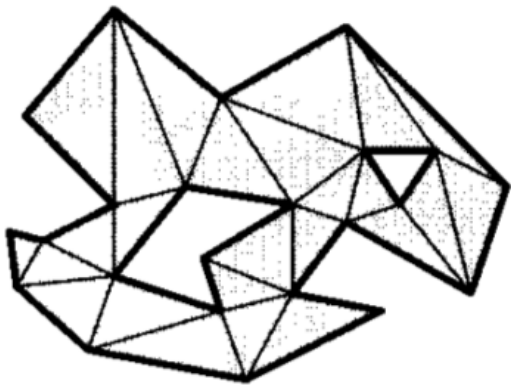


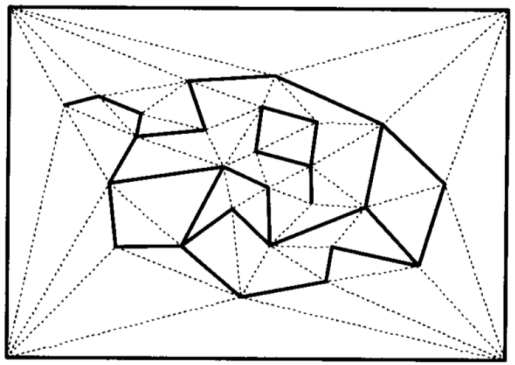
Algoritmus (Garey et al., 1978)

1. $u_1, \ldots, u_n$ a csúcsok felülről lefele (azon belül balról jobbra)
2. $S \leftarrow u_1$, $S \leftarrow u_2$
3. Ciklus $j = 3 \rightarrow n - 1$
   - 3.1 Ha u_j és az S tetején levő csúcs más oldalon vannak:
     - 3.1.1 Minden csúcsot pop-olunk S-ből
     - 3.1.2 Az utolsó kivételével mindegyikhez átlót húzunk
     - 3.1.3 $S \leftarrow u_{j-1}$, $S \leftarrow u_j$
   - 3.2 Egyébként:
     - 3.2.1 Egy csúcsot pop-olunk S-ből
     - 3.2.2 pop-olunk S-ből és átlót húzunk, amíg lehet
     - 3.2.3 Az utoljára pop-olt csúcsot visszatesszük S-be
     - 3.2.4 $S \leftarrow u_j$
4. Átlót húzunk u_n-ből az összes S-beli csúcshoz, kivéve az elsőt és az utolsót

Háromszögelés általánosabban

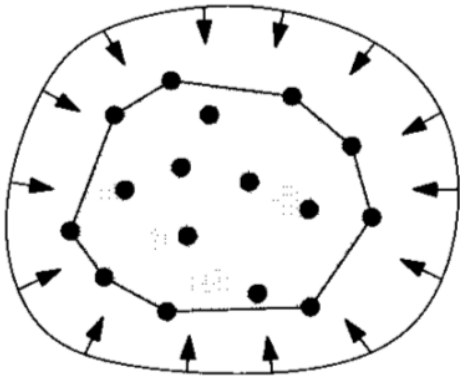
- Működik vízszintes élekre is
  - A baloldali csúcs „magasabban” van
  - ε dőlésszögű söprés
- Működik lyukas sokszögekre is
- Működik még általánosabban is
  - Határolódobozban levő síkfelosztásokra:
- Egyszerű sokszögekre létezik gyorsabb
  - $O(n \log \log n)$ Tarjan–Van Wyk (1988)
  - $O(n \log^* n)$ Clarkson et al. (1989)
    - Randomizálással, egyszerű
    - $\log^* n$ az iterált logaritmus
  - $O(n)$ Chazelle (1990/1991) – bonyolult

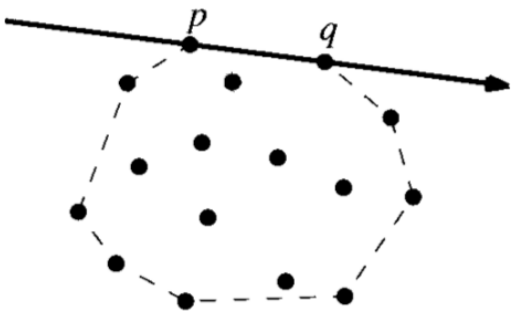




Feladat

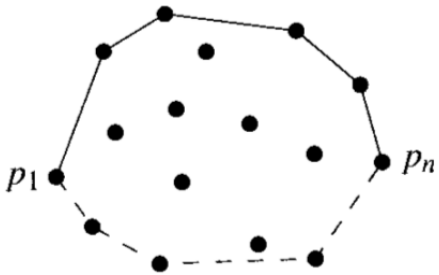
- Adott n pont
- Legszűkebb konvex sokszög körülötte?
- Mintha egy gumikarikát cuppantanánk rá
- Kimenet: burokpontok (CW) körbejárása
- Naív algoritmus – $O(n^3)$
  1. $E = \emptyset$
  2. Ciklus (p, q) rendezett párokon ($\overrightarrow{pq}$ irányított egyenes)
     - 2.1 Ha minden más r pont ettől *jobbra*, vagy a $\overline{pq}$ szakaszon belül van: $E \leftarrow \overrightarrow{pq}$
  3. E alapján rendezett lista készítése

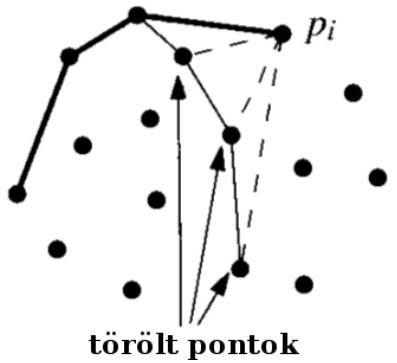


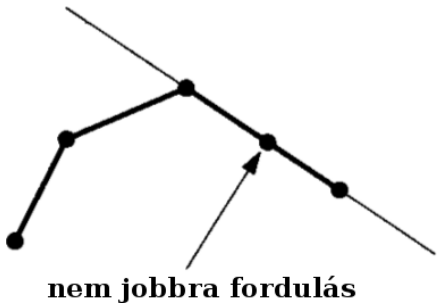


Inkrementális módszer

- Ötlet: haladjunk balról jobbra
  - Rendezés x koordináta szerint
  - Azon belül y koordináta szerint
- Két menet
  1. Felső burok
  2. Alsó burok
- Felső burok számítása (alsó szimmetrikusan)
  - Legbaloldalibb ponttól indulunk
  - Következő p_i pontot hozzáadjuk
  - Ha az utolsó 3 pont nem jobbra fordulás
    - Töröljük a középsőt
    - Ellenőrzés ismétlése





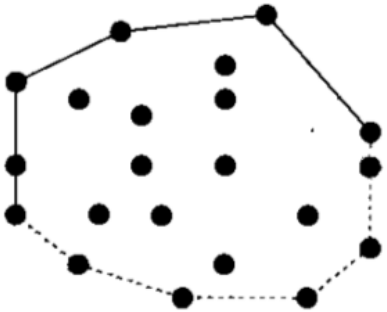


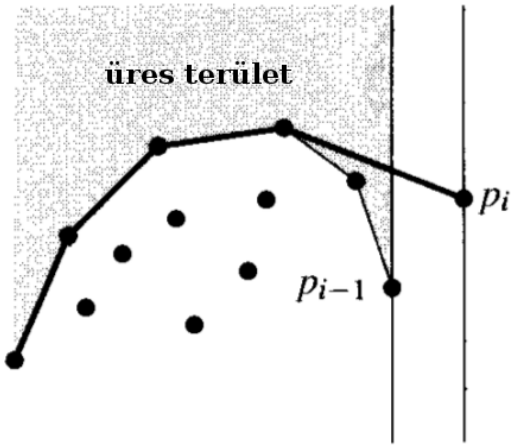
Algoritmus (Graham, 1972 / Andrew, 1979)

1. Pontok rendezése x-koordináta szerint: $p_1, \ldots, p_n$
2. $L_{\text{upper}} \leftarrow p_1$, $L_{\text{upper}} \leftarrow p_2$
3. Ciklus $i = 3 \rightarrow n$
   - 3.1 $L_{\text{upper}} \leftarrow p_i$
   - 3.2 Amíg $|L_{\text{upper}}| > 2$ és az utolsó 3 pont nem jobbra fordulás
     - 3.2.1 Utolsóelőtti pont törlése az L_{upper} listából
4. $L_{\text{lower}} \leftarrow p_n$, $L_{\text{lower}} \leftarrow p_{n-1}$
5. Ciklus $i = n - 2 \rightarrow 1$
   - 5.1 $L_{\text{lower}} \leftarrow p_i$
   - 5.2 Amíg $|L_{\text{lower}}| > 2$ és az utolsó 3 pont nem jobbra fordulás
     - 5.2.1 Utolsóelőtti pont törlése az L_{lower} listából
6. Első és utolsó pont törlése az L_{lower} listából
7. $L = L_{\text{upper}} + L_{\text{lower}}$ az eredmény

Bizonyítás & komplexitás

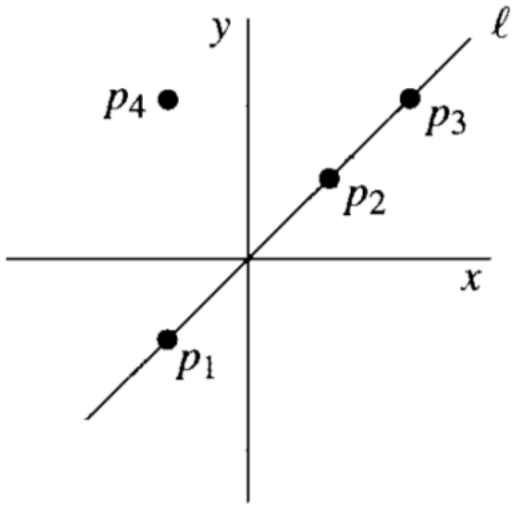
- Helyesség bizonyítása (felső burokra):
  - Indukció a feldolgozott pontok számán
  - Ind. feltétel: $p_1, \ldots, p_{i-1}$ közül egy sem volt a régi burok felett
  - Az új burok a régi felett van
    $\Rightarrow$ csak p_{i-1} és p_i közt lehetne hiba
- Műveletigény: $O(n \log n)$ – optimális
  - Preparata–Hong (1977) [D&C]
  - Kallay (1984) [inkrementális]
- Kimenet-érzékeny módszerek:
  - $O(n \cdot h)$ Jarvis (1973)
  - $O(n \cdot h)$ Eddy (1977) / Bykat (1978)
  - $O(n \log h)$ Kirkpatrick–Seidel (1986)
  - $O(n \log h)$ Chan (1996), egyszerűbb

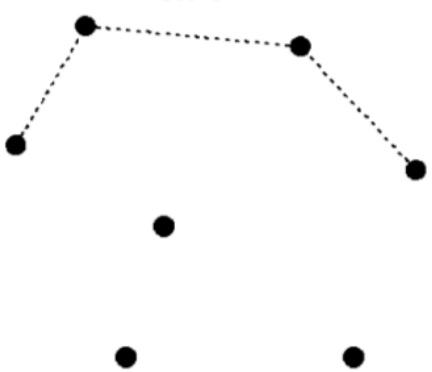


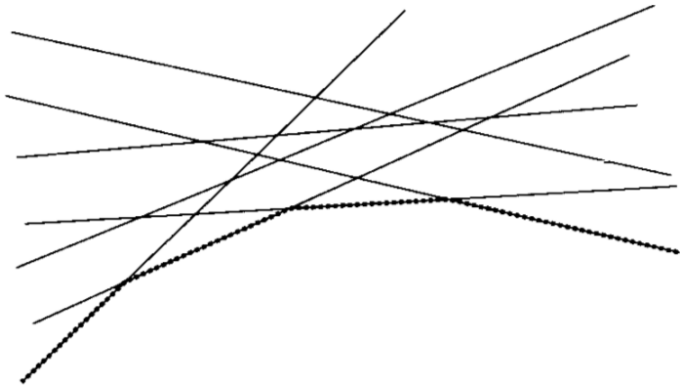


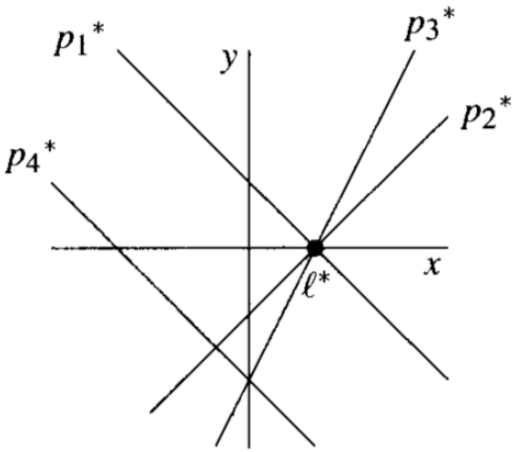
Duális sík

- Pont: p_x, p_y [koordináták]
- Egyenes: $y = p_x x - p_y$ [meredekség és y-metszés]
- $p \in l \equiv l^* \in p^*$
- p l felett van $\equiv$ l^* p^* felett van
- Felső burok $\equiv$ alsó féltér-metszet







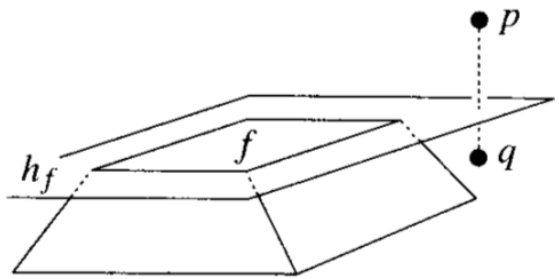


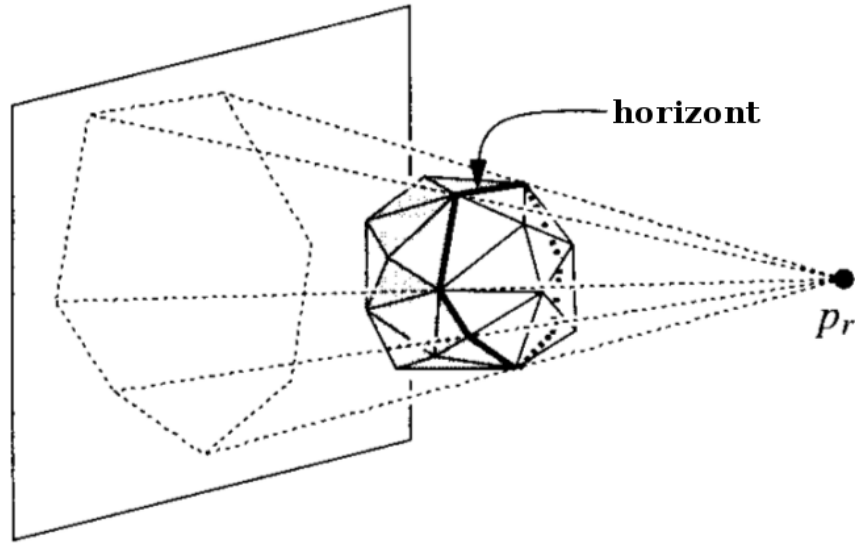
Randomizált inkrementális módszer

- Konvex burok: konvex lapok által határolt
- Válasszunk 4 pontot, ami nincs egy síkban
  - Vesszük az első két pontot (p_1 és p_2)
  - Keresünk egy harmadikat (p_3), ami nem egy egyenesen van
  - Keresünk egy negyediket (p_4), ami nincs egy síkban
  - Ha nincs ilyen $\Rightarrow$ síkban vagyunk, ld. előző algoritmus
- A maradék pontokat randomizáljuk ($p_5, \ldots, p_n$)
- Jelölje P_r a $\{p_1, \ldots, p_r\}$ halmazt, $CH(P_r)$ a konvex burkát
- Sorban vesszük a pontokat, és változtatjuk a burkot
- Ha a következő pont (p_r) benne van $CH(P_{r-1})$-ben, vagy a határán van $\Rightarrow$ nincs teendő
- Ha p_r kívül van $CH(P_{r-1})$-en, akkor bonyolultabb...

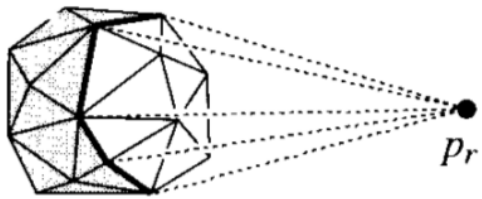
Láthatóság

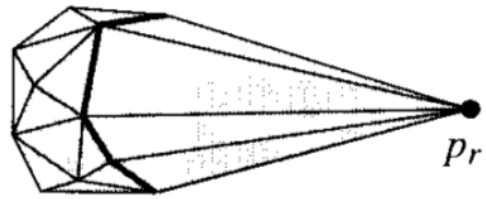
- Szétválasztjuk $CH(P_{r-1})$ lapjait aszerint, hogy láthatóak-e p_r-ből
- Egy f lap látható, ha p_r a h_f sík külső félterében van





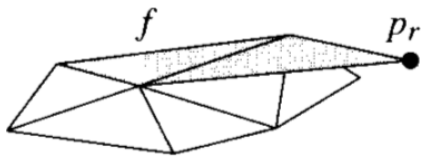
p_r-t a horizont pontjaihoz kötjük, látható lapokat eldobjuk

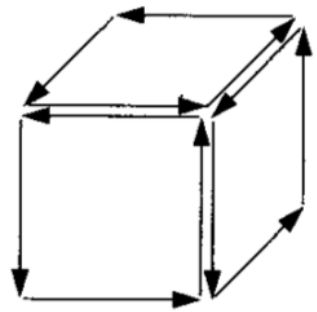




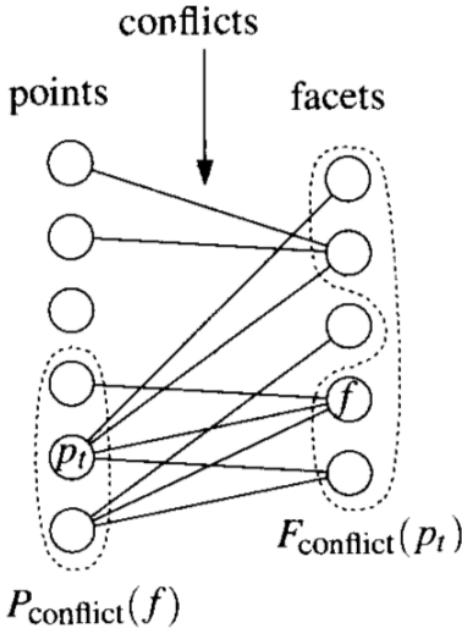
Részletek

- Ne csináljunk azonos síkú lapokat
  - Ha p_r a $CH(P_{r-1})$ egy lapsíkjában van
  - Össze kell vonni egy lappá





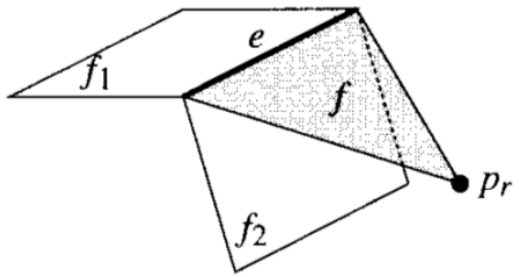
- Konvex burok fél-él struktúrában
  - A fél-élek kívülről nézve CCW körüljárással határolják a lapokat
- Látható lapok keresése az érdekes
  - Brute force: $O(r) \Rightarrow O(n^2)$
- Tároljuk a láthatóságokat (pont⇔lap)!
  - G páros „konfliktus” gráfban



Konfliktus gráf

- G inicializálása
  - Végigmegyünk a pontokon, hogy mit látnak a tetraéderből
- G frissítése
  - Kivesszük G-ből p_r-t és a szomszédait (hiszen ezek láthatóak)
  - Hozzáadjuk G-hez az új lapokat
    - Az összevont lapok konfliktusai nem változnak
    - Minden más új lap háromszög

- Legyen f egy új háromszög;
  p_t egy pont, ami látja f-et
- $CH(P_{r-1}) \subset CH(P_r)$
  $\Rightarrow$ p_t már $CH(P_{r-1})$-ben is látta e-t
- e egy volt szomszédja (f_1 vagy f_2) látható p_t-ből
- Elég tehát az f_1 és f_2 konfliktus-listájában szereplő pontokat ellenőrizni



Algoritmus (Clarkson–Shor, 1989)

1. Kezdő tetraéder keresése, $C \leftarrow CH(P_4)$
2. Maradék pontok randomizálása $(p_5, \ldots, p_n)$
3. G feltöltése (p_t, f) párokkal – $t \geq 5$, f egy lap C-ből
4. Ciklus $r = 5 \rightarrow n$
   - 4.1 Ha $F_{\text{conflict}}(p_r)$ üres $\Rightarrow$ tovább a köv. iterációra
   - 4.2 $C \leftarrow p_r$
   - 4.3 Horizont-élek listája (L) az $F_{\text{conflict}}(p_r)$ régió határaként
   - 4.4 Töröljük az $F_{\text{conflict}}(p_r)$ lapokat C-ből
   - 4.5 Ciklus $e \in L$
     - 4.5.1 Új háromszöglap (f) C-be, ami összeköti e-t p_r-el
     - 4.5.2 Ha f egy síkban van e másik lapjával $\Rightarrow$ egyesítés

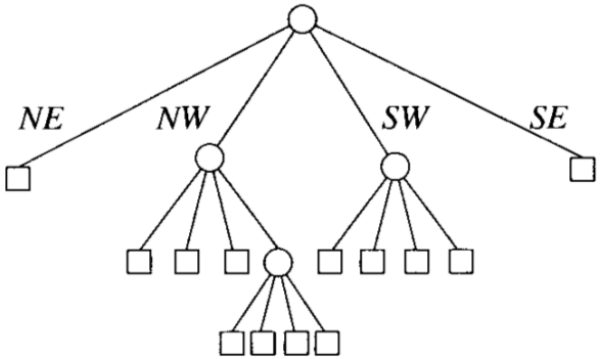
     Kül. f felvétele G-be; f_1 és f_2 az e régi szomszédai
     - $P(e) \leftarrow P_{\text{conflict}}(f_1) \cup P_{\text{conflict}}(f_2)$
     - (p, f) párok felvétele G-be, ha f látható p-ből, $p \in P(e)$
   - 4.6 p_r és $F_{\text{conflict}}(p_r)$ törlése G-ből

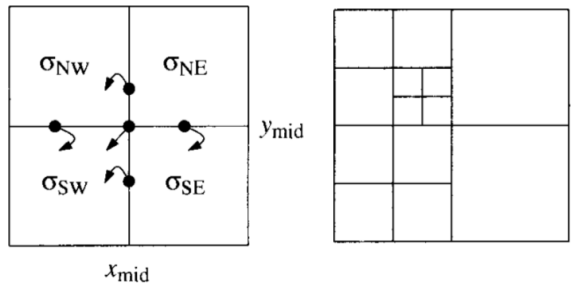
Komplexitás

- Műveletigény: $O(n \log n)$, ami optimális
- Memóriaigény: $O(n \log n)$, de könnyen javítható $O(n)$-re
- $\mathbb{R}^d$-ben is optimális (legrosszabb esetre nézve): $\Theta(n^{\lfloor d/2 \rfloor})$
- Kb. hány éle/lapja lesz a konvex buroknak?
  - Euler-formula: $n - e + f = 2$
  - Minden laphoz legalább 3 él tartozik
    és minden élhez pontosan 2 lap $\Rightarrow 2e \geq 3f$
  - Ebből $f \leq 2n - 4$ és $e \leq 3n - 6$, tehát $O(n)$
- Más algoritmusok:
  - $O(n \cdot f)$ Chand–Kapur (1970) [ajándékcsomagolás]
  - $O(n \log n)$ Preparata–Hong (1977) [D&C]
  - $\mathbb{R}^d$-ben kimenet-érzékeny:
    - $O(ndf)$ Avis–Fukuda (1992)
    - $O(n \log f + (nf)^{1-1/(\lfloor d/2 \rfloor + 1)} \log^{O(1)} n)$ Chan (1996)

Quadtree (Finkel–Bentley, 1974)

- Gyors geometriai lekérdezések
  - Szomszédos pontok
  - Adott távolságon belüli pontok
- Rekurzív felosztás 4 négyzetre
  - Amíg csak 1 pont marad
  - Más leállási feltételek is lehetnek
- Teljes: $\sigma = (x_\sigma : x'_\sigma] \times (y_\sigma : y'_\sigma]$
  - P a tárolt pontok halmaza
- Ha $|P| > 1$, akkor felosztás:
  - $x_{\text{mid}} = \frac{x_\sigma + x'_\sigma}{2}$, $y_{\text{mid}} = \frac{y_\sigma + y'_\sigma}{2}$

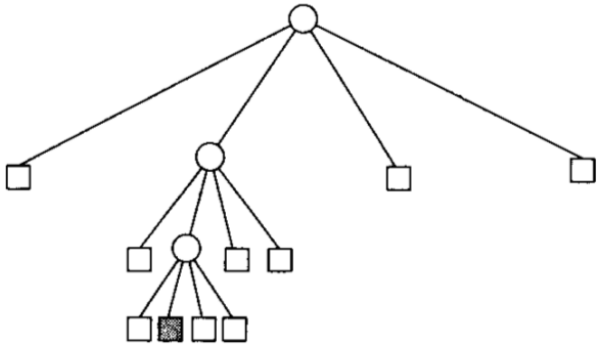




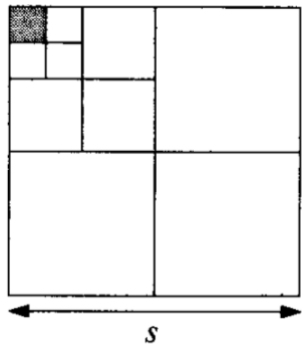
$P_{SE} = \{p \in P \ : \ p_x > x_{\text{mid}} \text{ és } p_y \leq y_{\text{mid}}\}$ stb.

Részletek

- Tároljuk minden szinthez:
  - A sarokpontokat
  - Vagy csak a középpontot
- Legfelső szint: befoglaló négyzet
- Legfeljebb $\log(s/c) + \frac{3}{2}$ szint
  - s a kiinduló négyzet élhossza
  - c a legkisebb távolság 2 pont közt



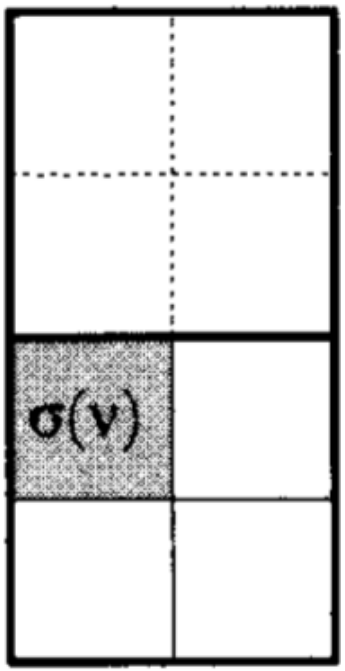
- Bizonyítás:
  - i-edik szinten élhossz: $s/2^i$
  - Egy négyzetben levő pontok max távolsága: $s\sqrt{2}/2^i \geq c \Rightarrow i \leq \log \frac{s\sqrt{2}}{c} = \log(s/c) + \frac{1}{2}$



Szomszéd négyzet

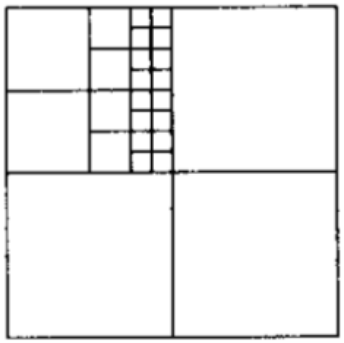
- Külön a négy irányban (példa: északi)
- Azonos szintű négyzet
  - Ha nincs $\Rightarrow$ a legmélyebb, ami van
- Input: v középpontú $\sigma(v)$ a T fában

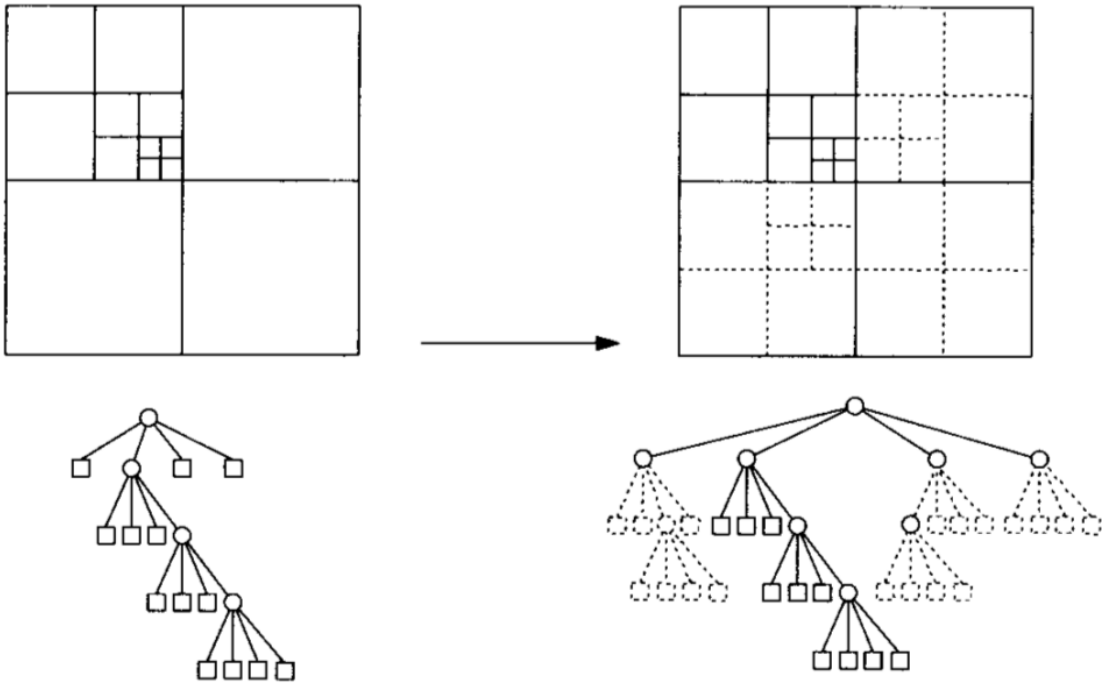
1. Ha $v = \texttt{root}(T) \Rightarrow \texttt{nil}$
2. Ha $v = SW(\texttt{parent}(v)) \Rightarrow NW(\texttt{parent}(v))$
3. Ha $v = SE(\texttt{parent}(v)) \Rightarrow NE(\texttt{parent}(v))$
4. $\mu \leftarrow \texttt{NorthNeighbor}(\texttt{parent}(v), T)$ [rekurzív]
5. Ha $\mu = \texttt{nil}$ vagy μ levél $\Rightarrow \mu$
6. Ha $v = NW(\texttt{parent}(v)) \Rightarrow SW(\mu)$
7. Ha $v = NE(\texttt{parent}(v)) \Rightarrow SE(\mu)$



Kiegyensúlyozás (1)

- A mélység nagyon változó lehet
  - Alkalmazástól függően ez lehet nem jó
- Szomszédos négyzetek mérete max. kétszeres
- Szomszédos levelek szintje max. ± 1





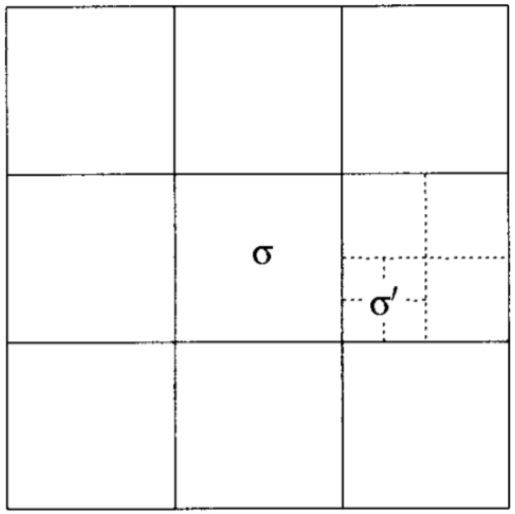
Kiegyensúlyozás (2)

1. L listába beletesszük T összes levelét
2. Amíg L nem üres
   - 2.1 Következő levél (μ) kivétele L-ből
   - 2.2 Ha μ-t fel kell osztani
     (van olyan szomszéd, ami ≥ 2 szinttel mélyebb)
     - 2.2.1 μ felosztása
     - 2.2.2 Ha μ-ben volt pont $\Rightarrow$ a megfelelő gyerekbe
     - 2.2.3 A négy új levelet L-be
     - 2.2.4 Ezáltal fel kell osztani vmelyik szomszédot?
       (van szomszéd, ami sekélyebb, mint μ volt)

- Ha T-nek m csúcsa volt...
  - A kiegyensúlyozott T_B-nek $O(m)$ csúcsa lesz
  - A kiegyensúlyozás $O((d+1)m)$ műveletigényű

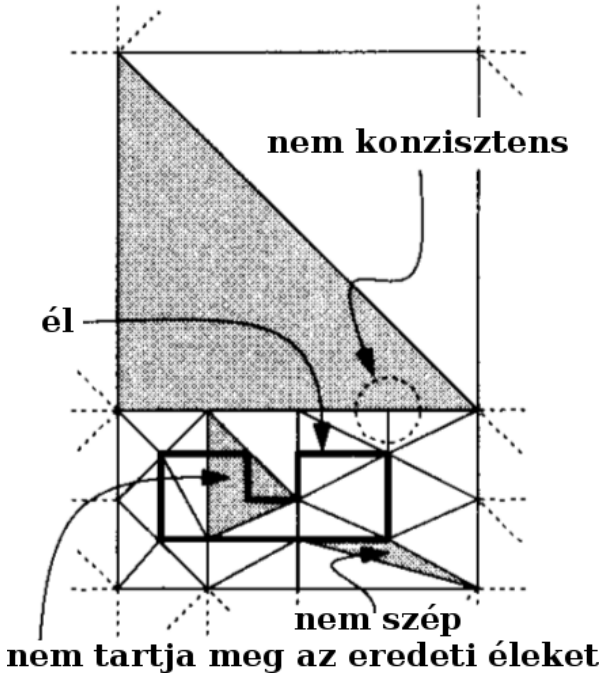
Kiegyensúlyozás (3)

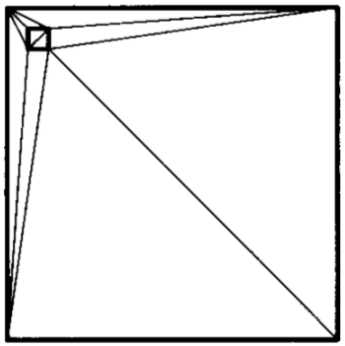
- Állítás: összesen $\leq 8m$ felosztás történik
  - Egy σ négyzet felosztásakor a környező négyzetekből egy biztosan eredeti
    - Ha ez nem lenne igaz:
    - Legyen σ a legmélyebb ilyen
    - Van ≥ 2 szinttel mélyebb szomszéd
    - σ' ennek a σ-nál eggyel mélyebb őse
    - Mivel $\texttt{parent}(\sigma')$ új, ezért σ' is
    - σ' szomszédai tehát mind újak:
      (i) σ'-vel együtt keletkezett
      (ii) σ szomszédjának a gyereke
      (iii) σ-nak a gyereke
    - Viszont akkor σ nem a legmélyebb
  - Ehhez „társítjuk” σ felosztását
  - Minden eredeti négyzethez ≤ 8 szomszéd

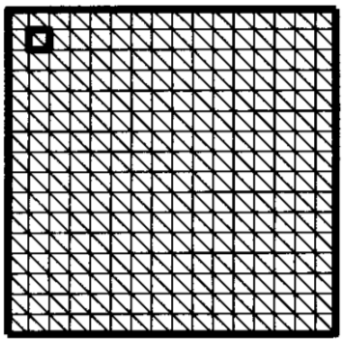


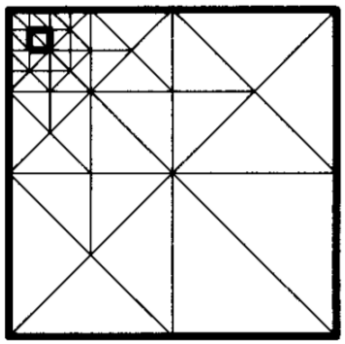
Alkalmazás – háromszögelés

- Célok:
  - Adott élek megtartásával
  - Konzisztens háromszögelés
  - Szép háromszögek
  - Non-uniform (sűrű ahol muszáj)
- Input élek:
  - 2^j felbontásban 0, 45, 90, 135 fokos
  - Pl. nyomtatott áramkörök
- Egyszerű △-elés / uniform / quadtree:



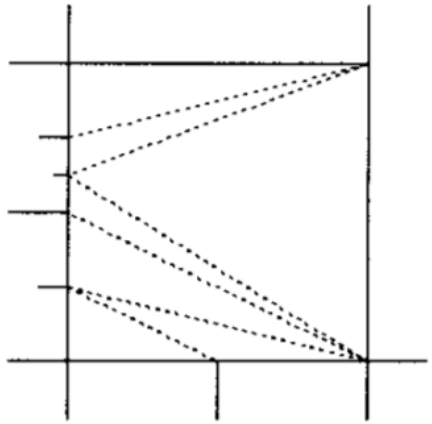


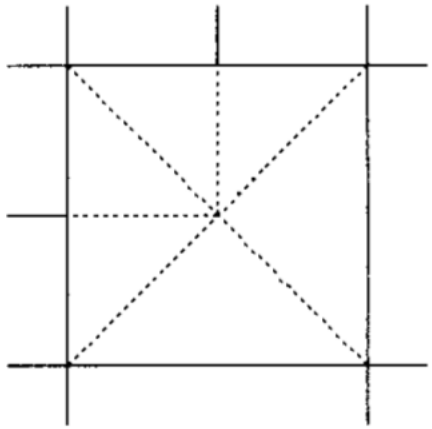




Részletek

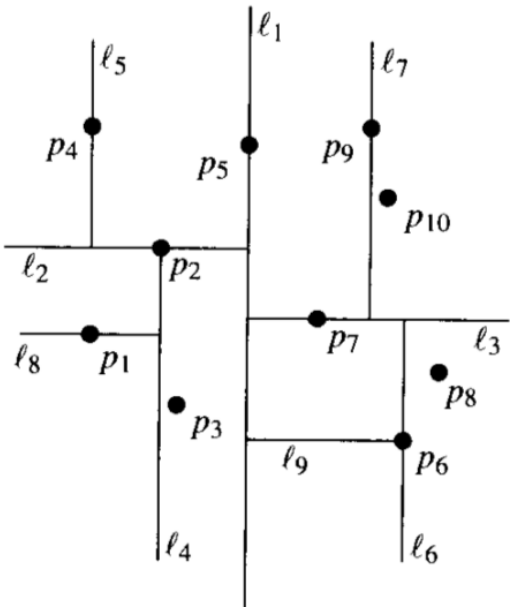
- Rekurzív osztás leállási feltétele:
  - Nincs él, ami metszi a négyzetet
  - Vagy egység oldalú a négyzet
- Nem elég a quadtree-be átlókat húzni
  - Nem lesz konzisztens
- Nem elég a belső osztásokat bekötni
  - Nem lesznek szép háromszögek
- Használjunk kiegyensúlyozott quadtree-t
  - Középső segédpont hozzáadásával javítható
  - (Segédpont $\approx$ „Steiner-pont”)
- $O(l \cdot \log^2 U)$ műveletigény, $O(l \cdot \log U)$ $\triangle$
  - l: élek összhossza, U: főnégyzet oldalhossza

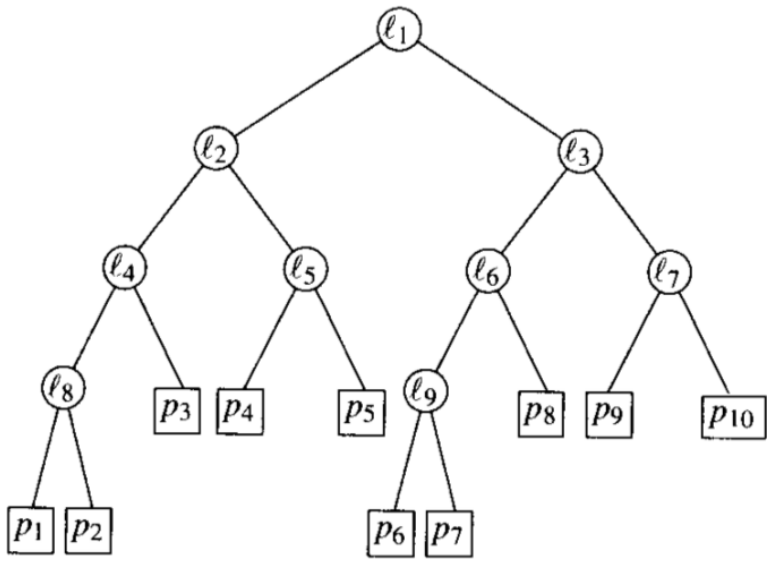




Kiterjesztés – *kd*-fa (Bentley, 1975)

- Tetszőleges helyen osztunk tengely-hipersíkokkal
  - A hipersíkok „lefelé” ciklikusan váltakoznak (x, y, z, $x \ldots$)
- Bináris fa struktúra
  - Belső csúcsok tartalmazzák a hipersík adatait
  - Levelek tartalmazzák a pontokat





2D *kd*-fa készítése – $O(n \log n)$ művelet, $O(n)$ memória

`BuildKdTree`(P, d)

- P a pontok halmaza, d az aktuális mélység (0 alapból)

1. Ha $|P| = 1 \Rightarrow$ visszatérés egy levéllel
2. Ha d páros:
   - 2.1 P felosztása függőleges l egyenessel a medián x-koordinátájú ponton át $\Rightarrow P_1$, P_2
3. Különben:
   - 3.1 P felosztása vízszintes l egyenessel a medián y-koordinátájú ponton át $\Rightarrow P_1$, P_2
4. $v_{\text{left}} \leftarrow$ `BuildKdTree`$(P_1, d+1)$, $v_{\text{right}} \leftarrow$ `BuildKdTree`$(P_2, d+1)$
5. Csúcs készítése l hipersíkkal és v_{left}, v_{right} gyerekekkel

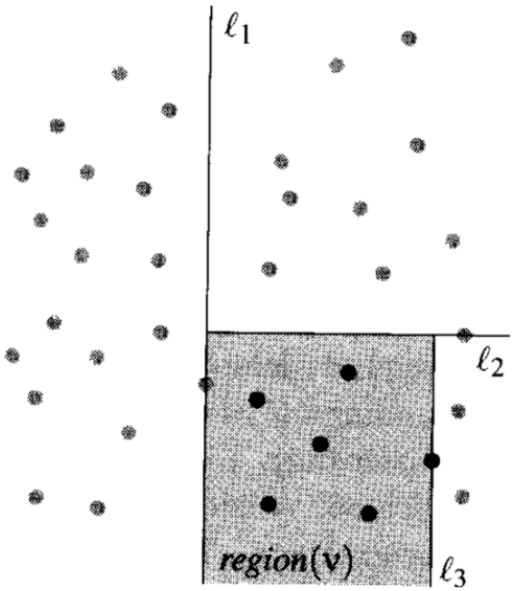
Medián (itt): az $\lceil n/2 \rceil$-dik legkisebb szám $\Rightarrow$ kettőből a kisebb

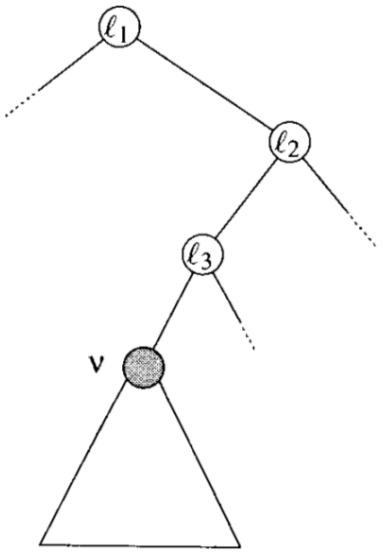
Keresés – adott régió összes pontja

`SearchKdTree`(v, R)

1. Ha v levél $\Rightarrow$ v-ben levő pont tesztje/kiírása, kilépés
2. Ha `region(left(`v`))` benne van R-ben:
   2.1 `left(`v`)` pontjainak kigyűjtése, kiírása
3. Kül. `region(left(`v`))` $\cap R \neq \emptyset \Rightarrow$ `SearchKdTree(left(`v`),` R`)`

4–5. [hasonlóan a `right(`v`)`-re]





Műveletigény

- $O(\sqrt{n}+k)$, ahol k a talált pontok száma
- (Komplexitás $\approx$ látogatott csúcsok száma)
- Bizonyítás:
  - Egy csúcs alatti pontok kigyűjtése lineáris $\Rightarrow O(k)$
  - Egy csak részben tartalmazott v csúcsnál:
  - Egy l függőleges egyenes hány csúcs-régiót metsz?
    - Vízszintes ugyanennyi lesz $\Rightarrow$ téglalap alakú régióé is
    - A legfelső (bal/jobb) osztásnak csak az egyik gyerekét metszi
    - Két szintet lejjebb megyünk $\Rightarrow$ csak 2 unokát metsz

$$Q(n) = \begin{cases} O(1), & \text{ha } n = 1, \\ 2 + 2Q(n/4), & \text{ha } n > 1. \end{cases} \quad \Rightarrow \quad Q(n) = O(\sqrt{n})$$

- d dimenzióban $O(n^{1-1/d}+k)$ [nagyon pesszimista]
- Általában sokkal kevesebb; egy pontra mindig $O(\log n)$

Keresés – legközelebbi szomszéd

- Egy p ponthoz legközelebbi P-beli pont

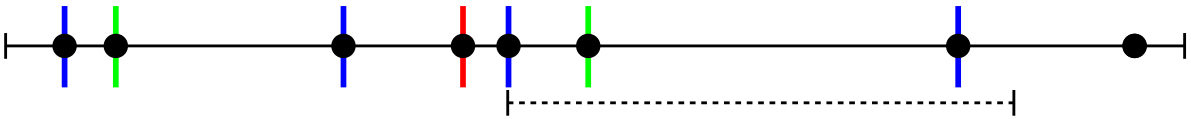
`NearestNeighbor`(v, p)

1. Sima keresés – az így talált pont az aktuális legjobb
2. Elindulunk felfele, és minden v' csúcsban:
   - 2.1 A hipersík másik oldalán lehet jobb pont?
     - 2.1.1 „Metszés" a pont körüli, aktuális legjobb sugarú hipergömbbel (egyszerű távolság-összehasonlítás a felosztás irányában)
   - 2.2 Lehet $\Rightarrow$ ha a `left`(v')-ből jöttünk, az aktuális legjobbat össze kell vetni a `NearestNeighbor`(`right`$(v'), p$) eredményével (ha a `right`(v')-ből jöttünk, akkor fordítva)

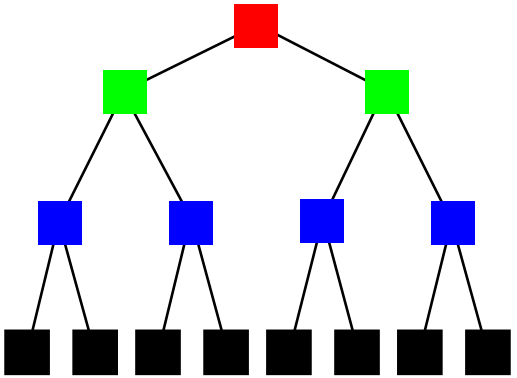
- k-NN feladathoz számon kell tartani a k legjobb pontot
  - Addig nem lehet eldobni egy ágat, amíg nincs min. k pontunk
- Átlagos esetben $O(\log n)$, legrosszabb esetben $O(n^{1-1/d} \cdot d)$
  - Magas dimenziószámnál alig jobb, mint a lineáris

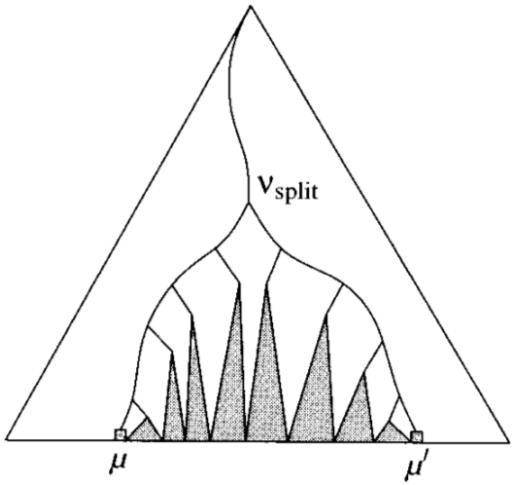
Intervallum-fák (Bentley, 1979)

- Régió-keresés gyorsabb: $O(\log^2 n + k)$
- 1D keresés: T bináris fa (medián alapján)

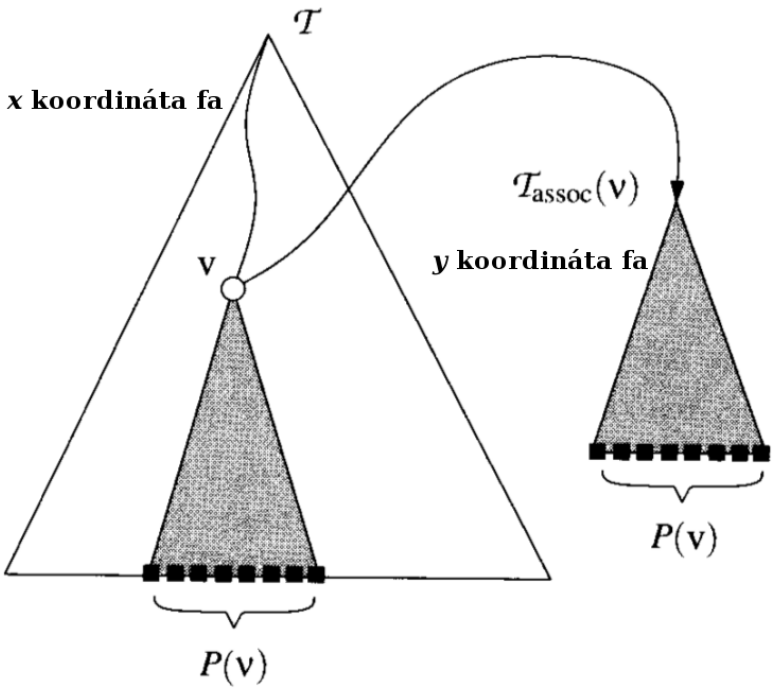


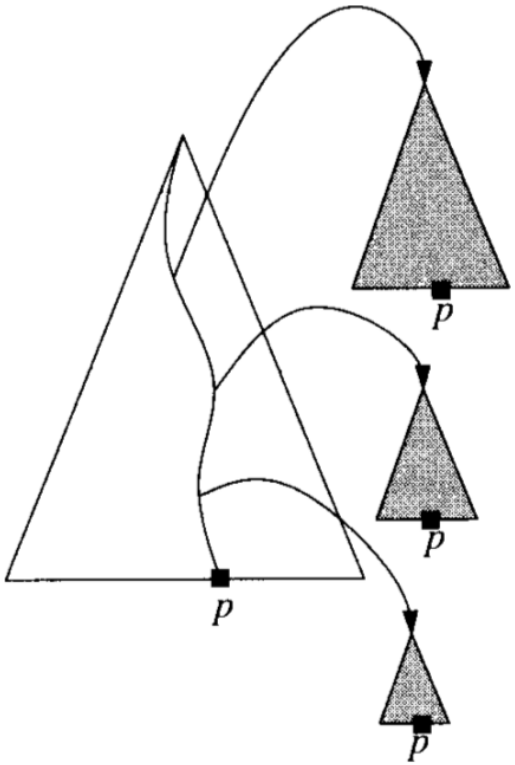
- $[x, x']$ intervallumba eső pontok?
  - Elindulunk lefelé az x-el és x'-vel
  - v_{split}: ahol szétválik a kettő
  - x (x') minden balra (jobbra) lépésénél kiírjuk a pontokat a jobb (bal) részfában
  - Ellenőrizzük a végső μ ill. μ' pontokat is
- $P(v)$ a v részfában levő összes pont
- Minden belső v csúcsban tároljuk $P(v)$-t egy T_{assoc} y koord. szerinti fában





Nagyobb memóriaigény: $O(n \log n)$

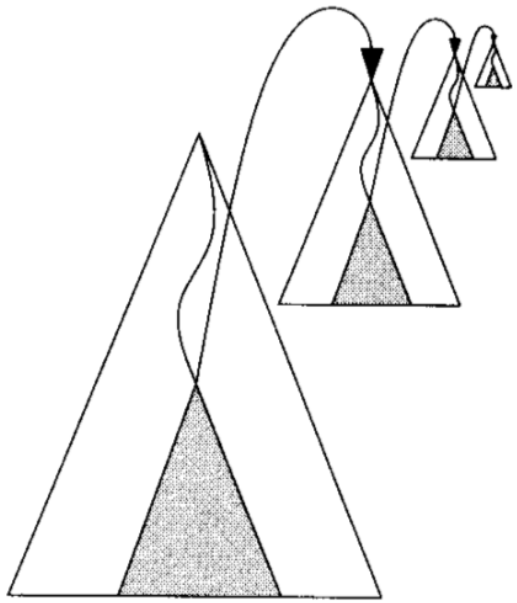




- Minden részfa lineárisan tárolható
- Egy p pontot csak a T-ben hozzá vezető úthoz asszociált T_{assoc}-okban tárolunk

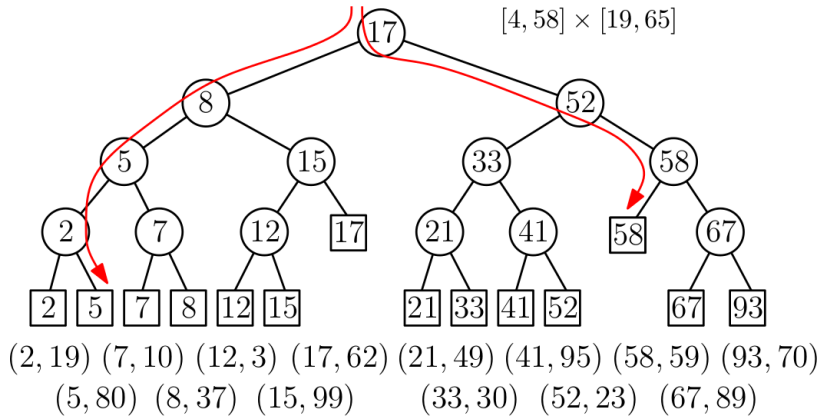
Magasabb dimenziók

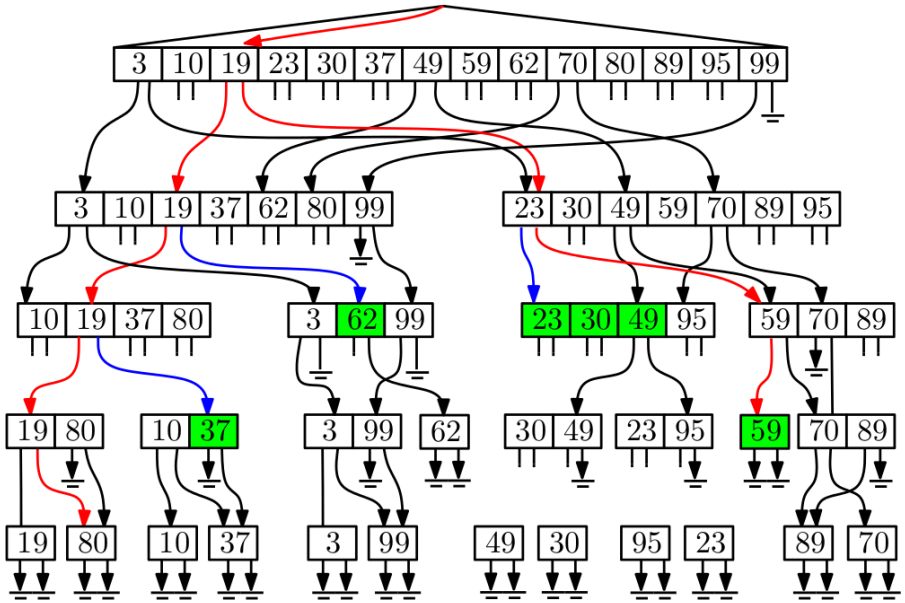
- Kiterjeszthető magasabb dimenziókra
  - 3D: x-fán belül y-fa, amin belül z-fa
- Készítés: $O(n \log^{d-1} n)$
- Keresés: $O(\log^d n + k)$
- Memóriaigény: $O(n \log^{d-1} n)$
- Réteges intervallum-fa:
  - Lueker/Willard (1978) [„cascading"]
  - Legbelső keresések újrafelhasználása
  - T_{assoc}-ok elemeiből 2-2 mutató a gyerekekben levő $\geq$ elemre
  - Keresés: $O(\log^{d-1} n + k)$
  - Memória: $O\left(n(\log n / \log\log n)^{d-1}\right)$



Javított réteges intervallum-fa (Chazelle, 1986/1990)

- „Fractional cascading”
- Memóriaigény optimális
  - $O(n \log^{d-1} n)$
- Legbelső T_{assoc}-ok tömbök
  - Kivéve a gyökéré
  - Utolsó koordináta szerint rendezett
- Az elemekben mutatók a gyerekek $\geq$ elemeire
- T_{assoc}-ban követjük az utolsóelőtti szinten megtett lépéseket

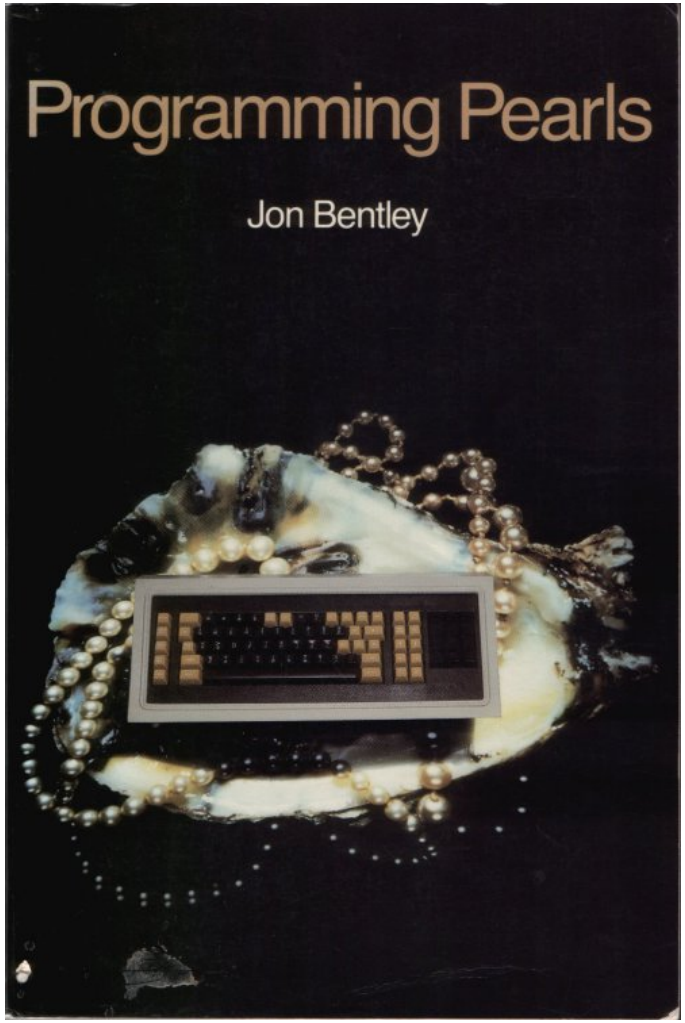




Ábra forrása: Marc van Kreveld
http://www.cs.uu.nl/docs/vakken/ga/

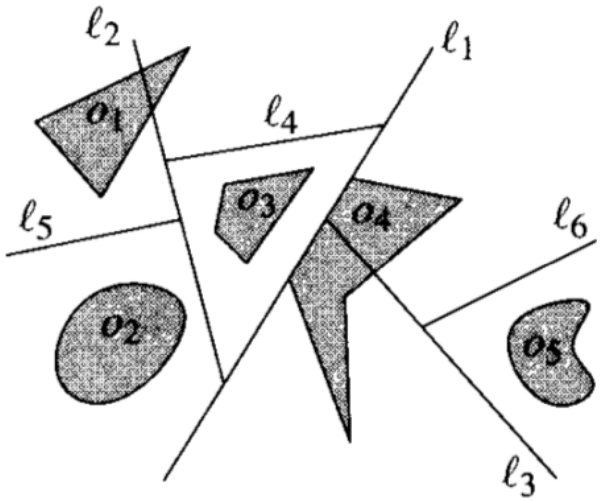
Kitérő – Jon Louis Bentley

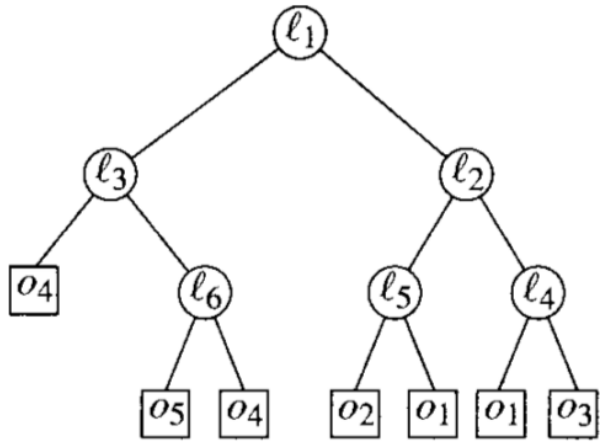
- Láttuk tőle:
  - Összes metszéspont keresése
  - Quadtree
  - *k*d-fa
  - Intervallum-fa
- CMU, Bell Labs
- *Dr. Dobb's Excellence in Programming* díj
- Nagyszerű könyvek:
  - Programming Pearls (1986)
  - More Programming Pearls (1988)



Bináris térfelosztás (Fuchs et al., 1980)

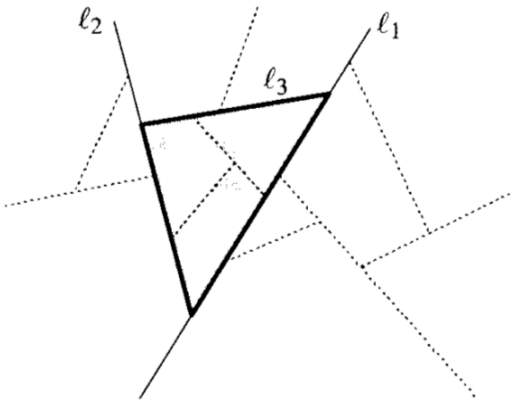
- A *k*d-fák általánosítása
- Tetszőleges irányú hipersíkok
- Alkalmazások:
  - Grafika (renderelés, árnyékszámítás ⇒ Doom, Quake)
  - CSG (Boolean-operációk poliédereken)
  - Robotika (ütközés-ellenőrzés)

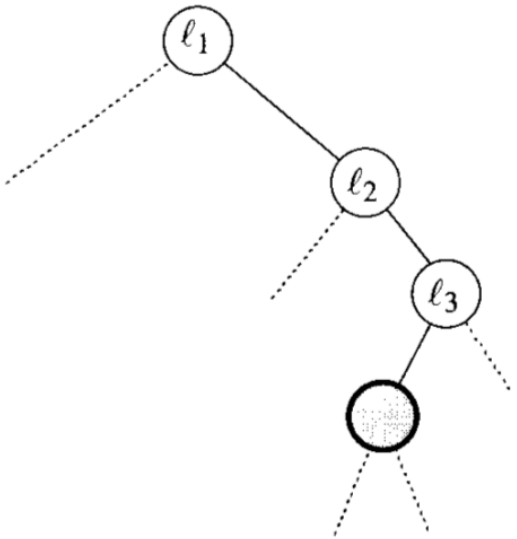




Részletek

- Belső csúcsokban hipersíkon belüli elemek
- Levelek konvex, nyílt régiók
- Hipersík: $h : a_1 x_1 + \cdots + a_d x_d + a_{d+1} = 0$
- Félterek:
  $h^+ = \{x : a_1 x_1 + \cdots + a_d x_d + a_{d+1} > 0\}$
  $h^- = \{x : a_1 x_1 + \cdots + a_d x_d + a_{d+1} < 0\}$
- Pl. 2D-ben egyenesek: $l_1^+ \cap l_2^+ \cap l_3^- \Rightarrow$
- BSP-fa mérete: objektumdarabok száma
  - Csúcsok száma ennek lineáris függvénye
  - Minél kisebb fát szeretnénk
  - 2D: (várható értékben) $O(n \log n)$
  - 3D: (várható értékben) $O(n^2)$
  - Javítható? [ld. később]



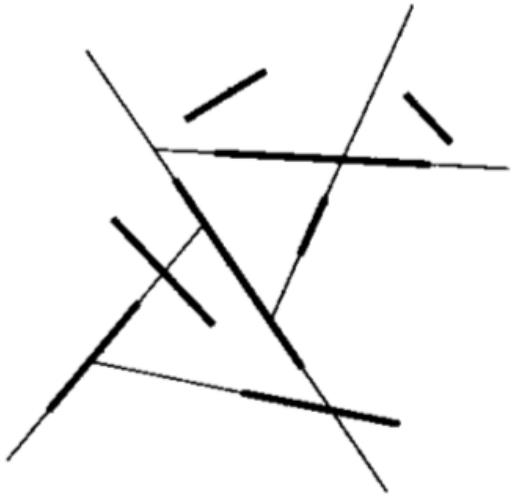


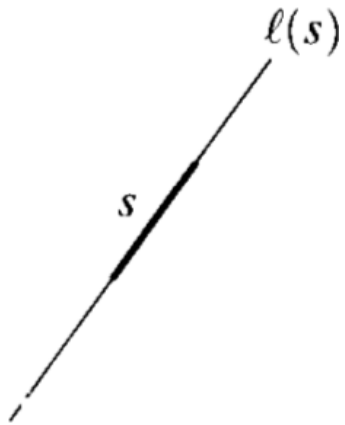
BSP-fa építése (2D) – $O(n^2 \log n)$

- Auto-partíció
  - Csak input éleket tartalmazó osztások
- Bemenet: $S = \{s_i\}$ szakaszok

$\texttt{2dBsp}(S)$

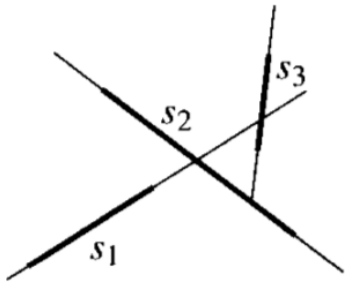
1. Ha $|S| \leq 1$
   1.1 Visszatérés egy levéllel
2. Legyen $l(s_1)$ az s_1-en átmenő egyenes
3. $S^+ = \{s \cap l(s_1)^+ : s \in S\}$
   $S^- = \{s \cap l(s_1)^- : s \in S\}$
4. Belső v csúcs készítése,
   $S(v) = \{s \in S : s \subset l(s_1)\}$,
   gyerekei $\texttt{2dBsp}(S^-)$ és $\texttt{2dBsp}(S^+)$

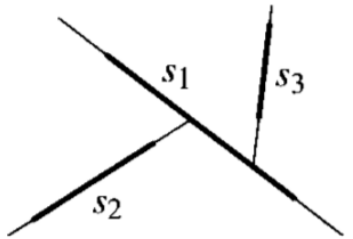


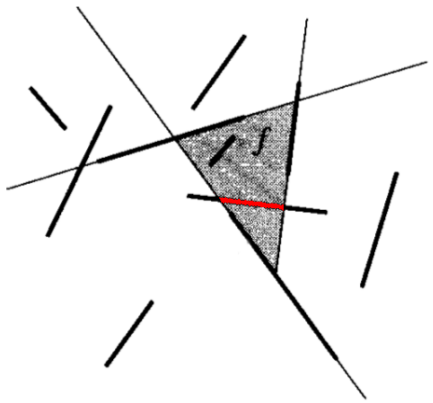


Ügyesebb szakaszválasztás

- Ötlet: legkevesebb metszésű szakasz
  - Nem mindig jó, és túl drága
- Jobb: véletlenszerű választás
  - Más sorrend más fát generál
- Ha van szakasz, ami végigmetsz egy régiót
  - „Free split” $\Rightarrow$ érdemes ezt választani
  - Biztosan nem növeli a darabok számát
- Ehhez minden szakasz(darab)hoz 2 boolean:
  - Végpontok osztóegyenesen vannak-e
  - Ha mindkettő `True`, akkor free split

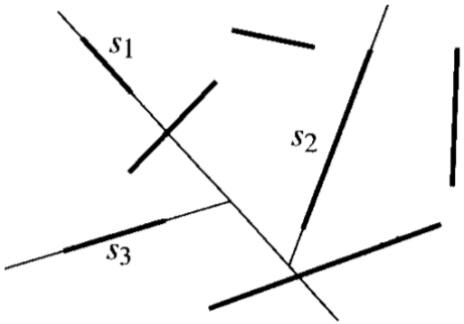


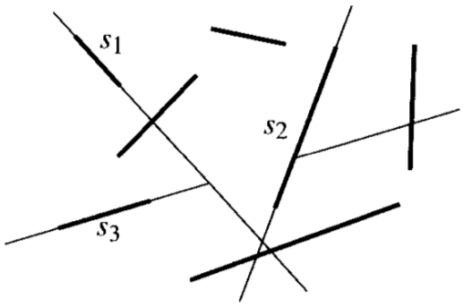


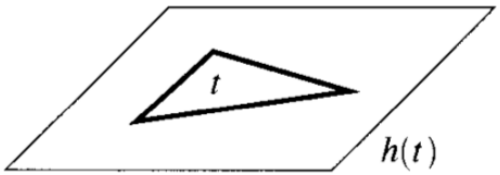


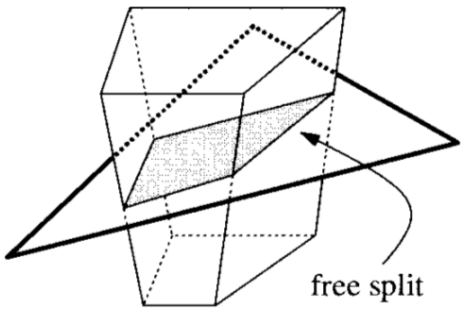
Módosítás & 3D kiterjesztés

- A hipersíkok végigvágnak minden régiót
  - Nem csak az adott régiót
  - Kivéve, ha a vágás üres régiót készítene
- Bonyolultabb kiszámítani
  - Nem egy sima rekurzív algoritmus
  - De 3D-ben kényelmesebb
- 3D-ben háromszögek a szakaszok helyett
  - Auto-partíció: a háromszögek síkjai
- Free split: a háromszög kettévág egy cellát



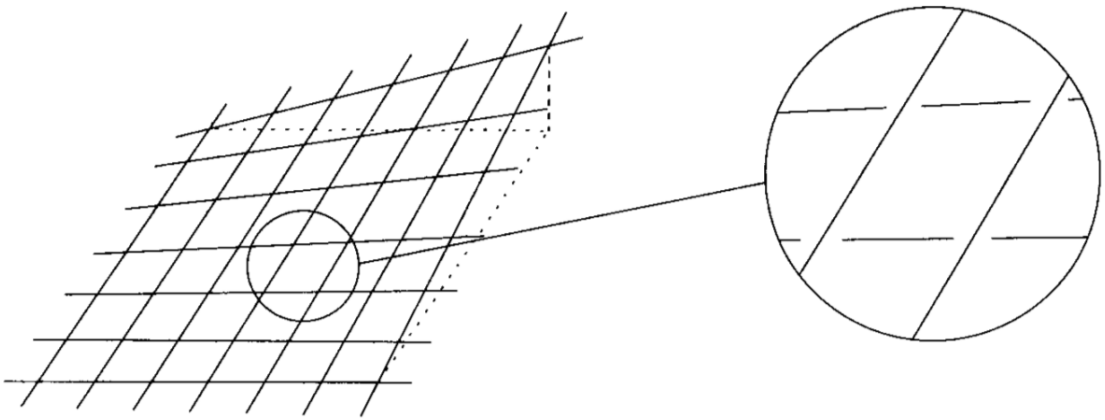






A BSP-fák méretéről

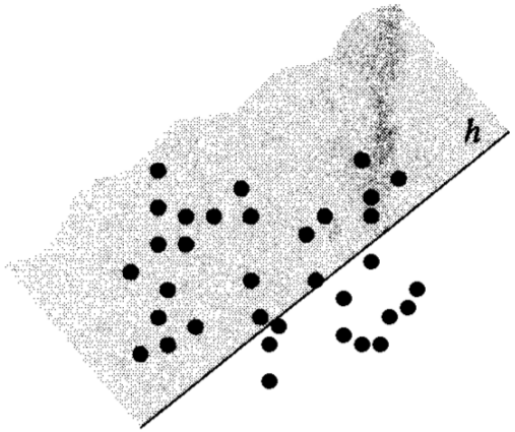
- 3D-ben $O(n^2)$-es az auto-particionáló algoritmus
  - Ez a legrosszabb esetre vonatkozik
  - Normális esetben sokkal jobban működik
- Belátható, hogy ez „optimális”
  - Van konfiguráció, amire minden BSP-fa $\Omega(n^2)$
    - Egy ferde (majdnem-)rácson minden „cella” egyik élét kell, hogy metssze egy hipersík a közelben
    - (Pontos bizonyítás hosszú és unalmas)
  - Nem az auto-particionálás hibája

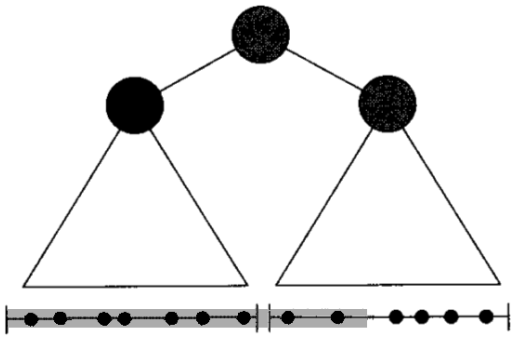


Feladat

$S = \{s_i\}$ pontok halmaza $\Rightarrow$ adott sokszögbe hány pont esik?

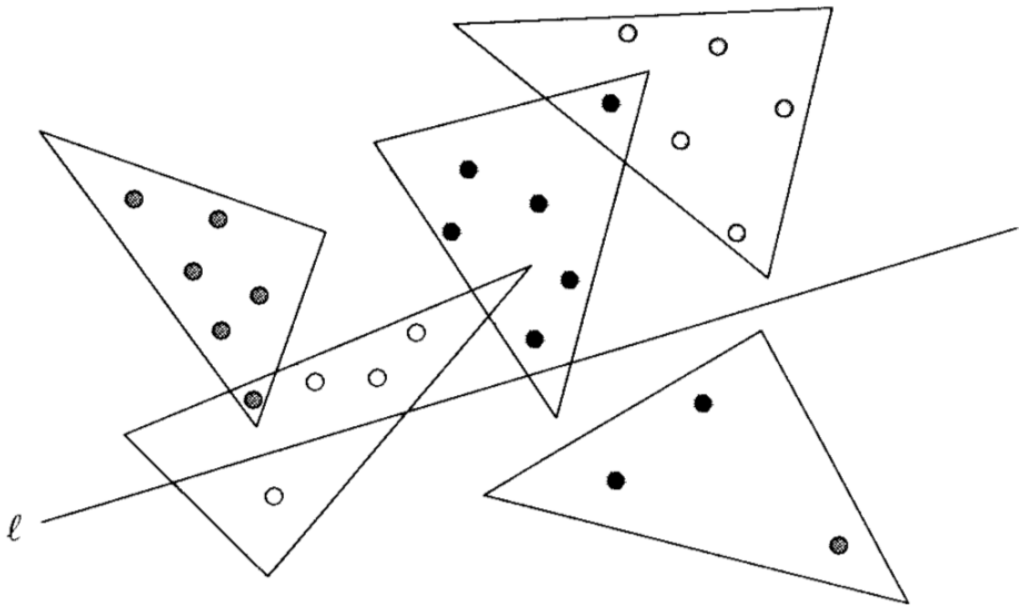
- Háromszögeljük a sokszöget
  - Elég háromszög-régiókra megoldani
- Először nézzük félterekre
- 1D: egy félegyenesbe eső pontok?
- Bináris keresőfa – $O(\log n)$
  - Csúcsokban a részfa pontjainak száma
  - Az egyik gyerek mindig egyszerű
    - Vagy teljesen benne van
    - Vagy egyáltalán nem
- 2D-ben nincs ilyen kettéosztás
  - Minden felosztásra van olyan féltér, ami mindkét oldalon részleges





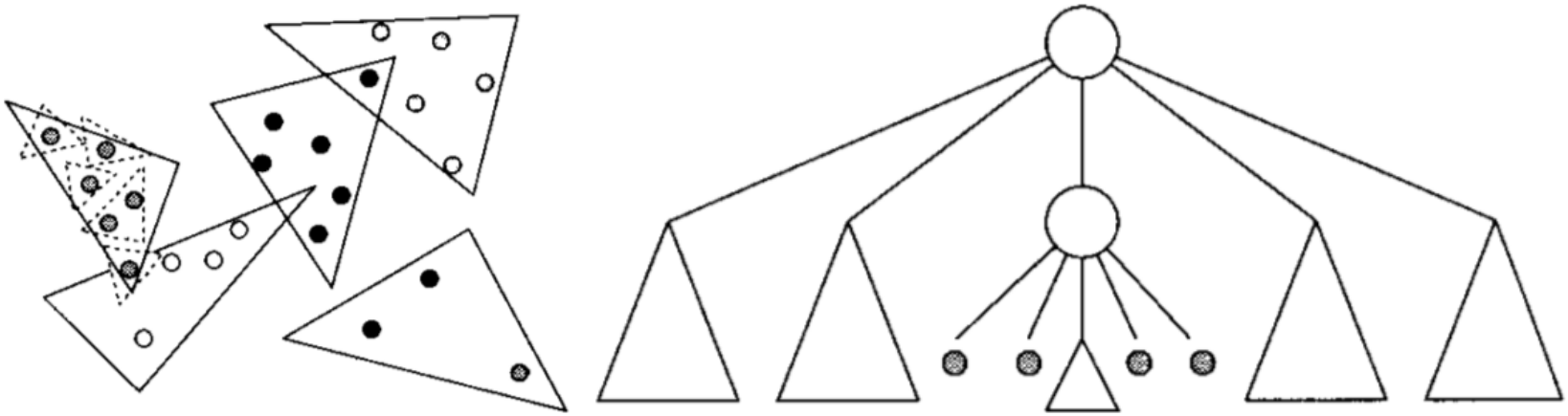
Szimplex-felosztás

- Ötlet: ne két részre osszunk, hanem többre!
  - Háromszögekre osztunk úgy, hogy $\cup_i S_i = S$ és $S_i \cap S_j = \emptyset$
  - Maguk a háromszögek nem feltétlenül diszjunktak
  - A felosztás „szép”, ha $|S_i| \leq 2n/r$,
    ahol r a háromszögek száma (a felosztás *mérete*)
  - Metszési szám: egy egyenes max. hány háromszöget metsz
  - Féltér-régiós keresésnél rekurzió csak a metsző háromszögeken



Partíciós fa (Willard, 1982)

- Tétel: n pont esetén tetszőleges $1 \leq r \leq n$-re létezik r méretű szép szimplex-felosztás $O(\sqrt{r})$ metszési számmal
  - A felosztás készítése $O(n^{1+\varepsilon})$, ahol $\varepsilon > 0$ tetszőleges
  - Inkrementálisan, súlyozott hipersíkokkal, ld. Matoušek (1992)
- Partíciós fa gyökerének r gyereke, azoknak is r gyereke stb.
  - Gyerekek sorrendje mindegy; v csúcshoz $t(v)$ a háromszög
  - Minden belső csúcshoz tároljuk a részfa pontjainak számát

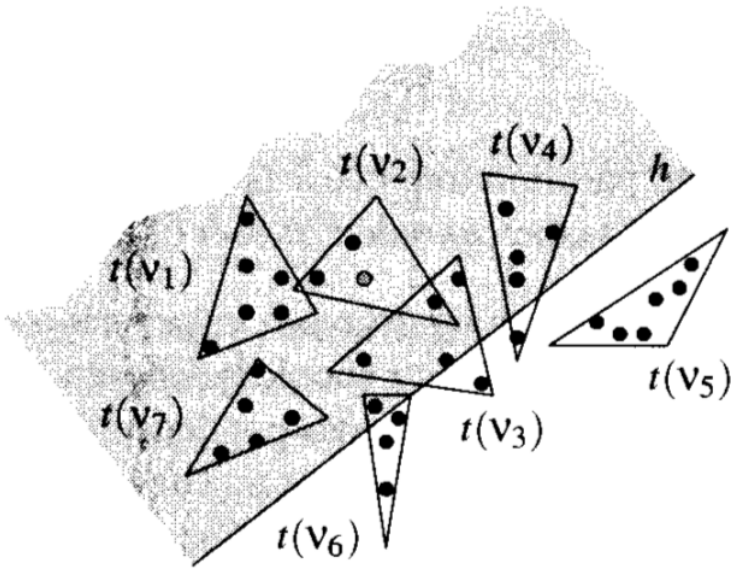


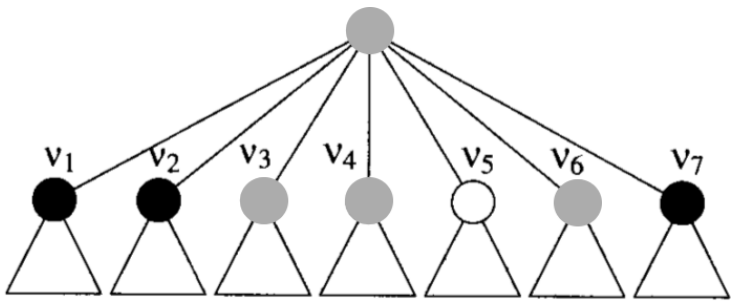
A h féltérbe eső pontok számlálása

- Kiválasztunk $V = \{v_i\}$ csúcsokat:
  - $v_i \subset h$
  - v_i minden μ ősére $\mu \not\subset h$
- Ekkor $S \cap h = \cup_i S(v_i)$
  - $S(v)$ a v alatti részfa pontjai

`SelectInHPlane`(h, T)

1. $V = \emptyset$
2. Ha $T = \mu$ levél:
   - 2.1 Ha h-ban van $\Rightarrow$ $V = \{\mu\}$
3. Kül. minden v gyerekre:
   - 3.1 Ha $t(v) \subset h \Rightarrow V \leftarrow v$
   - 3.2 Kül. ha $t(v) \cap h \neq \emptyset \Rightarrow$ $V \leftarrow$ `SelectInHPlane`(h, T_v)



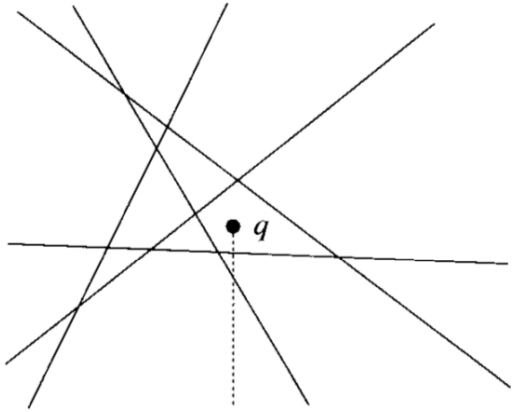


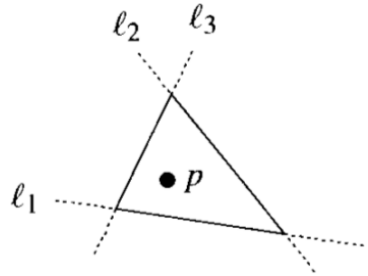
Komplexitás

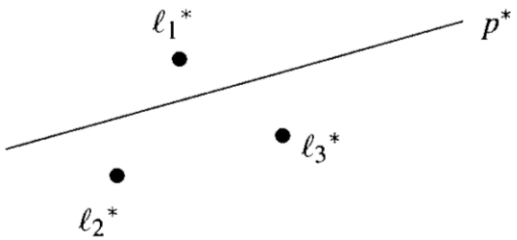
- A partícós fa mérete $O(n)$
- A kiválasztás műveletigénye $O(n^{1/2+\varepsilon})$, ahol $\varepsilon > 0$ tetszőleges
  - Ehhez $r = \lceil 2(c\sqrt{2})^{1/\varepsilon} \rceil$ kell, ahol c egy r-től és n-től független konstans úgy, hogy a metszési szám $< c\sqrt{r}$
  - Ha a pontokra is kíváncsiak vagyunk: $O(n^{1/2+\varepsilon} + k)$, ahol k a talált pontok száma
- Ha háromszög-régiókra nézzük, akkor is ugyanilyen bonyolult
  - Ugyanúgy működik, csak három féltérrel van teszt
  - Nagyobb r-et kell választani
- Az algoritmus könnyen átvihető más d dimenzióra is
  - Metszési szám: $O(r^{1-1/d})$
  - Műveletigény: $O(n^{1-1/d+\varepsilon})$

Alternatíva: vágó fa (Chazelle, 1993)

- Jobb műveletigény: $O(\log n)$
- Rosszabb méret: $O(n^2)$
- Féltér-keresés duális problémája:
  - Hány egyenes van a q pont alatt?
- Csak a q körüli laptól függ
  - A lapokra előre kiszámítható
  - Csak azt kell megtalálni, hogy q melyik laphoz tartozik
- Háromszög-régiós keresésnél?
  - Három pont felett/alatt levő egyenesek
  - Túl sok variáció, hogy előre számoljuk







Keresés háromszögeléssel

- Beháromszögeljük a duális síkot
  - Tétel: van olyan háromszögelés, hogy a metszési szám $< n/r$
  - Méret: $O(r^2)$, készítés: $O(nr)$
- Fa struktúra: minden $\triangle$-ben
  - Az alatta levő egyenesek száma
  - Rekurzív háromszögelés (csak a metsző egyenesekre)
- Kereséskor a q-t tartalmazó $\triangle$-ket kell csak végignézni
- $\triangle$-ben keresés $\Rightarrow$ 3 szintű fa
  - Hasonlóan az intervallum-fákhoz