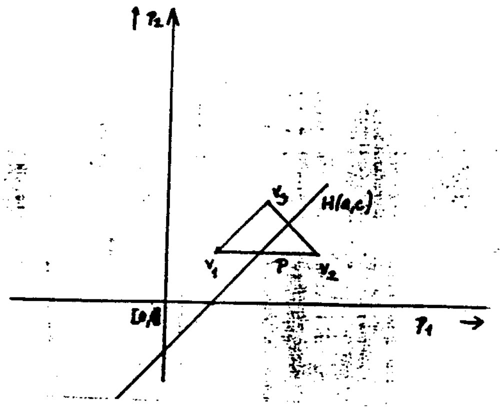
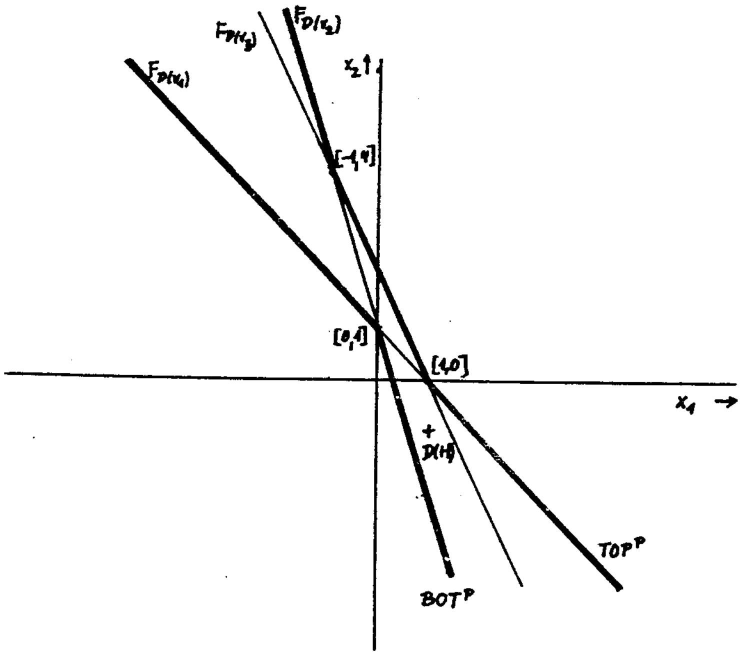
Obr.1 : Zadaný polygon P a přímka H



Obr.2 : Polygon P a přímka H po převodu do duálního prostoru



PREHĽAD VÝPOČTOVEJ GEOMETRIE

Roman Galbavý, Andrej Ferko, KAM MFF UK, 842 15 Bratislava

Kľúčové slová: výpočtová geometria, efektívne algoritmy, algoritmické paradigmy

Abstrakt.

Výpočtová geometria (computational geometry) završuje prvé desaťročie svojho prudkého rozvoja. V jej štruktúre sa stabilizovalo päť typov problémov: vyhľadávanie, konvexita, prieniky, proximita (Voronoiov diagram; zovšeobecnenia a aplikácie) a problémy na špeciálnej triede objektov (napr. geometria obdĺžnikov).

Konštrukcia efektívnych algoritmov na riešenie uvedených typov problémov sa líši jednak v algoritmických paradigmách resp. technikách, jednak podľa toho, či je vstup kompletný alebo dostupný postupne (on/line problem). Efektívnosť algoritmov sa hodnotí v štandardnom výpočtovom modeli.

1. Úvod

Existuje viacero možných predstáv o tom, čo je (alebo by mala byť) výpočtová geometria, od numericky orientovanej teórie splajnov, kriviek a povrchov, cez implementačné techniky tvorby softwaru v oblasti počítačovej grafiky, až po oblasť automatického dokazovania geometrických tvrdení. My budeme pod pojmom výpočtová geometria rozumieť disciplínu zaoberajúcu sa analýzou a návrhom efektívnych algoritmov na určovanie vlastností a vzťahov geometrických objektov (bod, priamka, mnohouholník, ...).

Cieľom tejto disciplíny je (v jej praktickej podobe) robiť veci premyslene, uvážene a v konečnom dôsledku efektívne (rýchlo a/alebo s malými pamäťovými nárokmi) a nie bezhlavo programovať prvý (ťažkopádny, pomalý a pamäťovo náročný) algoritmus, ktorý nás po zhliadnutí problému napadne.

Prostriedkami výpočtovej geometrie sú jednak (v teoretickej oblasti) aparát algebraickej, topologickej alebo kombinatorickej geometrie potrebný k tvorbe nástrojov na analýzu problémov, a jednak (v praktickej oblasti) techniky tvorby efektívnych algoritmov (dynamické programovanie, rozdeľuj a panuj, ...) a vhodné dátové štruktúry (vyvážené binárne stromy...), [AHU 76].

Popri teoretickej zaujímavosti mnohých problémov výpočtovej geometrie, ponúkané riešenia možno bezprostredne aplikovať v obrovskom množstve oblastí - databázové systémy, počítačová grafika, robotika, počítačové videnie, rozpoznávanie obrazcov atď.

Špecifickou črtou danej problematiky je pomerne široká prebádanosť riešení jednotlivých problémov v rovine a nárast zložitosti týchto problémov (a pomerne málo vedomostí o nich) pri prechode do priestoru resp. do ešte vyšších dimenzií.

Algoritmický problém vo výpočtovej geometrii predstavuje množinu geometrických úloh (zadaní) daného typu, ktoré možno riešiť daným algoritmom, ktorý v konečnom čase i pamäti zadanému vstupu jednoznačne priradí výstup, riešenie. Podľa kompletnosti vstupu rozoznávame problémy off-line (vstup zadaný naraz) a on-line (vstup zadávaný postupne).

[Typickým príkladom algoritmického problému je konštrukcia konvexného obalu N bodov v rovine. Ak sú body zadané naraz, ide o off-line problém - nájsť ich konvexný obal. Ak sa body postupne pridávajú a/alebo uberajú, ide o on-line problém - udržiavať konvexný obal.]

Riešiť algoritmický problém znamená nájsť riešenie problému v množine možných kandidátov (v stavovom priestore, v priestore prehľadávania). Triviálnym postupom je vyčerpávajúce prehľadanie (metóda hrubej sily), ktoré je zriedka efektívne, a preto ho možno použiť iba pre malé stavové priestory.

[Množina možných kandidátov pre konvexný obal N bodov v rovine sú napr. všetky konvexné mnohouholníky, ktoré obsahujú daných N bodov. Riešením je najmenší z nich. Hľadať ho vyčerpávajúcim prehľadaním (hrubou silou) nie je rozumné.]

Preto sa našlo niekoľko prístupov ku konštrukcii efektívnych algoritmov, ktoré sa nazývajú algoritmické paradigmy (alebo techniky). Efektívny algoritmus býva úspornejší ako metóda hrubej sily. Ak je pre daný problém známa dolná hranica zložitosti a niektorý algoritmus ju dosiahne, nazýva sa optimálnym algoritmom; takých je však dosiaľ známych veľmi málo.

"Vývoj v teórii zložitosti konkrétnych algoritmov viedol nielen k vzniku veľkého počtu algoritmov, ale i k vyšpecifikovaniu metód tvorby efektívnych algoritmov. Metódy, ktoré tu budeme prezentovať, bývajú úspešné pri tvorbe algoritmov pre navzájom sa podstatne líšiace výpočtové úlohy a sú dôvody predpokladať, že v sebe obsahujú hlbšie princípy pre efektívne riešenie úloh. Každopádne doterajšie praktické skúsenosti s návrhmi algoritmov poukazujú na to, že skôr ako sa navrhovateľ algoritmu pre riešenie

novej výpočtovej úlohy dá na hľadanie svojho originálneho návrhu oplatí sa mu vyskúšať, či niektorá z týchto metód nevedie k efektívnemu riešeniu danej úlohy," [Hrom92].

Paradigmám konštrukcie efektívnych algoritmov rôzni autori venujú pozornosť ako všeobecným nástrojom na konštrukciu algoritmov. Kurt Mehlhorn [Mehl84] im venuje zvláštnu kapitolu, Herbert Edelsbrunner [Edel86] appendix, Preparata so Shamosom [PrSh85] ich komentujú roztrúsene po texte svojej známej monografie, Mark H. Overmars a Emo Welzl v zborníku Sofsem [OvWe87] im venujú polovicu častí prehľadového článku (druhú venujú dátovým štruktúram). Počet paradigiem v týchto prameňoch nie je ustálený. V tomto článku z uvedených prameňov zhrnieme všetky.

Treba poznamenať, že podľa [Mehl84] sú dva základné spôsoby štrukturovania výkladu výpočtovej geometrie - orientované na problémy a orientované na paradigmy. Štrukturovanie podľa problémov dáva známe oblasti výpočtovej geometrie [PrSh85] - multidimenzionálne prehľadávanie, konvexné obaly, prieniky, problémy blízkosti (proximity) s centrálnym pojmom Voronoiovho diagramu, geometria obdĺžnikov. Výpočtová geometria od svojho vzniku v polovici 80. rokov tieto problémy postupne identifikuje, algoritmicky rieši a zlepšuje tieto algoritmické riešenia, pokiaľ možno až k nájdeniu optimálneho algoritmu.

Efektívnosť algoritmov sa vyjadruje v klasickom výpočtovom modeli so zaužívaným označením asymptotickej výpočtovej zložitosti. Túto konvenciu možno samozrejme opustiť, napr. ak pripustíme pojem približného algoritmu (approximation algorithm - rieši sa jednoduchší problém), alebo pravdepodobnostného algoritmu (probabilistic algorithm - použije sa silnejší výpočtový stroj). Týmto dvom možnostiam (ani paralelizmu vo výpočtovej geometrii) sa tu nebudeme venovať.

Pod pamäťovou zložitosťou algoritmu budeme rozumieť maximálny počet pamäťových miest, ktoré daný algoritmus bude v priebehu výpočtu používať. Nakoľko povačšine budeme pracovať s reálnymi číslami, pod pamäťovým miestom si môžeme predstaviť slovo konkrétneho počítača, v ktorom je tento schopný uchovať (v akejsi presnosti a v akomsi rozsahu) jedno reálne číslo.

Pod časovou zložitosťou budeme rozumieť počet základných operácií vykonaných daným algoritmom. Pod základné operácie môžeme zahrnúť:

- porovnanie dvoch reálnych čísel,
- súčet, rozdiel, súčin, podiel dvoch reálnych čísel,
- vykonanie goniometrickej funkcie nad reálnym číslom.

Čitateľ oboznámený s pojmom RAM si jednoducho môže predstaviť obdobný model, v ktorom každé jedno pamäťové miesto môžme použiť na zapamätanie si reálneho čísla, k základným inštrukciám pridáme goniometrické operácie a uvažujeme jednotkovú cenu jednotlivých inštrukcií.

Pri asymptotických odhadoch nás v podstate nezaujíma, či na vykonanie určitého kroku algoritmu (napr. zistenie prieniku dvoch úsečiek) potrebujeme jednu, dve alebo sto elementárnych operácií (podstatné je, že ich počet nezávisí od veľkosti vstupu). Preto ako o elementárnych operáciách môžme uvažovať o všetkých operáciách, ktoré vieme vykonať v konštantnom čase.

Niektoré základné pojmy, objekty a ich reprezentácia

Našim najzákladnejším objektom bude bod v rovine, resp. v d-rozmernom euklidovskom priestore, reprezentovaný ako dvojica (resp. usporiadaná d-tica) reálnych hodnot. Dvojicou bodov budeme reprezentovať úsečku , prípadne priamku. Horizontálne a vertikálne priamky možeme reprezentovať bodom s indikáciou príslušného smeru. Ďalším často používaným útvarom bude mnohouholník.

Mnohouholník je v rovine definovaný konečnou množinou úsečiek:

(i) Každý koncový bod úsečky je zdieľaný práve dvomi úsečkami.

(ii) Žiadna vlastná podmnožina úsečiek definujúcich mnohouholník nemá vlastnosť (i).

Úsečky budeme nazývať hranami a ich koncové body vrcholmi mnohouholníka. N-vrcholový mnohouholník nazveme N-uholník.

Mnohouholník je jednoduchý, ak žiadne dve nesusedné úsečky nemajú spoločný bod. Jednoduchý mnohouholník rozdeľuje rovinu na dve disjunktné oblasti; na vnútornú (ohraničenú) a vonkajšiu (neohraničenú), ktoré sú oddelené práve mnohouholníkom (Jordanova veta). (Pojem mnohouholník sa často používa aj na označenie vnútornej časti spolu s hranicou). Jednoduchý mnohouholník je konvexný, ak je jeho vnútorná časť konvexná množina.

Jednoduchý mnohouholník je hviezdicový, ak existuje bod z taký, že pre všetky body p mnohouholníka P úsečka zp celá leží v P. (Každý konvexný mnohouholník je hviezdicový). Množina bodov z majúcich uvedenú vlastnosť sa nazýva jadrom P. (Konvexný mnohouholník je totožný so svojim jadrom - každý bod z ležiaci v jeho vnútri alebo na jeho hranici má uvedenú vlastnosť).

N-uholník P budeme reprezentovať usporiadanou N-ticou $[v_1,\dots,v_N]$ jeho vrcholov tak, že každá úsečka $v_i v_{i+1}$ (uvažujeme sčítanie modulo N) je hranou P. U jednoduchého mnohouholníka má zmysel hovoriť o orientácii takéhoto uspo-riadania v smere (resp.proti smeru) hodinových ručičiek. Dohodnime sa, že jednoduchý mnohouholník budeme reprezentovať postupnosťou

$[v_1,\dots,v_n]$ jeho vrcholov orientovanou proti smeru hodinových ručičiek. Takto pre každú hranu $e=v_i v_{i+1}$ orientovanú z v_i do v_{i+1}, vnútro P leží vľavo od e.

Ďalším dôležitým pojmom pre nás bude pojem rovinného grafu. Graf G=(V,E) je rovinný, ak sa dá vnoriť do roviny tak, aby sa vnútra jeho hrán nepretínali. Každý rovinný graf sa dá do roviny vnoriť tak, že každá jeho hrana je úsečkou. Takéto vnorenie budeme ďalej nazývať RGPH (rovinný graf s priamymi hranami).

RGPH v rovine definuje rozklad roviny na oblasti. Nech v,e, a f označujú postupne počet vrcholov, hrán a oblastí (vrátane jednej neohraničenej oblasti). Tieto parametre sú zviazané známym vzťahom - Eulerovou formulou: v-e+f=2. Ak navyše v takomto grafe stupeň každého jeho vrchola je aspoň 3, tak:

$v \le \frac{2}{3}e, e \le 3v-6$

$e \le 3f-6, f \le \frac{2}{3}e$

$v \le 2f-4, f \le 2v-4$

Z hľadiska nášho cieľa (skúmanie zložitosti), je dôležité, že všetky tri parametre v, e, f sú navzájom proporcionálne.

Na reprezentáciu rovinného grafu s priamymi hranami použijeme tzv. DCEL štruktúru (doubly connected edge list) - dvojito pospájaný zoznam hrán.

Nech $V=\{v_1,\dots,v_N\}$ a $E=\{e_1,\dots,e_M\}$ sú vrcholy a hrany uvažovaného grafu G. DCEL štruktúra bude zoznam M prvkov - záznamov- jedno- jednoznačne zodpovedajúcich jdnotlivým hranám. T.j. každá hrana je reprezentovaná práve jedným záznamom. Jednotlivý záznam pre hranu e pozostáva zo štyroch polí: V1, V2, F1, F2 a dvoch smerníkov P1 a P2 s nasledovným významom:

V1 resp. V2: počiatočný resp. koncový vrchol hrany e;

F1 resp. F2: meno oblasti ležiacej vľavo resp. vpravo od e;

P1 resp. P2: smerník do DCEL na popis tej hrany incidentnej s V1 resp. V2, ktorá sa ako prvá vyskytne pri otáčaní e okolo V1 resp. V2 proti smeru hodinových ručičiek.

Pri práci s takouto štruktúrou nám často môžu byť užitočné polia HV[1..N] a HF[1..F] (N je počet vrcholov, F je počet oblastí grafu G) s významom:

HV[i] ukazuje do DCEL na nejakú hranu incidentnú s vrcholom v_i;

HF[j] ukazuje do DCEL na nejakú hranu tvoriacu hranicu oblasti f_j.

2. Zametacia technika (plane sweep)

je špecifickou metódou prístupu k riešeniu geometrických úloh. Používa v rovine (plane sweep) alebo v priestore (space sweep) s možným zovšeobecnením pre vyššie dimenzie. Základ tejto techniky ukážeme na konkrétnom príklade. Pre daný systém úsečiek v rovine máme zistiť všetky ich priesečníky. Uvažujme vertikálnu priamku l rozdeľujúcu rovinu na ľavú a pravú polovicu. Všimnime si, že

(i) Celkové riešenie problému je dané zjednotením riešenia v ľavej a riešenia v pravej časti roviny.

(ii) Riešenie zistené v ľavej časti roviny už nie je ovplyvniteľné úsečkami ležiacimi vpravo.

(iii) Dve úsečky môžu mať prienik, len ak existuje nejaká pozícia priamky l, pri ktorej sú priesečníky týchto úsečiek a priamky l susedné (v poradí výskytu na l).

Ak by sme vedeli vygenerovať <u>všetky</u> (nespočitateľne veľa) vertikálne rezy danou množinou úsečiek, určili by sme aj všetky prieniky týchto úsečiek. Našťastie z polôh sú zaujímavé len tie, pri ktorých sa "niečo podstatné mení" - a to poradie priesečníkov úsečiek s priamkou l (napr. v usporiadaní zhora nadol). Preto skúmame len také polohy priamky l,

1/ keď l prechádza koncovým bodom nejakej úsečky (vtedy táto úsečka buď "vchádza do alebo vypadáva z hry"), alebo

2/ keď l prechádza priesečníkom nejakých dvoch úsečiek (vtedy sa mení vzájomné poradie priesečníkov týchto úsečiek s l).

Týchto význačných pozícií typu 1 a 2 je len konečne veľa a výsledný algoritmus pracuje približne tak, že sa postupne prechádza zľava doprava po význačných pozíciách priamky l, v týchto pozíciách sa aktualizuje poradie priesečníkov úsečiek s priamkou l a pre úsečky so susednými priesečníkmi sa testuje, či nemajú spoločný prienik. Takýto prístup vedie k O(NlogN) algoritmu pre N vstupných úsečiek.

Predchádzajúci popis ukazuje princíp zametacej techniky v rovine: Vertikálna (horizontálna) priamka zametá rovinu zľava doprava (zhora dole) zastavujúc sa pritom v špecifických, tzv. významných bodoch. Prienik zametacej priamky s údajmi riešeného problému obsahuje úplnú informáciu pre generovanie časti výstupu a pre sposob ďalšieho pokračovania zametania. Takto máme dve základné štruktúry:

1/ Plán významných bodov, t.j. postupnosť bodov roviny usporiadaných podľa x-ovej súradnice (pre vertikálnu zametaciu priamku) zľava doprava. Tento plán netreba vopred stanoviť zo vstupu, môže sa dynamicky meniť v procese zametania. Priamka však nikdy necúva! (Z tohto hľadiska ide o metódu žravú - greedy).

2/ Stav zametacej priamky - zodpovedá popisu prieniku priamky so zametanou štruktúrou. Stav sa aktualizuje v každom významnom bode.

Príklad použitia ukazuje lokalizácia bodu v rovinnom grafe. Tento problém je jednou z dvoch úloh súhrnne označovaných pojmom geometrické vyhľadávanie. Prvá úloha - lokalizácia bodu - treba

určiť tú triedu daného rozkladu euklidovského priestoru, ktorá obsahuje zadaný bod. Druhá úloha - multidimenzionálne vyhľadávanie - spočíva v určení všetkých bodov danej množiny nachádzajúcich sa v istej zadanej oblasti viacrozmerného euklidovského priestoru (resp. v určení ich počtu). V oboch prípadoch ide o tzv. dotazovacie úlohy: daný je <u>súbor</u> prvkov a postupne zadávané <u>dotazy</u> (query) vzťahujúce sa k tomuto súboru.

Pre lokalizáciu bodu súbor predstavuje rozklad geometrického priestoru E^d na oblasti a dotaz bude reprezentovaný bodom $z \in E^d$. Treba určiť tú oblasť rozkladu, do ktorej patrí bod z.

Pre multidimenzionálne prehľadávanie súborom je množina bodov v E^d a dotazom bude oblasť R priestoru E^d. Treba vymenovať (resp. určiť počet) všetkých bodov súboru, patriacich do oblasti R.

Tieto dve úlohy sú v istom zmysle navzájom duálne. Kým v prvej úlohe k zadanému bodu hľadáme oblasť, do ktorej patrí, v úlohe druhej k zadanej oblasti hľadáme do nej patriace body.

Je užitočné uvedomiť si, že pokiaľ má byť v danej úlohe položený len <u>jediný</u> dotaz, nemá zmysel hlbšie sa zaoberať štruktúrou zadaného súboru a v oboch prípadoch nám neostáva nič iné, ako preskúmať vzťah dotazu ku <u>každému</u> prvku súboru (testovať každú oblasť, či neobsahuje bod z, resp. testovať každý bod, či sa nenachádza v R).

Situácia sa podstatne mení, ak bude dotazov na zadaný súbor viac. Vtedy môže byť výhodné vhodne preorganizovať (predspracovať) zadaný súbor do štruktúry umožňujúcej efek-tívnejšie zodpovedať jednotlivé dotazy. Takéto predspracovanie však niečo stojí. Vo všeobecnosti čím viac námahy si dá-me zo súborom pred započatím dotazovania naň, tým efektívnejšie je samotné dotazovanie a naopak, ak sa uspokojíme s "pomalším" odpovedaním na dotazy, nemusíme veľa pozornosti (času a pamäte) venovať predspracovaniu. Preto sa jednotlivé algoritmy prehľadávania hodnotia z hľadiska troch rozdielnych mier zložitosti:

1. čas zodpovedania dotazu:	koľko času algoritmus potrebuje na zodpovedanie jednotlivého dotazu
2. pamäť:	koľko pamäte potrebuje algoritmus na príslušnú údajovú štruktúru, umožňujúcu spracovávať dotazy
3. čas predspracovania:	koľko času treba na vytvorenie požadovanej údajovej štruktúry.

Ak by sme pripustili možnosť dynamickej zmeny súboru, pristúpila by štvrtá miera, čas potrebný na úpravu dátovej štruktúry pri vložení, resp. zrušení prvku súboru. V ďalšom sa však budeme zaoberať len statickými súbormi.

Špeciálne sa budeme zaoberať rovinným prípadom lokalizácie daného bodu do príslušnej oblasti E^2. Jednou z foriem rozkladu roviny je rozklad na oblasti definované RGPH a každá ohraničená oblasť takéhoto rozkladu je jednoduchý mnohouholník. V špeciálnom – triviálnom – prípade je rovina rozdelená jednoduchým mnohouholníkom $P=(p_1,p_2, \ldots, p_N)$ na dve oblasti – vnútornú (vzhľadom na P) a vonkajšiu. Problém lokalizácie v tomto prípade nazveme problémom inklúzie. Formulácie oboch problémov:

Problém INKLÚZIA PRE JEDNODUCHÝ MNOHOUHOLNÍK
Pre daný bod z a jednoduchý mnohouholník P rozhodnúť, či $z \in P$.

Problém LOKALIZÁCIA BODU
Pre zadaný RGPH graf G s oblasťami $R_1, \ldots R_n$ a bod z nájsť oblasť R_i takú, že $z \in R_i$.

Ak vieme napr. efektívne riešiť problém inklúzie a zostrojiť rozklad jednotlivých oblastí $R_1, \ldots R_n$ na mnohouholníky, máme nápad na prvý algoritmus pre lokalizáciu bodu. Ale samotná konštrukcia rozkladu oblastí $R_1, \ldots R_n$ môže byť netriviálna. Navyše, úspešnosť efektívnej lokalizácie bodu do oblasti spočíva v schopnosti rýchlo redukovať počet jednotlivých oblastí (kandidátov na odpoveď), ktoré môžu daný bod obsahovať a ktoré sa skutočne budú testovať. Najrýchlejšia známa metóda prehľadávania je bisekcia (binárne vyhľadávanie). Preto namiesto minimalizácie veľkosti prehľadávanej množiny, budeme sa snažiť organizovať ju do štruktúry, ktorá takéto vyhľadávanie umožní.

Tieto úvahy zhrnieme tak, že pre rozklad roviny definovaný zadaným RGPH grafom skonštruujeme nový rozklad s vlastnosťami:

(i) mnohouholníky nového rozkladu majú neprázdny prienik len s malým a fixovaným počtom oblastí pôvodného rozkladu (najlepšie s jednou oblasťou)

(ii) nový rozklad definuje množinu oblastí spôsobom, ktorý umožňuje jej binárne prehľadávanie.

Tak sa dostaneme ku jednoduchej metóde, tzv. metóde pásov. Pre zadaný RGPH G uvažujme horizontálne priamky prechádzajúce jednotlivými jeho vrcholmi. Tieto priamky vo všeobecnosti rozdelia rovinu na (N+1) (N je počet vrcholov G) horizontálnych pásov. Ak tieto pásy v rámci predspracovania utriedime podľa y-ových súradníc ich spodných okrajov, tak pri dotaze na bod z budeme v čase O(logN) vedieť určiť pás, v ktorom sa z nachádza. Teraz si pozornejšie všimnime štruktúru samotného pásu. Prienik pásu s grafom G pozostáva z (častí) hrán G, ktoré v danom páse definujú lichobežníky (prípadne degenerované na trojuholníky). Keďže G je rovinným vnorením rovinného grafu, jeho hrany sa pretínajú len vo

vrcholoch a keďže práve vrcholy definujú hranice pásov, žiadne dve hrany G sa nepretínajú vo vnútri pásu. Toto dovoľuje v každom páse usporiadať hrany (resp. ich časti) zľava doprava a tým pre daný pás v čase O(logN) určiť lichobežník, do ktorého patrí zadaný bod z. Uvedomme si, že výsledkom takéhoto prehľadávania v páse sú v danom páse susedné hrany e, e' (resp. len jedna z nich, ak z leží v neohraničenej oblasti) také, že z leží vpravo od e a vľavo od e'. Ak si spomenieme, že v reprezentácii DCEL, v ktorej sme dostali graf G zadaný, je ku každej hrane uvedená aj oblasť naľavo resp. napravo od nej, vidíme, že detekciou e, resp. e' sme zároveň zodpovedali pôvodnú úlohu – určiť oblasť grafu G obsahujúcu z. Vidíme teda, že pre daný bod z vieme

(i) vyhľadať pás obsahujúci z;

(ii) v tomto páse lokalizovať z, sumárne v čase O(logN).

Teraz už len zostáva zistiť, ako a za akú cenu vieme vyrobiť príslušnú dátovú štruktúru vhodnú pre vyššie uvedenú metódu lokalizácie. Ak sa rozhodneme usporiadať hrany v každom páse nezávisle, vidíme, že spotrebujeme čas $O(N^2.\log N)$ (usporiadavame N pásov po O(N) častiach hrán). Dá sa ľahko dokázať, že uvedená metóda vyžaduje $O(N^2)$ pamäte.

Teraz ukážeme, ako sa dá čas predspracovania redukovať z $O(N^2.\log N)$ na $O(N^2)$. Všimnime si, že ak hrana grafu G prechádza viacerými pásmi, tak tieto pásy nasledujú postupne za sebou. Ďalej si všimnime, že susedné pásy sa "veľmi nelíšia", t.j. že postupnosti hrán v susedných pásoch sa "veľmi nelíšia". Predstavme si dva susedné pásy, ktorých hranicu definuje vrchol $v \in G$ a predpokladajme, že poznáme usporiadanú postupnosť hrán z nižšieho pásu. Ako vyzerá usporiadaná postupnosť pre vyšší pás? Dostaneme ju jednoducho tak, že z pôvodnej postupnosti vynecháme hrany vchádzajúce do v "zdola" a na príslušné (po sebe nasledujúce) miesta do postupnosti vložíme hrany vychádzajúce z v smerom "nahor". Dostali sme zametací algoritmus, vidíme (zametajme zdola nahor):

(i) (postupnosť významných bodov) postupnosť vrcholov G, usporiadaná podľa y-nových súradníc;

(ii) (stav zametacej priamky) zľava doprava usporiadaná postupnosť hrán daného pásu.

Stav zametacej priamky budeme reprezentovať vyváženým stromom (napr. 2-3 stromom), ktorý v logaritmickom čase umožní realizovať operácie VLOŽ a ZRUŠ (operácie vkladania a vyberania prvkov do (z) usporiadanej množiny) [AHU 76])

Práca zametacieho algoritmu sa tu skladá z dvoch druhov činností; z vkladania a rušenia hrán do (z) postupnosti reprezentujúcej stav zametania a z generovania výstupu – výslednej štruktúry hrán pre každý pás. Z Eulerovej teorémy vieme, že hrán v rovinnom grafe s N vrcholmi je O(N); každú z týchto hrán v čase O(logN) do stromu reprezentujúceho stav zametacej priamky práve raz vkladáme a práve raz ju z neho v čase O(logN) vypúšťame.

Táto činnosť zaberie sumárne čas O(N.logN). Tomuto času dominuje čas $O(N^2)$ generovania výstupu - (N-1) pásov po O(N) hranách (resp.ich častí), ktorý zároveň určuje časovú zložitosť celého predspracovania.

Pre formálny zápis tejto metódy bude pre nás výhodné predstaviť si hrany grafu G orientované zdola nahor. Vrcholy majme v poli VERTEX usporiadané podľa rastúcej y-ovej súradnice. B[i] je množina hrán vchádzajúcich do VERTEX[i] zdola usporiadaných proti smeru hodinových ručičiek. A[i] je množina hrán vychádzajúcich z VERTEX[i] smerom hore usporiadaná v smere hodinových ručičiek.

<u>ALGORITMUS</u> PREDSPRACOVANIE PRE LOKALIZÁCIU

<u>Vstup</u>: polia VERTEX, A, B
<u>Výstup</u>: prehľadávacie stromy pre jednotlivé pásy

```
Procedure PREDSPRACOVANIE PRE LOKALIZÁCIU
begin  L:=0;
  for i:=1 to N do
     begin ZRUŠ(B[i]);
         VLOŽ(A[i]);
         GENERUJ VÝSTUP (L)
     end
end
```

Predchádzajúce úvahy možno zhrnúť do vety:

<u>Veta</u> 1 Problém lokalizácie bodu pre rovinné rozdelenie definované N-vrcholovým RGPH grafom vieme riešiť v čase O(logN) s pamäťou $O(N^2)$ a časom predspracovania $O(N^2)$.

Aj keď uvedená metóda vykazuje optimálny čas zodpovedania dotazu, jej pamäťové nároky a čas predspracovania neuspokojujú. Exisutjú aj lepšie výsledky. [PrSh85] dáva trojicu lepších metód - reťazí, trojuholníkov a lichobežníkov, ktoré teoreticky i prakticky dokazujú existenciu optimálnej metódy s θ(logN) časom zodpovedania dotazu, θ(N) pamäťou a θ(N.logN) časom predspracovania.

Doplňme ešte výsledok pre problém duálny k problému lokalizácie bodu - MULTIDIMENZIONÁLNE PREHĽADÁVANIE. Súborom tu je konečná množina bodov d-rozmerného priestoru, dotazom je oblasť tohoto priestoru a odpoveďou je buď vymenovanie všetkých bodov súboru, ktoré sa v oblasti nachádzajú (report mode) alebo len ich

počet (count mode). Takáto formulácia problému je abstrakciou množstva dôležitých aplikácií často označovaných za "prehľadávanie podľa viacerých kľúčov" (multikey searching). Napr. osobné oddelenie podniku, ktoré chce zistiť, koľko zamestnancov (resp. ktorí zamestnanci) vo veku medzi 30-40 rokov zarába 2500 až 3000 Kčs mesačne. Popri tomto vykonštruovanom príklade existuje množstvo serióznych aplikácií podobného charakteru v geografii, štatistike, v oblasti automatizácie návrhu a podobne. Výsledok:

<u>Veta</u> 2 Pre počet dotazov k a d≥2 vieme problém multidimenzionálneho prehľadávania N-prvkovej množiny riešiť v čase $O(dN^{1-1/d}+k)$ použijúc θ(dN) pamäti a θ(dNlogN) času na predspracovanie.

3. Triedenie

Mnohé efektívne algoritmy sa zakladajú na využití triedenia, napr. podľa súradníc v x, y, alebo podľa uhla vzhľadom k danému bodu. Príkladom optimálneho algoritmu na konštrukciu konvexného obalu je Grahamov algoritmus; tu uvedieme jednu jeho modifikáciu v podaní podľa [OvWe87].

Jedným z kľúčových problémov výpočtovej geometrie je konštrukcia konvexného obalu (convex hull) konečnej množiny bodov. Pojem konvexného obalu CH(S) konečnej množiny S bodov roviny je prirodzený a ľahko pochopiteľný. Podľa definície je to najmenšia konvexná množina nad S. Nanešťastie, táto jednoduchá definícia nemá konštruktívny charakter.

<u>Problém</u> Konvexný obal: Pre danú N-prvkovú množinu S bodov skonštruujte jej konvexný obal, t.j. úplný popis hranice CH(S).

Pod úplným popisom sa nemyslí len vymenovanie hrán (resp. vrcholov) výsledného konvexného mnohouholníka, ale <u>aj ich usporiadanie</u>. Teda výsledkom ľubovoľného algoritmu na hľadanie CH(S) S nie je množina, ale <u>usporiadaná množina</u> vrcholov konvexného obalu - mnohouholníka. To nás vedie k poznatku, že každý algoritmus riešiaci problém CH(S) musí vediť triediť, a preto vyžaduje aspoň Ω(NlogN) operácií.

<u>Veta 3</u> Konvexný obal N bodov v rovine vieme nájsť v optimálnom čase θ(NlogN) použijúc θ(N) pamäti.

Dôkaz: Skonštruujeme nezávisle spodnú (L-Hull) a hornú (U-Hull) časť konvexného obalu. Utriedime body S podľa súradníc x. Nech l a r sú najľavejší a najpravejší vrchol množiny S, potom priamka lr rozdeľuje S na dve podmnožiny nad a pod lr. Konvexný obal pozostáva z dvoch častí UC a LC (upper chain, lower chain).

Ukážeme, ako nájsť UC. LC možno nájsť analogicky. Hľadáme konvexný reťazec (chain) z l do i, kde i je i-ty bod UC. V bode i skúmame uhol daný bodmi i-1, i, i+1. Ak je konvexný, pokračujeme pre i:=i+1. Inak bod i nepatrí do UC a možno ho vynechať. Vtedy sa vrátime a pokrašujeme pre i:=i-1. Takto v čase O(N) nájdeme horný konvexný reťazec UC.

Ak sme použili optimálne triedenie v čase O(NlogN), tak sme dosiahli spodnú hranicu pre celý algoritmus, Θ(NlogN). Dôkaz pamäťovej zložitosti algoritmu je triviálny.

Poznámka. Na hľadanie konvexného obalu sú aj iné algoritmy, napr. Jarvis march dosahuje čas O(Nk), kde k je počet vrcholov CH(S) (output sensitive method).

4. Rozdeľuj a panuj (divide and conquer, DIVIDE_ET_IMPERA)

Táto metóda je pravdepodobne najúspešnejšou metódou na tvorbu efektívnych algoritmov vôbec. Neformálne ju možno popísať nasledovne. Nech V je nejaká výpočtová úloha a nech pre každé n∈N V(n) označuje výpočtovú úlohu pre vstupy veľkosti n. Potom pre vhodne veľké n∈N metóda "rozdeľuj a panuj" pracuje v nasledovných troch krokoch:

1. Rozdeľ úlohu V(n) na k úloh $V(c_1),\dots V(c_k)$ pre $1 \le c_i \le n-1$ a $c_i \in N$ (t.j. V(n) sa rozdeľuje na k problémov toho istého druhu V ale menšej veľkosti).
2. Vyrieš úlohy $V(c_1),\dots,V(c_k)$.
3. Poskladaj riešenia úloh $V(c_1),\dots V(c_k)$ tak, aby výsledkom zloženia bolo riešenie úlohy V(n).

Zložitosť takto navrhnutého algoritmu je súčet zložitostí na riešenie úloh $V(c_1)$, $,V(c_n)$ + zložitosť rozdelenia + zložitosť zloženia riešenia pre V(n) z riešení menších úloh. Pretože úlohy $V(c_1),\dots V(c_n)$ riešime tou istou metódou pokiaľ nie sú triviálne (malej veľkosti, napr. jednotkovej) a $c_i < n$ pre všetky $1 \le i \le k-1$, $i \in N$ celkovú zložitosť algoritmu môžeme vyjadriť rekurentným vzťahom. Prednosťou metódy "divide et impera" je i vlastnosť, že pre algoritmy ňou navrhnuté sa pomerne jednoducho dokazuje korektnosť metódou indukcie. Metódu ukážeme opäť na CH(S).

QUICKHULL algoritmus rozdelí množinu S na dve podmnožiny S_1 a S_2 tak, že konvexný obal S je jednoducho zreťazením konvexných obalov S_1 a S_2 (ako usporiadaných postupností bodov). Iniciálne rozdelenie je dané priamkou prechádzajúcou bodmi l (bod s naj-

menšou x-vou súradnicou) a r (bod s najväčšou x-vou súradnicou). Nech $S^{(1)}$ resp. $S^{(2)}$ je množina bodov S nad, resp. pod touto priamkou. Všeobecný rekurzívny krok môžme popísať nasledovne (popíšeme postup len pre $S^{(1)}$, pre $S^{(2)}$ je analogický).

Z $S^{(1)}$ vyberieme bod h, ktorý maximalizuje plochu trojuholník (h, l, r). Ak by takýchto bodov bolo viac, vyberieme ten, ktorý maximalizuje veľkosť uhla (h, l, r).

V ďalej popisovanom algoritme budeme bod h hľadať funkciou H(s, l, r).

Teraz skonštruujeme dve priamky: L_1 orientovanú od l ku h a L_2 orientovanú od h ku r. Testovaním každého bodu z $S^{(1)}$ rozdelíme $S^{(1)}$ na 3 časti:

$S^{(1.1)}$ - množina tých bodov z $S^{(1)}$, ktoré ležia vľavo od L_1;
$S^{(1.2)}$ - množina tých bodov s $S^{(1)}$, ktoré ležia vľavo od L_2;
a množinu bodov ležiacich v trojuholníku (lrh), ktorú môžeme z ďalších úvah vylúčiť (prečo?).

Množiny $S^{(1.1)}$ a $S^{(1.2)}$ sú vstupom do ďalšej úrovne rekurzie a výsledný konvexný obal množiny $S^{(1)}$ je daný zreťazením konvexných obalov $S^{(1.1)}$ a $S^{(1.2)}$.

Vyššie uvedený postup teraz viac sformalizujeme. Funkcia H(S, l, r) vracia bod h, funkcia QUICKHULL(S, l, r) vracia usporiadaný zoznam bodov a '*' označuje zreťazenie zoznamov.

<u>Algoritmus</u> QUICKHULL

<u>Vstup</u>: konečná množina S bodov roviny
<u>Výstup</u>: konvexný obal CH(S) množiny S

```
Function QUICKHULL(S, l, r)
begin
if  (S={l,r})
 then  return(l, r);  (*konvexný obal 2 bodov je úsečka
 else
  begin
   h:=H(S, l, r);
   S(1):= body z S ležiace vľavo od lh;
   S(2):= body z S ležiace vľavo od hr;
   return QUICKHULL(S(1),l,h) * (QUICKHULL(S(1),h,r)-h)
  end
end

begin
  l:= najľavejší bod z S;
  r:= najpravejší bod z S;
```

$S^{(1)}$:= body vľavo od lr;

$S^{(2)}$:= body vpravo od lr

CH:= QUICKHULL($S^{(1)}$,l,r)*(QUICKHULL($S^{(2)}$,r,l)-r)

<u>end</u>

Ďalší algoritmus MERGEHULL rovnako ako QUICKHULL rieši daný problém rozložením množiny S ľubovoľne na podmnožiny S_1 a S_2, z ktorých každá obsahuje polovicu bodov z S. Rekurzívne skonštruujme ich konvexné obaly $CH(S_1)$ a $CH(S_2)$. Aký vzťah majú tieto ku konvexnému obalu CH(S) množiny S ?

Jednoduchý: $CH(S) = CH(S_1 \cup S_2) = CH(CH(S_1) \cup CH(S_2))$.

Na prvý pohľad nič povzbudzujúce. Ale uvedomme si, že $CH(S_1)$ a $CH(S_2)$ už sú konvexné mnohouholníky a nie len neusporiadané množiny bodov. Keďže ale konvexný obal zjednotenia dvoch konvexných mnohouholníkov s m a n vrcholmi vieme skonštruovať v čase O(m+n), tak je časová zložitosť MERGEHULL takisto $\Theta(N\log N)$.

5. Geometrické miesto bodov (locus approach)

Na ilustráciu locus approach sa budeme venovať veľmi zaujímavému objektu - Voronoiovmu diagramu, ktorý vo veľmi kompaktnej forme udržiava informáciu o vzájomnej vzdialenosti bodov v rovine (resp. v priestoroch vyšších dimenzií). Aj keď je tento objekt zaujímavý aj sám o sebe, jeho hlavný význam spočíva v tom, že predstavuje prostriedok k efektívnemu riešeniu mnohých na prvý pohľad značne sa líšiacich úloh:

<u>Problém</u> Najbližší pár

Pre daných N bodov rozhodnite, ktoré dva sú najbližšie.

<u>Problém</u> Všetci najbližší susedia

Pre daných N bodov v rovine ku každému nájdite najbližší.

<u>Problém</u> Triangulácia

Daných N bodov roviny pospájajte nepretínajúcimi sa úsečkami tak, aby každá oblasť ležiaca vo vnútri konvexného obalu týchto bodov bola trojuholník.

<u>Problém</u> Vyhľadávanie najbližšieho suseda

Pre daných N bodov roviny s dovoleným predspracovaním, ktorý z nich je najbližšie k zadanému bodu q (dotazu)?

O všetkých týchto problémoch možno vysloviť pozorovania. Napr. ako vyzerá riešenie. Pre najbližší pár ide o dva body, všetci najbližší susedia určujú reláciu na S, tj. orientovaný graf. Keďže triangulácia N bodov je planárny graf s N vrcholmi, má najviac 3N-6 hrán. Riešením musí byť aspoň zoznam týchto hrán.

Vyhľadávanie najbližšieho suseda súvisí s lokalizáciou, odpoveďou je aspoň jeden bod.

Akú možno očakávať zložitosť algoritmov na riešenie týchto problémov? Napr. pre obsluhu riadiacej veže na letisku sú (s istým zjednodušením) najviac ohrozené práve tie dve lietadlá, ktoré sú k sebe najbližšie. Treba preskúmať každú dvojicu bodov? To vedie k triviálnemu $O(N^2)$ algoritmu. V jednorozmernom prípade vieme skonštruovať algoritmus rýchlejší: ak vstupných N bodov usporiadame (v čase O(NlogN)), tak dva najbližšie nájdeme v čase O(N) - musia ležať vedľa seba.

Odpoveď na všetky tieto otázky dáva Voronoiov diagram. K zadaným bodom $p_1, \ldots p_N$ roviny budeme chcieť zostrojiť rozklad roviny na oblasti V(1),...V(N) tak, že oblasť V(i) bude množinou práve tých bodov roviny, ktoré sú k bodu p_i bližšie ako ku ktorémukoľvek inému bodu p_j.

<u>Problém</u> Voronoiov diagram Vor(S)

Pre danú N-prvkovú množinu S bodov roviny a pre každý bod $p_i \in S$ treba určiť oblasť tých bodov roviny, ktoré sú bližšie k p_i ako k ľubovoľnému inému bodu z S.

Treba poznamenať, že Vor(S) dáva také rozdelenie roviny na konvexné oblasti, že odpoveď na vyhľadávanie najbližšieho suseda je v nich konštantná. Vor(S) pre jeden bod je rovina, pre dva body dve polroviny, rozdelené osou úsečky, danej tými dvoma bodmi. Pre rad bodov na priamke je Vor(S) rad pásov, pre pravidelne rozložené body v rovine je Vor(S) teseláciou (dlaždičkovaním), napr. trojuholníkovým, štvorcovým, šesťuholníkovým. Vor(S) dáva geometrické miesta bodov najbližších k bodom S. Uvedieme výsledok:

<u>Veta</u> 4 Voronoiov diagram N prvkovej množiny bodov v rovine vieme skonštruovať v optimálnom čase $\Theta(N\log N)$.

Vor(S) sa dá vypočítať mnohými spôsobmi, niektoré z nich sú optimálne. Sila metódy geometrického miesta bodov dáva v tomto prípade optimálne algoritmy pre všetky horeuvedené problémy: predspracovanie dá Vor(S) a ďalej už možno v lineárnom čase získať riešenie všetkých horeuvedených piatich problémov.

6. Dualizácia

Mnohé riešenia úloh výpočtovej geometrie používajú transformácie, medzi ktorými vyniká dualizácia. Táto transformácia transformuje body d-rozmerného priestoru na nadroviny a naopak.

Napr. ak treba určiť, či sú v danej množine S tri body kolineárne, dualizáciou dostaneme úlohu, či z danej množiny priamok prechádzajú tri jedným bodom. Tento duálny problém možno riešiť lepšie, napr. s použitím procedúry na hľadanie prieniku úsečiek.

7. Kombinatorická analýza

Tento prístup zdôrazňuje [Edel86]. Príkladom tohoto prístupu môže byť pokračovanie v úvahe zhora: "Keďže triangulácia N bodov je planárny graf s N vrcholmi, má najviac 3N-6 hrán. Riešením musí byť aspoň zoznam týchto hrán." Všetkých možných hrán je $O(N^2)$. Ak hľadáme napr. akúkoľvek trianguláciu, môžeme ihneď napísať neefektívny algoritmus: vygenerujeme všetky hrany, usporiadame ich podľa rastúcej dĺžky, najkratšiu z nich akceptujeme a potom pridávame v rastúcom poradí ďalšie hrany, ak sa nepretínajú. Ak sme akceptovali 3N-6 hrán, stop. Ešte treba ošetriť možnosť, že ak sa vyprázdnil zoznam utriedených hrán, stop. Tento algoritmus sa nazýva "greedy triangulation" a popisuje ho [PrSh85], s. 228.

8. Prune and search

Tento prístup uvádza [Edel86]. Princípom je sústrediť sa nie na hľadané riešenie, ale na rýchle identifikovanie tých častí stavového priestoru, kde riešenie určite neleží. Tento prístup je vhodný pre prípady, keď objem výstupnej informácie je malý. Metódu prune and search môžeme ilustrovať na riešení už spomínaného problému multidimenzionálneho prehľadávania (Veta 2, s. 11).

<u>Problém</u> Multidimenzionálne prehľadávanie

Pre zadaný súbor S' a pre zadaný hyperpravouholník D vymenovať všetky prvky S ležiace v D.

Pre jednorozmerný prípad je súborom množina S - N bodov na osi x, dotazom je interval [x', x''], odpoveďou tie body z S, ktoré ležia v danom intervale. Vhodnou dátovou štruktúrou je prešitý binárny strom - vyvážený strom, ktorého listy sú prepojené súhlasne s usporiadaním prvkov množiny S. Každý vrchol stromu reprezentuje niektorý interval na osi x. Koreň stromu reprezentuje celú os x. Ak interval reprezentovaný vrcholom v obsahuje aspoň 2 body S, tak má 2 synov. Ľavý z nich reprezentuje podinterval obsahujúci "ľavú" polovicu vrcholov S, pravý reprezentuje podinterval obsahujúci "pravú" polovicu vrcholov S. Ak prehľadávaný interval (dotaz) je podintervalom iba jedného zo synov, proces prehľadávania sa druhým synom už nezaoberá.

Zovšeobecnením tohoto postupu pre vyššie dimenzie je tzv. metóda multidimenzionálneho binárneho stromu (k-D strom), o ktorej hovorí Veta 2.

Záver

Uvedený prehľad výpočtovej geometrie nie je a nemôže byť úplny. Nové problémy, prístupy i algoritmy sa stále objavujú. Aplikácií výpočtovej geometrie len v počítačovej grafike je neprehľadný počet. Vhodné je sledovať časopisy Algorithmica a Computational Geometry i každoročnú konferenciu ACM Computational Geometry, ktorá má už 8 ročníkov. Na MFF UK v Bratislave sa výpočtová geometria považuje za súčasť vzdelania absolventov počítačovej grafiky a postgraduálneho štúdia.

Literatúra

[AHU 76] Aho, A.V. - Hopcroft, J.E. - Ullman, J.D.: The Design and Analysis of Computer Algorithms, Addison-Wesley 1974

[Mehl84] Mehlhorn, K.: Data Structures and Algorithms 3: Multidimensional Searching and Computational Geometry, Springer 1984

[PrSh85] Preparata, F. - Shamos, M.J.: Computational Geometry, Springer 1985

[Edel86] Edelsbrunner, H.: Algorithms in Combinatorial Geometry, Springer 1987

[OvWe87] Overmars, M.H. - Welzl, E.: Basic Techniques in Computational Geometry, Sofsem 1987, pp. 189-214

[Hrom92] Hromkovič a kol.: Teória výpočtovej zložitosti, skriptum MFF UK (v tlači)