UNIVERZA V LJUBLJANI

Fakulteta za strojništvo

REKONSTRUKCIJA POVRŠIN

DIPLOMSKA NALOGA VISOKOŠOLSKEGA ŠTUDIJA

Aleš DROLC

Mentor: izr. prof. dr. Jože Duhovnik, dipl. inž.

Ljubljana, 1997

To diplomsko nalogo posvečam svojim staršem, ki so mi študij omogočili in mi ves čas stali ob strani.

D Tek. štev.: 4747 UDK 519.68

REKONSTRUKCIJA POVRŠIN

Aleš Drolc

Ključne besede: Računska geometrija, računalniška grafika, rekonstrukcija površin, zajem prostorskih točk, Delaunayeva triangulacija, Voronoi diagram, konveksna lupina, nekonveksna lupina.

Izvleček: Diplomska naloga predstavlja mehanski sistem za zajem prostorskih točk in metodo za rekonstrukcijo površine iz razpršenih podatkov. V prvem delu so predstavljene teoretične osnove računske geometrije in osnovni algoritmi, v drugem pa mehanski sistem za zajem prostorskih točk, z opisom povezave z računalnikom, in lgoritem za rekonstrukcijo površine, ki ni nujno konveksna.

D Tek. štev.: 4747 UDC 519.68

SURFACE RECONSTRUCTION

Aleš Drolc

Key words: Computational geometry, computer graphics, surface reconstruction, canning of 3D points, Delaunay triangulation, Voronoi diagram, convex hull, nonconvex hull.

Abstract: In the thesis a mechanical system for scanning 3D points and a method for surface reconstruction from scattered data are presented. In the first part theoretical rounds of computational geometry and some ground applications are described. In the second part a mechanical system for scannin 3D points with description of connection between the system and the computer and the application for surface reconstruction, which is not necessary convex, are presented

Kazalo

1 Uvod

Pri proučevanju različnih dogodkov, pojavov, lastnosti elementov, ipd., si je velikokrat pot do rezultata mogoče olajšati s pomočjo različnih analiz ali rekonstrukcij. Podatke, ki jih je mogoče zbrati na različne načine, je potrebno po določenih zakonih ali predpisih urediti. Podatki so lahko:

- temperature, izmerjene s temperaturnim senzorjem na različnih mestih,
- temperature, izmerjene s temperaturnim senzorjem na istem mestu v časovnih presledkih,
- izmerjeni ekonomskosocialni kazalci,
- izmerjeni pretoki skozi cevovode,
- točke, določene z odčitavanjem površine, itd [9].

Za vse zgoraj naštete podatke (in tudi ostale neomenjene) velja, da so le redko zbrani v neko urejeno obliko, ki bi že sama po sebi ponujala rešitev, ampak jih je potrebno v urejeno obliko šele spraviti. To velja tudi za podatke, ki se jih dobi z odčitavanjem površine. Ti podatki so ponavadi neurejeni in razpršeni.
Podatki, na podlagi katerih je potrebno narediti rekonstrukcijo ploskve, so lahko dobljeni na različne načine: z mehanskim sistemom za odčitavanje 3D točk, z laserskim 3D skenerjem, s pomočjo ultrazvoka, itd. Te točke so lahko razpršene po ravnini (2D ali ravninske), ali pa so razpršene v prostoru (3D ali prostorske).

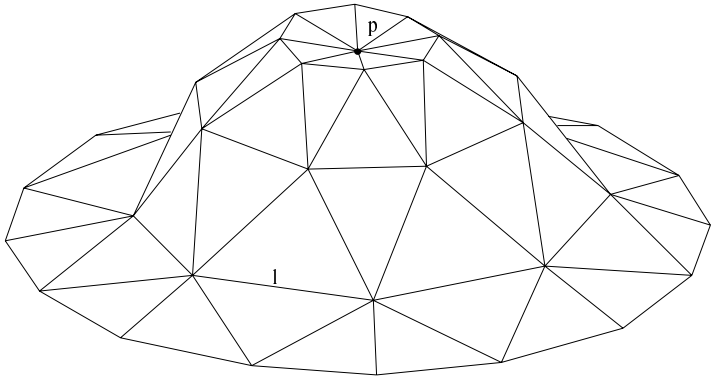
Namen je torej, da bi bilo iz vhodnih neurejenih podatkov, ki so lahko ravninski ali prostorski, mogoče sklepati o geometriji in topologiji površine, oz. z načrtnim odčitavanjem točk zagotoviti digitaliziran model površine ali modela, ki bi ga bilo mogoče uporabiti pri analiziranju ali pri modeliranju večjih, bolj zahtevnih oblik. Pri tem je ena možnost način, pri katerem se ugotavljajo odnosi med podatki, rezultat pa je popis površine s funkcijo (v odvisnosti od kompleksnosti površine) in drugi, ki temelji na različnih algoritmih računske geometrije, in točke v prostoru povezuje v "dobre" trikotnike, te pa naprej v optimalno trikotniško mrežo.

V nadaljevanju je pozornost posvečena drugi možnosti, t.j. z uporabo različnih algoritmov računske geometrije izdelati algoritem, ki bo sposoben rekonstrukcije površine, ki ne bo nujno konveksna.

Za pridobivanje razpršenih podatkov je bil izbran mehanski sistem za odčitavanje 3D točk. Pri tem načinu odčitavanja je največji problem natančnost sistema. Za opisan primer je bilo odločeno, da je poudarek na prikazu postopka odčitavanja in ne na natančnosti sistema. Sistem je sestavljen iz treh palic in treh potenciometrov v treh vrtiščih. Potenciometri odčitavajo relativni kot med posameznimi palicami, kar zadošča za določitev koordinat točke v prostoru. Z uporabo optičnih enkoderjev v zgibih in po naročilu izdelanih elementov, bi se natančnost bistveno povečala, vendar bi se s tem sistem podražil.

V računski geometriji vsa teorija temelji na konveksnosti. Tako na stopnji lupin na ravnini, kot tudi kasneje pri konstruiranju lupin v prostoru. Večina algoritmov, ki gotovo rešujejo probleme pri nedvoumni konveksnosti, pri konstrukciji ali rekonstrukciji površine, ki ni nujno konveksna, v popolnosti odpove. Tako je bilo potrebno za rešitev združiti različne načine reševanja in ideje, ki jih uporabljajo različni algoritmi, pri tem pa postaviti tudi nekaj omejitev, ki jih na tej stopnji ni mogoče preseči.

Algoritmi za določanje konveksnih lupin temeljijo na ovijanju (deterministični način) ali na dodajanju posameznih točk in določanju, ali leži točka znotraj ali zunaj lupine in v odvisnosti od tega popravljanju lupine (naključen način). Ta dva načina konkavnosti ne prepoznavata; vsako konkavno teme postavita v notranjost lupine. Na Sliki 1.1 je prikazano konkavno področje površine. Zgoraj omenjena načina točke *p* ne bi prepoznala kot del lupin, ampak bi jo postavila v njeno notranjost. Konkavna so tudi vsa temena na robu *l*. Algoritem, ki bo sposoben prepoznavanja konkavnosti, bo v principu podoben načinu ovijanja, vendar bo hkrati upošteval odnose med točkami, kar popisujeta Voronoi diagram in Delaunayeva triangulacija.



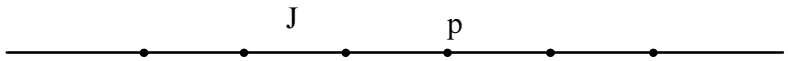
Slika 1.1 Konkavno področje površine je v točki *p* in po robu *l*.

Lupino je potrebno glede na točke v prostoru oblikovati iz površinskih elemetnov. To so lahko trikotniki, štirikotniki ali kakšni drugi konveksni mnogokotniki. V nadaljevanju je prikazan postopek, ki lupino v prostoru oblikuje iz optimalnih trikotnikov. Tako razdelitev omogoča Delaunayeva triangulacija, ki zagotovi maksimiziranje najmanjših kotov (Slika 1.1).

V naslednjem poglavju je predstavljenih nekaj pogledov, ki so v zgodovini vodili, da se je geometrija kot taka sploh rodila in se kasneje pod različnimi vplivi razvijala do nivoja, na kakršnem je danes. Z uveljavitvijo računalnikov se je v sklopu geometrije razvila nova veja, t.i. računska geometrija, ki si pri reševanju geometrijskih problemov pomaga s posebnimi podatkovnimi strukturami in z različnimi metodami reševanja (algoritmi) s pomočjo računalnika.

Splošno gledano je računska geometrija proučevanje algoritmov, s pomočjo katerih se rešujejo geometrijski problemi na računalniku.
Pri ožjem pojmovanju računske geometrije je ta opredeljena kot proces iskanja in razvrščanja preko določene množice elementov. Najpreprostejši primer je razvrščanje enodimenzionalnih točk po koordinatnih vrednostih. Naj bo niz N sestavljen iz n elementov na realni osi R. Pri tem naj R predstavlja enodimenzionalen prostor R^1. Naloga je razvrstiti elemente po njihovih koordinatah. Temu ustreza (Slika 1.2):

Problem razvrščanja:
Potrebno je poiskati ureditev $H(N)$ na R, ki bo oblikovana iz danega niza točk N in bo zadostila zahtevanim pogojem [7].

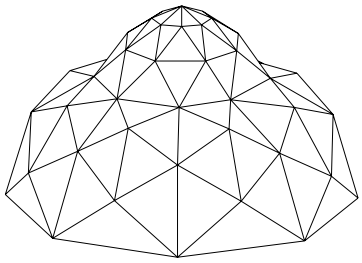


Slika 1.2 Razvrstitev $H(N)$ predstavlja ureditev točk p_i glede na koordinatno vrednost.

S problemom razvrščanja se povezuje problem iskanja.

Problem iskanja:
Potrebno je združiti ureditev $H'(N)$ z ureditvijo $H(N)$ tako, da je pri katerikoli dani točki $q \in R$ mogoče določiti interval v $H(N)$, ki to točko vsebuje.

V primeru prostorske triangulacije je ureditev $H(N)$ triangualcija $T(N)$ preko danega niza točk v R^3. Ureditev $T(N)$ enaka $D(N)$, t.j. Delaunayevi triangulaciji, ki je najboljša možna razdelitev, saj maksimizira najmanjše kote trikotnikov (Slika 1.3).



Slika 1.3 $D(N)$; $n = 51$ niza N v R^3.

Pri dinamičnih procesih se niz N spreminja z dodajanjem oziroma odvzemanjem točk oz. katerihkoli drugih elementov. Tu je potrebno hitro prilagajanje ureditev $H(N)$ in $H'(N)$ med vsakim dejanjem dodajanja ali odvzemanja.

Problema razvrščanja in iskanja se lahko združita: dan je niz N, sestavljen iz n elementov na osi R. Oblikovati je mogoče ureditvi $H(N)$ in $H'(N)$. S pomočjo slednje je mogoče določiti položaj katerekoli točke v $H(N)$.

Razvrščanje točk na osi R je najpreprostejši primer računske geometrje. Pri premiku v višje dimenzije, niz elementov N ni več niz točk, temveč niz linijskih segmentov, ploskev, polprostorov, itd., v R^d prostoru.
R^d predstavlja d-dimenzionalen prostor. Razvrstitev $H(N)$ tako predstavlja ureditev vsega ali samo del prostora R^d, v katerem je niz N. $H(N)$ se imenuje tudi geometrijski kompleks. Pod tem se razume zbir razčlenjenih nizov, različnih dimenzij, skupaj z medsebojnimi povezavami v prostoru R^d. Najpreprostejši primer je planarni graf, ki je sestavljen iz temen, robov in ploskev ter povezav med njimi. Element, v geometrijskem kompleksu, dimenzije j se imenuje j-ploskev. 0-ploskev ali ničdimenzionalna ploskev je teme, 1-ploskev ali enodimenzionalna ploskev je rob, itd. V tem primeru je terminologija prilagojena planarnemu grafu.

Vsakega geometrijskega problema se je mogoče lotiti s pomočjo deterministčnih ali naključnih algoritmov [7].

Deterministični načini:

- Inkrementalna metoda: Dodajanje elementov določenega niza enega za drugim po znanem vrstnem redu.

- Deljenje in združevanje: Delitev niza po nekem določenem načinu in ponovna združitev.
- Iskanje med zaporednimi razvrstitvami, ki so določene.
- Pregledovanje.

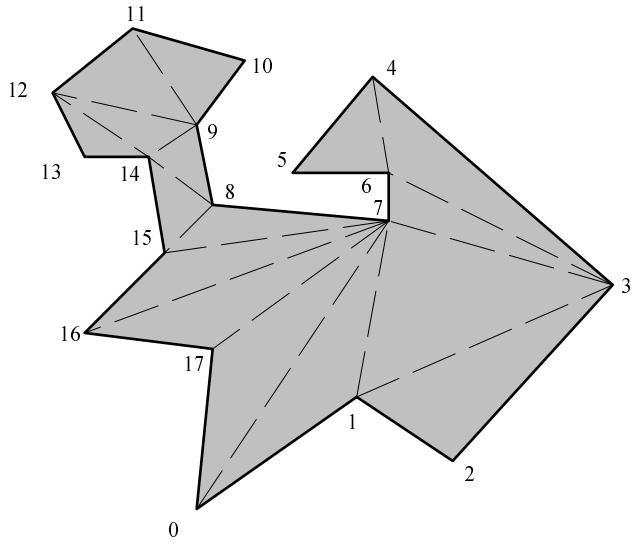
Naključni načini:
- Slučajna inkrementalna metoda: Dodajanje elementov določenega niza enega za drugim brez znanega vrstnega reda.
- Naključno deljenje in združevanje: Naključna delitev niza in ponovna združitev.
- Iskanje med zaporednimi razvrstitvami, ki so naključne.

Slabost deterministčnih načinov je v tem, da so težko prilagodljivi razmeram v večdimenzionalnih prostorih. Celo v dvodimenzionalnem prostoru so lahko naključni algoritmi učinkovitejši kot deterministični.

V tretjem poglavju so predstavljeni osnovni elementi računske geometrije in zakonitosti njihove postavitve na ravnini.

Osnovni element računske geometrije je točka, ki je lahko postavljena:
- na premico $p_i = [x_i] \in R^1$,
- na ravnino $p_i = [x_i, y_i] \in R^2$,
- v prostor $p_i = [x_i, y_i, z_i] \in R^3$.

Z različnimi postavitvami in načini združevanja teh točk je mogoče oblikovati ravninske in prostorske elemente. Na ravnini so to linijski segmenti in iz njih sestavljeni mnogokotniki. Mnogokotniki so območja na ravnini, ki so od okolice razmejeni z nizom linijskih segmentov. Linijski segmenti ali robovi se med seboj dotikajo v temenih. Temena so lahko vbočena ali izbočena, kar vpliva na obliko mnogokotnika in na možnost postavitve diagonale med dve temeni. S postavljanjem diagonal v mnogokotnik se doseže triangulacija ali razdeljevanje na trikotnike. Mogoči so še drugi načini razdeljevanja mnogokotnikov: na trapeze, na konveksne površinske elemente. Razdeljevanje mnogokotnikov na površinske podelemente je izvedljivo s pomočjo algoritmov. Na Sliki 1.4 je prikazan mnogokotnik z osemnajstimi temeni, preko katerega je izvedena triangulacija.

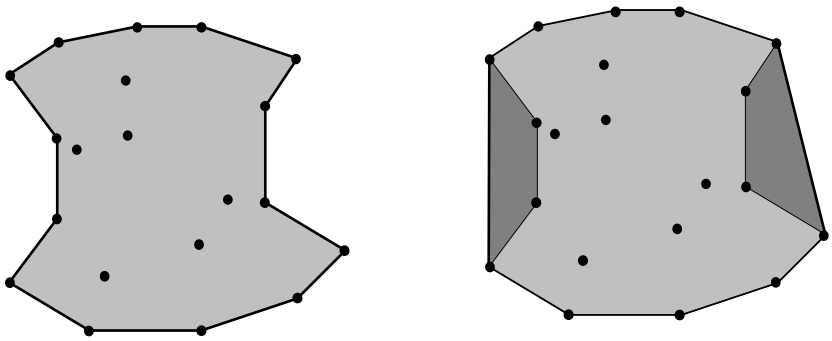


Slika 1.4 Mnogokotnik z $n = 18$ temeni.

V poglavjih 4 in 5 so predstavljene lupine na ravnini in v prostoru. Okoli niza točk *N* na ravnini je mogoče oviti ovoj, ki se dotika samo skrajnih točk, medtem ko so vse ostale točke v njegovi notranjosti. Te geometrijske forme se imenujejo lupine. Pri lupinah je pomembno, kako so predstavljene. Lupina je lahko prikazana kot niz neurejenih skrajnih točk niza *N*, ki so med seboj sicer povezane, vendar ta povezava ni nakazana. Ali pa je lupina prikazana kot urejen niz skrajnih točk niza v nasprotni smeri urinih kazalcev, ki so med seboj povezane z robovi. Unija vseh takih robov je lupina niza točk *N* na ravnini.

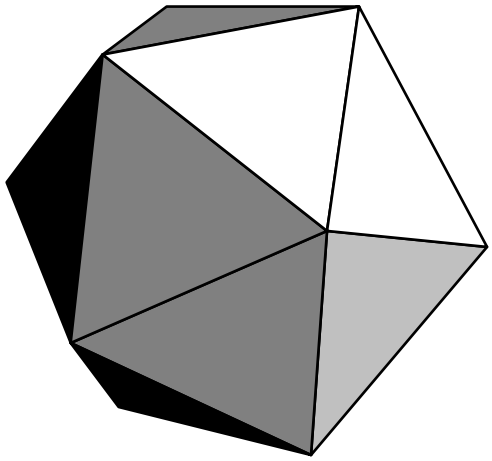
Podobno velja tudi za niz točk N v prostoru. Osnovni princip je enak: okoli vseh skrajnih točk niza *N* v prostoru je potrebno oviti ovoj tako, da bodo vse točke niza v njegovi notranjosti ali vsaj na njegovi meji. Pri predstavitvi prostorske lupine je dilema enaka kot na ravnini, le da je tu dodana možnost predstavitve s površinskimi elementi, ki se med seboj dotikajo v robovih in temenih, ki jih določajo skrajne točke niza *N*.

Zgoraj povedano velja za konveksne lupine. Pri lupinah, ki niso nujno konveksne, navedeni opisi odpovejo, saj gre lupina lahko tudi skozi točke, ki niso ekstremne, zaradi tega opisani način ovijanja ovoja okoli niza točk *N* za nekonveksne lupine odpove. V nekonveksnih temenih je notranji kot $>\pi$. V tem primeru je potrebno lupino graditi na medsebojnih odnosih najbližjih si točk in upoštevati določene omejitve. Na Sliki 1.5 je prikazana nekonveksna lupina niza točk *N* in konveksna lupina, kakršna bi nastala iz istega niza točk *N* po načinu ovijanja ovoja okoli vseh skrajnih točk.



Slika 1.5 Lupina na ravnini. (a) Nekonveksna lupina; (b) konveksna lupina.

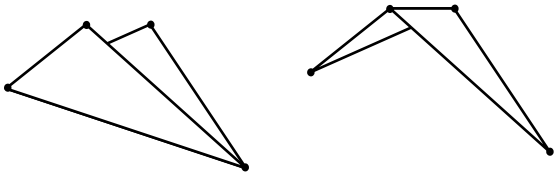
Enak problem nastane pri konstrukciji nekonveksene lupine v prostoru. Vsako nekonveksnost je potrebno prepoznati skozi postavitev prostorskih točk oz. s pomočjo določenih omejitev. Na Sliki 1.6 je prikazana konveksna lupina v 3D, ikozaeder. Sestavljen je iz dvajsetih trikotnikov, ki se med seboj dotikajo v dvanajstih temenih in tridesetih robovih. Njegova konstrukcija je mogoča z ovijanjem ovoja okoli skrajnih točk niza točk *N* v prostoru.



Slika 1.6 Konveksna lupina v 3D, ikozaeder.

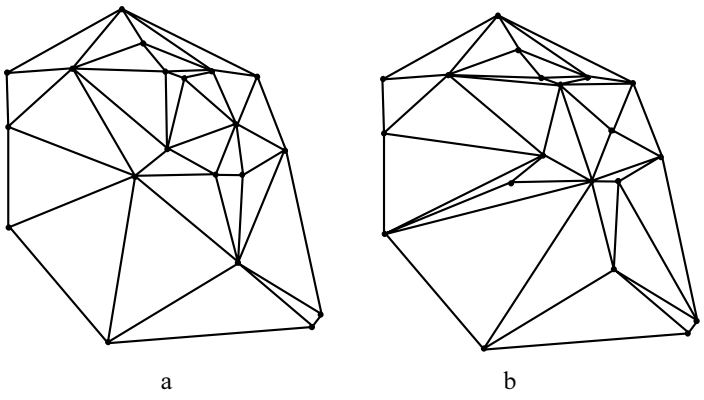
Na ravnini je lupina sestavljena iz robov, ki povezujejo med seboj pare točk. V prostoru je lupina sestavljena iz površinskih elementov: trikotnikov (Slika 1.6), kvadratov ali drugih mnogokotnikov. Površinske elemente se prilagaja namenu uporabe modela določenega telesa. Vrsti uporabljenih površinskih elementov je potrebno prilagoditi način odčitavanja točk z modela. Pri trikotniških mrežah je možnosti postavitev trikotnikov več, vendar le ena daje najboljšo možno rekonstrukcijo površine. Na Sliki 1.7 je prikazano, kako postavitev trikotnikov med

istimi štirimi točkami vpliva na površino. V tem primeru izbira vpliva tudi na konveksnost površine.



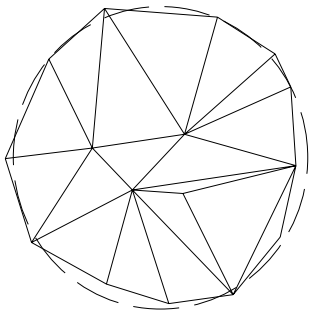
Slika 1.7 Različna postavitev trikotnikov za štiri točke v prostoru.

V 6. poglavju so obravnavani odnosi med točkami niza *N*, kar popisujejo Voronoi diagrami. Voronoi diagram ravnino razdeli tako, da v vsakem Voronoi območju leži samo ena točka niza *N*, rob tega območja pa leži natančno na polovici med dvema sosednjima točkama. Omenjena lastnost vodi do najboljše možne triangulacije, ki se imenuje Delaunay-eva triangulacija in je v bistvu dvojnost Voronoi diagrama. Delaunayeva triangulacija zagotavlja tako razdelitev površine, da so vsi trikotniki kar se da "debeli", to pomeni, da imajo čim večje notranje kote.
To lastnost je mogoče v kombinaciji z ostalimi normativi uporabiti pri konstrukciji nekonveksne lupine v prostoru. Na Sliki 1.8 je prikazana Delaunayeva triangulacija in neka poljubna triangulacija za niz točk *N* na ravnini.



Slika 1.6 Triangulacija točk na ravnini: (a) Delaunayeva; (b) poljubna.

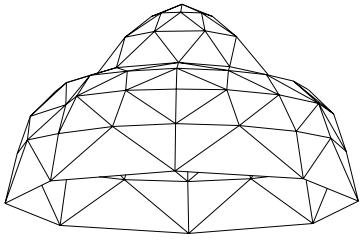
V 7. poglavju je predstavljen sistem za zajemanje prostorskih točk. Narejen je bil z namenom, da se pokaže možnost mehanskega zajemanja 3D točk, kar je bistveno ceneje od laserskega 3D skenerja. Uporabljeni elementi so najcenejši možni in prav tako izdelava. Zaradi tega je natančnost razmeroma slaba, a še vedno zadovoljiva za predstavitev delovanja sistema in kasneje za prikaz rekonstrukcije površine iz zajetih prostorskih točk. Pokazana je pričakovana deformacija oblike zaradi nenatančnosti potenciometrov in v podpoglavju 7.6 dejanska deformacija valjaste oblike (Slika 1.8).



Slika 1.7 Deformacija valjastega dna.

A/D konverter digitalizira analogne podatke dobljene preko potenciometrov. Ta je povezan z računalnikom preko LPT porta. Prikazani sta sliki napajalnika konverterja in njegova povezava z računalnikom.
Digtalizirane podatke prečita program napisan v C, ki upošteva časovni diagram A/D pretvorbe in jih sproti zapisuje v datoteko "*tocke.dat*". Diagram A/D pretvorbe je prikazan v podpoglavju 7.3.2.
Iz geometrije sistema in vrednosti na potenciometrih je mogoče določiti koordinatne vrednosti točke v prostoru.

V 8. poglavju je predstavljen celoten potek rekonstrukcije površine. Najprej osnovni elementi geometrije, ki so podrobneje opisani v 3. poglavju. V podpoglavju 8.2 je opisan princip triangulacije v 2D. Temelji na pregledovanju ravnine (uporabljen pri algoritmih opisanih v 3. poglavju) in lastnostih Delaunayeve triangulacije. To je predvsem lastnost največjega praznega kroga, ki je natančneje opisana v podpoglavju 6.4.3. Postopek je zastavljen tako, da ga je mogoče uporabiti tudi v prostoru, kar je prikazano v podpoglavjih 8.3 in 8.4. V podpoglavju 8.4 je natančno opisan celoten postopek triangulacije oz. rekonstrukcije površine v 3D. V podpoglavju 8.5 sta prikazani vhodna in izhodna oblika rekonstrukcije in v podpoglavju 8.6 še nekaj primerov in načinov uporabe rekonstrukcij. Na Sliki 1.8 je prikazana rekonstrukcija površine.



Slika 1.8 Rekonstrukcija površine.

2 GEOMETRIJA IN OSNOVE ALGORITMOV

2.1 Zgodovinska pričakovanja

Osnovna motiviranost k reševanju geometrijskih problemov izvira že iz starega Egipta in antične Grčije.
Za reševanje geometrijskih problemov je bil pomemben izum Evklidove strukture, to je sheme, ki na slogovno čist način vsebuje prepletanje algoritma in njegovega dokaza. Evklidova shema izpolnjuje vse, kar se zahteva od algoritma: je jasna, nedvoumna, natančna in predvsem zaključena celota, ki da končni rezultat. Shema je pomembna tudi zato, ker določa zbir orodij in operacij, ki se jih sme pri reševanju problemov uporabljati. Vprašanje, ki je v dobi reševanja problemov računske geometrije s pomočjo računalnika še vedno aktualno je, ali je z Evklidovimi primitivi moč izpeljati do konca vse geometrijske "račune". Nove možnosti reševanja so se pokazale, ko je Descartes predstavil koordinate in možnost prikazovanja geometrijskih problemov algebraično [3].

2.1.1 Kompleksnost v klasični geometriji

Evklidove sheme so, razen za trivialne primere, zaradi nerazvitih primitivov, ki so dovoljeni, precej zapletene. Matematiki so se v kasnejših obdobjih precej posvečali vprašanju, kako zmanjšati število zaporedno potrebnih operacij na minimum, ki še vedno zagotavlja rešitev danega problema. Končno število potrebnih operacij za rešitev problema predstavlja enostavnost rešitve oziroma merilo kompleksnosti. Tu pa je že podobnost s sodobno idejo časovne kompleksnosti algoritma. Prav tako je vseskozi prisotno vprašanje prostora, ki ga potrebujemo za dokončno rešitev problema.

2.2 Algoritem

V zadnjih petindvajsetih letih so se vzporedno z razvojem računalnikov pospešeno razvijali tudi algoritmi. Temeljna dela so v začetku sedemdesetih let pripomogla k sistematiziranemu uvajanju pravil, ki so v tem času prerasla v standard tega znanstvenega področja, ki se mu reče računska geometrija.
Glavna elementa računske geometrije sta: algoritem in podatkovna struktura. Algoritmi so programi, ki jih je mogoče izvajati na računalniku oziroma na primerni abstrakciji dejanskega računalnika. Podatkovne strukture so ureditve podatkov na način, ki v povezavi z algoritmi omogočajo učinkovito in elegantno rešitev računskih problemov.

2.2.1 Algoritem: Izraz in ovrednotenje učinka

Algoritmi so oblikovani glede na določen način računanja; način računanja je primerna in zadostna abstrakcija fizičnega orodja - računalnika - na katerem se algoritem oziroma program izvaja. Ni potrebno niti zaželjeno, da se algoritmi pišejo v strojnem jeziku. Nasprotno, višji programski jezik Pidgin Algol, ki je standard na tem področju, teži k jasnosti, učinkovitosti in zgoščenosti izraza. Pidgin Algol je neformalna in prilagodljiva oblika Algolu in njemu podobnim programskim jezikom. Jezik je precej strog pri kontrolnih strukturah, na drugi strani pa daje veliko svobode pri oblikovanju drugih izjav, kjer dopušča običajne matematične izraze v povezavi z navadnim besednim izražanjem. Algoritem napisan v Pidgin Algolu je moč enostavno preoblikovati v katerikoli višji programski jezik [10].

Program se imenuje procedura in ima obliko

procedure ime (parametri) izjava.

Izjava je lahko izpisana kot niz dveh ali večih izjav, ki jih je potrebno med seboj združiti:

begin izjava;
:
:
izjava
end.

Izjava je lahko oblikovana z navadnim besednim izrazom, ali pa z bolj formalno oblikovanim izrazom.

1. Prireditveni stavek:

spremenljivka := vir

2. Pogojni stavek:

if pogoj **then** izjava (**else** izjava)

3. Zanka, ki je lahko v eni od naslednjih oblik:

3a. **for** spremenljivka := vrednost **until** vrednost **do** izjava
3b. **while** pogoj **do** izjava
3c. **repeat** izjava **until** pogoj

Razlika med **while** in **repeat** zankama je v tem, da je pri while-zanki pogoj testiran pred izvajanjem izjave, pri repeat-zanki pa je pogoj testiran kasneje.

4. Vrnitev vrednosti:

return izraz

Izjava te vrste mora biti uporabljena v funkciji, ki je oblike

function ime (parametri) izjava.

Izraz, ki ga vrne *return*, postane vir prireditvenega stavka, kot je pokazano v 1 zgoraj.

Vsi v nadaljevanju opisani algoritmi so oblikovani v Pidgin Algolu.

Čas, ki je potreben za izvršitev algoritma, je vsota vseh delnih časov, v katerih se izvršijo posamezne opracije.Pri tem nas zanima tisti čas, ki je neodvisen od vrste računalnika, na katerem se algoritem izvaja ter predvsem, kako zahtevnost problema in količina vhodnih podatkov vplivata na čas izvajanja. Pri oblikovanju in analizi algoritma je potrebno za ovrednotenje izvajalnega časa določiti število ključnih operacij, ki se izvedejo. Ta način je uporaben pri določanju spodnje meje, pri čemer lahko vsaka neupoštevana for-zanka čas izvajanja samo poveča. Pri določanju zgornje meje ta način ne da zadovoljivih podatkov. V tem primeru je potrebno upoštevati prav vse operacije, ki so se in ki bi se lahko izvedle v algoritmu. Sistem, ki opredeljuje spodnji, zgornji in vmesni čas izvajanja algoritmov je [10]:

- $O(f(N))$ označuje niz vseh funkcij $g(N)$ tako, da obstajata pozitivni konstanti C in N_0 ter da velja $|g(N)| \leq Cf(N)$ za vse $N \geq N_0$.
- $\Omega(f(N))$ označuje niz vseh funkcij $g(N)$ tako, da obstajata pozitivni konstanti C in N_0 ter da velja $g(N) \geq Cf(N)$ za vse $N \geq N_0$.
- $\theta(f(N))$ označuje niz vseh funkcij $g(N)$ tako, da obstajajo pozitivne konstante C_1, C_2 in N_0 ter da velja $C_1 f(N) \leq g(N) \leq C_2 f(N)$ za vse $N \geq N_0$.

$O(f(N))$ tako označuje funkcije, ki nikakor niso večje kot nek konstanten čas $f(N)$. Ta način se uporablja za označevanje zgornje časovne meje izvajanja. Nasprotno $\Omega(f(N))$ označuje funkcije, ki so vsaj tako velike kot nek konstanten čas $f(N)$. Analogno se ta način uporablja za označevanje spodnje časovne meje izvajanja. $\theta(f(N))$ označuje funkcije, ki so iste velikosti kot je $f(N)$. Ta način se uprablja za določevanje "optimalnih" algoritmov.

Poleg časa je drugo merilo kvalitete izvajanja algoritma prostor, ki je ponavadi definiran kot količina spomina, ki je potreben za izvedbo algoritma. Časovna in prostorska kompleksnost sta tako dve glavni merili pri analizi izvajanj algoritmov. Pri izbiri algoritmov za posamezne aplikacije je potrebno upoštevati tudi dejstvo, da algoritem, ki je uspešen pri reševanju velikih problemov, ni nujno uspešen pri reševanju majhnih problemov, in obratno.

2.2.2 Podatkovne strukture

Algoritmi, ki se uporabljajo na področju računske geometrije, manipulirajo z objekti, ki niso na nivoju strojnega jezika. Podatke je zato potrebno urediti v čim preprostejšo strukturo, ki v povezavi z algoritmom omogoča rešitev problema z računalnikom. Najpogosteje so podatki predstavljeni kot urejeni ali neurejeni nizi elementov.

Naj bo S niz elementov predstavljen s podatkovno strukturo in naj bo u poljubni element neke splošne množice, katere podmnožica je niz S. Glavne operacije, ki jih je mogoče izvesti preko niza S, so:

1. MEMBER(u, S). Ali velja $u \in S$? (možen odgovor da/ne.)
2. INSERT(u, S). Doda element u v niz S.
3. DELETE(u, S). Zbriše element u iz niza S.

Če je $\{S_1, S_2, ..., S_k\}$ zbir nizov, katerih medsebojni preseki so prazne množice, potem sta uporabni operaciji:

4. FIND(u). Vrne j, če velja $u \in S_j$.
5. UNION(S_i, S_j; S_k). Združi niza S_i in S_j in ga poimenuje S_k.

Kadar je določena splošna množica, so pomembne naslednje operacije:

6. MIN(S). Vrne najmanjši element niza S.

7. SPLIT(u, S). Razdeli niz S na $\{S_1, S_2\}$ tako, da velja $S_1=\{v: v\in S \text{ in } v\leq u\}$ ter $S_2=S-S_1$.
8. CONCATENATE(S_1, S_2). Če za poljubna $u'\in S_1$ in $u''\in S_2$ velja $u'\leq u''$, potem naredi niz $S=S_1\cup S_2$.

Podatkovno strukturo si je abstraktno moč predstavljati kot premico, na kateri ima vsak podatek svojo točko. V tem primeru je brisanje in dodajanje elementov povsem poljubno. Pri nekaterih aplikacijah se zahtevajo dodatne omejitve: elemente se dodaja na eni strani premice, briše se jih na drugi strani; brisanje in dodajanje elementov se izvaja na istem mestu. Neurejen niz je vedno mogoče urediti upoštevaje lastnosti posameznih elementov oziroma s poimenovanjem elementov in abecednim razvrščanjem.

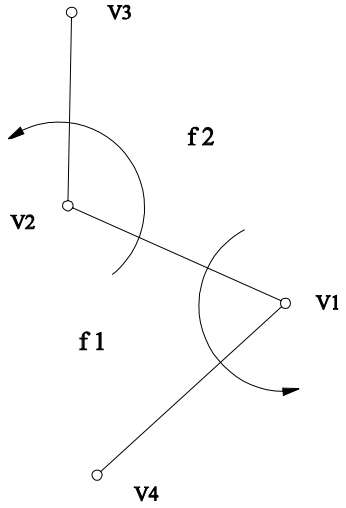
Pri računski geometriji je bilo potrebno običajne podatkovne strukture prilagoditi specifičnosti problemov. Pri tem sta nastali nekonvencionalni strukturi DCEL (Double-Connected-Edge-List) in odsekovna drevesna struktura.

2.2.2.1 Podatkovna struktura DCEL

Struktura DCEL je primerna za predstavitev planarnih grafov na ravnini, kjer je $G=(V, E)$ preslikava temen V_i v točke na ravnini in E_i v povezave med temi točkami tako, da se povezave med seboj ne sekajo, razen v svojih krajiščih. Planarni graf je tista oblika, pri kateri so povezave med točkami ravni linijski elementi - robovi.
Naj bo $V=\{v_1, ..., v_n\}$ in $E=\{e_1, ..., e_n\}$. Graf predstavlja zaporedni zapis robov in temen, v katerih se robovi sekajo. Vsak rob je predstavljen natančno enkrat. Primer DCEL podatkovne strukture je prikazan v Tabeli 2.1 in na Sliki 2.1 del planarnega grafa, ki ustreza podatkom v tabeli.

Tabela 2. 1: Primer DCEL podatkovne strukture.

	V1	V2	F1	F2	P1	P2
1						
2						
:						
a1	1	2	1	2	a2	a3
:						
a2	4	1	1			
:						
a3	2	3		2		



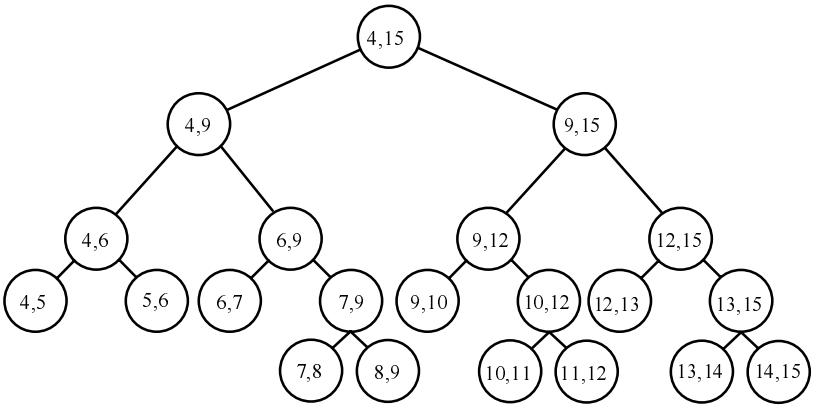
Slika 2.1 Del planarnega grafa za zgornjo DCEL podatkovno strukturo.

V prvih štirih poljih $V1$, $V2$, $F1$ in $F2$ so zapisani podatki, v zadnjih dveh poljih $P1$ in $P2$ so zapisani kazalci na podatkovna polja. V polju $V1$ je zapisano začetno teme roba, v polju $V2$ je zapisano končno teme roba. V poljih $F1$ in $F2$ sta zapisani imeni ploskev, ki ležita na levi in desni strani gledano iz $V1$ proti $V2$. Kazalec $P1$ kaže na rob, ki v nasprotni smeri urinih kazalcev z vrtiščem v $V1$ naslednji sledi robu ($V1$, $V2$). Podobno velja za kazalec $P2$, le da je tu vrtiščno teme $V2$.

V algoritmu opisanem v osmem poglavju je uporabljena poenostavljena struktura DCEL. Poenostavitev je potrebna, ker ne predstavlja planarnega grafa, ampak triangulacijo v prostoru.

2.2.2.2 Odsekovna drevesna struktura

Odsekovna drevesna struktura je statično oblikovana za manipuliranje z odseki na realni osi, katerih ekstremi so del določenega niza na abscisi v obsegu [1, n]. Za celi števili l in r, za kateri velja $l<r$, odsekovna drevesna struktura $T(l, r)$ nastaja: odsek v je sestavljen iz vrednosti $B[v] = l$ in $E[v] = r$, pri čemer sta B začetek odseka in E konec odseka. Če velja $r\text{-}l>1$, potem je mogoče odsek v preurediti v dva pododseka. V levega $T(l,\ \text{int}((B[v]+E[v])/2))$ in desnega $T(\text{int}((B[v]+E[v])/2),\ r)$. Parametra $B[v]$ in $E[v]$ določata interval $[B[v],\ E[v]] \subseteq [l,\ r]$ v povezavi z odsekom v. Odsekovno drevesna struktura za l=4 in r=15 $T(4, 15)$ je prikazana na Sliki 2.2.



Slika 2.2 Odsekovna drevesna struktura $T(4, 15)$.

2.3 Definicije osnovnih geometrijskih elementov

Elemetni, ki se uporabljajo za reševanje problemov s področja računske geometriji, so v glavnem točke (niz točk), ki so postavljene v evklidov prostor. Koordinatni sistem je določen tako, da je vsaka točka predstavljena z vektorjem primerne dimenzije v kartezijevem koordinatnem sistemu. Za geometrijske elemetne ni nujno, da so sestavljeni iz končnega števila točk, pomembno je, da te točke zagotovijo končno določljivost posameznega elementa. Tako velja, razen za točko, ki določa sama sebe: premico določata dve poljubni točki na njej, linijski segment določata točki na njegovih krajiščih, ravnino določajo tri poljubne točke te ravnine, mnogokotnik določa (urejen) niz točk, itd. [7].

Točka. Točko p glede na dimenzijo prostora določa $(x_1, ..., x_d)$ v E^d, pri čemer je E^d d-dimenzionalen evklidov prostor. Prav tako je točko moč razumeti kot d-dimenzionalen vektor v E^d z vrhom v točki p.

Premica. Dve različni točki q_1 in q_2 v E^d določata premico v tem prostoru kot linearno kombinacijo

$$\alpha q_1+(1-\alpha)q_2; (\alpha \in R).$$

Bolj splošna oblika za k neodvisnih točk pri $(k \leq d)$ je

$$\alpha_1 q_1+\alpha_2 q_2+...+\alpha_{k-1} q_{k-1}+(1-\alpha_1-...-\alpha_{k-1})q_k; (\alpha_j \in R, j=1,...k-1).$$

Linijski segment. Dve različni točki q_1 in q_2 v E^d določata segment podobno kot premico, le da je potrebno zadostiti dodatnemu pogoju za koeficient α

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

Taka kombinacija opisuje ravni linijski segment, ki povezuje točki q_1 in q_2. Segment je lahko označen tudi kot neurejen par $\overline{q_1 q_2}$.

Konveksen niz. Področje D v E^d je konveksno, če je linijski segment $\overline{q_1 q_2}$ med katerimakoli dvema točkama q_1 in q_2 v E^d, v celoti v D.

Konveksna lupina. Konveksna lupina niza točk S v E^d je meja okoli konveksnega niza in povezuje med seboj vse ekstremne točke tega niza.

Mnogokotnik. V E^2 je mnogokotnik določen kot končen niz linijskih segmentov, ki se v parih dotikajo med seboj samo v svojih končnih točkah. Linijski segmenti so robovi mnogokotnika, končne točke pa temena mnogokotnika.

Planarni graf. Graf $G=(V, E)$, pri čemer je V niz temen in E niz robov, je planarni, če se ga da položiti na ravnino, ne da bi prekrival sam sebe. Planarni graf določa razdelitev ravnine oziroma področja na ravnini na manjša podpodročja.

Triangulacija. Razdelitev ravninskega področja je triangulacija, če so vsa njegova podpodročja trikotniki. Triangulacija je planarni graf z največjim možnim številom robov, pri čemer se niti dva med seboj ne smeta sekati.

Polieder. V E^3 je polieder določen s končnim nizom mnogokotnikov v prostoru, pri čemer se po dva in dva mnogokotnika med seboj dotikata samo v skupnem robu. Temena in robovi mnogokotnikov so hkrati tudi temena in robovi poliedra.

3 Mnogokotniki

V računski geometriji imajo velik pomen t.i. mnogokotniki. Z njimi je mogoče popisovati objekte v realnem svetu, hkrati pa so primerni za računalniško obdelavo. Po zgradbi so lahko povsem enostavni (tak je s tremi linijskimi elementi trikotnik), lahko pa so sestavljeni iz več tisoč elementov. Večje, kompleksnejše mnogokotnike je ponavadi potrebno razstaviti na več enostavnejših.

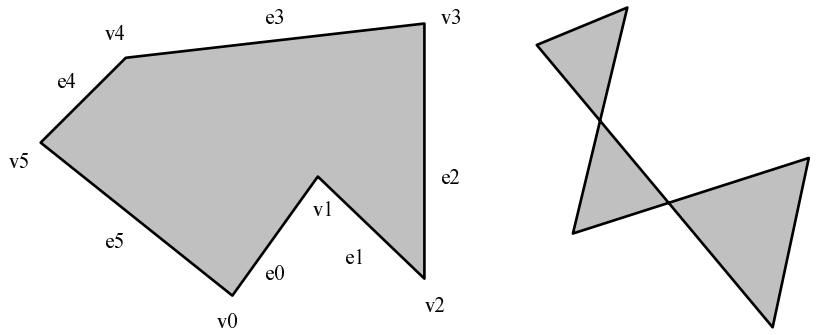
3.1 Definicija mnogokotnika

Mnogokotnik je območje na ravnini, ki je od okolice razmejeno z nizom linijskih elementov. Topološko gledano je mnogokotnik homeomorfična oblika kroga, oziroma njegova deformacija.

Naj bodo v_0, v_1, v_2, ..., v_{n-1} točke na ravnini, naj bodo $e_0 = v_0v_1$, $e_1 = v_1v_2$, ..., $e_i = v_iv_{i+1}$, ..., $e_{n-1} = v_{n-1}v_0$ linijski segmenti, ki povezujejo točke na ravnini med seboj. Obojih je n in določajo mnogokotnik, če [8]:

1. Je v krožnem redu presečišče med dvema sosednjima linijskima segmentoma ena sama skupna točka: $e_i \cap e_{i+1} = v_{i+1}$, za vse $i = 0, ..., n-1$.
2. Se nesosednji segmenti ne sekajo med seboj: $e_i \cap e_j = \varnothing$, za vse $j \neq i+1$.

Prvi pogoj določa mnogokotniku podobnost krogu zaradi krožnega stikanja linijskih segmentov med seboj po sistemu konec z začetkom, drugi pogoj pa definira enostavnost mnogokotnikov.



Slika 3.1 Mnogokotnik: (a) enostaven mnogokotnik - označevanje temen in robov v nasprotni smeri urinih kazalcev; (b) neenostaven mnogokotnik.

Mnogokotniki, ki ne zadoščajo tema dvema pogojema, so neenostavni mnogokotniki (Slika 3.1b). Točke na ravnini v so temena mnogokotnika, linijske povezave med njimi (temeni) e so robovi mnogokotnika. Vsak enostaven mnogokotnik P razdeli ravnino na dva dela. Meja mnogokotnika P je δP. Velja $\delta P \subseteq P$.

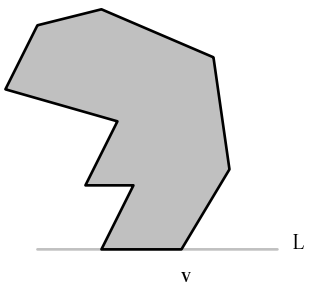
3.2 Triangulacija - razdeljevanje na trikotnike

Pri reševanju geometrijskih problemov s pomočjo mnogokotnikov, je te potrebno večkrat deliti na manjše, enostavnejše mnogokotnike. Osnovna delitev je delitev na trikotnike ali t.i. triangulacija. Glavno vprašanje, ki se pri tem zastavlja je, ali je na trikotnike mogoče razdeliti vsak mnogokotnik.
Obstoj triangulacije je tesno povezan z obstojem diagonale v mnogokotniku. Postavitev diagonale v mnogokotnik je mogoča le, če ima mnogokotnik vsaj eno izbočeno teme. Iz tega sledijo trije logični zaključki:

1. Vsak mnogokotnik mora imeti najmanj eno izbočeno teme.
2. Vsak mnogokotnik z $n \geq 4$ temeni ima diagonalo.
3. Vsak mnogokotnik P z n temeni se da razdeliti na trikotnike (triangulirati) z dodajanjem diagonal.

3.2.1 Obstoj izbočenega temena

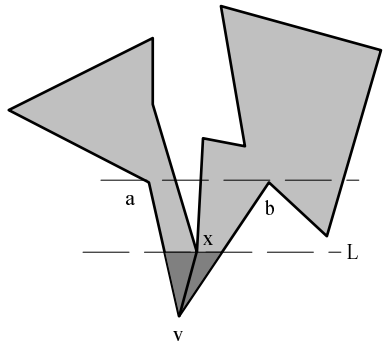
Za dokaz prve postavke je potrebno privzeti mejo δP mnogokotnika P kot pot, po kateri hodi hodec v nasprotni smeri urinih kazalcev. Vsak njegov obrat v levo označuje izbočeno teme mnogokotnika, vsak njegov obrat v desno pa vbočeno teme mnogokotnika. Naj bo L premica, ki teče skozi najnižje teme mnogokotnika v; najnižje glede na koordinato y. Če je takih temen več, naj bo to najbolj desno, najnižje teme. notranjost mnogokotnika P je nad premico L. V temenu v hodec naredi obrat v levo, kar pomeni, da ima vsak mnogokotnik najmanj eno izbočeno teme (Slika 3.2).



Slika 3.2 V najbolj desnem, najnižjem temenu se hodec obrne v levo; teme v je izbočeno.

3.2.2 Obstoj diagonale

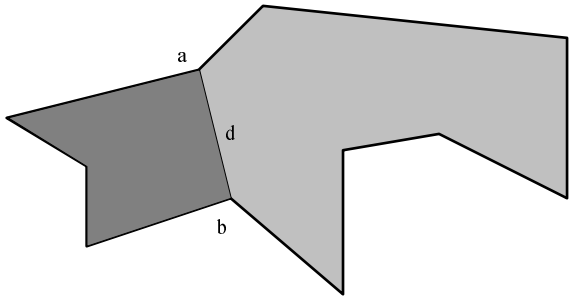
Naj bosta a in b temeni mnogokotnika P, ki ležita na meji mnogokotnika δP pred in za temenom v. Če daljica ab meje mnogokotnika δP ne seka, niti se je ne dotika, potem je ab nedvoumno diagonala mnogokotnika. Če pa se daljica ab dotika ali seka mejo mnogokotnika δP, potem za ta mnogokotnik velja $n \geq 4$, v trikotniku Δavb pa je še najmanj eno teme mnogokotnika P, ki ni a, v ali b. Naj bo x teme mnogokotnika P v trikotniku Δavb, ki je najbližje temenu v v pravokotni smeri na ab. Tako je teme x prvo teme, ki ga zadane premica L, ko se pomika iz temena v proti daljici ab ($L \parallel ab$). V prostoru Δavb, ki je omejen s premico L (temno senčeno na sliki), ni nobene točke meje mnogokotnika δP, razen temena x. Tako je daljica vx gotovo diagonala mnogokotnika P (Slika 3.3).



Slika 3.3 vx mora biti diagonala mnogokotnika P.

3.2.3 Obstoj triangulacije in njene lastnosti

Dokaz o obstoju diagonale posredno dokazuje, da se da vsak mnogokotnik razdeliti na trikotnike. Naj bo $d = ab$ diagonala mnogokotnika P. Po definiciji diagonala d seka mejo mnogokotnika δP samo v svojih krajiščih. Diagonala d razdeli mnogokotnik P na dva manjša mnogokotnika P', ki imata vsak manj kot n temen, in s tem postane del meje novih mnogokotnikov $\delta P'$ (Slika 3.4). V vsakem mnogokotniku P' (če velja $n' \geq 4$) je mogoče poiskati novo diagonalo $d' = ef$, ki mnogokotnik P' razdeli na dva nova manjša mnogokotnika. Tako deljenje je mogoče ponavljati, dokler začetni mnogokotnik P ni razdeljen na same trikotnike.

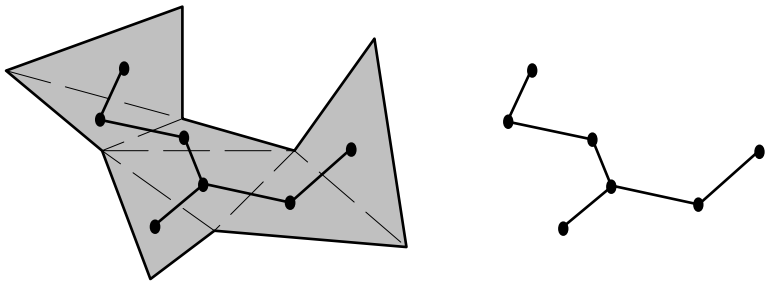


Slika 3.4 Diagonala *ab* razdeli mnogokotnik *P* na dva manjša mnogokotnika.

Triangulacijo preko mnogokotnika je mogoče narediti na več načinov, število katerih narašča s številom temen *n*. Za vse velja, da je za triangulacijo mnogokotnika *P* potrebno *n* - 3 diagonal, ki mnogokotnik *P* razdelijo na *n* - 2 trikotnikov.

3.2.4 Dualnost triangulacije

V teoriji grafov ima pomembno mesto t.i. dualnost ali dvojnost grafa. Gre za lastnost, da je določene podatke moč predstaviti na več načinov (dva-dualnost). Triangulacija mnogokotnika *P* je ponavadi predstavljena z vrisanimi diagonalami, lahko pa tudi z grafom *T*, ki je druga oblika predstavitve istih podatkov. Graf *T*, dualnost triangulacije mnogokotnika, je graf drevesne oblike, ki je sestavljen iz vozlišč in povezav med njimi. Vozlišča predstavljajo trikotnike triangulacije, povezave med njimi pa skupne stranice trikotnikov triangulacije oziroma digonalo, ki je skupna dvema trikotnikoma (Slika 3.5).



Slika 3.5 Dualnost triangulacije: (a) triangulacija mnogokotnika; (b) graf *T*.

Velja, da je graf *T* sestavljen iz vozlišč, v katerih se stikajo največ tri povezave, in povezav med njimi. Vozlišča grafa *T* stopnje ena so končne točke grafa *T* oziroma uhlji poligona *P*. Vozlišča druge stopnje ležijo na veji *T* in jo podaljšujejo. Vozlišča tretje

stopnje so razvejišča grafa T. Iz povedanega sledi, da ima vsak mnogokotnik z $n \geq 4$ temeni vsaj dva neprekrivajoča se uhlja.

3.3 Površina mnogokotnika

Pri računanju površin mnogokotnikov si je mogoče pomagati s triangulacijo. Mnogokotnik je potrebno razdeliti na trikotnike, izračunati površino posameznih trikotnikov in vsota vseh je površina mnogokotnika.

3.3.1 Vektorski produkt

V splošnem je površina trikotnika osnovnica krat višina polovic. Vendar v pogojih, ko so podana temena a, b, c trikotnika T na ravnini, ta formulacija ni več primerna. V tem primeru je uporaben vektorski produkt, katerega velikost predstavlja površino paralelograma s stranicama A in B. Polovica paralelograma je trikotnik s stranicama A in B, pri čemer velja $A = b - a$ in $B = c - a$. Vektorski produkt je izračunljiv s pomočjo determinante:

$$\begin{vmatrix} \vec{i} & \vec{j} & \vec{k} \\ A_0 & A_1 & A_2 \\ B_0 & B_1 & B_2 \end{vmatrix} = (A_1B_2 - A_2B_1)\cdot\vec{i} + (A_2B_0 - A_0B_2)\cdot\vec{j} + (A_0B_1 - A_1B_0)\cdot\vec{k}\,.$$

$\vec{i}$, $\vec{j}$ in $\vec{k}$ so enotski vektorji v x, y in z smeri. V dvodimenzijskem prostoru velja $A_2 = B_2 = 0$, od zgornjega izraza ostane $(A_0B_1 - A_1B_0)\cdot\vec{k}$. Površina trikotnika $T(a, b, c)$ je:

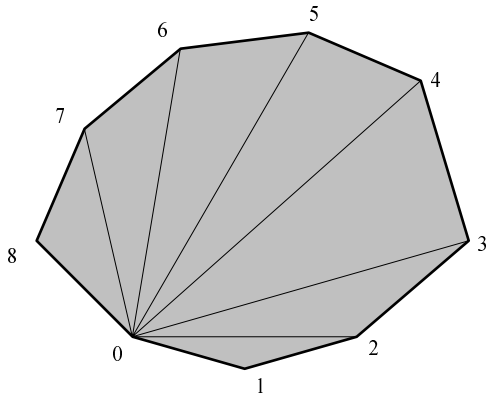
$$\mathcal{A}(T) = 0.5(A_0B_1 - A_1B_0),$$
$$2\mathcal{A}(T) = a_0b_1 - a_1b_0 - a_1c_0 - a_0c_1 - b_0c_1 - c_0b_1$$

ali

$$2\mathcal{A}(T) = \begin{vmatrix} a_0 & a_1 & 1 \\ b_0 & b_1 & 1 \\ c_0 & c_1 & 1 \end{vmatrix}.$$

3.3.2 Površina konveksnega mnogokotnika

Vsak konveksen mnogokotnik je mogoče trivialno razdeliti na trikotnike tako, da se eno krajišče vseh diagonal d pripne v eno teme v_k (Slika 3.6).



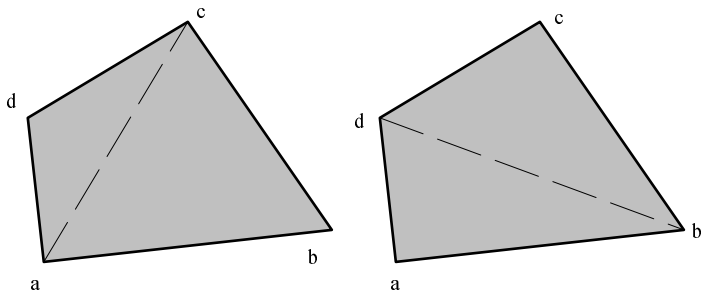
Slika 3.6 Triangulacija konveksnega mnogokotnika; vse diagonale imajo eno skupno krajišče v_0.

Površina tako razdeljenega poligona, ki ima temena označena v nasprotni smeri urinih kazalcev z $v_0, v_1, ..., v_{n-1}$, je lahko zapisana kot

$$\measuredangle(P) = \measuredangle(v_0, v_1, v_2) + \measuredangle(v_0, v_2, v_3) + ... + \measuredangle(v_0, v_{n-2}, v_{n-1}).$$

3.3.3 Površina štiristranega konveksnega mnogokotnika

Ker je razdeljevanje na trikotnike izvedljivo na več načinov, je tudi površino mnogokotnika mogoče zapisati na več načinov. Naj bo $Q = (a, b, c, d)$ štiristrani konveksni mnogokotnik, ki ga je mogoče triangulirati na dva načina (Slika 3.7).



Slika 3.7 Dve možni triangulaciji štiristranega konveksnega mnogokotnika.

V prvem primeru velja

$$\measuredangle(Q) = \measuredangle(a, b, c) + \measuredangle(a, c, d)$$

$$2\measuredangle(Q) = a_0b_1 - a_1b_0 + a_1c_0 - a_0c_1 - b_0c_1 - c_0b_1 + a_0c_1 - a_1c_0 + a_1d_0 - a_0d_1 + c_0d_1 - d_0c_1,$$

po ureditvi:

$2\measuredangle(Q) = a_0b_1 - a_1b_0 + c_0d_1 - d_0c_1.$

Do enakega izraza pripelje računanje površine tudi v drugem primeru. Razvidno je, da izraz ne vsebuje segmenta *ad*, kar pomeni, da postavitev diagonale ne vpliva na končni rezultat $\measuredangle(Q)$.
Če so temena v_i štiristranega konveksnega mnogokotnika popisana s koordinatama x_i in y_i, je splošna enačba za izračun dvojne površine mnogokotnika enaka

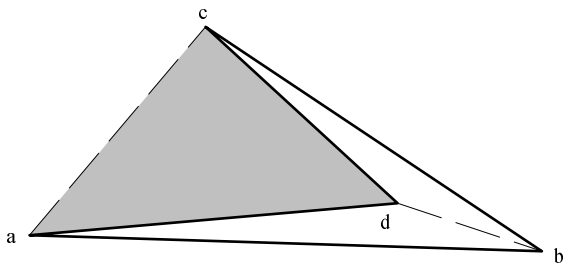
$$2\measuredangle(P) = \sum_{i=0}^{n-1}(x_i y_{i+1} - y_i x_{i+1}).$$

3.3.4 Površina štiristranega nekonveksnega mnogokotnika

Pri nekonveksnem štiristranem mnogokotniku (Slika 3.8) je možen en sam način triangulacije. Diagonala *ac* je izven poligona, ostane le *ab*. Površina mnogokotnika je

$\measuredangle(Q) = \measuredangle(a, b, c) + \measuredangle(a, c, d).$

Zaradi lastnosti vektorskega produkta je površina Δacd negativna in se od Δabc odšteje. Tako prej omenjena enačba za izračun površine štiristranih konveksnih mnogokotnikov velja tudi za štiristrane nekonveksne mnogokotnike.



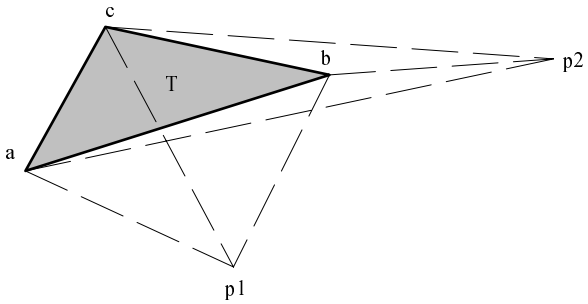
Slika 3.8 Nekonveksen mnogokotnik; senčena površina $\measuredangle(a, c, d)$ je negativna.

Izkaže se, da je enačba za izračun površine štiristranih mnogokotnikov splošna in velja tudi za mnogokotnike $n \geq 5$.

3.3.5 Površina mogokotnika, računana iz poljubnega centra

Pri posplošitvi omenjenega načina, pri katerem se seštevajo površine trikotnikov, naj bo izhodišče točka p, ki je povsem poljubna in naj leži zunaj poligona P (ni nujno). Naj bo $T(a,b,c)$ trikotnik, katerega temena so orientirana v nasprotni smeri urinih kazalcev, naj bo p katerakoli točka na ravnini. Potem velja:

$$\measuredangle(T) = \measuredangle(p, a, b) + \measuredangle(p, b, c) + \measuredangle(p, c, a).$$



Slika 3.9 Površina trikotnika določljiva preko dveh zunanjih izhodiščnih točk p_1 in p_2.

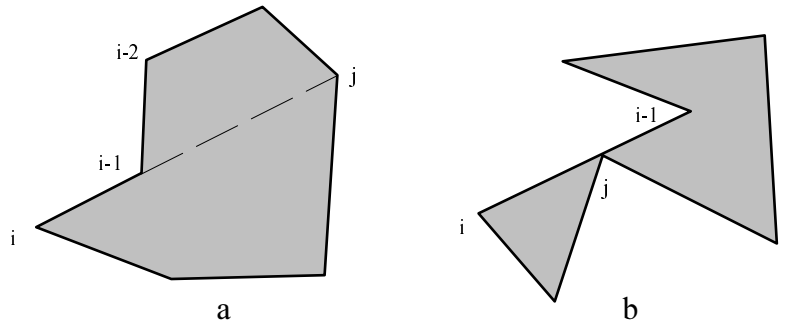
Naj bo izhodiščna točka $p = p_1$ (Slika 3.9). Potem je prvi člen enačbe za izračun površine $\measuredangle(p_1, a, b)$ negativen zaradi svoje orientiranosti; ostala dva člena sta pozitivna. Površina $\measuredangle(p_1, a, b)$ predstavlja natanko tisto površino štiristranega mnogokotnika (p_1, b, c, a), ki je zunaj trikotnika T. Tako je končna površina natančno $\measuredangle(T)$. Podobno velja tudi za izhodiščno točko $p = p_2$. Obe površini, $\measuredangle(p_2, a, b)$ in $\measuredangle(p_2, b, c)$ sta zaradi svoje orientiranosti negativni in se od $\measuredangle(p_2, c, a)$, ki je pozitivna, odštejeta. Ostanek je natančno $\measuredangle(T)$. Podobno velja za katerokoli poljubno točko na ravnini, znotaj ali zunaj trikotnika T.

3.4 Izvedba triangulacije

3.4.1 Diagonala - notranja ali zunanja

Pri triangulaciji je potrebno najprej poiskati diagonalo mnogokotnika. Diagonala razdeli mnogokotnik na dva dela. Postopek je potrebno ponavljati, dokler mnogokotnik ni razdeljen na same trikotnike. Da je daljica v_iv_j, $i \neq j$ diagonala mnogokotnika P, mora veljati dvoje: da ne seka meje mnogokotnika δP in, da je v vsej svoji dolžini v notranjosti mnogokotnika P. Diagonale mnogokotnika se med seboj ne sekajo.
Določanje diagonal [8] je preprosta aplikacija, ki se ponavlja, dokler niso določene vse diagonale mnogokotnika: za vsak rob e mnogokotnika P, ki se ne dotika nobenega

krajišča diagonale *s*, se preveri, če *e* seka *s*. Ko je ugotovljeno presečišče med *e* in *s*, daljica *s* ni diagonala mnogokotnika. Če noben tak rob ne seka *s*, potem je *s* diagonala mnogokotnika *P*. V primeru kolinearnosti točk ta način ni uspešen (Slika 3.10).

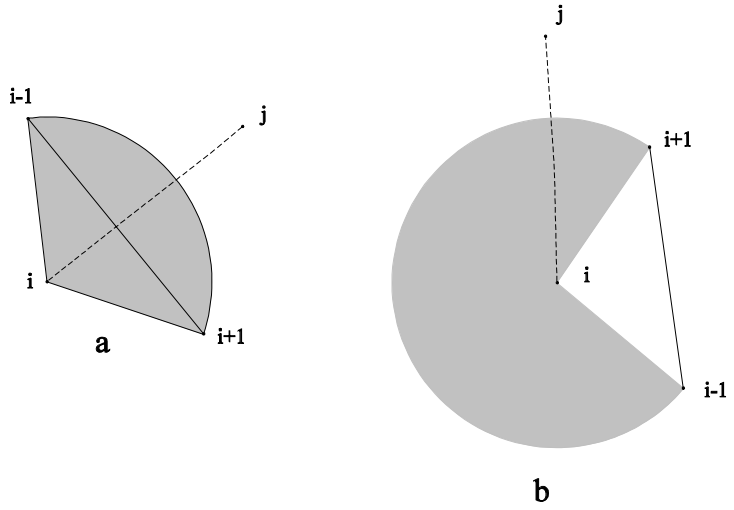


Slika 3.10 Teme, sosednje *i*, je kolinearno z *ij*.

Če je rob v_iv_{i-1} kolinearen z v_iv_j, potem v_iv_j ni diagonala mnogokotnika. Naj bo rob mnogokotnika $e = (v_{i-1}, v_j)$ kolinearen z v_iv_j. Mogoča sta primera, ko je daljica v_iv_j daljša od *e*; pri $j \neq i-1$ (Slika 3.10a) in ko je daljica v_iv_j krajša od *e* (Slika 3.10b), vendar to ni več enostaven poligon.

Vzporedno z aplikacijo določanja diagonal je potrebno za vsako diagonalo preveriti, če ne seka druge diagonale. To se preveri z medsebojnim primerjanjem koordinat krajišč obeh diagonal. Na koncu je potrebno za vsako diagonalo preveriti, ali leži v notranjosti mnogokotnika *P*.

Vprašanje notranjosti in zunanjosti je test lokalne narave. Če je diagonala $s = v_iv_j$ v okolici temen v_i in v_j v notranjosti mnogokotnika *P*, potem je v notranjosti mnogokotnika v vsej svoji dolžini. Diagonala v vsej svoji dolžini namreč ne seka meje mnogokotnika δP. Zadostno je testiranje ene same končne točke diagonale. Če je diagonala *s* v okolici točke v_i v notranjosti mnogokotnika, potem je v notranjosti mnogokotnika tudi v okolici točke v_j in obratno. Podobno velja tudi za zunanjost. Test se tako zoži na pregled okolice točke v_j. Možna sta primera, ko je teme v v_j izbočeno (Slika 3.11a) ali vbočeno (Slika 3.11b).



Slika 3.11 Diagonala $s = ij$ izhaja iz trikotnika določenega s temeni v_{i-1}, v_i, v_{i+1}: (a) i je izbočeno teme;(b) i je vbočeno vbočeno teme.

Na sliki 3.11a je diagonala ij s krajiščem v izbočenem temenu v_i. Jasno je, da je diagonala v notranjosti mnogokotnika P, če je v notranjosti trikotnika z vrhom v temenu v_i. To pomeni, da mora biti teme v_{i-1} na levi strani diagonale v_iv_j in teme v_{i+1} ne levi strani v_jv_i. Ta pogoj se preverja s pomočjo računanja površine med temeni i, j in i-1 ter i, i+1 in j. Če je površina (računana s pomočjo vektorskega produkta) pozitivnega predznaka, potem predpostavka levega drži, če je negativnega predznaka, predpostavka levega ne drži.
V primeru vbočenosti temena i ta način odpove, saj sta lahko temeni i-1 in i+1 hkrati obe na desni strani, hkrati obe na levi strani ali vsako na svoji strani diagonale ij. Podobnost s primerom a je ta, da je notranjost pri izbočenem temenu enaka zunanjosti pri vbočenem temenu. Notrajnost in zunanjost sta zamenjani. Pri drugem primeru torej velja (test po istem načinu kot pri izbočenem temenu), če diagonala ni zunanja, potem je notranja. Testira se zunanjost. Pred testiranjem notranjosti oziroma zunanjosti je tako potrebno določiti, ali je teme v_i izbočeno ali vbočeno. Teme v_i je izbočeno, če je teme v_{i+1} na levi strani ali na robu $v_{i-1}v_i$. Po dogovoru je teme v_i, pri katerem je notranji kot enak π, tudi izbočeno.

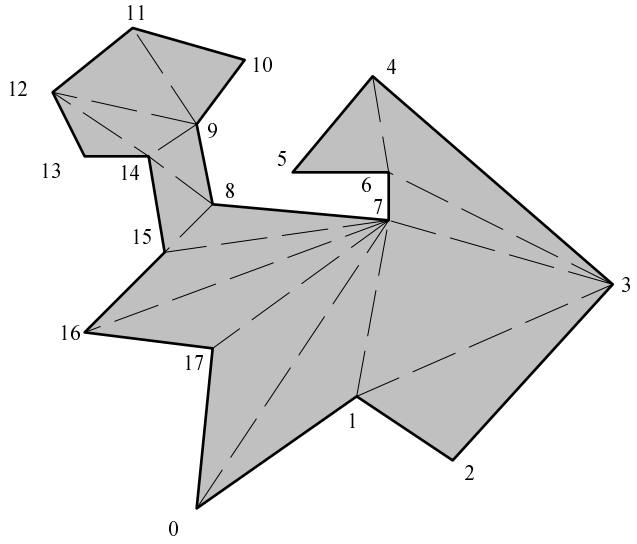
3.4.2 Triangulacija z odstranjevanjem ušes

Zgoraj opisani postopek triangulacije mogokotnikov se ne uporablja prav pogosto, saj je zaradi vsakokratnega pregledovanja pravilnosti diagonale precej počasen.
Hitrejši je način, pri katerem se postavlja diagonalo med temeni v_i in v_{i+2}, pri čemer mora biti v_{i+1} izbočeno teme. Imenuje se način odstranjevanja ušes mnogokotnika [8]. Po definiciji ima vsak mnogokotnik z $n \geq 4$ vsaj dva neprekrivajoča se uhlja.

Triangulacija se začne v izhodiščnem temenu v_0. Poiskati je potrebno prvo trojico v_i, v_{i+1}, v_{i+2}, (uhelj mnogokotnika), t.j. izbočeno teme z vrhom v v_{i+1}. Ko je taka trojica najdena, je daljica v_i, v_{i+2} gotovo diagonala mnogokotnika. Trikotnik v_i, v_{i+1}, v_{i+2} se 'odreže' od preostanka mnogokotnika. Postopek se ponavlja, dokler od mnogokotnika ne ostane samo (zadnji) trikotnik. Triangulacija po načinu odrezovanja ušes je s tem končana (Slika 3.12). Izhoden podatek je seznam diagonal, ki so definirane z lastnimi krajišči glede na označevanje vhodnega podatka mnogokotnika (Tabela 3.1).

Tabela 3.1: Izhodni podatek: seznam diagonal za sliko 3.12.

Diagonale:				
1 - 3	3 - 7	9 - 11	9 - 14	7 - 15
4 - 6	1 - 7	9 - 12	8 - 14	7 - 16
3 - 6	0 - 7	12 - 14	8 - 15	7 - 17

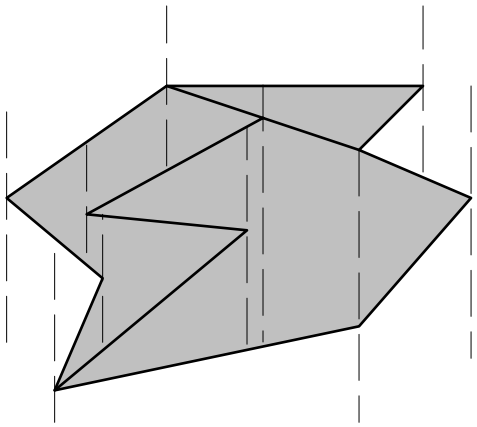


Slika 3.12 Mnogokotnik razdeljen na trikotnike po načinu odstranjevanja uhljev.

3.5 Razdeljevanje na trapeze

Poleg razdeljevanja na trikotnike, je mnogokotnik mogoče razdeliti tudi na trapeze (na konveksne površinske elemente). Ta razdelitev je lahko osnova za nadaljnje razdeljevanje na trikotnike. Razlika med triangulacijo in razdeljevanjem na trapeze je ta, da so pri prvi razdelitvi razdelitveni elementi diagonale, pri drugi razdelitvi pa so razdelitveni elementi horizontalne ali verikalne (horizontalna ali vertikalna trapezna dekompozicija) linije skozi temena *v*, ki so pri dekompoziciji mnogokotnika omejene z njegovo mejo δP (Slika 3.13), pri dekompoziciji ureditve črt pa s črtami samimi ali pa se nadaljujejo v neskončnost (Slika 3.14). Na Sliki 3.13 je prikazana vertikalna

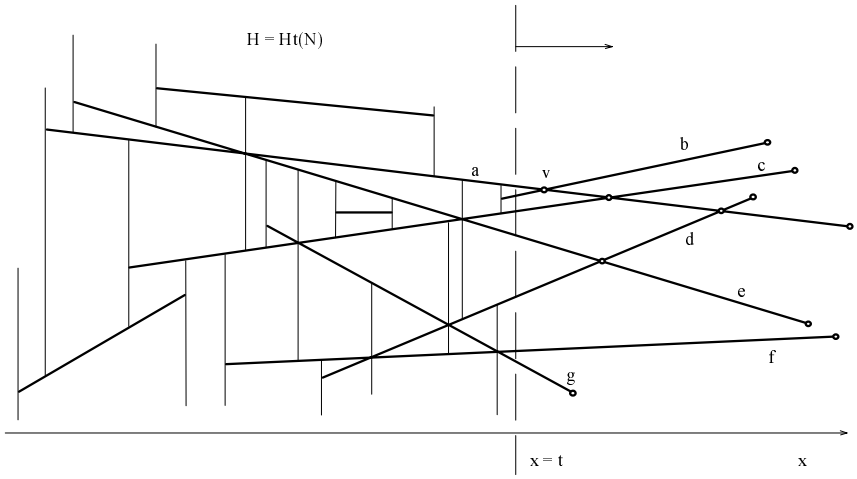
trapezna dekompozicija. Pri horizontalni trapezni dekompoziciji so razdelitvene črte postavljene horizontalno.



Slika 3.13 Vertikalna trapezna dekompozicija.

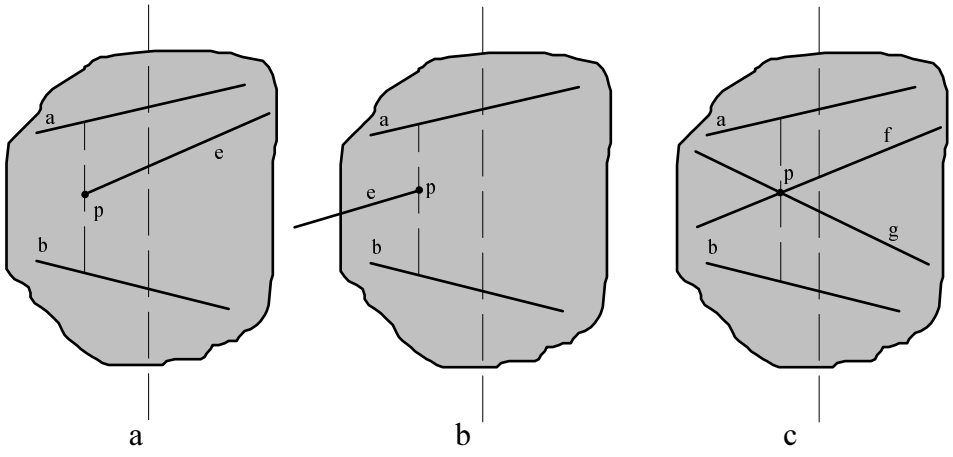
3.5.1 Trapezna dekompozicija s pregledovanjem ravnine

Razdeljevanje na trapeze je izvedljivo s pomočjo različnih deterministčnih ali naključnih inkrementalnih algoritmov. Pri determinističnem načinu gre za pregledovanje poligona oziroma planarnega grafa s pomočjo horizontalne ali vertikalne linije [7]. Če je pregledovalna linijo vertikala, potem koordinata x predstavlja čas, v katerem se pregledovalna linija L pomika od $x = -\infty$ proti $x = +\infty$. Naj bo N niz n-tih linearnih segmentov na ravnini (Slika 3.14).



Slika 3.14 Pregledovanje ravnine z vertikalno linijo L.

Naj bo rezultat trapezne dekompozicije $H(N)$. Naj bo $H_t(n)$ presek med polravnino $x \leq t$ in dekompozicijo $H(N)$. $H_t(n)$, ki se med procesom pregledovanja ravnine ves čas spreminja oziroma obnavlja, postane pri $x = +\infty$ enak $H(N)$. $H_t(n)$ se spreminja oziroma obnavlja vsakič, ko pregledovalna linija L preide kakšno izmed karakterističnih točk (krajišče linijskega segmenta, presečišče dveh segmentov). Pregledovalna linija L tako napreduje od $x = -\infty$ proti $x = +\infty$, od ene karakteristične točke do druge. V vsakem trenutku je potrebno vedeti, katera bo naslednja karakteristična točka, ki jo bo pregledovalna linija L zadela. Vse karakteristične točke so popisane v t.i. popisu dogodkov. Razvrščene so po naraščujoči koordinati x. Tako je v vsakem trenutku $x = t$ znano, katera karakteristična točka je prva na desni strani pregledovalne linije L_t. Presečišč linijskih segmentov v prvi fazi v tem seznamu ni. Ta so določljiva s pomočjo mejnega seznama. V mejnem seznamu so zapisani vsi linijski segmenti, ki jih v danem trenutku $x = t$ pregledovalna linija L_t seka. Za Sliko 3.14 je mejni seznam {a, b, c, e, d, f, g, }. Do presečišča lahko pride le med segmentoma, ki sta si v mejnem seznamu sosednja. Glede na to (Slika 3.14), da se razdalja med segmentoma a in b, ko se pregledovalna linija L približuje presečišču v, manjša, se lahko naslednje presečišče natančno določi vnaprej. Isto velja za vse ostale pare v mejnem seznamu. Ko je presečišče določeno, se ga pripiše v popis dogodkov. Na začetku pregledovanja ravnine sta $H(N)$ in mejni seznam prazna. V popisu dogodkov so krajišča vseh elementov niza N razvrščena po naraščujoči koordinati x. Vsakič, ko pregledovalna linija L preide katero karakterističnih točk, se ta v popisu dogodkov zbriše. To pomeni, da se zbriše točka z najmanjšo vrednostjo x. Ko pregledovalna linija L preide karakteristično točko p, se dogaja naslednje (Slika 2.17):



Slika 3.15 Pregledovanje (a) leve krajiščne točke linijskega segmenta e; (b) desne krajiščne točke linijskega segmenta e; (c) presečišča med linijskima segmentoma g in f.

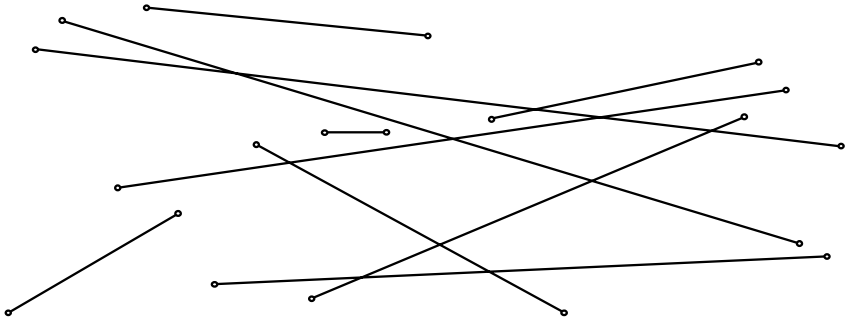
1. Karakteristična točka p je levo krajišče linijskega segmenta $e \in N$. V tem primeru se segment e doda mejnemu seznamu. Segmenta a in b, za katera se predpostavi, da vedno obstajata (lahko se ju umetno doda nad in pod vse ostale segmente niza N), sta v popravljenem mejnem seznamu pred in za segmentom e. V tem trenutku se spremeni tudi $H_t(n)$, ki se mu doda segment e in vertikalna povezava med segmentoma a in b, ki gre skozi karakteristično točko p. V mejnem seznamu sta sedaj segmenta a in e sosednja. Če se sekata na desni strani točke p, je to presečišče natančno določljivo. Ob predpostavki, da ga v popisu dogodkov še ni, se novo presečišče tja vpiše. Isto se naredi tudi za segmenta b in e.

2. Karakteristična točka p je desno krajišče linijskega segmenta $e \in N$. V tem primeru se segment e zbriše iz mejnega seznama. V $H_t(n)$ se doda vertikalna povezava med segmentoma a in b, ki gre skozi karakteristično točko p. Segmenta a in b postaneta v popravljenem mejnem seznamu sosednja. Če se sekata ne desni strani točke p, je to presečišče natančno določljivo. Ob predpostavki, da ga v popisu dogodkov še ni, se novo presečišče tja vpiše.

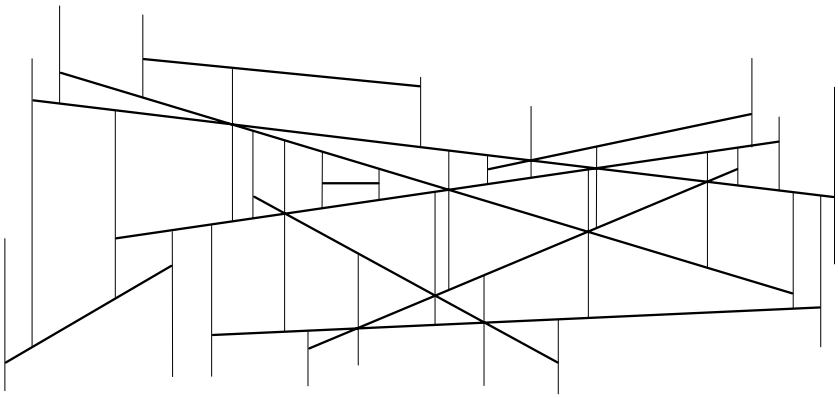
3. Karakteristična točka p je presečišče dveh linijskih segmentov e, $f \in N$. V popravljenem mejnem seznamu segmenta f in g zamenjata mesti. Popravi se $H_t(n)$, kamor se vpiše presečišče p in vertikalna povezava med segmentoma a in b, ki gre skozi karakteristično točko p. V popravljenem mejnem seznamu sta sedaj sosednja segmenta a in f ter b in g. Za oba para je potrebno preveriti obstoj presečišča desno od karakteristične točke p (kot opisano v točki 1.) in ob predpostavki, da ga v popisu dogodkov še ni, se novo presečišče tja vpiše.

3.5.2 Inkrementalni algoritem trapezne dekompozicije

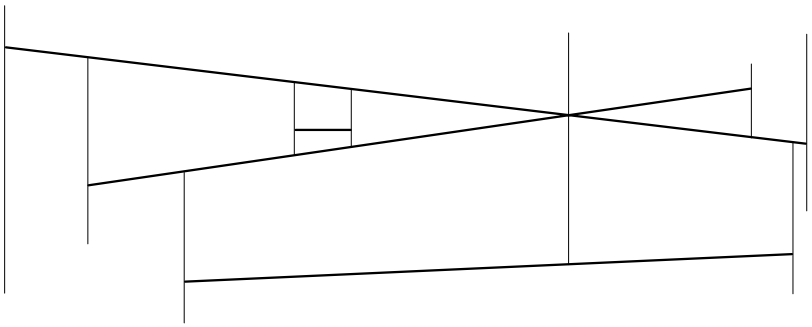
Inkrementalni algoritem trapezne dekompozicije [7] temelji na dodajanju linijskih segmentov v niz N po naključnem vrstnem redu. Za vsak na novo dodan linijski segment se dodajo karakteristične vertikale, ki so del trapezne dekompozicije. Naj bo N niz n linijskih segmentov na ravnini (Slika 3.16) in $H(N)$ končni rezultat razdelitve na trapeze (Slika 3.17). $H(N)$ nastaja inkrementalno z dodajanjem segmentov iz niza N, enega za drugim po naključnem vrstnem redu. V i-tem koraku algoritma je zgrajen $H(N^i)$, pri čemer N^i predstavlja niz prvih i naključno izbranih linijskih segmentov. Na Sliki 3.18 je prikazana trapezna dekompozicija za prve štiri linijske segmente.



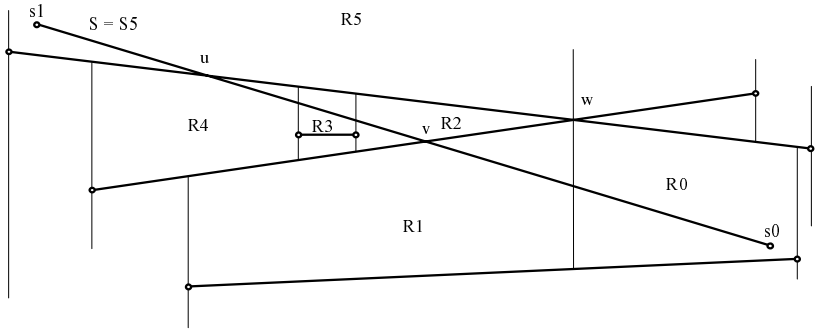
Slika 3.16 *N*: niz *n* linijskih segmentov na ravnini.



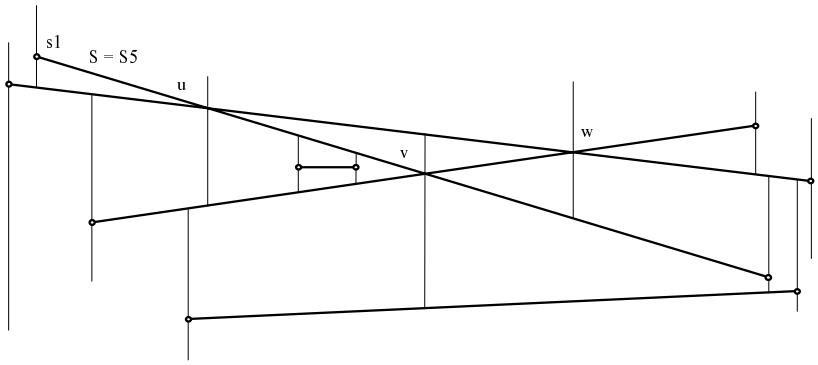
Slika 3.17 *H*(*N*): končni rezultat trapezne dekompozicije.



Slika 3.18 Trapezna dekompozicija $H(N^4)$ za prve štiri linijske segmente.

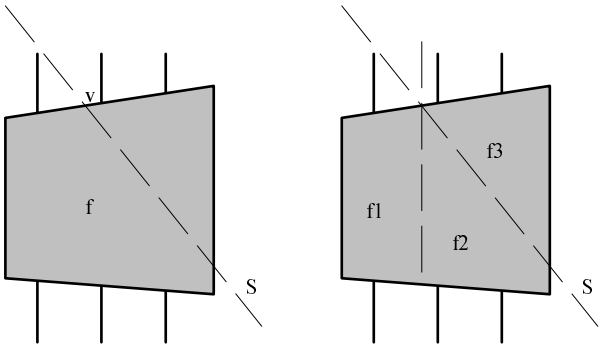


Slika 3.19 Potovanje vzdolž linijskega segmenta $S = S_5$ od točke s_0 proti točki s_1v razdelitvi $H(N^4)$.



Slika 3.20 $H'(N^4) = H(N^5)$.

Naj bo $H(N^4)$ trapezna dekompozicija po štirih dodanih linijskih segmentih. Naj bo točka s_0 eno dveh krajišč linijskega segmenta $S = S_5$. Predpostavimo, da točka s_0 leži v trapezu, ki je v $H(N^4)$ že določen. Za določitev karakterističnih točk je potrebno pregledati vse že določene trapeze R_0, R_1, ... v $H(N^4)$, preko katerih gre novododani linijski segment S_5. Potrebno je potovati vzdolž linijskega segmenta $S = S_5$ od krajišča s_0 proti krajišču s_1 in pri tem pregledovati območja R_0, R_1, ... (Slika 3.19). Naj bo f katerikoli trapez v $H(N^i)$, ki ga linijski segment S_{i+1} seka. V vsakem pregledanem trapezu je potrebno ugotoviti karakteristične točke (Slika 3.20) in skozi njih dodati vertikale (za vertikalno dekompozicijo).



Slika 3.21 Razdeljevanje trapeza *f*.

Kaj se dogaja, ko linijski segment S_{i+1} seka že določene trapeze, je prikazano na Sliki 3.21. Če linijski segment S preseka zgornjo ali spodnjo stranico trapeza f, potem je potrebno skozi to presečišče postaviti vertikalo, ki je sestavni del trapezne dekompozicije in je na obeh koncih omejena z dvema drugima že postavljenima linijskima segmentoma (na enem koncu se lahko konča v neskončnosti). Če krajišče segmenta S leži v notranjosti trapeza f, kot v primeru R_0 in R_5 na Sliki 3.19, potem je potrebno skozi to krajišče postaviti vertikalo. Če novododani segment S seka že postavljeno vertikalo v $H(N^4)$, je potrebno to vertikalo odrezati zgoraj, če je karakteristična točka pod S, ali spodaj, če je karakteristična točka nad S. Ko so pregledani vsi trapezi vzdolž segmenta S in skozi karakteristične točke dodane vse vertikale, je določena nova razvrstitev $H'(N^4) = H(N^5)$. Prikazana je na Sliki 3.20. Sledi dodajanje naslednjega linijskega segmenta iz niza N. Rezultat trapezne dekompozicije je prikazan na Sliki 3.17.

3.6 Razdeljevanje mnogokotnikov na konveksne površinske elemente

Mnogokotnike je mogoče, poleg triangulacije in razdelitve na trapeze, razdeliti tudi na konveksne površinske elemente. Pri tem se je potrebno držati dveh načel:

1. Razdeliti mnogokotnik na čim manj konveksnih površinskih elementov.
2. Izvesti to razdelitev v čim krajšem času.

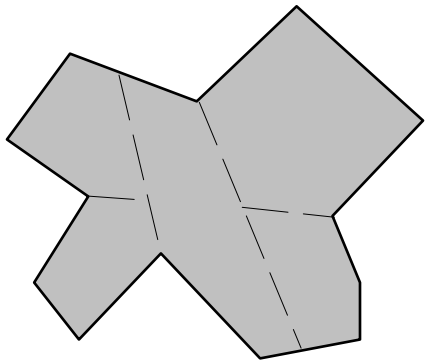
Ti dve načeli si nasprotujeta. Tako se možne rešitve razdeljevanja razlikujejo po tem, katero od obeh načel je upoštevano v večji meri. Poiskati je potrebno kompromisno rešitev med obema načeloma.
Dekompozicijo mnogokotnika P je mogoče narediti s postavljanjem diagonal ali s postavljanjem linijskih segmentov. Razlika je v tem, da so krajišča diagonal vedno temena v mnogokotnika P, za segmente pa je pomembno samo to, da njihova krajišča

ležijo nekje na meji mnogokotnika δP. Delitev s pomočjo segmentov je precej bolj zapletena, vendar ponuja optimalnejše rezultate.

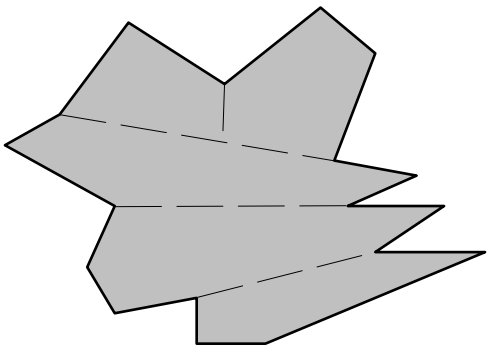
Teorem (Chazzele) [8] pravi:

Naj bo Φ najmanjše število konveksnih površinskih elementov. Za mnogokotnik z r vbočenimi temeni velja, $\lceil r/2 \rceil + 1 \leq \Phi \leq r + 1$.

Če se skozi temena, ki imajo notranji kot > π (vbočeno), potegnejo diagonale, nujno nastaneta dva konveksna površinska elementa. Število takih elementov je $r + 1$. Z eno diagonalo je torej mogoče dobiti največ dva taka elementa. Rezultat je $\lceil r/2 \rceil + 1$ konveksnih elementov (Slika 3.22, Slika 3.23).



Slika 3.22 $r + 1$ konveksnih elementov: $r = 4$; 5 elementov.

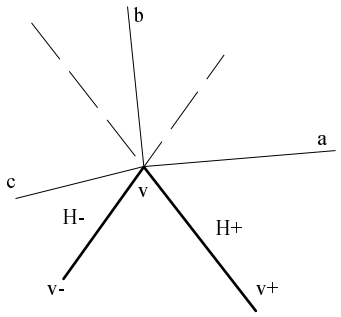


Slika 3.23 $\lceil r/2 \rceil + 1$ konveksnih elementov: $r = 7$; 5 elementov.

Tako dekompozicijo je mogoče dobiti z odvzemanjem nepomembnih diagonal. Diagonala d je za teme v nepomembna, če njena odstranitev ne ustvari površinskega elementa, ki bi bil v temenu v vbočen. Teme v mora biti tako v vsakem primeru vbočeno. Vsako vbočeno teme ima največ dve pomembni diagonali. Diagonala, ki ni pomembna za nobeno teme, je nepomembna digonala.

Algoritem je preprost. Najprej je potrebno mnogokotnik P razdeliti na trikotnike in po vrsti odvzemati vse nepomembne diagonale. Na Sliki 3.24 je prikazano vbočeno teme v. Sosednji temeni sta v_+ in v_-. Levo od v_-, v je polravnina H_-, desno od v, v_+ je

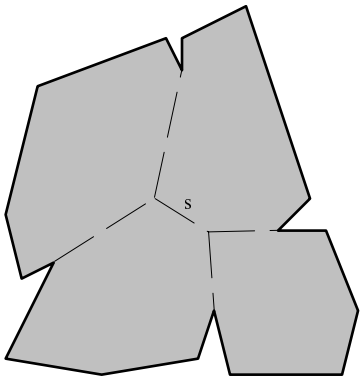
polravnina H_+. Na polravnini H_+ je lahko največ ena pomembna diagonala. Če sta diagonali dve, je tista, ki je bližja (v, v_+), nepomembna. Njena odstranitev ne povzroči vbočenosti površinskega elementa v temenu v. Podobno velja tudi za polravnino H_-.



Slika 3.24 Pomembne diagonale. Diagonala a je nepomembna, saj diagonala b v polravnini H_+ še vedno zagotavlja izbočenost v temenu v. Podobno je tudi c nepomembna diagonala.

3.6.1 Optimalna konveksna razdelitev mnogokotnika

Razdeljevanje s segmenti je precej bolj zapleteno, vendar ponuja optimalnejše rezultate. Mogoča je tudi delitev, pri kateri se delitveni segmenti s meje mnogokotnika δP ne dotikajo (Slika 3.25).



Slika 3.25 Optimalna konveksna razdelitev mnogokotnika.

3.7 Ravninski elementi v prostoru

Za popisovanje teles v prostoru se prav tako uporabljajo zgoraj našteti elementi, le da so njihove funkcije nekoliko spremenjene. Osnovni element je še vedno točka, določena s tremi kordinatnimi vrednostmi in ne dvema, kot na ravnini:

točka v prostoru $p_i = [x_i, y_i, z_i] \in R^3$.

Mnogokotnik je še vedno območje na ravnini, ki je lahko katerakoli ravnina v prostoru; razmejitev med notranjostjo in zunanjostjo mnogokotnika določa niz linijskih segmentov. Razmejitev med notranjostjo in zunanjostjo prostorskih teles pa ne določajo več linijski segmenti mnogokotnika, ampak površinski elementi, ki jih ti mnogokotniki določajo. Načeloma so lahko ti površinski elementi poljubni mnogokotniki, vendar so to ponavadi konveksni mnogokotniki, največkrat trikotniki ali konveksni štirikotniki.
Pri rekonstrukciji površine, opisani v 8. poglvju, so uporabljeni površinski elementi, ki ločujejo prostorsko notranjost od zunanjosti, Delaunayevi trikotniki.

4 Lupine v 2D

Če gre pri mnogokotnikih v grobem za razdeljevanje ravnine na dva dela, t.j. na zunanjost in notranjost mnogokotnika, so lupine že bolj kompleksne strukture. Ne samo, da predstavljajo eno osrednjih področij računske geometrije, temveč je z njimi mogoče rešiti tudi velik del ostalih problemov, ki se pojavljajo z razvojem računske geometrije.
Konveksna lupina je uporabna kot samostojna struktura, ali pa kot pripomoček za graditev kompleksnejših oblik in preko tega za proučevanje njihove geometrije in topologije.
Pojem konveksnosti je načeloma določljiv nedvoumno, za prepoznavanje konveksnosti in konveksnih lupin pa kljub temu obstaja več različnih definicij. Nekatere izmed njih so:

1. Nizu elementov R^d pravimo konveksen, če povezava med katerimakoli elementoma (točkama) tega niza leži v vsej svoji dolžini v njegovi notranjosti. Najenostavnejši niz v R^d je polprostor (polravnina).
2. Konveksna lupina, sestavljena iz niza točk N, je presek vseh polravnin, ki vsebujejo ta niz N.
3. Konveksna lupina, sestavljena iz niza točk N na ravnini, je najmanjši konveksni mnogokotnik, ki obdaja niz N. V takem primeru ne obstaja noben manjši mnogokotnik P', za katerega bi veljalo $P \supset P' \supseteq N$.
4. Konveksna lupina, sestavljena iz niza točk N na ravnini je unija vseh trikotnikov, ki jih določajo točke niza N.
5. Konveksen niz točk $x_1,..., x_k$ je vsota $\alpha_1 x_1 + ... + \alpha_\kappa x_k$, pri čemer mora veljati $a_i \geq 0$ za vse i in $\alpha_1 + ... + \alpha_\kappa = 1$. Linijski segment je konveksna kombinacija lastnih krajišč; $k = 2$, trikotnik je konveksna kombinacija lastnih temen, $k = 3$, tetraeder je konveksna kombinacija lastnih temen; $k = 4$. Vsi ti elementi so sami po sebi konveksni.

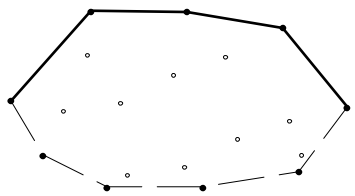
Pri reševaju problemov s področja lupin je glavno vprašanje, kaj naj bo rezultat oz. kako naj bo oblikovan izhodni podatek. Naj bo N niz n-tih neurejenih točk na ravnini. Kot izhoden podatek je možen seznam vseh skrajnih točk niza N, ali pa določena meja mnogokotnika. Skrajne točke niza N so temena konveksne lupine, meja mnogokotnika pa je konveksen mnogokotnik, ki niz točk na ravnini s konveksno lupino omejuje - razmejuje od okolice. Razlika je v tem, da se s podatkom o meji dobi niz točk, ki po

vrstnem redu določajo lupino, ponavadi v nasprotni smeri urinih kazalcev, pri seznamu skrajnih točk pa so ti podatki neurejeni.

Za določanje konveksnih lupin obstaja cela vrsta algoritmov. V splošnem se delijo na deterministične in naključne. Tu se pokaže prednost naključnih algoritmov, saj so v splošnem razširljivi na večdimenzionalne prostore, medtem ko ni nujno, da bodo deterministični algoritmi, ki so uporabni na ravnini, uporabni tudi v prostoru. Če že, je lahko njihovo izvajanje zapleteno ali zelo počasno. V nadaljevanju sta na kratko predstavljena po dva deterministična in dva naključna algoritma.

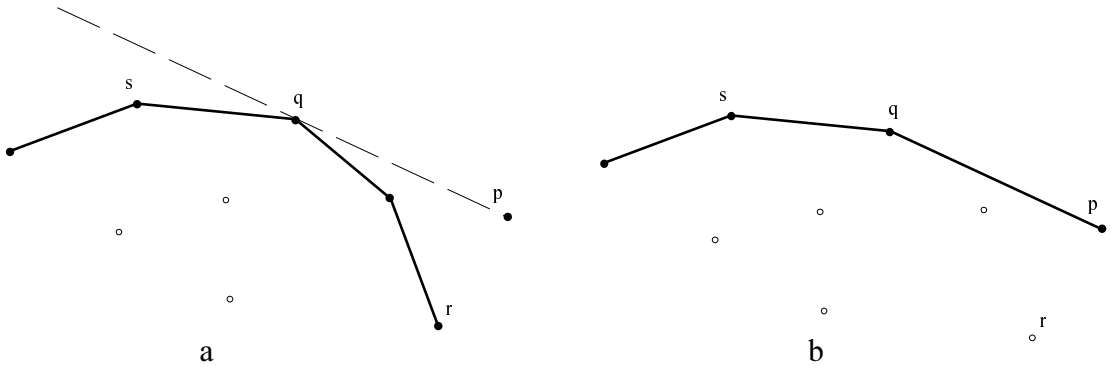
4.1.1 Deterministični algoritem za konstrukcijo konveksne lupine v 2D

Pri prvem deterističnem načinu je potrebno konveksno lupino razdeliti na zgornjo in spodnjo verigo [7] (Slika 4.1).



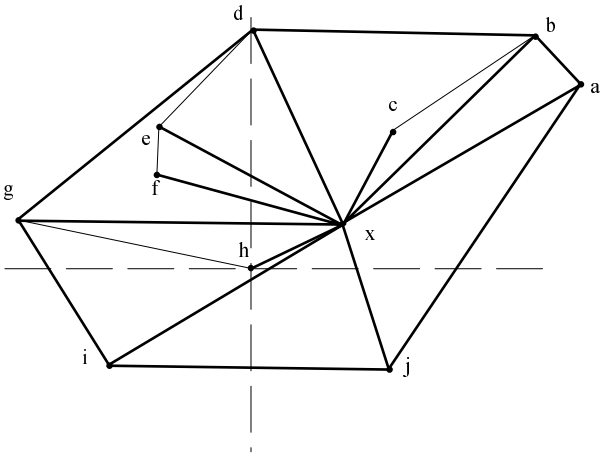
Slika 4.1 Zgornja in spodnja veriga.

Ločnici med njima sta točki z najmanjšo in največjo vrednostjo koordinate x. Točke iz niza N je potrebno razporediti po naraščujoči koordinati x. Glavna ideja je, da se nizu N^i, ki vsebuje prvih i že pregledanih točk, dodajajo naslednje točke, za katere se preverja, ali ležijo na konveksni lupini ali v njeni notranjosti.
Naj bo N^i niz prvih i točk, naj bo $U(N^i)$ zgornja veriga, ki je že določena v okviru N^i. Naslednja točka, ki jo je potrebno pregledati, je $p = p_{i+1}$. Zadnja, za katero je ugotovljeno, da je del verige $U(N^i)$, naj bo točka r. Iz točke r je potrebno potovati po meji $U(N^i)$ v levo in pregledovati že postavljene točke na zgornji verigi. Pri točki q je pregledovanje končano, saj leži naslednja točka s pod linijo pq. Očitno je, da je v zgornjo verigo $U(N^{i+1})$ potrebno pripisati povezavo med točkama p in q ter zbrisati vse točke iz že določene verige $U(N^i)$, ki so desno od točke q (Slika 4.2). Enak postopek je potrebno ponoviti še za določanje spodnje verige.



Slika 4.2 Dodajanje nove točke. (a) $U(N^i)$; (b) $U(N^{i+1})$.

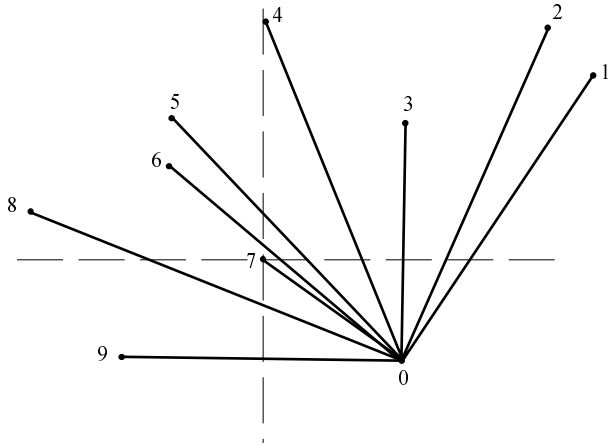
Drugi deterministične algoritem je znan kot Graham-ov algoritem [8]. Vhodni podatek je niz *n*-tih točk *N*, za katerega je potrebno ugotoviti konveksno lupino. Ideja temelji na izbiri neke poljubne točke *x* v notranjosti meje mnogokotnika in ureditvi vseh točk niza *N* po naraščujočem relativnem kotu v nasprotni smeri urinih kazalcev (Slika 4.3).



Slika 4.3 Točka *x* je poljubna, ostale so označene glede na naraščujoči kot α; $\alpha_a \leq \alpha_b \leq \alpha_c \leq \alpha_d \leq \ldots \leq \alpha_j$.

Naslednji korak je pregledovanje že urejenih točk na ravnini. Prvi točki niza *S*, ki določa konveksne lupine, sta $S = (a, b)$. Naslednja točka, ki sledi glede na naraščujoče kote, je točka *c*. Konveksnost v vsaki točki se določa z že omenjeno metodo hodca po meji mnogokotnika. Ker hodec po že postavljeni lupini naredi na poti *a*, *b*, *c* v točki *b* zavoj v levo, je točka *b* gotovo izbočeno teme in točka *c* se pripiše nizu $S = (a, b, c)$. Kot naslednjo je potrebno preveriti točko *d*. Na poti *b*, *c*, *d* naredi hodec v temenu *c* zavoj v desno. To pomeni, da je *c* gotovo vbočeno teme in kot tako ne more biti del meje konveksnega mnogokotnika oz. konveksne lupine. Točka *c* se iz niza *S* izbriše,

vpiše se točka *d*, *S* = (*a*, *b*, *d*). Ta postopek je potrebno ponavljati, dokler v nizu *S* prvi in zadnji element nista enaka; *S* = (*a*, *b*, *d*, *e*, *g*, *i*, *j*, *a*).
Glavni problem je, da ni moč z gotovostjo trditi, da sta točki *a* in *b* na meji konveksnega mnogokotnika oz. na konveksni lupini. Tako bi se v primeru, da nista, postopek preverjanja nadaljeval naprej, kljub temu, da je bil pregledan že ves niz *N*. Med začetkom in koncem niza *S* ne bi bilo logične povezave.
Problem nastane tudi v primeru, če je že v prvem koraku, pri trojici *S* = (*a*, *b*, *c*), ugotovljeno, da je teme *b* nedvoumno vbočeno, t.j. da hodec v tem temenu naredi obrat v desno. Točka *b* tako ne more biti del niza *S*. V tem primeru je potrebno *b* iz niza *S* = (*a*, *b*) zbrisati. Ostane samo *S* = (*a*). Z eno samo točko naslednje točke ni mogoče preverjati, potrebni sta najmanj dve.
Rešitev je, če se za naključno točko *x* izbere najnižjo (po koordinati *y*) točko niza *N*. Če sta taki točki dve, naj bo izbrana najbolj desna, najnižja točka niza *N*. Zaporedno urejanje po naraščujočih kotih je enako kot prej omenjeno. Glede na velikost relativnega kota naj bodo točke zdaj označene s $p_0, p_1, ..., p_{n-1}$ v nasprotni smeri urinih kazalcev. V tem primeru je logično,da so točke p_0, p_1 in p_{n-1} na meji konveksnega mnogokotnika oz. na konveksni lupini. S tem sta odpravljena oba prej omenjena problema. Novo označevanje in urejanje je prikazano na Sliki 3.4.

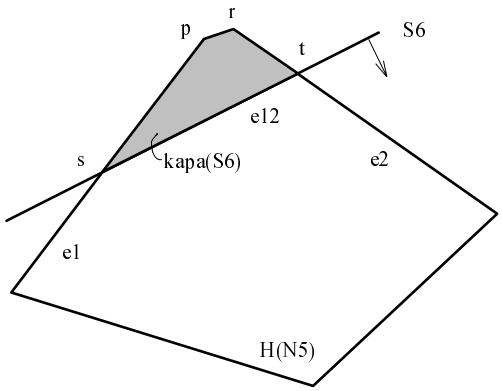


Slika 4.4 Osnova urejanja je točka 0, (na Sliki 4.3 točka *j*).

4.1.2 Naključni algoritem za konstrukcijo konveksne lupine v 2D

Prvi naključni algoritem temelji na dodajanju polprostorov R^d, v tem primeru polravnin R^2, v ureditev $H(N)$. $H(N)$ predstavlja presek vseh že dodanih polravnin. Polravnina je najosnovnejša konveksna lupina. Presek konveksnih lupin da vedno novo konveksno lupino. Presek polravnin je torej konveksna lupina [7].

Naj bo N niz polravnin in $H(N^i)$ konveksna lupina, t.j. presek vseh polravnin po dodani polravnini $S = S_i$. Naslednji korak je dodajanje (i+1). polravnine $S = S_{i+1}$ k ureditvi $H(N^i)$ (Slika 4.5).



Slika 4.5 Dodajanje polravnine $S = S_{i+1}$ k ureditvi $H(N^i)$.

Ureditev $H(N^{i+1})$ bo nastala iz $H(N^i)$ po odstranitvi kape i+1, ki je presek $H(N^i) \cap \overline{S}$, pri čemer je polravnina $\overline{S}$ komplementarna polravnini S. Najprej je potrebno izbrati poljubno teme $p \in H(N^i)$, ki leži na komplementarni ravnini $\overline{S}_{i+1}$. Če tako teme ne obstaja, se lahko ravnino $S = S_{i+1}$ izpusti, saj ne vpliva na $H(N^{i+1}) = H(N^i) \cap S_{i+1}$. Če tako teme obstaja, je potrebno pregledati robove ureditve $H(N^i)$ in ugotoviti presečišči (vedno sta dve) med $H(N^i)$ in mejo polravnine S_{i+1} (oziroma $\overline{S}_{i+1}$).

Pregledovanje $H(N^i)$ se začne v temenu p. Naj bo f prvi rob, ki ga je potrebno pregledati. Če f v celoti leži na polravnini $\overline{S}$, potem ga je potrebno s pripadajočima temenoma odstraniti. Prvi rob f, ki leži na $\overline{S}$ samo delno, je hkrati tudi zadnji, ki ga je potrebno pregledati v tej smeri. Na Sliki 4.5 je to rob e1. Ta rob f se razdeli na dva dela tako, da se tisti del, ki leži na polravnini $\overline{S}$, izbriše, druga polovica pa je hkrati del ureditve $H(N^{i+1})$. Pregledovanje je potrebno ponoviti tudi v drugi smeri, da se ugotovi še drugi rob, ki leži hkrati na polravninah $\overline{S}$ in S. Drugi rob na Sliki 4.5 je e2. Od ureditve $H(N^i)$ je potrebno odstraniti kapo, del δS_{i+1}, ki leži med obema odrezanima robovoma f, ki postaneta nova robova ureditve $H(N^{i+1})$. Tretji nov rob ureditve $H(N^{i+1})$ je rob polravnine S_{i+1}, ki je na obeh koncih omejen z že določenima robovoma e1 in e2.

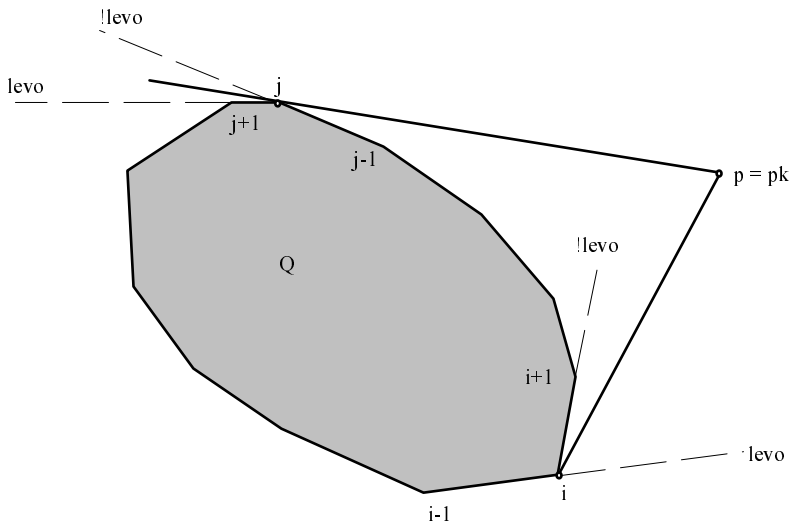
Drugi naključni algoritem je prav tako kot prejšnji inkrementalen, vendar gre v tem primeru za dodajanje točk [8] (v prejšnjem primeru je šlo za dodajanje polravnin). Algoritem je uporaben tudi v večdimenzionalnem prostoru, pri konstruiranju konveksnih lupin v 3D, kar je eno pomembnejših področij računalniške geometrije.

Osnovna ideja je dodajanje točk ene za drugo h konveksni lupini, ki jo že določa prvih k dodanih točk oziroma dodajanje točke k obstoječi konveksni lupini. Naj bo $P = \{p_0,$

p_1, ..., p_{n-1}} niz točk na ravnini. Prva konveksna lupina je trikotnik, sestavljen iz točk $konv\{p_0,p_1,p_2\}$. Naj bo $Q = H_{k-1}$ in $p = p_k$. Določanje konveksne lupine $konv\{Q \cup p\}$ se razširi na dve možnosti v odvisnosti ali $p \in Q$ ali $p \notin Q$:

1. $p \in Q$. Če je ugotovljeno, da leži točka p v območju Q, potem se jo lahko izpusti, saj ne vpliva na obliko konveksne lupine. Isto velja, če točka p leži na meji Q t.j. na δQ. Za določanje notranjosti oziroma zunanjosti se uporabi že omenjeno t.i. predpostavko levega (3.4.1). Za vsak rob konveksne lupine Q posebej se preverja ali je točka na njegovi levi strani. Če to velja preko cele meje območja δQ, potem je točka p v notrajnosti Q.

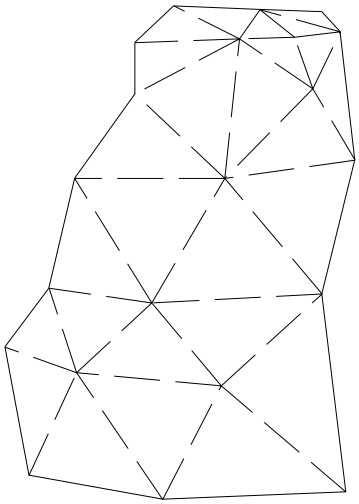
2. $p \notin Q$. Če se za katerikoli rob izkaže, da predpostavka levega ne drži, potem velja $p \notin Q$ in mogoča je določitev $konv\{Q \cup p\}$. Potrebno je poiskati dve tangenti, ki povezujeta točko p in območje Q in oblikovati novo konveksno lupino (Slika 4.6).



Slika 4.6 Tangenti od točke p h Q; "levo" pomeni, da je točka p levo od nakazane smeri in "!levo", da točka p ni levo od nakazane smeri.

Tangenta, ki povezuje točko p_k z območjem Q, se dotika meje konveksne lupine Q v eni sami točki. Naj bo p_i ena izmed takih točk. Tudi v tem primeru se točko lahko določi z uporabo predpostavke levega. Na Sliki 4.6 je vidno, da je točka p levo od smeri $p_{i-1}p_i$ in desno oziroma "nelevo" od smeri p_ip_{i+1}. Za zgornjo tangento velja ravno obratno. Točka p desno od smeri $p_{j-1}p_j$ in levo od smeri p_jp_{j+1}. Sledi, da je točka p_j točka tangente, če se za dve sosednji točki dobi drugačen rezultat predpostavke levega. Ostane še konstrukcija nove konveksne lupine, ki je (p_0, p_1, ..., p_{i-1}, p_i, p, p_j, p_{j+1}, ..., p_{n-1}) .

4.2 Nekonveksna lupina niza točk na ravnini



Slika 4.7 Lupina niza točk N, $n = 21$.

Na Sliki 4.7 je prikazana nekonveksna lupina niza točk N. Narejena je s pomočjo algoritma, ki je opisan v osmem poglavju, in predstavlja projekcijo točk v prostoru na ravnino. Črtkani robovi med posameznimi točkami niza N predstavljajo Delaunayevo triangulacijo.

5 Konveksne lupine v 3D

Na ravnini je konveksna lupina določena kot ovoj, ki se ga lahko ovije okoli vseh ekstremnih točk niza N tako, da so vse točke niza v notranjosti ovoja ali vsaj na njegovi meji (te so ekstremne točke niza N). Lupina je torej sestavljena iz točk na ravnini (temen), ki so med seboj povezane z linijskimi segmenti (robovi). Tudi v prostoru je lupina ovoj, ki ga je mogoče oviti okoli vseh ekstremnih točk niza N tako, da so vse prostorske točke v njegovi notranjosti ali vsaj na njegovi meji. V prostoru je lupina sestavljena iz množice točk (temen), ki so med seboj povezane z linijskimi segmenit (robovi), ti pa med seboj tvorijo ploskve. Temena robov so hkrati tudi temena ploskev. Podobno kot je lupina v 2D določena kot presek polravnin R^2, je konveksna lupina v 3D določena kot presek polprostorov R^3.

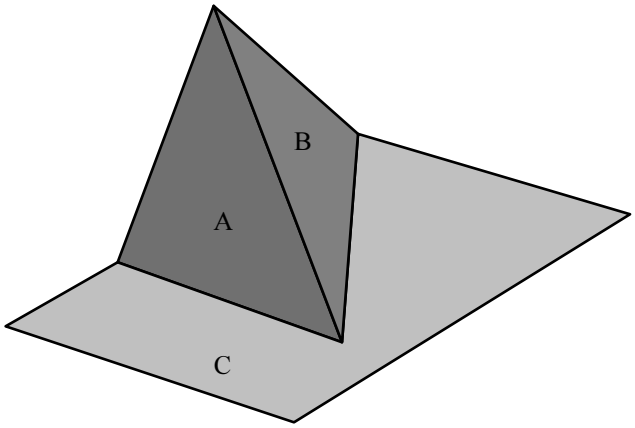
5.1 Polieder

Polieder je naravno posploševanje ravninskega mnogokotnika v prostor, je območje v prostoru, katerega meja vsebuje končno število večstranih ploskev, ki so lahko nepovezane, ali pa se med seboj dotikajo v robovih in temenih. Ta definicija je precej nejasna, še posebej, kadar ima opraviti z lupinami na splošno.
Meja poliedra je sestavljena iz treh vrst geometrijskih objektov: ničdimenzionalnih temen (točke), enodimenzionalnih robov (linijski segmenti) in dvodimenzionalnih površin (mnogokotniki). Površino poliedra je mogoče natančno označiti z odnosi med posameznimi geometrijskimi elementi. Pri tem so upoštevani trije pogoji: elementi se morajo med seboj sekati "pravilno", lokalna topologija mora biti "pravilna", globalna topologija mora biti "pravilna" [8].

1. Elementi se sekajo "pravilno".

Za vsak par ploskev se zahteva, ali
(a) se med seboj ne dotikata, ali
(b) imata eno skupno teme, ali pa
(c) imata dve skupni temeni in en skupen rob, ki ti dve temeni povezuje med seboj.

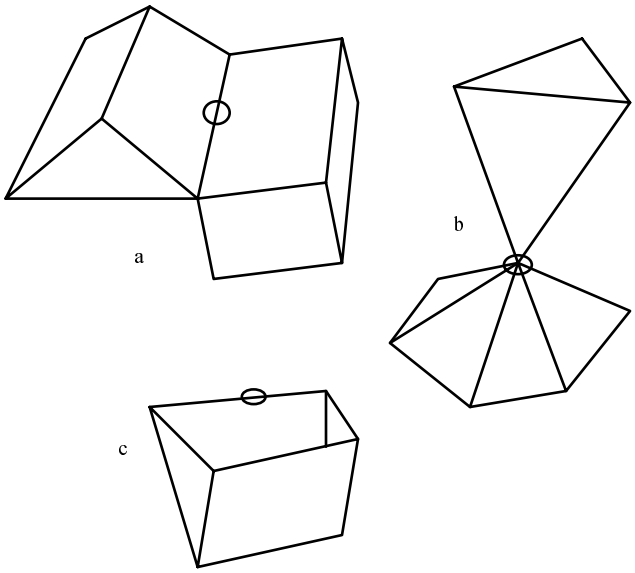
Nepravilno sekanje je torej prediranje ploskev in napačno dotikanje ploskev med seboj (Slika 5.1). S tem, ko so določena presečišča med ploskvami, so določena tudi presečišča med robovi in temeni.



Slika 5.1 Ploskev *B* prebada ploskev *C*; ploskvi *A* in *C* se dotikata napačno.

2. Lokalna topologija je "pravilna".

Lokalna topologija popisuje površino v okolici točke. Tehnično se ta omejenost popiše, da naj bo okolica vsake točke na površini homeomorfična disku. Homeomorfnost dopušča raztezanje in upogibanje, ne dopušča pa pretrganja (Slika 5.2).



Slika 5.2 Telesa na sliki niso poliedri. V vseh treh primerih okolica točke ni homeomorfična disku: (a) točka leži hkrati na dveh ploskvah; (b) triangulacija ne da enostavno zaprte povezave med elementi; (c) objekt ni zaprt, okolica točke je pol disk.

3. Globalna topologija je "pravilna".

Površina mora biti zvezna, zaprta in omejena. To pomeni, da namišljeni hodec lahko prehodi katerokoli pot med katerikolima točkama na tej površini. V tem smislu so

dovoljeni tuneli oziroma različne udrtine v lupino. Obravnavanje konveksnih lupin ne zajema teh dveh pojavov.

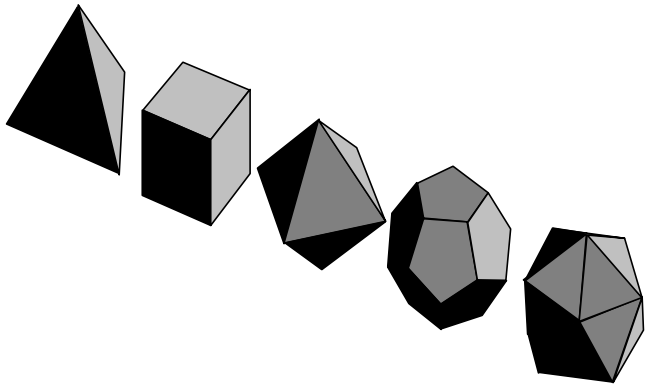
Upoštevaje vse skupaj sledi: meja poliedra je končen zbir ravninskih, konveksnih, omejenih poligonov na način, da:

1. se ploskve sekajo pravilno,
2. je okolica vsake točke homeomorfična oblika diska oziroma, da je povezava med temeni preprosta poligonalna veriga,
3. da je površina zvezna.

Meja je torej zaprta in obdaja oziroma omejuje območje prostora. Vsak rob si delita natančno dve ploskvi, ki se imenujeta sosednji. Za konveksen polieder velja, da je segment, ki povezuje katerikoli točki poliedra v notranjosti tega poliedra. Konveksnost je popisljiva tudi z notranjimi koti. Notranji kot med dvema sosednjima ploskvama je vedno $\leq \pi$ in vsota vseh takih kotov okoli vsakega temena je $\leq 2\pi$.
Pravini poliedri so tisti, katerih ploskve so skladni, pravilni mnogokotniki: enakostraničen trikotnik, kvadrat, pravilni petkotnik, itd. Število ploskev, ki se stikajo v enem temenu je enako za vsa temena pravilnega poliedra. Obstaja samo pet značilno pravilnih poliedrov, kar je prvi odkril Platon [8]. Če je p število temen na ploskev in v število ploskev, ki se dotikajo v temenu, mora za pravilni polieder veljati $(p - 2) \cdot (v - 2) \leq 3$ (Slika 5.3). V Tabeli 5.1 so prikazane vrednosti za pravilne poliedre.

Tabela 5.1: Seznam vrednosti p in v.

p	***v***	***(p-2)(v-2)***	**ime**	**opis**
3	3	1	tetraeder	trije trikotniki na teme
4	3	2	kocka	trije kvadrati na teme
3	4	2	oktaeder	štirje trikotniki na teme
5	3	3	dodekaeder	trije petkotniki na teme
3	5	3	ikozaeder	pet trikotnikov na teme



Slika 5.3 Pravilna Platonova telesa.

Medsebojen odnos med številom temen, robov in ploskev popisuje Eulerjeva formula $V - E + F = 2$ (Tabela 5.2).

Tabela 5.2: Število temen, robov in ploskev petih pravilnih poliedrov.

Ime	***(p, v)***	***V***	***E***	***F***
tetraeder	(3, 3)	4	6	4
kocka	(4, 4)	8	12	6
oktaeder	(3, 4)	6	12	8
dodekaeder	(5, 3)	20	30	12
ikozaeder	(3, 5)	12	30	20

5.2 Inkrementalni algoritem za konstrukcijo konveksne lupine v 3D

Iz množice točk je mogoče določiti polieder s pomočjo inkrementalnega algoritma. Prej opisani algoritem je zadostoval za konstrukcijo konveksne lupine iz množice točk na ravnini, naslednji pa omogoča konstruiranje konveksne lupine iz množice točk v prostoru [8].
Glavna ideja je identična: pri *i*-tem dodajanju točke je potrebno določiti $H_i \Leftarrow konv(H_{i-1} \cup p_i)$. Problem določanja se razdeli na dva dela. Naj bo točka p = p_i in Q = H_{i-1}. Ugotoviti je potrebno, če velja $p \in Q$. Če to drži, se lahko točko p izpusti, če ne, je potrebno določiti 'stožec', ki je tangenten na Q in ima vrh v točki p ter skonstruirati novo konveksno lupino. Test $p \in Q$ se izvede podobno kot v dveh dimenzijah. Točka p je v notranjosti Q, če je p na pozitivni strani vsake izmed ploskev, ki določajo lupino Q. Testiranje predpostavke levega omogoča volumen tetraedra, kakor to pri dveh dimenzijah omogoča površina trikotnika. Če je točka p v notranjosti lupine Q, morajo imeti vsi volumni enak predznak. Problem nastane, kadar je točka zunaj lupine Q, saj je potrebno le to spremeniti.

V ravninskem primeru je bilo potrebno poiskati dve tangenti med točko p in lupino Q. V prostoru pa tangentne linije zamenjajo tangentne ploskve. Te ploskve skupaj oblikujejo stožčasto telo, ki je sestavljeno iz trikotnih ploskev, katerih skupni vrh je v točki p, osnova pa rob e na lupini Q. Iz točke p je nekaj ploskev na Q vidnih, nekaj pa nevidnih. Vse tiste ploskve, ki so vidne je potrebno odstraniti pri spreminjanju Q = H_{i-1} v H_i. Robovi vidnosti postanejo osnova za konstrukcijo stožca z vrhom v točki p. Rob e na lupini Q je torej tak rob, da je ploskev, ki povezuje e s p, tangenta na Q. Rob e je soseden dvema ploskvama. Ena teh je iz točke p vidna, druga ne. Rob e je torej na meji med vidnim in nevidnim delom Q.
Podobno je mogoče p razumeti kot izvor svetlobe. Del lupine Q bo osvetljen, drugi del pa bo ostal v senci. Rob e je tista linija, ki ločuje osvetljeni del od senčnega.
V nadaljevanju algoritma je potrebno poiskati vse ploskve, ki so vidne iz točke p. To se ugotovi s pomočjo volumna, ki ga določajo točke na lupini Q, to je trojica (a, b, c) in točka p. Če je ploskev (a, b, c) iz točke p vidna, potem je volumen tetraedra (a, b, c, p) gotovo negativen. Taka ploskev se odstrani, H_i pa se oblikuje iz nevidnega dela lupine in ploskovne povezave med mejnim robom e in točko p. V naslednjem koraku se izbere novo točko $p = p_i$ in celoten postopek preverjanja in ponovnega konstruiranja se ponovi, dokler niso pregledane vse točke niza N v prostoru.

Osnovni algoritem za računalniško konstrukcijo lupine v 3D iz niza točk:

```
    procedure: 3D inkr_alg(i)
begin pripisati H4 tetraedru (p0,p1,p2,p3);
for i := 4 until i < n do
    begin
    for za vsako ploskev f stopnje Hi-1 do
        begin
        vol := vol (f, pi);
        if vol < 0 then f je vidna
        if nobena ploskev vidna then
            izločiti pi; (* je v notranjosti Hi-1 *)
        else
            for za vsak mejni rob e stopnje Hi-1 do
                begin
                konstrukcija ploskve e,pi;
                end.
            for za vsako vidno ploskev f do
```

```
                begin
                briši f;
                end.
            Popravi H_i;
        end.
    end.
end.
```

V višjih dimenzijah postanejo problemi bolj zapleteni in težje predstavljivi. "Hiperprostor" si je najlaže predstavljati, če si znamo predstavljati preskoke iz ničdimenzionalnega prostora v enodimenzionalen in naprej v dvo in trodimenzionalnen prostor.

5.3 Višje dimenzije

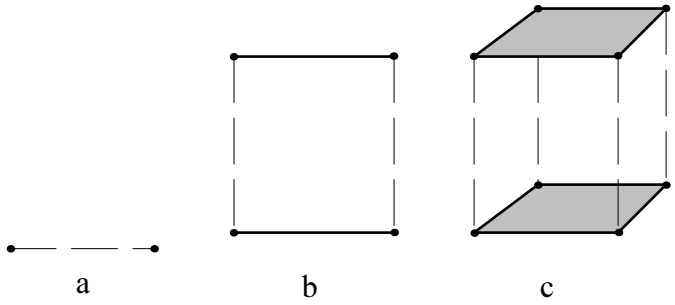
Predstavljanje prve, druge in tretje dimenzije je preprosto. Problem nastane pri višjih dimenzijah. Najboljši je postopen pristop skozi nižje dimenzije. Točka na številski premici je predstavljena z eno samo številko: je vrednost oziroma lokacija. Lahko se razume kot enodimenziinalen objekt, saj leži na premici, ki je enodimenzionalen element. Točka v dveh dimenzijah je določljiva z dvema koordinatama (x, y), v treh dimenzijah pa s tremi (x, y, z). Iz tega sledi, da so za točko v štirih dimenzijah potrebne štiri koordinate (x, y, z, t). Če se prve tri koordinate (x, y, z) razumejo kot prostorske koordinate in t kot čas, potem vse štiri skupaj predstavljajo dogodek v prostoru in času. Poleg prostor-časa obstaja še več možnih izpeljav višjih dimenzij. Ena izmed njih je primer hiperkocke. Ničdimenzionalna kocka je točka. Enodimenzionalna kocka je daljica. Dvodimenzijska kocka je kvadrat. Tridimenzionalna kocka je normalna kocka v prostoru. Pri tem veljajo naslednji podatki:

Tabela 5.4: Izpeljava točke v višje dimenzije.

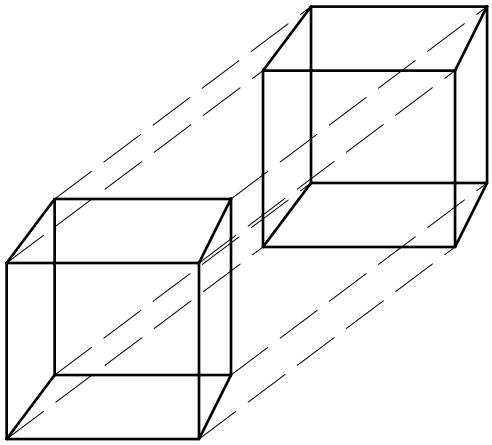
Dimenzija	**Ime**	V_d	E_d
0	točka	1	0
1	daljica	2	1
2	kvadrat	4	4
3	kocka	8	12
4	hiperkocka	16	31
d	d-kocka	2^d	$2E_{d\text{-}1}+V_{d\text{-}1}$

V je število temen in E število robov.

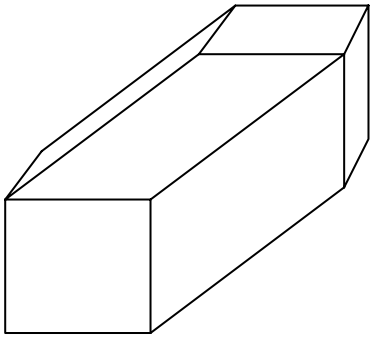
Kocka v *d* dimenzijah je zgrajena iz dveh kock (druga je kopija prve) dimenzije *d*-1. Če se enodimenzionalno kocko (točko) raztegne v drugo dimenzijo, nastane dvodimenzionalna kocka, daljica. Če se to naprej raztegne v smeri pravokotno sebi, nastane kvadrat. Če se kvadrat preslika na ravnino, ki je vzporedna prvi, nastane kocka (Slika 5.4). Če se kocko z osmimi temeni in dvanajstimi robovi preslika v nov položaj in obe kocki poveže z osmimi robovi, nastane t.i. hiperkocka (Slika 5.5). To je kocka v četrti dimenziji. Koordinate temen (vseh je šestnajst) predstavljajo konveksno lupino (Slika 5.6).



Slika 5.4 Kocka kot kvadrat potegnjen v tretjo dimenzijo.



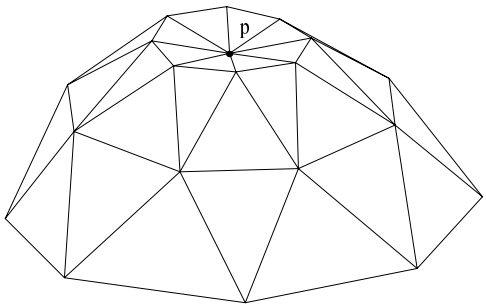
Slika 5.5 Hiperkocka kot kocka potegnjena v četrto dimenzijo.



Slika 5.6 Konveksna lupina hiperkocke.

5.4 Nekonveksna lupina niza točk v prostoru

Iz točk v prostoru je mogoče določiti tudi nekonveksno lupino. V tem primeru zoraj opisani algoritmi odpovejo, saj vsako točko, ki določa lokalno konkavnost, postavijo v notranjost lupine. Za konstrukcijo nekonveksne lupine (Slika 5.7) je bil uporabljen algoritem, opisan v poglavju osem.



Slika 5.7 Lupina v prostoru z lokalno konkavnostjo okoli točke *p*.

6 Voronoi diagram

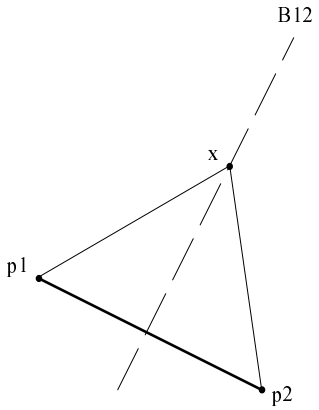
Voronoi diagrami obravnavajo sosedske odnose med točkami niza *N*: katera točka je kateri najbližja, katera je od katere najbolj oddaljena, itd.

6.1 Definicija voronoi diagrama

Naj bo $P = \{p_1, p_1, ..., p_n\}$ niz točk na ravnini. Voronoi diagram $V(p_i)$ ravnino razdeli tako, da je diagram $V(p_i)$ sestavljen iz vseh točk, ki so vsaj tako blizu točki p_i, kot so blizu katerikoli drugi točki p_k [8]:

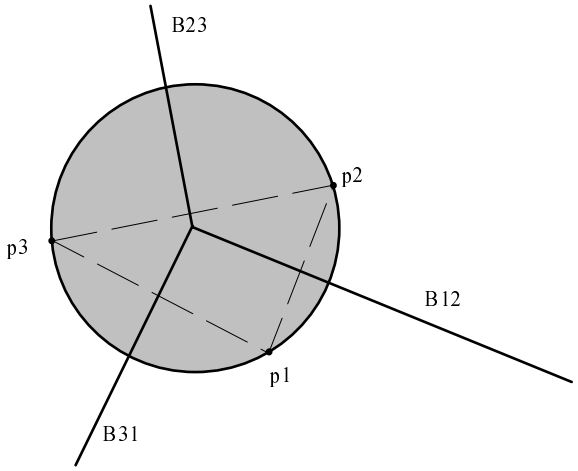
$$V(p_i) = \{x: |p_i - x| \leq |p_j - x|, \forall j \neq i\}.$$

Naj bosta na ravnini samo dve točki p_1 in p_2. Naj bo premica $B(p_1, p_2) = B_{12}$, ki pravokotno razdeli segment p_1p_2 na dva enaka dela. Potem je vsaka točka x na premici B_{12} enako oddaljena od točke p_1, kakor tudi od točke p_2. Velja: $|p_1x| = |p_2x|$ (Slika 6.1).



Slika 6.1 Dve točki na ravnini: $|p_1x| = |p_2x|$.

Če ležijo na ravnini tri točke (p_1, p_2, p_3), te definirajo trikotnik. Razpolovnice stranic trikornika B_{12}, B_{23} in B_{31} se sekajo v točki, ki je središče očrtanega kroga (Slika 6.2). Ta točka ni nujno v notranjosti trikotnika.

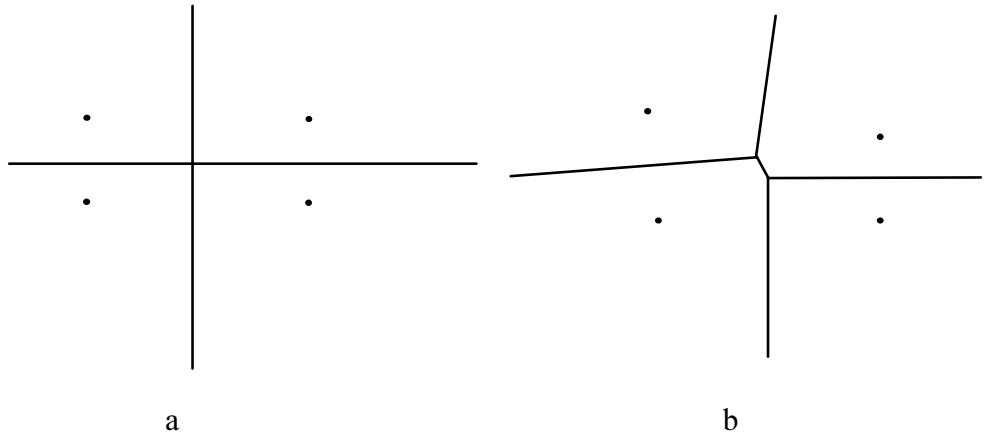


Slika 6.2 Tri točke na ravnini: razpolovnice se sekajo v središču trikotniku očrtanega kroga.

Iz zgoraj omenjenih primerov je razvidno, da imajo razpolovnice B_{ij} pomembno vlogo pri proučevanju odnosov med točkami na ravnini. Naj bo $H(p_i, p_j)$ zaprta polravnina, ki je omejena z B_{ij} in vsebuje točko p_i. Potem se lahko $H(p_i, p_j)$ razume kot niz vseh točk, ki so bliže točki p_i kot točki p_j. Kot že omenjeno, je $V(p_i)$ niz točk, ki so bliže točki p_i, kot katerikoli drugi točki ali z drugimi besedami, to so točke, ki so bliže p_i kot p_1, bliže p_i kot p_2, bliže p_i kot p_3, itd. Upoštevaje navedeno je mogoče napisati enačbo za Vornoi diagram $V(p_i)$:

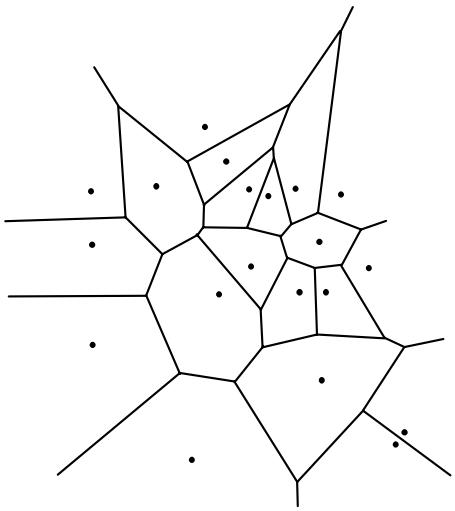
$$V(p_i) = \bigcap_{i \neq j} H(p_i,p_j),$$

pri čemer je potrebno upoštevati presek preko vseh i in j, za katere velja $i \neq j$. Iz enačbe sledi lastnost Voronoi diagrama: območja Voronoi diagrama so konveksna, ker nastanejo kot presečišča večih polravnin. Kadar so ta območja omejena, so to konveksni poligoni. Robovi se imenujejo Voronoi robovi in temena Voronoi temena. Katerakoli točka na Voronoi robu ima dve najbližji točki niza točk na ravnini, Voronoi teme pa ima najmanj tri najbližje take točke.
V primeru štirih točk na ravnini, ki oblikujejo vogale kvadrata, je to Voronoi diagram kot je prikazano na Sliki 6.3a. Voronoi teme je v tem primeru četrte stopnje. Če se eno točko nekoliko premakne (Slika 6.3b), postane diagram normalen. Prvi primer je izrojen zaradi centričnega položaja štirih točk na ravnini.



Slika 6.3 Izrojen (a) in normalen (b) Voronoi diagram.

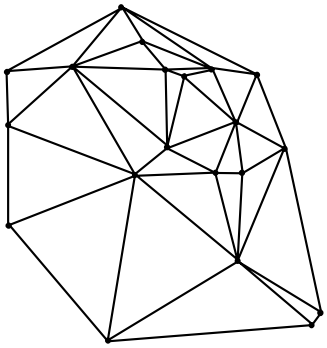
Voronoi diagram $V(p_i)$ za $i = 20$ točk na ravnini bi izgledal kot na Sliki 6.4.



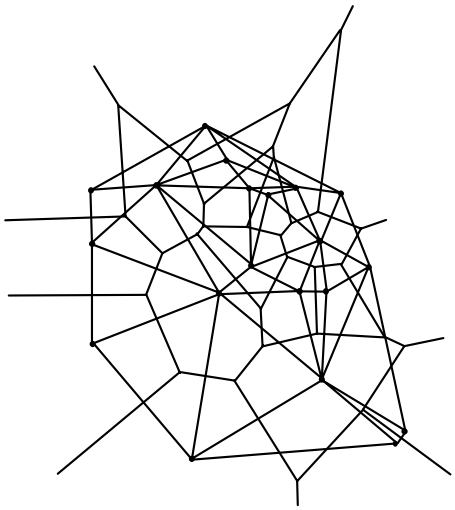
Slika 6.4 Voronoi diagram za $i = 20$ točk na ravnini.

6.2 Delaunayeva triangulacija

Delaunayeva triangulacija nastane takrat, kadar se med seboj povežejo točke na ravnini, ki imajo skupen Voronoi rob. Delaunayeva triangulacija $D(P)$ in Voronoi diagram $V(P)$ predstavljata dvojnost; isti podatki so predstavljeni na dva različna načina.
Na Sliki 6.4 je prikazan Voronoi diagram za dvajset točk na ravnini. Na Sliki 6.5 je prikazana Delaunayeva triangulacija za teh istih dvajset točk. Na Sliki 6.6 sta prikazana Delaunayeva triangulacija in Voronoi diagram skupaj.



Slika 6.5 Delaunayeva triangulacija za iste točke na ravnini kot na Sliki 6.4.



Slika 6.6 Delaunayeva triangulacija in Voronoi diagram: Sliki 6.4 in 6.5 skupaj.

6.2.1 Odnos med Delaunayevo triangulacijo in Voronoi diagramom

Za razumevanje obeh, v računski geometrijij pomembnih struktur $D(P)$ in $V(P)$, je potrebno poznati odnos med njima [8]. Naj bo P nespremenljiv niz točk na ravnini.

Za Delaunayevo triangulacijo veljajo lastnosti:

D1. $D(P)$ je ravnočrtna dvojnost $V(P)$, po definiciji.
D2. $D(P)$ je triangulacija, če ne obstajajo štiri centrične točke na ravnini: vsaka površina je trikotnik. Ploskve $D(P)$ so Delaunayevi trikotniki.
D3. Vsak trikotnik grafa $D(P)$ ustreza temenu grafa $V(P)$.
D4. Vsak rob grafa $D(P)$ ustreza robu grafa $V(P)$.
D5. Vsak vozel grafa $D(P)$ ustreza območju grafa $V(P)$.
D6. Meja $D(P)$ je konveksna lupina
D7. Notranjost trikotnikov $D(P)$ ne vsebuje nobenih točk niza P.

Za Voronoi diagram veljajo lastnosti:

V1. Vsako Voronoi območje $V(p_i)$ je konveksno.
V2. $V(p_i)$ je neomejen, če je p_i na konveksni lupini niza točk P.
V3. Če je Voronoi teme na stičišču $V(p_1)$, $V(p_2)$ in $V(p_3)$, potem je v središču kroga $C(v)$, ki ga določajo točke p_1, p_2 in p_3.
V4. $C(v)$ je očrtan krog pri Delaunayevi triangulaciji, ki ustreza v.
V5. V notranjosti $C(v)$ ni nobene točke niza P.
V6. Če je p_j najbližji p_i, potem je (p_j,p_i) rob triagulacije $D(P)$.
V7. Če obstaja krog skozi točki p_j in p_i, ki ne vsebuje nobenih drugih točk niza P, potem je (p_j,p_i) rob triangulacije $D(P)$.

6.3 Konstrukcija Voronoi diagrama

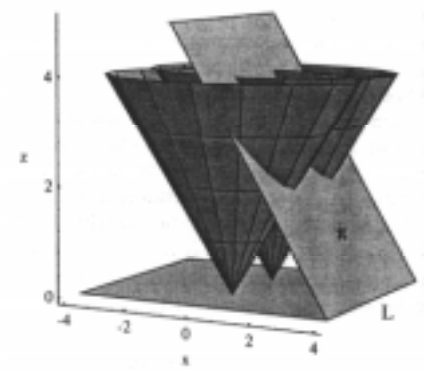
6.3.1 Deterministične algoritem za konstrukcijo Voronoi diagrama

Za konstrukcijo Voronoi diagrama obstaja cela vrsta algoritmov. Leta 1985 je Fortune [8] izdelal algoritem, ki temelji na principu pregledovanja ravnine. Na prvi pogled je nerazumljivo, kako lahko nastaja diagram za linijo, če nanj vplivajo točke na ravnini, ki so pred pregledovalno linijo in še niso bile pregledane.
Naj bodo točke na xy ravnini del tridimenzionalnega koordinatnega sistema. Nad vsako točko naj bo postavljen stožec z nagibom 45°, ki ima vrh v točki p. V tretji dimenziji je to mogoče razumeti kot čas. Stožec nad točko p predstavlja krog, ki se veča: po času t ima radij t. Naj imata točki p_1 in p_2 vsaka svoj stožec. Presečišče teh dveh stožcev je krivulja v prostoru, njena projekcija pa je ravna črta na xy ravnini (Slika 4.7).

Slika 6.7 Projekcija prostorske krivulje na ravnino je daljica

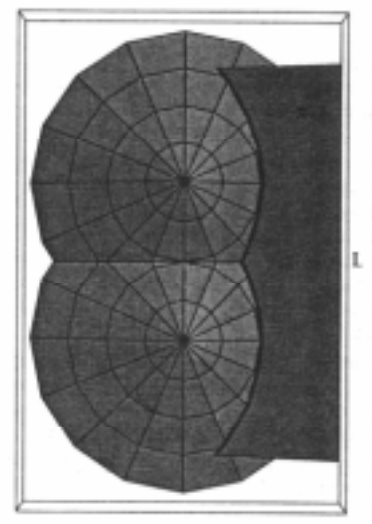
Gledano iz $z = -\infty$, bi projekcije vseh prostorskih krivulj, ki so presečišča stožcev nad točkami p_i, predstavljale Voronoi diagram na ravnini xy. To lastnost je Fortune uporabil pri svoji metodi. Ravnina se pregleduje s pomočjo ravnine π, ki je proti

ravnini *xy* nagnjena za 45^{o}. Pregledovalna črta *L* je presečišče med ravninama π in *xy*. Naj bo pregledovalna črta *L* vzporedna *y* osi, koordinata *x* pa naj bo *l* (Slika 6.8).



Slika 6.8 Stožca presekana s pregledovalno ravnino. Ravnina π in črta *L* se pomikata v desno.

Na strani *x>l* črte *L* je od spodaj vidna samo ravnina π. Ta zakriva del stožcev, obenem pa predstavlja del ravnine, ki jo je potrebno še pregledati. Na strani *x<l* črte *L*, je viden Voronoi diagram vse do presečišča ravnine π s stožcema. Presečišče ravnine s stožcem je parabola in tako je tudi presečišče ravnine π z desno stranjo stožcev parabola. Ta se na ravnino *xy* (gledano iz $z = -\infty$) projecira kot parabolično čelo, ki je sestavljeno iz večih elementarnih parabol (Slika 6.9).



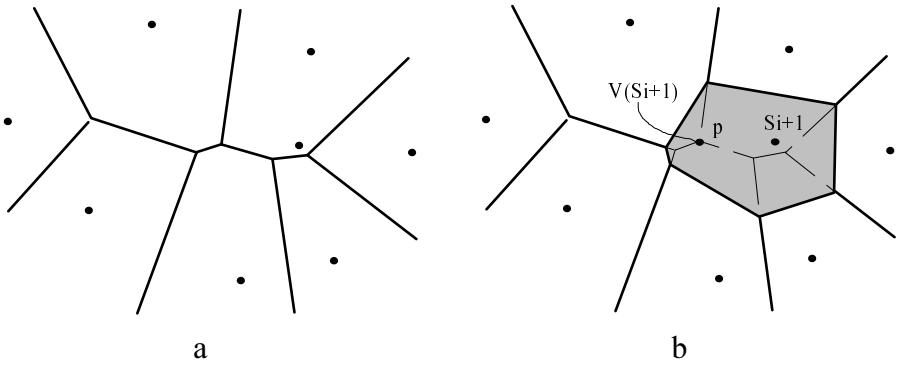
Slika 6.9 Gledano iz $z = -\infty$. Odebljena črta je parabolično čelo.

Dve paraboli se stikata v točki, kjer se ravnina π dotika obeh stožcev. Iz prej navedenih ugotovitev je na tem mestu Voronoi rob. Ker je pregledovalna ravnina π nagnjena pod istim kotom kot stranica stožca, sreča *L* točko *p* v trenutku, ko pregledovalna ravnina

zadane ob stožec točke *p*. Tako Voronoi diagram ne nastaja samo levo od pregledovalne črte *L*, ampak ves čas tudi pod ravnino π. Torej: Vornoi diagram nastaja levo od pregledovalne črte *L*, navzgor do paraboličnega čela, ki za *L* nekoliko zaostaja.

6.3.2 Inkrementalni algoritem za konstrukcijo Voronoi diagrama

Inkrementalni algoritem za konstrukcijo Voronoi diagrama [7] temelji na postopnem dodajanju točk niza *P* po naključnem vrstnem redu; podobno kot že opisani inkrementalni algoritmi. Naj bo *P* niz točk na ravnini in *V*(*P*) Voronoi diagram, ki ga te točke določajo. Naj bo P^i niz prvih *i* naključno izbranih točk. Na vsaki stopnji *i* algoritem določi Voronoi diagram $V(P^i)$. Diagram $V(P^{i+1})$ nastane iz $V(P^i)$ kot je prikazano na Sliki 6.10.



Slika 6.10 (a) $V(P^i)$; (b) $V(P^{i+1})$; (b) območje S_{i+1}.

Naj bo S=S_{i+1} i+1 točka, ki je na ravnino dodana naključno. Najprej je potrebno poiskati točko v $V(P^i)$, ki leži v Voronoi območju točke *S*. V tem primeru je *S* bližja točki *p* kot katerikoli drugi sosednji točki v $V(P^i)$. Jasno je, da točka *p* ne bo več del diagrama $V(P^{i+1})$. V vsakem trenutku dodajanja i+*1* je možnih več točk *p*, vendar zadostuje ena sama, ki je izbrana naključno. Ostale točke (temena Voronoi diagrama), ki so v območju *S*, je moč poiskati v diagramu $V(P^i)$. Iskanje se začne v točki *p* in je preprosto izvedljivo, saj so sosednja temena med seboj povezana z Voronoi robovi. Robovi, ki so temenom *p* sosednji, se skrajšajo ali odstranijo. Če ima Voronoi rob obe krajišči v točkah *p*, se odstrani, če pa je točka *p* samo eno krajišče, se rob skrajša. Na koncu je potrebno določiti še robove Voronoi območja dodane točke *S*. Ti robovi povezujejo med seboj krajišča odrezanih robov v krožnem vrstnem redu. Mesta, kjer se robovi odrežejo, so določljiva z razpolavljanjem povezav med točko *S* in ostalimi sosednjimi točkami. Tako dobljen diagram je $V(P^{i+1})$.

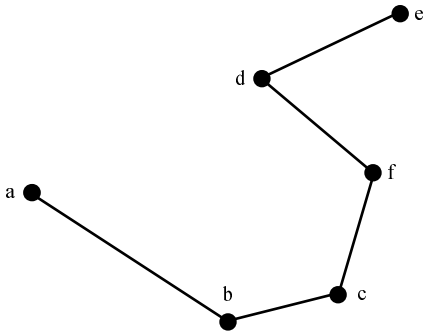
6.4 Aplikacije povezane z Voronoi diagrami

S konstrukcijo Voronoi diagrama je povezanih več aplikacij: najbližji sosed, maksimiziranje najmanjših kotov pri triangulaciji, največji prazen krog, najmanjše razvejišče, problem trgovskega potnika, itd.

6.4.1 Najbližji sosed

Problem najbližjega soseda je iskalne narave. Potrebno je poiskati najbližjega soseda določeni točki oziroma poiskati najbližjega soseda vsaki izmed točk niza *P*. S tem problemom se ukvarjajo na različnih področjih: biologija, ekologija, geografija, fizika, itd.

Definicija problema:
b je najbližji sosed *a*-ja, če $|a - b| \leq \min_{c \neq a} |a - c|$, pri $c \in P$. Lahko se tudi zapiše $a \rightarrow b$ ali najbližji sosed *a*-ja je *b*. Ta trditev ni simetrična, saj ni nujno, da hkrati velja tudi $b \leftarrow a$ (Slika 6.11).



Slika 6.11 $a \rightarrow b$, $b \rightarrow c$,$d \rightarrow e$ in $d \rightarrow f$. $b \rightarrow a$ ne velja.

Naj bo *P* niz točk, preko katerega se lahko določi Voronoi diagram. Pri iskanju najbližjega soseda točki *q* se problem zreducira na iskanje Voronoi območja, v katerem ta točka leži. Vse točke v tem območju so najbližji sosedi točki *q*.

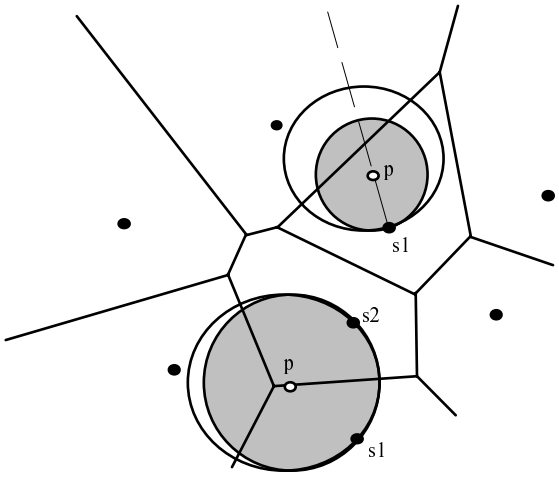
6.4.2 Maksimiziranje najmanjših kotov triangulacije

Pri analiziranju kompleksnejših oblik se uporabljajo metode končnih elementov (pri načrtovanju avtomobilskih lupin). Stabilnost numeričnega postopka je odvisna od kvalitete razdelitve. Izkaže se, da spada Delaunayeva triangulacija med dobre razdelitve. Naj bo *P* niz točk, preko katerih je izvedena triangulacija. Linijski segmenti

se med seboj sekajo (v svojih krajiščih) samo v točkah niza *P*. Konveksna lupina je razdeljena na trikotnike. Z vidika končnih elementov je dobra tista razdelitev, ki uporablja "debele" trikotnike. Izogniti se je potrebno trikotnikom z majhnimi koti. Iz tega sledi postopek maksimiziranja najmanjših kotov. Natančen odgovor na to je Delaunayeva triangulacija. Naj bo *T* triangulacija niza *P*, zapis kotov te triangulacije naj bo $(\alpha_1, \alpha_2,...,\alpha_{3t})$, razvrščenih od najmanjšega do največjega, pri čemer je *t* število trikotnikov v *T*. Število *T* je konstantno za vsak niz *P*. Med dvema triangulacijama istega niza *P* je določljivo razmerje $T \geq T'$ glede na njuno 'debelost', če velja: $\alpha_1>\alpha_1'$ ali $\alpha_1=\alpha_1'$ in $\alpha_2>\alpha_2'$ ali $\alpha_1=\alpha_1'$, $\alpha_2=\alpha_2'$ in $\alpha_3>\alpha_3'$ itd. Iz tega vsega sledi, da je Delaunayeva triangulacija $T=D(P)$ največja glede na velikost kotov v odnosu $T \geq T'$ glede na katerokoli razdelitev *T'* preko niza točk *P*.

6.4.3 Največji prazen krog

Potrebno je poiskati največji prazen krog, čigar center je v notranjosti konveksne lupine, ki jo določa niz točk *S*. Prazen v tem smislu, da v njegovi notranjosti ne leži nobena točka niza *S* in največji, da ne obstaja noben tak krog, ki bi imel večji radij. V praksi je to primerljivo z iskanjem lokacije za postavitev jedrskega reaktorja, ki naj bo zgrajen čim bolj stran od naseljenih krajev. Naj bo $f(p)$ polmer največjega praznega kroga s centrom v točki *p*. Poiskati je potrebno maksimum te funkcije preko vseh *p*-jev v konveksni lupini *S*, $H=H(S)$. Navidezno obstaja neskončno možnosti za točko ekstrema. Izkaže pa se, da jih je končno mnogo.
Naj bo nekje v notranjosti *H* prazen krog s središčem v *p*, ki se mu lahko povečuje polmer. V nekem trenutku bo ta krog zadel ob eno izmed točk na ravnini $S = \{s_1,...,s_n\}$ in imel pri tem vrednost $f(p)$. Če krog zadane samo ob eno točko, npr. s_1, potem je jasno, da $f(p)$ ni največja možna vrednost. Če se točka *p* po premici iz p_1 pomakne stran od s_1 v točko *p*', potem je $f(p') > f(p)$ (Slika 6.12).



Slika 6.12 Krog skozi eno ali dve točki na ravnini.

Recimo, da pri razdalji $f(p)$ krog zadane ob natančno dve točki s_1 in s_2. Tudi v tem primeru $f(p)$ ne more biti največja možna vrednost. Točko p je mogoče po razpolovnici S_1S_2 pomakniti stran od s_1 in s_2 v točko p'. Potem velja $f(p')>f(p)$ (Slika 6.12). Iz tega sledi, da funkcija $f(p)$ doseže maksimum samo v primeru, če krog v enem trenutku zadane ob najmanj tri točke hkrati. Vsako premikanje točke p bi v tem primeru pomenilo približevanje k neki točki in s tem manjšanje $f(p)$. Če je torej center p največjega praznega kroga v notranjosti lupine $H(S)$, potem mora p sovpadati z Voronoi temenom. Če p leži na lupini $H(S)$, potem mora točka p ležati na Voronoi robu. S tem je določenih končno število možnih točk. Med vsemi je potrebno poiskati tisto, ki ima $f(p)$=max. Algoritem za največji prazen krog:

procedure RMax(p)
begin
določi Voronoi diagram $V(S)$ za S;
določi konveksno lupino H;.
for za vsako Voronoi teme n **do**
begin
if $v \in H$ **then**
max $f(p)$ s centrom v v;
end.
for za vsak Voronoi rob e **do**
begin
$p=e \cap \delta H$;
max $f(p)$ s centrom v p;
end.
return $f(p)$max;

end.

6.4.4 Najmanjše razvejišče

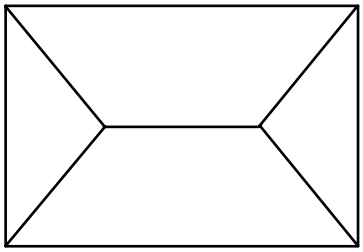
Najmanjše razvejišče je problem, pri katerem je potrebno točke niza *S* povezati med seboj tako, da bo imela nastala drevesna struktura najmanjšo mogočo dolžino. V praksi je to primerljivo z načrtovanjem lokalnih mrež, kjer je potrebno s kablom povezati med seboj uporabnike na različnih lokacijah. Ideja algoritma je, da je potrebno med seboj združiti vse najkrajše robove *e*, ki povezujejo med seboj vse točke niza *G*.

6.4.5 Problem trgovskega potnika

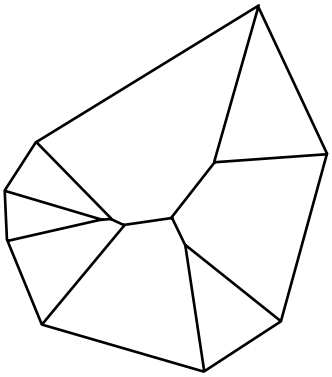
Problem trgovskega potnika je eden bolj proučevanih v računski geometriji. Je povsem praktične narave in ga je moč uporabiti pri številnih primerih. Glavno vprašanje je, po kateri poti naj hodi trgovski potnik, da bo obiskal vsa mesta in da bo pot najkrajša možna. Ali, kako z neprekinjeno črto povezati med seboj vse točke niza *M* tako, da bo ta črta najkrajša možna.

6.5 Posplošitve Voronoi diagrama

Lastnosti Voronoi diagrama omogočajo posplošitve v večih smereh, kar se izkaže kot zelo uporabno na različnih praktičnih področjih. Voronoi diagram je v svoji osnovi določen kot niz točk, za katere velja, da ima vsaka najmanj dve najbližji točki iz niza točk *P*. Ena izmed posplošitev je sistem srednjih osi. Srednje osi so skup točk v notranjosti *P*, ki imajo najmanj dve najbližji točki na meji δP mnogokotnika *P*. V tem primeru prej omenjene točke zamenja meja mnogokotnika δP. Srednje osi pravokotnika so prikazane na Sliki 6.13. Vsaka točka horizontalnega segmenta v kvadratu je enako oddaljena od zgornjega in spodnjega roba mnogokotnika *P*. Vsaka točka na diagonalnem segmentu je enako oddaljena od točk na sosednjih robovih mnogokonika *P*. Bolj kompleksen primer je prikazan na Sliki 6.14. Srednje osi imajo drevesno strukturo. Vsaka točka na srednjih oseh je središče kroga, ki se meje mnogokonika δP dotika v najmanj dveh točkah.



Slika 6.13 Srednje osi kvadrata



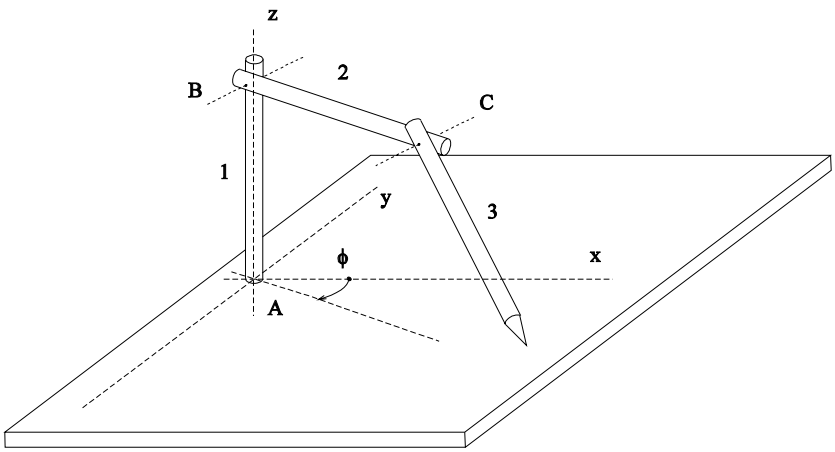
Slika 6.14 Srednje osi konveksnega mnogokotnika z $n == 8$ temeni

7 Zajem prostorskih točk

Proces modeliranja v prostoru je mogoč tudi na osnovi modelov, ki so dobljeni z zajemom točk v prostoru. Tako dobljene površine je mogoče vključiti v že obstoječa stanja ali jih uporabiti kot osnovo za nove modele. Zajem in modifikacija obstoječega stanja je pomemben način modeliranja na področjih, kjer so površine kompleksne in za katere popis površine ne obstaja. Vhodne koordinate točk so lahko urejene in popisujejo skupine površin, ali pa so neurejena množica točk, za katere površine niso opredeljene. V splošnem so površine kompleksne.
Digitalizirane prostorske točke je mogoče z uporabo algoritmov računske geometrije združiti v mnogokotniške mreže, ki tvorijo površine. Za zahtevne površine je najbolj uporabna trikotniška mreža.

7.1 Mehanski sistem za zajemanje prostorskih točk

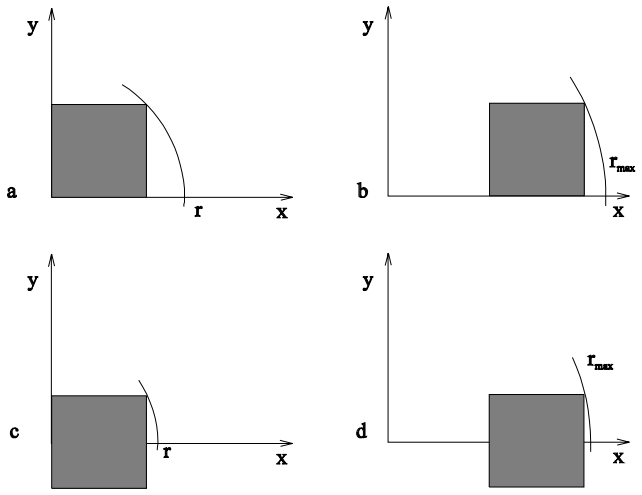
Mehanski sistem je zasnovan za zajemanje prostorskih točk. V povezavi z računalnikom je sposoben digitalizacije seznama urejenih ali neurejenih točk. Sistem je sestavljen iz koordinatne mize, na katero je pritrjen mehanizem, sestavljen iz treh palic. Palica, ki je pritrjena na osnovno ploščo, je vrtljiva okoli svoje vertikalne osi, ostali dve pa sta vrtljivi okoli horizontalne osi, ki gre skozi krajišče palice. V vsakem izmed treh vrtišč je pritrjen potenciometer, preko katerega je mogoče določiti kot zasuka posamezne palice. Z upoštevanjem vseh treh relativnih kotov in dolžin palic, je mogoče določiti koordinate točke v prostoru.
Trije potenciometri so priključeni vsak na svoj kanal analogno digitalnega konverterja. Ta komunicira (podatke pošilja za vsak kanal posebej) preko LPT porta z računalnikom s pomočjo programa, ki upošteva karakteristike A/D konverterja. Rezultat je število med 0 in 255, s katerim je mogoče določiti kot v vsakem vrtišču in preko tega koordinate določene točke v prostoru.



Slika 7.1 Mehanski sistem za zajemanje prostorskih točk.

Na sliki 7.1 je prikazan mehanski sistem za zajemanje prostorskih točk. Sestavljen je iz treh palic (1, 2 in 3) in treh potenciometrov, ki omogočajo vrtenje v treh vrtiščih (A, B in C). Na koncu tretje palice je pritrjeno tipalo. Konico tipala je potrebno postaviti na tisto točko v prostoru, ki se ji želi odčitati koordinatne vrednosti.

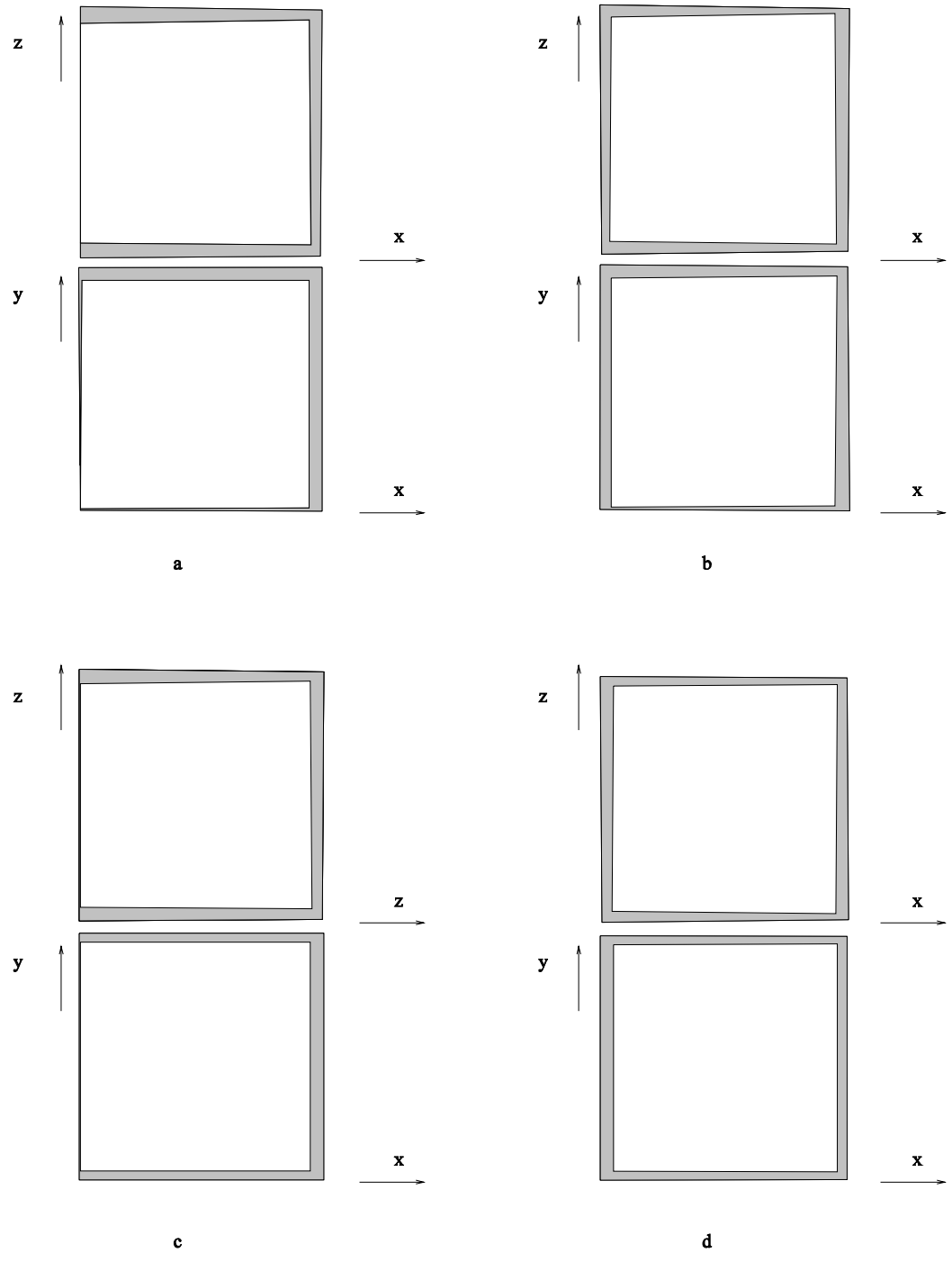
7.1.1 Napake pri zajemanju prostorskih točk



Slika 7.2 Štiri različne postavitve modela na koordinatno mizo.

Do napak pri zajemanju prostorskih točk pride v večji meri zaradi zračnosti v potenciometrih in spojih, zanemarljivo majhen del pa prispeva netogost palic. Zračnost

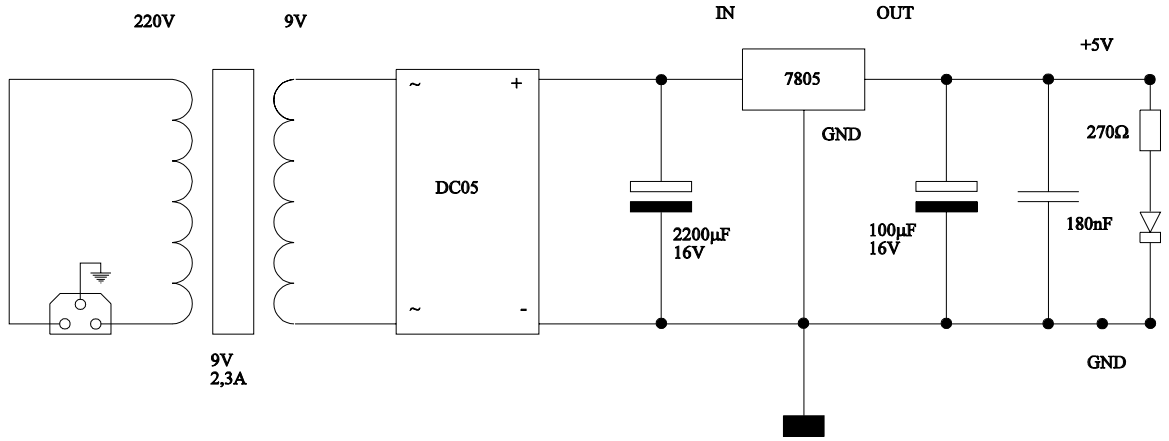
v spojih je mogoče s primerno izvedbo zmanjšati, medtem ko se na zračnost v potenciometrih ne da vplivati. Zaradi naštetega izmerjene vrednosti odstopajo od realnih, kar vpliva na popačenje oblike modela, ki se ga želi posneti. Na Sliki 7.2 so prikazane štiri različne postavitve modela na koordinatno mizo in na Sliki 7.3 štiri predvidena popačenja oblike zaradi zračnosti v vrtiščih. Senčeno področje predstavlja možna odstopanja oblike za kocko, ki je bila uporabljena kot model.



Slika 7.3 Popačenja oblike za štiri postavitve modela iz Slike 7.2.

7.2 Napajalnik

Analogno digitalni konverter deluje pri napetosti petih voltov (Slika 7.4).



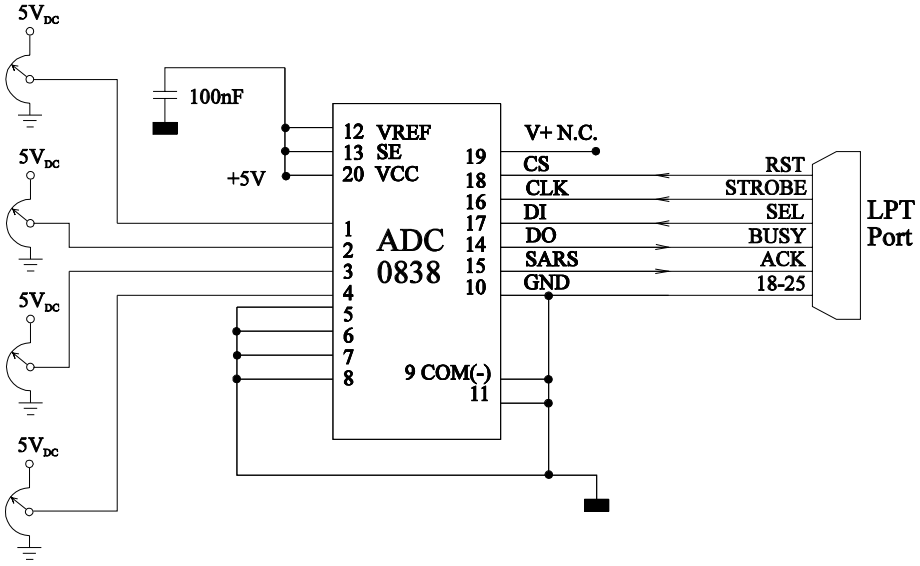
Slika 7.4 Shema napajalnika za ADC.

7.3 Analogno digitalni konverter

Za pretvarjanje podatkov iz analogne oblike v digitalno se uporablja 8-bitni analogno digitalni konverter ADC0838 [1] s serijskim I/O vhodom in možnostjo uporabe do osmih kanalov. Pri tem se lahko uporablja vsak kanal posebej (rezultat je absolutna vrednost) ali v parih (rezultat je diferenca vrednosti na obeh kanalih).

7.3.1 Povezava A/D konverterja z računalnikom

A/D konverter je z računalnikom povezan preko kabla na LPT port. Shema povezave je prikazana na Sliki 7.5.



Slika 7.5 Povezava konverterja z računalnikom (LPT).

V tabeli 7.1 je prikazana vezava na LPT port.

Tabela 7.1 LPT port [12].

Pin	Povezava		Naslov	Bit	Povezava z A/D
1	Not Data Strobe	$\overline{\text{STROBE}}$	0x37a	0	CLK
10	Not Printer Acknowledge	$\overline{\text{ACK}}$	0x379	6	SARS
11	Printer Busy	BUSY	0x379	7	DO
16	Not Reset Printer	$\overline{\text{RST}}$	0x37a	2	$\overline{\text{CS}}$
17	Not Printer Selected	SEL	0x37a	3	DI
18-25	GND	GND			

7.3.2 Komunikacija med računalnikom in ADC.

Delovanje AD konverterja je pod nadzorom programske opreme. Pretvorba se začne s postavitvijo linije $\overline{\text{CS}}$ (chip select) na nič (Slika 7.4). Ta mora ostati na ničli ves čas pretvorbe. Tako je konverter pripravljen sprejeti prvi bit, t.j. prvi bit določitve kanala, s katerega naj pošlje pretvorjeno vrednost. Čas, ki ga je potrebno aktivirati na tem mestu, je določen s procesorjem in ga je potrebno poslati na CLK (clock) vhod konverterja.
Posamezno vhodno konfiguracijo se določi z MUX naslavljanjem. MUX naslov določi, s katerega analognega kanala bo potekala pretvorba in ali bo to samostojen podatek ali pa bo v paru določal diferenco. MUX naslov je potrebno poslati na DI (data in) vhod A/D konverterja. V Tabeli 7.2 je prikazano MUX naslavljanje za posamezne kanale in v Tabeli 7.3 diferencialno Mux naslavljanje.

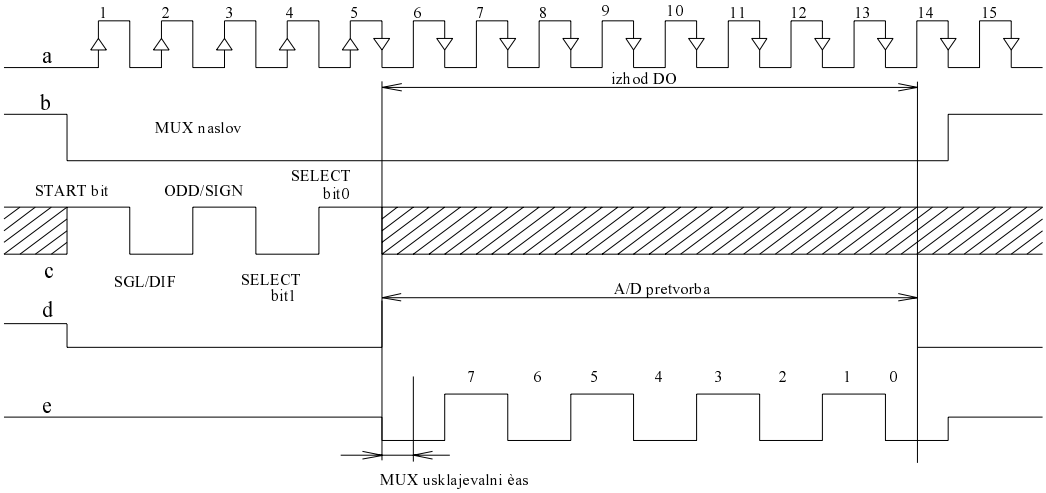
Tabela 7.2 Mux naslavljanje za posamezne kanale.

MUX Naslov		Posamezen analogni kanal									
SGL/ $\overline{\text{DIF}}$	ODD/ SIGN	SELECT 1 0	0	1	2	3	4	5	6	7	COM
1	0	0 0	+								-
1	0	0 1			+						-
1	0	1 0					+				-
1	0	1 1							+		-
1	1	0 0		+							-
1	1	0 1				+					-
1	1	1 0						+			-
1	1	1 1								+	-

Tabela 7.3 Diferencialno Mux naslavljanje.

MUX Naslov		Diferencialni analogni par								
SGL/ $\overline{\text{DIF}}$	ODD/ SIGN	SELECT 1 0	0	1	2	3	4	5	6	7
0	0	0 0	+	-						
0	0	0 1			+	-				
0	0	1 0					+	-		
0	0	1 1							+	-
0	1	0 0	-	+						
0	1	0 1			-	+				
0	1	1 0					-	+		
0	1	1 1							-	+

Začetni bit je prva logična "1" (ničle pred tem se izpustijo), ki se pojavi na vhodu konverterja DI. Po prvem bitu konverter pričakuje še štiri bite za določitev MUX naslova. Ko je začetni bit prestavljen na začetno lokacijo MUX registra (t.j. za pet mest v levo), je s tem določen analogni kanal, s katerega bo potekala pretvorba. Na tem mestu je potreben čas 1/2 časovne periode, da izbrani MUX kanal uskladi svoje delovanje. SAR linija se dvigne na ena, kar pomeni, da je pretvorba v teku in da vhod DI do konca pretvorbe ne sprejme nobenega signala več. Po koncu pretvorbe se SAR linija spusti na nič. Izhodna linija DO pride iz tristanjskega položaja in v času usklajevalnega časa odda začetno ničlo. Izhodni kanal DO začne oddajati osem bitov, vsakega v trenutku padca linije CLK. Vsi notranji registri se izpraznijo z dvigom linije $\overline{\text{CS}}$ na ena.



Slika 7.6 Časovni diagram A/D pretvorbe za ADC0838 [1].

Na Sliki 7.6 je prikazan časovni diagram poteka določanja analognega kanala in A/D pretvorbe. Linija a predstavlja CLK (clock), b je $\overline{CS}$ (chip select), c je DI (data in), d je SARS (SAR status) in e je DO (data out). Šrafirano področje predstavlja čas, ko DI ne sprejema nobenih signalov.

Algoritem določanja analognega kanala in branja z DO A/D pretvornika:

```
        procedure ReadAD(channel)
begin   CS := 0;
        CLK := 0;
        i := števec bitov;
        a := naslov analognega kanala po Tabeli 7.2 - največ pet bitov;
        for i:= 0 until i<5 do              (* branje MUX naslova *)
                begin  if a[peti bit] := 1 then pošlji 1 na DI else 0 na DI;
                       časovna perioda;
                       pomik a za en bit v levo;
                end.                        (* konec branja MUX naslova *)
        a := vseh osem bitov na 0;
        pavza za usklajevanje delovanja;
        for i := 0 until i<8 do             (* branje z DO *)
                begin  časovna perioda;
                       a := prebere bit z DO po Sliki 7.4 in ga vpiše v prvi bit;
                       pomik a za en bit v levo;
                end.
```

$\overline{CS}$:= 1; (* konec branja z DO *)
end.

Funkcija je napisana v programskem jeziku C.
Na A/D konverter ($\overline{CS}$, CLK in DI) pošilja z naslova 0x379 in bere (DO in SARS) z naslova 0x379.

```
int iport = 0x379;
int oport = 0x37a;
```

Signal s $\overline{STROBE}$ na CLK je invertiran, zato je za postavitev CLK na nič potrebno postaviti (hkrati tudi $\overline{CS}$ na nič):

```
b = 0x01;
outportb( oport, b );
```

Čas je določen s spreminjanjem vrednosti na prvem bitu "0→1→0". Zaradi invertiranosti velja:

```
outportb( oport, b &= 0xFE ); /* prižge bit 0 */
outportb( oport, b |= 0x01 ); /* ugasne bit 0 */
```

Čas usklajevanja po MUX naslavljanju je določen z

```
delay(4);
```

in ga je potrebno prilagajati izbranemu A/D konverterju.

Z DO po vrsti prihajajo osmi, sedmi, šesti itd. biti rezulatata A/D pretvorbe na BUSY (invertiran), t.j. osmi bit. Tako je potrebno prebrano vrednost najprej prestaviti za sedem bitov v desno (t.j. v prvi bit) in pred naslednjim branjem ves zapis za en bit v levo.

```
a = (( ~inportb( iport ) & 0x80 ) >> 7 ) | ( a<< 1);
```

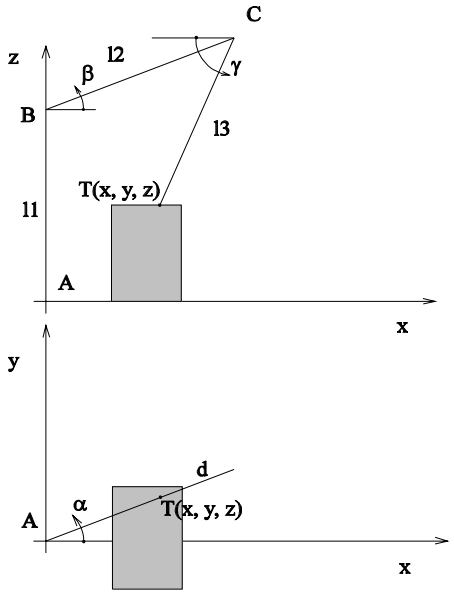
Po končani pretvorbi se $\overline{CS}$ postavi na ena:

```
b = 0x02;
```

outportb(oport, b);

7.4 Določanje koordinatnih vrednosti iz geometrije sistema

Koordinatne vrednosti točk v prostoru so določljive v odvisnosti od geometrije sistema za zajemanje točk v prostoru. Na Sliki 7.7 je prikazana geometrija sistema.



Slika 7.7 Geometrija sistema za zajemanje prostorskih točk.

Določitev koordinat:
$d = l_2 * cos\beta - l_3 * cos(\gamma - \beta)$,

koordinata *x*:
$x = d * cos\alpha$,

koordinata *y*:
$y = d * sin\alpha$,

koordinata *z*:
$z = l_1 - l_2 * sin\beta - l_3 * sin(\gamma - \beta)$.

7.5 Zapis zajetih prostorskih točk v datoteko

Program Read.C zapisuje zajete prostorske točke v datoteko "tocke.dat" preko funkcije ReadAD(ch), ki je predstavljena v podpoglavju 7.3.2 Komunikacija med računalnikom in ADC:

unsigned char ReadAD(int channel).

Program je narejen tako, da se v datoteko zapisujejo samo tiste točke, ki se jih potrdi. Pri vsakem zapisovanju se točke zapisujejo na konec datoteke, s čimer je omogočeno vmesno umerjanje sistema ali zamenjava tipala zaradi nedosegljivosti posameznih delov površine.

```
zapis = fopen( "\\ \\ \\tocke.dat", "a" );
```

Funkcija ReadAD(channel) vrne vrednost med 0 in 255.

```
a = ( ( ~inportb( iport ) & 0x80 ) >> 7 ) | (a << 1);
return a;
```

Ker vsi potenciometri ne delujejo v istem območju, je bilo za vsakega posebej potrebno umerjanje z linearno odsekovno interpolacijo.

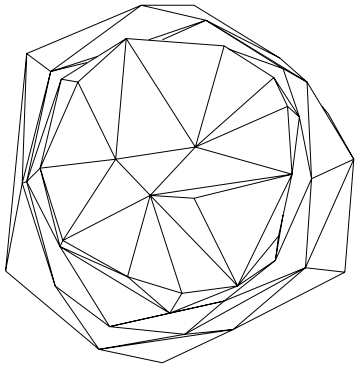
A - α [-1/6π, +1/6π],
B - β [-1/12π, +1/12π],
C - γ [1/12π, 1/2π].

```
a = ka * ReadAD(0) + na;
b = kb * ReadAD(1) + nb;
c = kc * ReadAD(2) + nc;

alfa = a * pi / 180;
beta = b * pi / 180;
gama = c * pi / 180;

d = l2 * cos( beta ) - l3 * cos( gama - beta );
x = d * cos( alfa );
y = d * sin( alfa );
z = l1 - l2 * sin( beta ) - l3 * sin( gama - beta);
```

7.6 Popačenje oblike zaradi napake pri odčitavanju

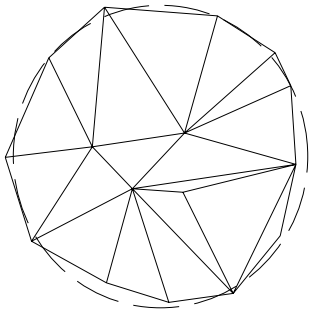
Napake, ki nastanejo zaradi netočnosti sistema pri zajemanju posameznih točk, se pri oblikovanju površine seštevajo tako, da je popačenje oblike večje, kot bi bilo mogoče

pričakovati glede na napako pri odčitavanju koordinat posamezne točke. Na Siki 7.8 je prikazana rekonstrukcija površine plastičnega lončka, narejena z algoritmom, opisanim v 8. poglavju. Točke so določene z mehanskim sistemom za zajem prostorskih točk.



Slika 7.8 Rekonstrukcija površine plastičnega lončka.

Na Sliki 7.9 je prikazano popačenje oblike dna lončka zaradi napake pri zajemanju posameznih točk. Črtkana črta predstavlja dejansko obliko in velikost dna.



Slika 7.9 Dno plastičnega lončka.

Kljub napaki, se je dejanski obliki mogoče približati s čim večjim številom odčitanih točk.

8 Rekonstrukcija površin

Podan je prostorski model, ki se mu želi digitalizirati površino. Osnovni podatek so prostorske točke

$p_i = [x_i, y_i, z_i] \in R^3$, $i = 1,...,n$

na površini telesa, ki so bile dobljene s pomočjo mehanskega sistema za odčitavanje 3D točk ali na kakšen drug način. Na podlagi odčitanih točk je potrebno narediti rekonstrukcijo telesa oz. rekonstrukcijo njegove površine ali samo dela njegove površine.

Načeloma odčitane točke niso urejene in same po sebi ne določajo površine, če pa že, ta razdelitev ni nujno najboljša. Z delnim urejanjem pri odčitavanju točk je mogoče razširiti možnosti rekonstruiranja in celo izboljšati kvaliteto rekonstruirane površine. S tem je mišljena uporaba Delaunayeve triangulacije, ki je že sama po sebi najboljša, vendar je njen rezultat s primernim odčitavanjem točk mogoče še izboljšati. Pri urejenem odčitavanju se lahko določi osnovni rob telesa, ki se ga želi rekonstruirati ali pa se določi meja površine, ki predstavlja samo del površine celega telesa. O ostalih točkah površine, ki jo je potrebno rekonstruirati, v večini primerov ni nobenih podatkov. Ne o sosednosti, ne katerih koli drugih podatkov o povezanosti, naklonih in podobno.
Na splošno imajo lahko odčitane točke tudi kakšno urejenost. Lahko so urejene tako, da so za vsako ravnino posebej ($y = 0$, $y = 1$, $y = 2$, ...) podane vrednosti ostalih dveh koordinat in predstavljajo zaporedje ravninskih prečnih prerezov.

V nadaljevajnu je predstavljena metoda, ki iz neurejenih oz. delno urejenih točk, ki ležijo na površini telesa, omogoča rekonstrukcijo telesa oz. dela njegove površine.

8.1 Triangulacija površine

Pri rekonstrukciji površine so uporabljeni osnovni geometrijski elementi, ki so opisani v poglavju 3. Osnovni element je točka v prostoru

$p_i = [x_i, y_i, z_i] \in R^3$, $i = 1, ..., n$.

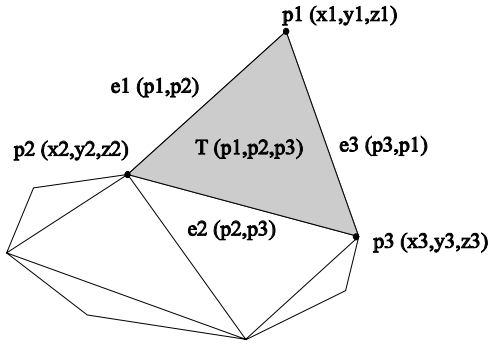
Dve točki v prostoru se lahko med seboj poveže z robom

$e_i = p_i p_j$ za $i \neq j$.

Trije robovi, ki med seboj povezujejo tri točke prostora, določajo površino trikotne oblike

$T_i = [p_i, p_j, p_k]$ za $i \neq j \neq k$ ali $T_i = [e_i, e_j, e_k]$ za $i \neq j \neq k$.

Z množico trikotnikov je mogoče rekonstruirati celotno ali samo delno površino modela (Slika 8.1).



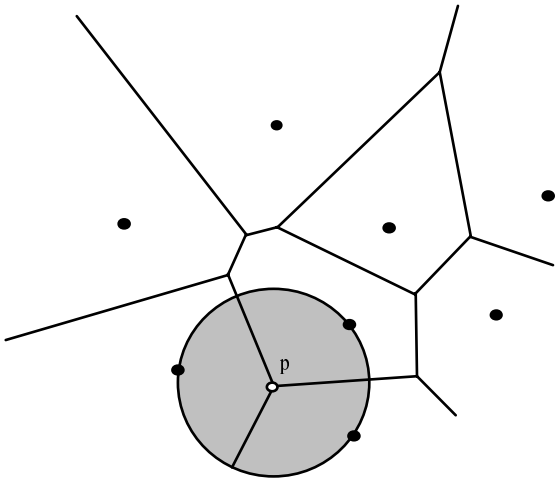
Slika 8.1 Površina sestavljena iz trikotnikov.

8.2 Delaunayeva triangulacija v 2D

V šestem poglavju je predstavljen Voronoi diagram in njegova dvojnost - Delaunayeva triangulacija. Tako Voronoi diagram kot Delaunayevo triangulacijo je mogoče narediti tudi preko niza točk N v prostoru. Rekonstrukcija površine, ki je opisana v nadaljevanju, je triangulacija, ki temelji na pravilih Delaunayeve triangulacije na ravnini, prenesene v prostor. Optimalni trikotniki se določajo s pomočjo lokalnega postopka.

8.2.1 Največji prazen krog

V poglavju 6.4.3 je opisan princip največjega praznega kroga, ki ga je mogoče postaviti v nek niz točk N. Pokazano je, da center p največjega praznega kroga sovpada z Voronoi temenom (Slika 8.2).

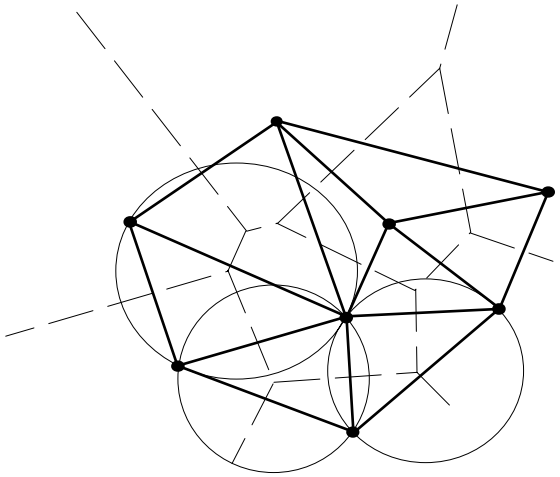


Slika 8.2 Center največjega praznega kroga sovpada z Voronoi temenom.

Za odnos med Voronoi diagramom in Delaunayevo triangulacio, ki se jo želi doseči, veljajo lastnosti, kot so opisane v podpoglavju 4.2.1.

D3. Vsak trikotnik grafa $D(P)$ ustreza temenu grafa $V(P)$.

Iz napisanega sledi, da postavljanje največjih (gledano lokalno) praznih krogov v niz točk N hkrati pomeni tudi posredno določanje Delaunayeve triangulacije. Na Sliki 8.3 je prikazanih nekaj največjih praznih krogov in Delaunayeva triangulacija.

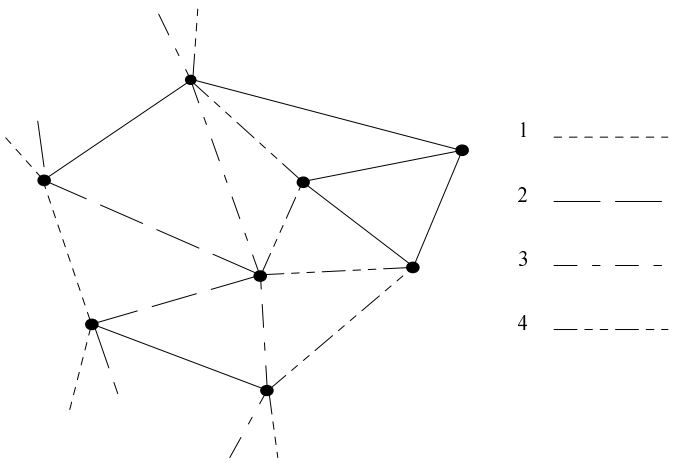


Slika 8.3 Delaunayeva triangulacija in največji prazen krogi.

Delaunyjev trikotnik $T_i = [p_i, p_j, p_k]$ je tako mogoče določiti iz nekega roba $e_i = p_i p_j$ in tiste tretje točke p_k, skozi katero gre največji prazen krog, ki gre hkrati tudi skozi obe krajišči roba p_i in p_j.

8.2.2 Pregledovanje ravnine

Ko so določene lokalne odvisnosti med posameznimi točkami in trikotniki Delaunayeve triangulacije, je potrebno tako ureditev izpeljati čez ves niz *N* točk na ravnini. V ta namen je potrebno ves niz točk pregledati s pregledovalno črto. Princip je podoben tistemu, ki je opisan v podpoglavju 3.5.1. Pomembna razlika je ta, da je bila pri trapezni dekompoziciji s pregledovanjem ravnine pregledovalna črta ravna in je potovala od točke do točke. V primeru gradnje triangulacije je primerneje, da se pregledovalno črto prilagodi točkam in s tem robovom, ki jih povezujejo. Tako pregledovalna črta ne potuje od točke do točke, ampak od enega pregledovalnega roba, ki je sestavljen iz večih robov triangulacije, do naslednjega pregledovalnega roba, ki je sestavljen iz novo nastalih robov, ki so bili dobljeni z določanjem tretje točke vsakega roba triangulacije posebej. Na Sliki 8.4 je prikazan niz točk in pregledovalni rob, ki se prilagaja točkam in že določenim robovom triangulacije. Pregledovalni rob potuje od leve proti desni. Štiri različne črte kažejo štiri različne položaje pregledovalnega roba, ki se prilagaja obliki Delaunayeve triangulacije.

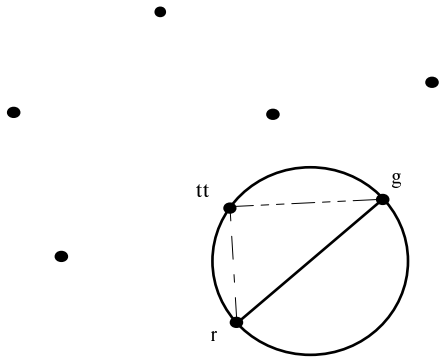


Slika 8.4 Pregledovalni rob potuje preko ravnine in se prilagaja že izgrajeni triangulaciji.

Načeloma niti ni tako pomembno, kje se pregledovanje začne. V tem primeru je privzet način iz Grahaovega algoritma, ki je opisan v podpoglavju 4.1.1.

8.2.3 Princip tretje točke

Začetek pregledovanja je tako rob, ki ga določata najnižja, najbolj desna točka in točka, ki je prva najbližja po naraščujočem kotu glede na os *x* (Slika 8.5).



Slika 8.5 Začetek pregledovanja niza točk *N*.

Tretja točka je določljiva glede na položaj središča očrtanega kroga. Izmed vseh točk, ki so levo od aktivnega roba *rg*, je potrebno izbrati tisto, katere odsek med središčem očrtanega kroga in razpoloviščem aktivnega roba je najmanjši, oz. največji, če je središče očrtanega kroga na desni strani aktivnega roba. Na Sliki 8.5 je to točka *tt*. Orientiranost robov trikotnika je pomembna pri iskanju naslednjih točk. Naslednji pregledovalni rob za Sliko 8.5 bi bil (*r*-*tt*, t*t*-*g*). Za vsak rob posebej je potrebno poiskati tretjo točko in tako naprej, dokler ni trianguliran celoten točk *N*.

8.2.3.1 Računanje polmera največjega praznega kroga

Določevanje tretje točke se skrči na določitev središča očrtanega kroga. Izračun se poenostavi, če se vse tri točke prestavi v drug koordinatni sistem tako, da leži rob *rg* na osi *x* in je *r*(0,0).

8.3 Delaunayeva triangulacija v 3D

Izvajanje triangulacije v 3D je precej podobno opisani za 2D. Odnos med tremi točkami v prostoru se lokalizira in prevede v problem na ravnini. Pravilnost izvajanja algoritma je mogoče povečati s primernim vzorčenjem. To pomeni, da morajo biti odčitane točke posejane dovolj na gosto, še posebej, kadar je ukrivljenost ploskve velika.

8.4 Proces rekonstrukcije površine - Delaunayeva triangulacija

V nadaljevanju je predstavljen algoritem, ki je sposoben rekonstrukcije površine, ki ni nujno konveksna. Temelji na osnovnih elementih računske geometrije, predstavljenih v tretjem poglavju, na lastnostih Voronoi diagrama in Delaunayeve triangulacije ter kombinaciji rašitev iz posameznih algoritmov, predstavljenih v posameznih poglavjih.

8.4.1 Podatkovne strukture

Uporabljene podatkovne strukture so podobne strukturi DCEL, ki je opisana v podpoglavju 2.2.2.1, vendar so zaradi specifičnosti problema nekoliko poenostavljene. Osnovni podatek so točke v prostoru. Vrednosti koordinat x_i, y_i in z_i so shranjene v polju

int T[*STT*][*3*],

pri čemer je *STT* število vseh točk v prostoru in je postavljeno na začetek datoteke *tocke.dat*.
Pregledovalni rob, ki potuje preko površine, je določen s poljema

int PR[*stPR*][*2*],
int NPR[*stNPR*][*2*],

PR je aktiven pregledovalen rob, *NPR* je nov pregledovalen rob, ki nastaja z določanjem tretjih točk posameznemu robu v PR. Pred premikom *PR* se *NPR* prepiše v *PR*. Pregledovalni rob so kazalci na točke polja *T*.

Pri vsaki novo določeni tretji točki je določen nov trikotnik, ki je del končne rekonstrukcije površine. Trikotniki so določeni v polju

int TRIK[*stTRIK*][3],

s tem, da se lahko *stTRIK* poljubno spreminja glede na število vseh točk *STT*. *TRIK* so kazalci na točke polja *T*; vseh je *stTRIK*.

Druge strukture so enake osnovnim in so uporabljene za dodatne podatke. Niz vseh robov triangulacije - *KN*[*stKN*][2], lupina osnovne ploskve telesa, ki leži na ravnini $z = 0$ - *OP*[*stOP*][2], normale za posamezne trikotnike - *NOPR*[*stPR*][3] in *NONPR*[*stNPR*][3] ter novo nastala robova trikotnika za vsak rob *PR* - *R*1[2] in *R*2[2].

8.4.2 Algoritem rekonstrukcije površine

```
        procedure Rekonstrukcija površine(STT)
begin   stNPR := 0;
        stTRIK := 0;
```

```
stPR := stOP;
T [STT][3] := prebrani podatki iz datoteke tocke.dat;
PR[stPR][2] := prvi rob aktivnega regledovalnega roba;

while stPR > 0 do
        begin
        for iPR := 0 until iPR < stPR do
                begin
                ii := tretja točka za posamezen rob PR[iPR][2];
                if stTRIK := 0 then
                        begin
                        TRIK[stTRIK][0] := PR[iPR][0];
                        TRIK[stTRIK][1] := PR[iPR][1];
                        TRIK[stTRIK][2] := ii;
                        end.
                else
                        begin
                        pregled, če novo določeni trikotnik že obstaja
                                če ne
                                TRIK[stTRIK][0] := PR[iPR][0];
                                TRIK[stTRIK][1] := PR[iPR][1];
                                TRIK[stTRIK][2] := ii;
                        end.
                (* tretja točka ii določi dva nova robova *)
                R1[0] := PR[iPR][0];
                R1[1] := ii;
                R2[0] := ii;
                R2[1] := PR[iPR][1];
                pregled, če novo določen rob R1 že obstaja v PR ali NPR
                        če ne
                        NPR[stNPR][0] := R1[0];
                        NPR[stNPR][1] := R1[1];
                        stNPR := stNPR+1;
                pregled, če novo določen rob R2 že obstaja v PR ali NPR
                        če ne
                        NPR[stNPR][0] := R2[0];
                        NPR[stNPR][1] := R2[1];
                        stNPR := stNPR+1;
```

```
                end.
            PR[stPR][ ] := NPR[stNPR][ ];
            stPR := stNPR;
            stNPR := 0;
            end.
        izpis TRIK[stTRIK][ ];
end.
```

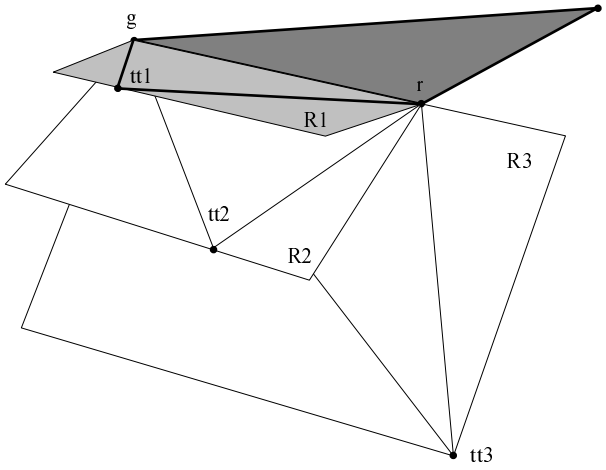
8.4.3 Začetek pregledovanja v prostoru

Začetek pregledovanja točk v prostoru je odvisen od urejenosti odčitanih točk. Če točke niso urejene, je začetni rob lahko takšen, kot je opisano zgoraj. Druga možnost je, da se prepoznajo točke, ki ležijo na osnovni ploskvi in določajo obod osnovne ploskve modela. Tako prepoznane točke je mogoče urediti v pregledovalni rob. Tâko pregledovanje ne izhaja iz enega samega roba, ampak iz celotnega niza. Ta način je primeren predvsem za simetrične oblike.

Pri branju podatkov zapiše funkcija *BranjePod()* vse tiste točke, ki imajo koordinato *z* manjšo od dopustne napake *delta*, v polje *OP*[*stOP*].

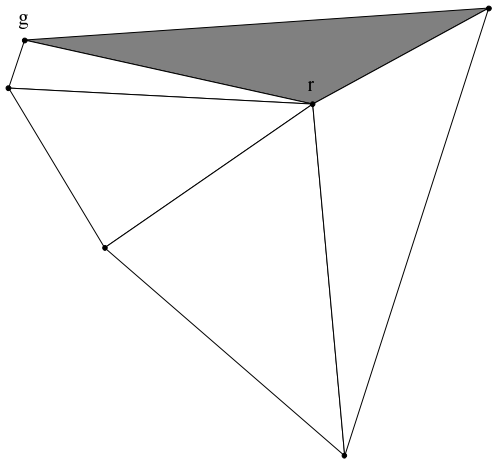
8.4.4 Princip tretje točke

Za vsak rob aktivnega pregledovalnega roba *PR*[*stPR*][2] je potrebno poiskati tretjo točko, ki z njim določi trikotnik Delaunayeve triangulacije. Postopek poteka toliko časa, dokler ni triangulirana celotna površina telesa. Tretja točka se za posamezen rob določi z ravnino, ki se zavrti okoli aktivnega roba *rg*. Odnos med tremi točkami v prostoru se prevede v ravninski problem, pri čemer gre ravnina skozi točke *r*, *g* in *tt*. Na Sliki 8.6 je prikazana določitev tretje točke v prostoru. Skozi točke *r*, *g* in tt_i se postavi ravnina (t.j. prostorski odnos se prevede v ravninskega). Kot je vidno na sliki, je tretja točka novega trikotnika točka tt_1.



Slika 8.6 Določitev tretje točke v prostoru.

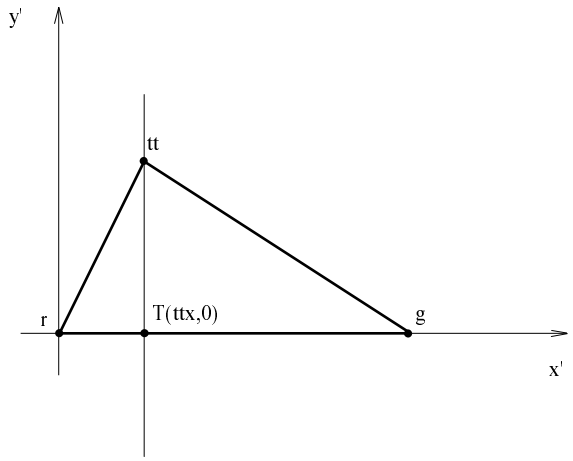
Na Sliki 8.7 je prikazana triangulacija za točke s Slike 8.6.



Slika 8.7 Triangulacija točk s Slike 8.6.

8.4.4.1 Postavljanje lokalnega koordinatnega sistema

Koordinatni sistem se iz 3D v 2D prestavi tako, da leži rob *rg* na os *x* tako, da je *rx* = 0 (Slika 8.8).



Slika 8.8 Lokalni koordinatni sistem.

Naj bodo:

$p_1 = [x_1, y_1, z_1]$,
$p_2 = [x_2, y_2, z_2]$,
$p_3 = [x_3, y_3, z_3]$

točke v prostoru, ki jih je potrebno prestaviti v ravninski koordinatni sistem. Po transformaciji so točke:

$p_1 \Rightarrow$ r = [0,0],
$p_2 \Rightarrow$ g = [*gx*,0],
$p_3 \Rightarrow$ tt = [*ttx*,*tty*],

pri čemer velja:

$gx = \text{sqrt}\ \{ (x_2-x_1)^2 + (y_2-y_1)^2 + (z_2-z_1)^2 \}$,
$ttx = (\vec{t}_{13} \cdot \vec{t}_{12}) / | \vec{t}_{12} |$,
$tty = | (\vec{t}_3 - \vec{t}_1) \times \vec{e}_{12} |$.

V novem koordinatnem sistemu je potrebno določiti *odsek*, t.j. razdaljo med središčem trikotniku očrtanega kroga in točko *T*(*ttx*,0). Odsek računa funkcija *Odsek()*, ki vrne +*d*, če središče trikotniku očrtanega kroga leži levo od *rg* in -*d*, če to leži desno od *rg*.

Poiskati je potrebno najmanjši pozitiven odsek ali največji negativen odsek. To naredi funkcija *TretjaTocka()*.

8.4.5 Računanje normale

Za vsak določen trikotnik triangulacije se postavi normalo, ki je določena kot vektorski produkt dveh vektorjev. Če je znan naklon posameznega trikotnika, se lahko določi kriterij, katere točke pridejo v poštev za tretjo točko in katere ne. To je omejitev, ki narekuje, da se pregledujejo samo točke, ki so levo od pregledovalnega roba. Na ravnini se je predpostavka levega testirala z vektorskim produktom, v prostoru pa se testira z mešanim produktom treh vektorjev. Glede na njihovo orientiranost in predznak mešanega produkta je moč ugotoviti, na kateri strani se nahaja tretja točka.

Normala za posamezen trikotnik je določena s:

```
for( n=0; n<3; n++ ) {
        RG[n] = T[R2[1]][n] - T[R1[0]][n];
        RII[n] = T[R1[1]][n] - T[R1[0]][n];
        }
NONPR[stNPR][0] = RG[1]*RII[2] - RG[2]*RII[1];
NONPR[stNPR][1] = RG[2]*RII[0] - RG[0]*RII[2];
NONPR[stNPR][2] = RG[0]*RII[1] - RG[1]*RII[0];
```

Mešani produkt določa šestkratnik volumna, ki ga določajo trije vektorji. V odvisnosti od predznaka mešanega produkta se lahko določi, ali tretja točka leži levo ali desno od pregledovalnega roba.

Mešani produkt za tri vektorje

T[g][ ]-T[r][ ], *NOPR[n][ ]-T[r][ ]* in *T[tt][ ]-T[r][ ]* je

```
      ( T[g][X]-T[r][X] ) * ( NOPR[n][Y]-T[r][Y] ) * ( T[tt][Z]-T[r][Z] )
+     ( T[g][Y]-T[r][Y] ) * ( NOPR[n][Z]-T[r][Z] ) * ( T[tt][X]-T[r][X] )
+     ( T[g][Z]-T[r][Z] ) * ( NOPR[n][X]-T[r][X] ) * ( T[tt][Y]-T[r][Y] )
-     ( T[tt][X]-T[r][X] ) * ( NOPR[n][Y]-T[r][Y] ) * ( T[g][Z]-T[r][Z] )
-     ( T[tt][Y]-T[r][Y] ) * ( NOPR[n][Z]-T[r][Z] ) * ( T[g][X]-T[r][X] )
-     ( T[tt][Z]-T[r][Z] ) * ( NOPR[n][X]-T[r][X] ) * ( T[g][Y]-T[r][Y] );
```

8.5 Vhod in izhod

8.5.1 Vhod

Vhodni podatki so pridobljeni s pomočjo mehanskega sistema za zajemanje prostorskih točk (7.5 Zapis zajetih prostorskih točk). Zapisane so v datoteki "*tocke.dat*". Prva se prebere vrednost STT, ki je število vseh točk v prostoru, nato

sledijo kordinatne vrednosti posameznih točk. Vrednosti so med seboj ločene s presledkom. Primer datoteke "*tocke.dat*" za točke v prostoru:

81
50 -20 15
45 -15 15
: : :
: : :

Vrednosti se shranijo:

STT
T[0][*X*] T[0][*Y*] T[0][*Z*]
T[1][*X*] T[1][*Y*] T[1][*Z*]
: : :
: : :

8.5.2 Izhod

Program zapiše izhodne podatke v datoteko "*3DPovrs.dxf*" v formatu *.dxf. S tem je omogočeno, da se rezultat Delaunayeve triangulacije lahko pogleda in uporabiti pri modeliranju npr. v AutoCAD-u. *.dxf datoteka je standardna ASCII tekst datoteka in jo je mogoče poljubno spreminjati za uporabo v drugih CAD okoljih [2].

Za vnos datoteke "*3DPovrs.dxf*" je potrebno v AutoCAD-u vpisati

Command: **dxfin** File name: **3DPovrs**

Poleg omenjene datoteke naredi program tudi datoteko "*3Dtocke.dxf*", v kateri so zapisane vse prebrane točke iz datoteke "*tocke.dat*" oz. polje *T*[*STT*][2].

Datoteka *.dxf je setavljena iz petih delov:

1. **Header**, v katerem so podani splošni podatki o risbi. Vsak parameter je sestavljen iz spremenljivke in pripadajoče vrednosti.
2. **Tables**, v katerem so definirani uporabljeni koordinatni sistemi, črte, oblike uporabljenih tekstov, pogledi, itd.
3. **Blocks**, v katerem so definirani bloki.

4. **Entities**, ki je jedro risbe. V tem delu so določeni vsi elementi, ki so vidni na sliki.
5. **End of file**.

Datoteki "*3DPovrs.dxf*" in "*3Dtocke.dxf*" imata že vpisane prve tri dele. Program na konec dopise še četrti del in zaključek. Vsi parametri v datoteki *.dxf, tudi nepomembni, ki bi jih lahko izpustili, so nastavljeni nevtralno.

Oblika dela 5. Entities za izris točke v prostoru je:

```
POINT
  8
0
 10                    //x
156.0
 20                    //y
31.0
 30                    //z
0.0
```

Oblika dela 5. Entities za izris trikotne ploskve v prostoru je:

```
3DFACE
  8
0
 10                    // prva točka
156.0
 20
31.0
 30
0.0
 11                    // druga točka
156.0
 21
44.0
 31
0.0
 12                    // tretja točka
155.0
 22
36.0
 32
15.0
 13                    // četrta točka je pri trikotnikih enaka tretji
155.0
 23
36.0
 33
15.0
```

Oblika datoteke *.dxf [2]:

```
  0                          // začetek dela Header
SECTION
  2
HEADER
                             // vpisani splošni podatki o risbi
    0
ENDSEC                       // konec dela Header
  0                          // začetek dela Tables
SECTION
```

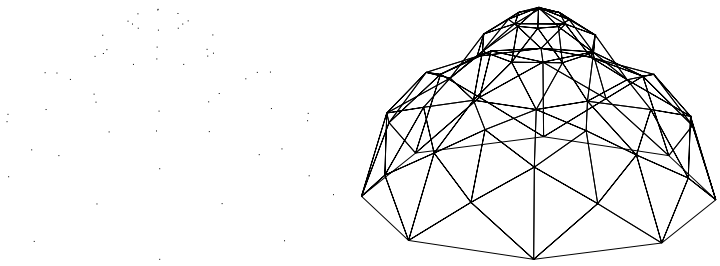
```
  2
TABLES
                        // vpisane definicije
    0
ENDSEC                  // konec dela Tables
  0                     // začetek dela Blocks
SECTION
  2
BLOCKS
                        //vpisane definicije
  0
ENDSEC                  // konec dela Blocks
  0                     // začetek dela Entities
SECTION
  2
ENTITIES
                        // vpisani podatki
  0
ENDSEC                  // konec dela Entities
  0
EOF                     // konec datoteke *.dxf
```

8.6 Rekonstrukcija površine

Iz površine modela so prostorske točke posnete s pomočjo sistema za zajem prostorskih točk in zapisane v datoteko "*tocke.dat*". Program naredi preko teh točk Delaunayevo triangulacijo v prostoru, ki je najboljša možna triangulacija in je hkrati tudi rekonstrukcija površine. Rezultat se zapiše v datoteko "*3DPovrs.dxf*"

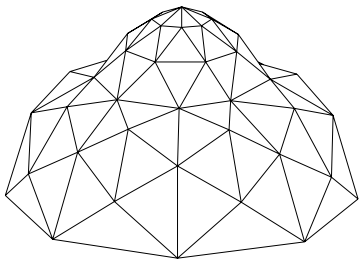
8.6.1 Oblika in število točk

Kakovost rekonstrukcije površine je odvisna od števila in izbire odčitanih točk. Na Sliki 8.9a je prikazan niz $n = 57$ točk v prostoru.



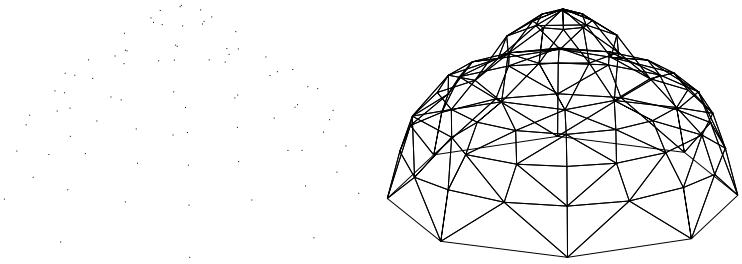
Slika 8.9 (a) Niz točk v prostoru N, $n = 57$; (b) robovi triangulacije.

Na Sliki 8.10 je prikazana rekonstrukcija površine. Vidi se, da je zaradi premajhnega števila odčitanih točk, površina nazobčana in ne gladka, kot bi bilo pričakovati glede na model.

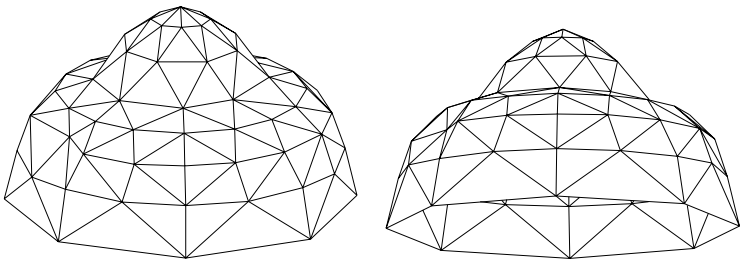


Slika 8.10 Rekonstrukcija površine za $n = 57$ prostorskih točk.

Obliko smo izboljšali s povečanjem odčitanih točk (Sliki 8.11 in 8.12).



Slika 8.11 (a) Niz točk v prostoru N, $n = 81$; (b) robovi triangulacije.

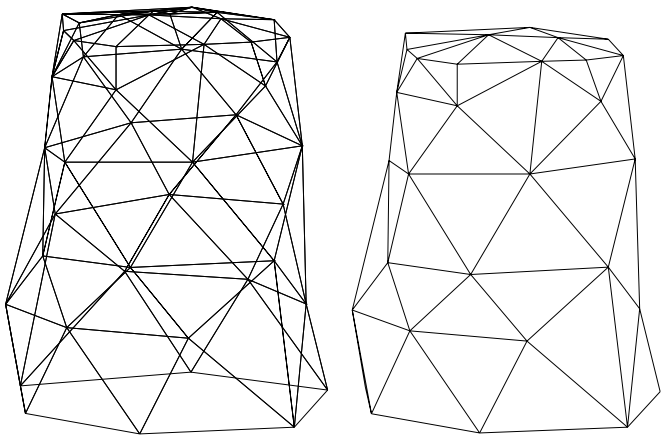


Slika 8.12 Rekonstrukcija površine za $n = 51$ prostorskih točk.

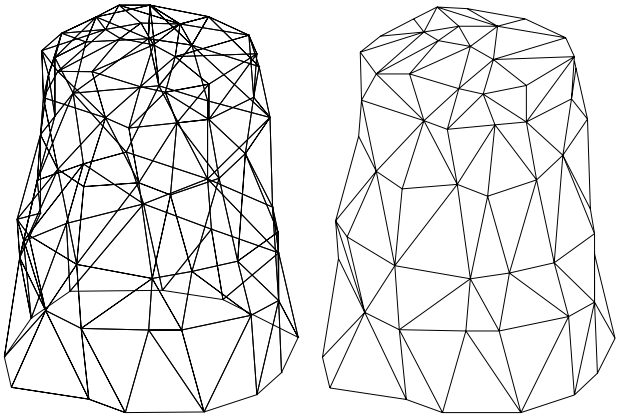
Na Sliki 8.12 je vidna napaka pri odčitavanju prostorskih točk.

8.6.2 Primer in uporaba

Na Slikah 8.13 in 8.14 je prikazana rekonstrukcija površine plastičnega lončka. Popačenje oblike je bilo obravnavano v poglavju 7.6. Prostorske točke so bile odčitane s pomočjo sistema za zajem prostorskih točk.V prvem primeru je bil niz točk N $n = 46$, v drugem pa $n = 81$. Vidi se vpliv števila točk niza N na kvaliteto rekonstrukcije. Napaka pri odčitavanju je zanemarljiva, saj doseganje natančnosti sistema za zajem prostorskih točk ni bila prioriteta naloge.

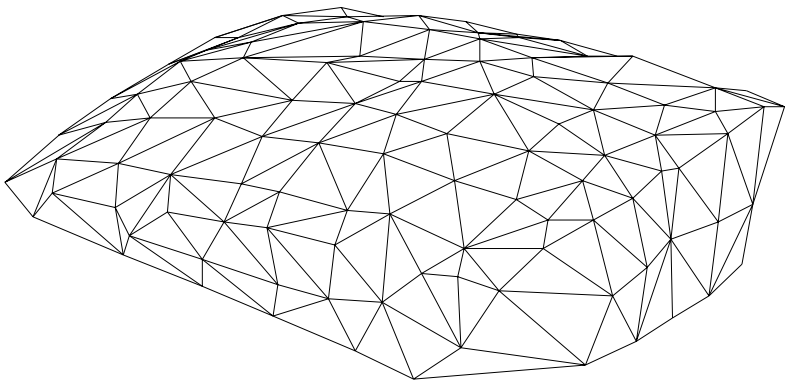


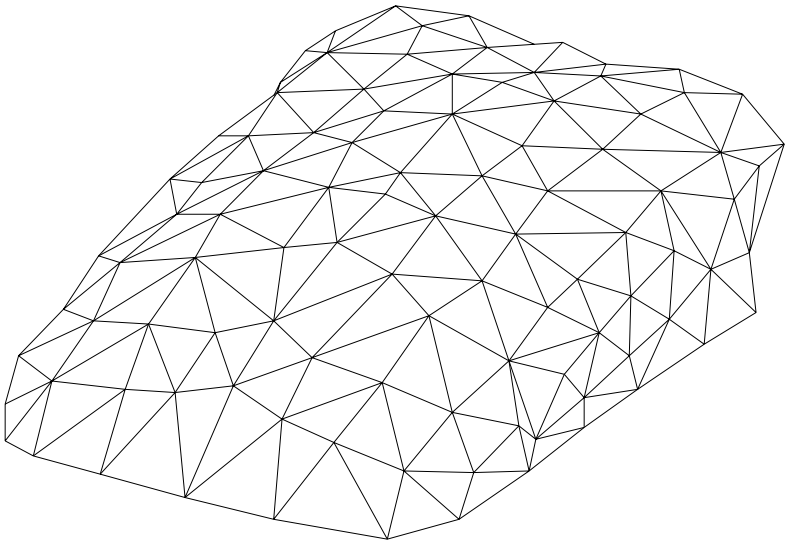
Slika 8.13 Rekonstrukcija površine plastičnega lončka $n = 46$.



Slika 8.14 Rekonstrukcija površine plastičnega lončka $n = 81$.

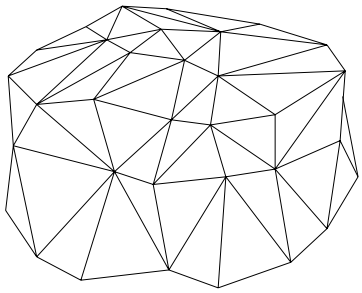
Na Slikah 8.15a in 8.15b je prikazana rekonstrukcija površine telefonskega aparata. Na osnovi te je mogoče v modelirniku oblikovati končno obliko.



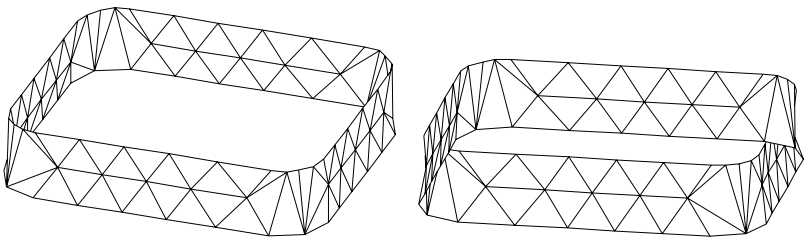


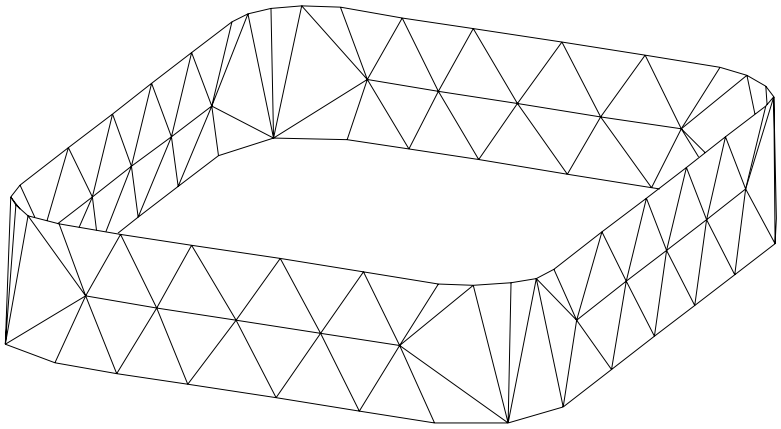
Slika 8.15 Rekonstruirana površina telefonskega aparata.

Na Slikah 8.16 in 8.17 je prikazana uporaba samo dela rekonstruirane površine. Na prvi je to del plastičnega lončka, na drugi pa površina podstavka.



Slika 8.16 Del rekonstruirane površine plastičnega lončka.





Slika 8.17 Del rekonstruirane površine podstavka.

8.7 Problemi

Pri rekonstrukciji nekonveksnih površin nastanejo problemi, ki jih pri konstrukciji konveksnih površin ni. Da do napak ne pride, je potrebno poskrbeti že pri zajemanju prostorskih točk. Predvsem mora biti gostota točk večja pri večji ukrivljenosti površine.

9 Zaključek

V sklopu diplomske naloge je bil izdelan mehanski sistem za zajem prostorskih točk, ki je v povezavi z računalnikom sposoben digitalizacije seznama neurejenih točk, in na osnovi algoritmov računske geometrije izdelan nov algoritem, ki je sposoben rekonstrukcije poljubne površine, ter program, ki zajete točke prebere in iz njih na osnovi novega algortima generira optimalno trikotniško mrežo.

Uporabljene so osnove računske geometrije in deli posameznih algoritmov. Noben že napisan algoritem ni zagotavljal rekonstrukcije poljubne (ne nujno konveksne) površine, zato je bilo potrebno izdelati nov algoritem.

Za triangulacijo se uporablja Delaunayeva triangulacija, ki da optimalno trikotniško mrežo, rezultat pa je v obliki *.dxf, kar omogoča uvoz v prostorski modelirnik.

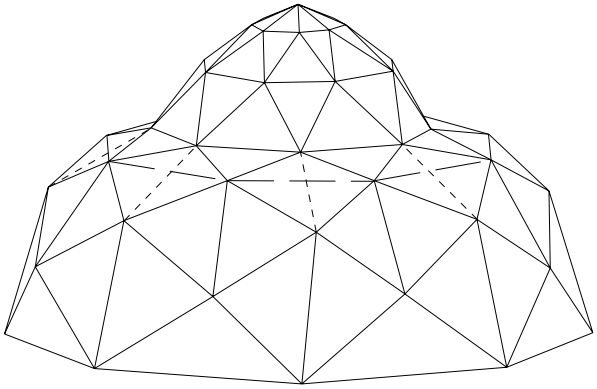
Izdelan je sistema za zajem prostorskih točk in algoritem za rekonstrukcijo površine ter program za rekonstrukcijo površine, napisan v programskem jeziku C, ki je prenosljiv med različnimi platformami.

Podrobno je predstavljen sistem za zajem prostorskih točk in način digitalizacije ter povezava z računalnikom. Prioriteta je v prikazu funkcionalnosti in uporabnosti in ne v natančnosti sistema. Natančnost je povezana s ceno in odločili smo se za najcenejšo možno varianto.

Predstavljen je algoritem, ki je bil preizkušen na praktičnih primerih. Rekonstruirane površine so uporabne samostojno ali v sklopu večje površine. V obeh primerih je površino v prostorskem modelirniku mogoče popravljati ali prilagajati, če v celoti ne ustreza zahtevam nadaljne uporabe.

Prostorske trikotniške mreže zaradi neprimerno odčitanih točk niso nujno najboljši posnetek realnega stanja. Izboljšava bi bila mogoča s posttriangulacijskimi metodami. Tu gre predvsem za glajenje površine in optimiranje trikotniških mrež [5]. Pri glajenju gre za to, da so trikotniki postavljeni po pravilih Delaunayeve triangulacije, vendar to ni nujno posnetek dejanske oblike (Slika). Navpični črtkani robovi so robovi Delaunayeve triangulacije, vendar rekonstrukcija površine ni natančen posnetek modela. Vodoravni črtkani robovi bi bili robovi nove, zglajene površine.

Pri optimiranju trikotniške mreže pa gre za zmanjšanje števila trikotnikov na ravnejših delih površine in za povečanje števila trikotnikov na delih površine z večjo ukrivljenostjo.



Slika Glajenje trikotniške mreže. Črtkana črta je rob po glajenju.

LITERATURA

[1] *ADC0831/0832/0834/0838 8-Bit Serial I/O A/D Converters with Multiplexer Options*, National Semiconductor Corporation, U.S.A., 1995.

[2] *AutoCAD® Release 11*, Reference Manual, Autodesk Ltd., England, August 1990.

[3] J. N. Bronštejn, K. A. Semendjajev, *Matematični priročnik*, Tehniška Založba Slovenije, Ljubljana, 1990, str. 226 in 246.

[4] Brian W. Kernighan, Denis M. Ritchie, *Programski jezik C*, Univerza Edvarda Kardelja v Ljubljani, Fakulteta za Elektrotehniko, Ljubljana, 1988.

[5] Leon Kos, Jože Duhovnik, *Physically based relaxation of unstructured meshes*, DMMI 97, 3rd int. conf., str. 241 - 247.

[6] Pavlina Mizori - Oblak, *Matematika za študente tehnike in naravoslovja - prvi del*, Univerza Edvarda Kardelja v Ljubljani, Fakulteta za Strojništvo, Ljubljana 1983. Tretja dopolnjena izdaja, 1989, str. 93 - 99.

[7] K. Mulmuley, *Computational Geometry: an Introduction Through Randomized Algorithms*, Prentice - Hall, Inc., 1994.

[8] Joseph O'Rourke, *Computational Geometry in C*, Cambridge University Press, 1993. Reprinted with corrections, 1994.

[9] Č. Oblonšek, *Rekonstrukcija funkcij, ploskev in teles iz razpršenih podatkov*, Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko, Maribor, Marec 1995.

[10] Franco P. Preparata, Michael Ian Shamos, M. I., *Computation Geometry: an Introduction (Texts and monographs in computer science)*, Springer - Verlag, New York, 1985. Third Corrected Printing, 1990.

[11] Dave Watson, watson@maths.uwa.edu.au, *Delaunay triangulation & convex hull in 2D or 3D (nnsort.c)*, Department of Mathematics, The University of Western Australia, Perth, December 1993.

[12] Bill Weidenauer, *Amstrad personal computer PC 1640*, Tehnical Reference Manual, Amstrad, 1987.

Dodatek

V dodatku so prikazani pomembnejši deli programske kode. Najprej del, ki omogoča komunikacijo računalnika z AD konverterjem in zapis koordinat v datoteko tocke.dat in na koncu še del, ki naredi rekonstrukcijo površine in rezultat vpiše v datoteko 3Dpovrs.dxf.

Funkcija omogoča komuniciranje z AD konverterjem.

```
unsigned char ReadAD(int channel)
{
	int iport = 0x379;
	int oport = 0x37a;
	int i;
	unsigned char a, b;
	char value;

	channel = (channel >> 1) | (channel << 2);

	outportb( oport, 0x05 );

	a = 0x18 | ( unsigned char ) channel;	/* določi kanal */
	b = 0x01;				/* clk in cs na nič */
	outportb( oport, b );

	for( i=0; i<5; i++ ) {
		if( a & 0x10 )
			outportb( oport, b &= 0xF7 );
		else
			outportb( oport,  b |= 0x08 );

		outportb( oport,  b &= 0xFE );		/* ugasne bit 0 */
		outportb( oport,  b |= 0x01 );		/* prižge bit 0 */

		a = a << 1;
	}

	a = 0x0;

	delay(4);

	for( i=0; i<8; i++ ) {
		outportb( oport,  b &= 0xFE );		/* ugasne bit 0 */
		outportb( oport,  b |= 0x01 );		/* prižge bit 0 */
		a = ( ( ~inportb( iport ) & 0x80 ) >> 7 ) | (a << 1);
	}
	b = 0x02;					/* cs na ena */
	outportb( oport,  b );
	return a;
}
```

Datoteka, v katero se zapisujejo z mehanskim sistemom odčitane točke.

```
FILE *zapis;
zapis = fopen( "\\ \\ \\tocke.dat", "w" );
```

Vpisane dolžine posameznih palic mehanskega sistema.

```
l1 = 198;
l2 = 200;
l3 = 197;
```

Odsekovno linearna interpolacija za kot alfa. Podobno je tudi za ostala dva kota.

```
if( ( ReadAD(0) >= (A-44) ) && ( ReadAD(0) < (A-29) ) ) {
        ka = -1;
        na = 45+A-44; }
else if( ( ReadAD(0) >= (A-29) ) && ( ReadAD(0) < (A-15) ) ) {
        ka = -1.071428571;
        na = 30-ka*(A-29); }
else if( ( ReadAD(0) >= (A-15) ) && ( ReadAD(0) < (A+15) ) ) {
        ka = -1;
        na = A; }
else if( ( ReadAD(0) >= (A+15) ) && ( ReadAD(0) < (A+29) ) ) {
        ka = -1.071428571;
        na = -15-ka*(A+15); }
else if( ( ReadAD(0) >= (A+29) ) && ( ReadAD(0) <= (A+44) ) ) {
        ka = -1;
        na = -30+A+29; }
```

Določanje kota v stopinjah.

```
        a = ka * ReadAD(0) + na;
        b = kb * ReadAD(1) + nb;
        c = kc * ReadAD(2) + nc;
```

Določitev koordinat točke v prostoru.

```
        d = l2 * cos( beta ) - l3 * cos( gama - beta );
        x = d * cos( alfa );
        y = d * sin( alfa );
        z = l1 - l2 * sin( beta ) - l3 * sin( gama - beta);
```

Zapis koordinat v datoteko.

```
fprintf( zapis, "\n%d\t%d\t%d", x, y, z );
```

BranjePod prebere podatke iz datoteke tocke.dat. Točke, pri katerih je koordinatna vrednost manjša od dopustne napake delta, popravi in vpiše v OP[stOP]. S pomočjo tega polja se določi začetek pregledovanja.

```
void    BranjePod( void ) {
        FILE *podatki;
        int i;
        podatki = fopen( "\\ \\ \\tocke.dat", "rt" );
```

```
	fscanf( podatki, "%i", &STT );

	T = ( int** ) malloc( sizeof( int* ) *STT );
	for( i=0; i<STT; i++ )
		T[i] = (int*) malloc( sizeof( int ) *DIM );
	for( i=0; i<STT; i++ ) {
		fscanf( podatki, "%i%i%i", &T[i][X], &T[i][Y], &T[i][Z] );
		if( T[i][Z] < delta ) {
			T[i][Z] = 0;
			OP[stOP] = i;
			stOP++;
		}
	}
	fclose(podatki);
}
```

Zapiše trikotnike mreženja v datoteko 3Dpovrs.dxf. Ta oblika zagotavlja uvoz v prostorski modelirnik. Podobna je funkcija, ki zapisuje vse odčitane točke v 3Dtocke.dxf.

```
int	PisanjE( stTRIK ) {
	FILE *dxf;
	int i;

	/* prvi del datoteke je že narejen */

	dxf = fopen( "\\ \\ \\3dpovrs1.dxf", "a" );

	fprintf( dxf, "ENTITIES" );

	/* zapiše trikotnie v datoteko *.dxf */

	for( i=0; i<stTRIK; i++ ) {
		fprintf( dxf, "\n  0\n3DFACE\n  8\n0" );
		fprintf( dxf, "\n 10\n%d.0\n 20\n%d.0\n 30\n%d.0", T[TRIK[i][0]][0], T[TRIK[i][0]][1],
	T[TRIK[i][0]][2] );
		fprintf( dxf, "\n 11\n%d.0\n 21\n%d.0\n 31\n%d.0", T[TRIK[i][1]][0], T[TRIK[i][1]][1],
	T[TRIK[i][1]][2] );
		fprintf( dxf, "\n 12\n%d.0\n 22\n%d.0\n 32\n%d.0", T[TRIK[i][2]][0], T[TRIK[i][2]][1],
	T[TRIK[i][2]][2] );
		fprintf( dxf, "\n 13\n%d.0\n 23\n%d.0\n 33\n%d.0", T[TRIK[i][2]][0], T[TRIK[i][2]][1],
	T[TRIK[i][2]][2] );
	}

	/* napiše konec *.dxf datoteke */

	fprintf( dxf, "\n  0\nENDSEC\n  0\nEOF" );
}
```

Naslednja funkcija računa dvojno površino trikotnika. To je uporabljeno pri določevanju lege središča trikotniku očrtanega kroga.

```
float	Povrsina2( float rx, float ry, float gx, float gy, float ttx, float tty ) {
```

```
        return
                ( rx * gy ) - ( ry * gx )   +
                ( ry * ttx ) - ( rx * tty ) +
                ( gx * tty ) - ( ttx * gy );
}
```

Volumen6 računa šestkratni volumen med štirimi točkami. Za določanje položaja točke glede na ravnino.

```
float   Volumen6( int r, int g, int n, int tt ) {

        return
                ( T[g][X]-T[r][X] ) * ( NOPR[n][Y]-T[r][Y] ) * ( T[tt][Z]-T[r][Z] )
            +   ( T[g][Y]-T[r][Y] ) * ( NOPR[n][Z]-T[r][Z] ) * ( T[tt][X]-T[r][X] )
            +   ( T[g][Z]-T[r][Z] ) * ( NOPR[n][X]-T[r][X] ) * ( T[tt][Y]-T[r][Y] )
            -   ( T[tt][X]-T[r][X] ) * ( NOPR[n][Y]-T[r][Y] ) * ( T[g][Z]-T[r][Z] )
            -   ( T[tt][Y]-T[r][Y] ) * ( NOPR[n][Z]-T[r][Z] ) * ( T[g][X]-T[r][X] )
            -   ( T[tt][Z]-T[r][Z] ) * ( NOPR[n][X]-T[r][X] ) * ( T[g][Y]-T[r][Y] );
}
```

Naslednja funkcija določi najnižjo točko, t.j. prvo točko prvega pregledovalnega roba.

```
int     NajnizTocka( void ) {

        int i;                          /*indeks točke*/
        int m = 0;              /*indeks najnižje točke doslej*/

        for ( i = 1; i < stOP; i++) {
                if ( (T[OP[i]][Y] < T[OP[m]][Y]) ||
                   ((T[OP[i]][Y] == T[OP[m]][Y]) &&
                    (T[OP[i]][X] > T[OP[m]][X]))
                  )
                        m = i;
                }
        return OP[m];
}
```

Odsek izračuna odsek za pregledovalni rob in tretjo točko. Glede na velikost in predznak odseka se določi tretja točka.

```
float   Odsek( int r, int g, int tt ) {

        int a, i;
        float pov2,  sx, sy, d2,  rx, ry, gx, gy, ttx, tty,  tx2, ty2, kkt2, n2;

        /* za transformacijo */

        float Tn[3][2];                 /* nov koord. sist. */
        float t12[3], t13[3]; /* vektorja 12 in 13 */
        float abs_t12;                  /* abs vrednost 12 */
        float e12[3];                   /* enotski vektor 12 */
        float p, d, di, dj, dk;

        for( i=0; i<=2; i++ ) {
                t12[i] = T[g][i] - T[r][i];     /* doloci vektor 12 */
```

```
}

abs_t12 = sqrt( t12[0]*t12[0] + t12[1]*t12[1] + t12[2]*t12[2] );

for( i=0; i<=2; i++)
        e12[i] = t12[i] / abs_t12;

/* p je projekcija */

p = ( ( t13[0]*t12[0] + t13[1]*t12[1] + t13[2]*t12[2] ) / abs_t12 );

/* t31 x e12 */
di = t13[1]*e12[2] - t13[2]*e12[1];
dj = t13[2]*e12[0] - t13[0]*e12[2];
dk = t13[0]*e12[1] - t13[1]*e12[0];

/* d je razdalja od premice do točke */

d = sqrt( di*di + dj*dj + dk*dk );

/* točke v novem koordinatnem sistemu */

sx = 0;
sy = 0;
rx = 0;
ry = 0;
gx = abs_t12;
gy = 0;
ttx = p;
tty = d;

a = 2;
if(  ( rx == ttx )  || ( gx == ttx ) ) a = 1;
switch( a ) {
        case 1:
                d2 = ( tty * tty ) / 4;
                sx = gx / 2;
                sy = tty / 2;
                break;
        case 2:
                tx2 = ttx / 2;
                ty2 = tty / 2;
                kkt2 = -( tx2 / ty2 );
                n2 = ty2 - kkt2 * tx2;
                sx = gx / 2;
                sy = kkt2 * sx + n2;
                d2 = sy * sy;
                break;
}
pov2 = Povrsina2( rx, ry, gx, gy, sx, sy );

/* poz d2, če lezi (sx,sy) levo oz. neg d2, če lezi (sx, sy) desno */
```

```
        if( pov2 < 0 ) return -(d2);
        else return d2;
}
```

Funkcija vrne indeks tretje točke.

```
int     TretjaTocka( int r, int g, int iPR ) {

        int i;
        int iNeg, iPoz;
        float dPozMin, dNegMax;
        float Ods, Vol;

        dPozMin = 20000;
        dNegMax = 0;
        iNeg = 9999;
        iPoz = 9999;

        /* zanka preko vseh točk na ravnini */

        for( i=0; i<STT; i++ ) {
                if( (i==r) || (i==g) ) goto Naslednja;
                if( (l == 1) &&
                        (Povrsina2( T[r][X], T[r][Y], T[g][X], T[g][Y], T[i][X], T[i][Y] ) == 0 )
                         ) goto Naslednja;
                if ( l == 1 ) goto PrviRob;
                Vol = Volumen6( r, g, iPR, i );
                if( Vol >= 0 ) goto Naslednja;          /* točka je pred ravnino mora biti zadaj */
                PrviRob:;

                Ods = Odsek( r, g, i );

                /* če je središče kroga levo - i za sx in sy */

                if ( ( Ods >= 0 ) && ( Ods < dPozMin ) ) {
                        dPozMin = Ods;
                        iPoz = i;
                }

                /* če je središče kroga desno - i za sx in sy */

                if( ( Ods < 0 ) && ( Ods < dNegMax ) ) {
                        dNegMax = Ods;
                        iNeg = i;
                }
                Naslednja:;
        if( iNeg != 9999 ) return iNeg;
        else return iPoz;
}
```

Preveri, če so novonastali robovi pregledovalnega roba že bili upoštevani. Podoben način je pri preverjanju tikotnikov.

```
int     Pregled( int R[2], int Niz[stt][2], int stEl ) {
```

```
        int i;
        if( stEl == 0 ) return 1;
        for( i=0; i<stEl; i++ ) {
                if( ( R[0] == Niz[i][0] ) && ( R[1] == Niz[i][1] ) ||
                  ( R[1] == Niz[i][0] ) && ( R[0] == Niz[i][1] ) )
                        return 0;
        }
        return 1;
}
```

Začetne vrednosti števcev:

```
        stPR = 1;
        stNPR = 0;
        stKN = 0;
        stTRIK = 0;

        BranjePod();
```

Določi se prvi pregledovalni rob.

```
        PR[0][0] = NajnizTocka();
        PR[0][1] = DrugaTocka();
```

Pregledovanje poteka, dokler je v PR še kakšen aktiven pregledovalni rob.

```
        while( stPR>0 ) {

                for( iPR=0; iPR<stPR; iPR++ ) {
```

Določi se tretja točka posameznem robu.

```
                        ii = TretjaTocka( PR[iPR][0], PR[iPR][1], iPR );
                        if( ii == 9999 ) goto NaslednjiRob;
```

Pregleda trikotnike. Če novo določen trikotnik še ne obstaja, se ga vpiše v niz.

```
                        if( PregleD( stTRIK, PR[iPR][0], PR[iPR][1], ii ) ) {
                        TRIK[stTRIK][0] = PR[iPR][0];
                        TRIK[stTRIK][1] = PR[iPR][1];
                        TRIK[stTRIK][2] = ii;
                        stTRIK++;
```

Pregleda vsak novo nastali rob in določi normalo trikotnika.

```
                        if( ( Pregled( R1, KN, stKN ) )   &&
                          ( Pregled( R1, PR, stPR ) )   &&
                          ( Pregled( R1, NPR, stNPR ) ) ) {

                        for( n=0; n<3; n++ ) {
                                RG[n] = T[R2[1]][n] - T[R1[0]][n];
                                RII[n] = T[R1[1]][n] - T[R1[0]][n];
```

```
}
NONPR[stNPR][0] = RG[1]*RII[2] - RG[2]*RII[1];
NONPR[stNPR][1] = RG[2]*RII[0] - RG[0]*RII[2];
NONPR[stNPR][2] = RG[0]*RII[1] - RG[1]*RII[0];
```

Vpisuje v novo nastali pregledovalni rob.

```
NPR[stNPR][0] = R1[0];
NPR[stNPR][1] = R1[1];
stNPR++;
```

Prepiše robove in trikotnikom pripadajoče normale iz NPR v PR

```
for( iNPR=0; iNPR<stNPR; iNPR++ ) {
        PR[iNPR][0] = NPR[iNPR][0];
        PR[iNPR][1] = NPR[iNPR][1];

NOPR[iNPR][0] = NONPR[iNPR][0];
NOPR[iNPR][1] = NONPR[iNPR][1];
NOPR[iNPR][2] = NONPR[iNPR][2];
stPR = stNPR;
stNPR = 0;
```

Zapiše vhodne in izhodne poratke v datoteko *.dxf

```
PisanjE( stTRIK );
PisanjET( STT );
```

IZJAVA

Diplomsko delo sem samostojno izdelal pod vodstvom mentorja izr. prof. dr. Jožeta Duhovnika, dipl. ing.

Aleš DROLC

Ljubljana, dne 11.9.1997