TARTU ÜLIKOOL
MATEMAATIKA-INFORMAATIKATEADUSKOND
ARVUTITEADUSE INSTITUUT
TARKVARASÜSTEEMIDE ÕPPETOOL
INFORMAATIKA ERIALA

Dan Bogdanov

Optimaalse teekonna leidmise ülesande lahendamine geomeetriliste objektide puude abil

Semestritöö

Juhendaja: dots. A. Isotamm

Autor: …… "….." mai 2004
Juhendaja: ……. "….." mai 2004
Õppetooli juhataja: …… "….." …….. 2004

TARTU 2004

Sisukord

Sissejuhatus

Teekonna planeerimine on aktuaalne probleem mitmes eluvaldkonnas. Vaatamisväärsusi külastava matkaraja ettevalmistamine või luurerühma vaenlase eest varjatud liikumine on kaks maastikule tuginevat otsinguülesannet. Kui soovime koostada lennukile marsruuti, mis oleks igal hetkel ülimalt 100 kilomeetri kaugusel mõnest lennuväljast või programmeerime kõrghoone jaoks majajuhti, mis juhiks külastaja kõige kiiremat teed mööda sihini, peame otsinguruumi laiendama kolmemõõtmelisse ruumi.

Käesolevas töös üldistatakse teeotsinguülesanne maastikul ja objektide ruumis ning jagatakse see alamülesanneteks, mille lahendamiseks võib kasutada mitmeid erinevaid meetodeid. Eraldiseisvate etappidena vaadeldakse lähteandmete põhjal mudeli loomist, selle põhjal graafi koostamist ja otsingut loodud graafil.

Esitatakse kaks geomeetriliste objektide puustruktuuril põhinevat mudelit maastiku modelleerimiseks: kolmnurkade kahendpuu ja ruutude neljandpuu. Mõlemat struktuuri on siiani kasutatud peamiselt maastiku visualiseerimisel ja nähtavuse arvutamisel. Töös kirjeldatakse lihtsaid algoritme, mille abil saame luua maastikuga hästi kohanevaid võresid, millele saab peale ehitada graafistruktuuri.

Graafi koostamise ja otsingu etappide lahendustena pakutakse välja kõrguskaardil põhinev graafi serva kaalu arvutamise meetod ja otsingualgoritm A*.

Tulemuste illustreerimiseks kasutatakse arvutiprogrammi 'Reisija', mille kirjeldus ja kood on esitatud töö lisana.

1. Ülesande püstitus ja lähteandmete kirjeldus

Tee leidmine on otsinguülesande üks tüüpe. Käesolevas töö eesmärgiks on eelnevalt seatud optimaalsuse tingimustele vastava tee leidmine maastikul või objektidega asustatud ruumis. Lähteandmeteks on otsinguruum, teekonna algus- ja lõpp-punkt ja täiendavad tingimused otsingu tulemusele. Otsinguruumi taustsüsteemiks on kolmemõõtmeline suunatud sirglõikude vektorruum. Maastiku kui otsinguruumi defineerime kui deformeerunud tasandi, mida saab esitada kõrguskaardina. Kõrguskaart on arvude maatriks, mis seab iga maastiku punktiga vastavusse selle vertikaalse kauguse nulltasandist.

Objektide ruumi defineerime geomeetriliste kehade hulgana, kus igal kujundil on oma kindlad koordinaadid ja paigutus antud vektorruumis. Lähteandmed esitatakse nende kehade nimekirjana. Keha võib esitada hulknurkade hulgana või definitsiooni ja atribuutidega (kera keskpunktiga koordinaadil (-2,5; 0,3; 7,4) ja raadiusega 3,8 ühikut).

Teekonna algus- ja lõpp-punkt esitatakse nende koordinaatidega otsinguruumis. Teekonnale saab esitada ka nõudeid vahepeatuste ja nähtavuse kohta. Vahepeatusteks on punktid, mida otsitav teekond peaks kindlasti läbima. Teekonnalt võib nõuda kindla vaatluspiirkonna sisse jäämist või vastupidi – selle piirkonna vältimist. Näiteks võime soovida, et lennumarsruutide puhul oleks lennuk pidevalt mõne juhtimistorniga kontaktis või inimrühm läbiks maastiku vaenlase luurajate vaatevälja sattumata.

Optimaalsuskriteeriumid määratakse otsinguruumis liikuvast agendist lähtuvalt. Oluline on teada, kas agent vajab liikumisel enda all kandvat pinda või suudab lennata. Vajalik on välja uurida milliseid tõusunurki ja takistusi suudab agent ületada, kui ta liigub kandval pinnal, ning kui palju energiat selleks kulub. Ka on tarvis teada, kui palju energiat kulub langeval teel pidurdamiseks.

Kui otsinguruumis on eri piirkondadele omistatud erinevad läbimistingimused (maastikul näiteks sile tee või raskestiläbitav võsa), siis peab ka agendi liikumisprofiili koostamisel neid arvestama. Maastikul saab erineva läbitavusega piirkondi märkida maapinnal hulknurgaga piiratud alana, ruumis peab kasutama hulktahukat, mis vastava ruumiosa kataks.

2. Otsinguruumi töötlemine

Otsinguruumi töötlemine ja graafi koostamine on jaotatud kolme etappi: otsinguruumi mudeli loomine, sellel rajapunktide (graafi tippude) märkimine ja rajapunktide ühendamine otsingugraafiks. Igal etapil on oma sisend- ja väljundandmed, mille vormingud ei ole kindlaks määratud. Selline jaotus võimaldab kasutatavaid meetodeid korrigeerida ülejäänud protsessi muutmata.

2.1 Mudeli loomine

Reaalne maailm ei ole sobiv otsinguruum, sest seal pole tee algoritmiliseks leidmiseks sobilikke struktuure. Graafi koostamiseks peame lähteandmete (mis on ka vaid reaalsuse mudel) põhjal koostama võimalikult efektiivse ja detaile säilitava mudeli – teise lähenduse reaalsele maailmale.

Uuele mudelile esitatakse järgmised tingimused:

1) Kooskõla reaalse maailmaga

Otsingul kasutatud mudelil saavutatud tulemused peavad olema rakendatavad reaalses maailmas. Olulisel kohal on siin lähteandmete täpsus – kui esimene lähend on ebatäpne, siis sisaldab teine samu vigu.

2) Arvutuslihtsus

Lähteandmete põhjal uue mudeli loomine ei tohi olla väga keerukas ja aeganõudev protsess. See seab piirangud uue mudeli saamisel kasutatavatele analüüsimeetoditele.

3**) Salvestusmahu kokkuhoid**

Mudel ei tohi olla liiga mahukas. Võib juhtuda, et mudel on kergesti arvutatav, kuid selle edasine töötlemine järgmistel etappidel ei ole enam otstarbekas.

Näitena võiksime vaadelda järgnevat mudelit: kolmemõõtmelises ruumis muudame iga eristatava koordinaadi graafi tipuks. Tegemist on reaalse maailmaga kooskõlas oleva mudeliga, mida on väga lihtne koostada. Kuid mäluvajadus on liiga suur - kuubikujulise ruumi puhul küljepikkusega n vajaksime kirjeldatud meetodil koostatud mudeli hoidmiseks mälu $O(n^3)$. Koostatavas graafis oleks suurusjärgus $O(n^3)$ tippu ja vähemalt $O(n^3)$ serva, mida on selgelt liiga palju

2.2 Mudeli analüüs graafi tippude määramiseks

Otsinguruumina kasutame loodud mudelit. Järgmise sammuna leiame mudelit analüüsides koostatava graafi tipud. Tippude valimise meetodid sõltuvad kasutatavast mudelist, kuid iga mudeli tüübi jaoks võib leiduda mitmeid meetodeid.

Sellel etapil tehtav töö sõltub mudeli koostamisel tehtust. Kui mudeli koostamise meetod oli keerukas ja analüütiline, siis võib graafi tippude määramine sellel mudelil olla väga lihtne. Samas võib tippude määramine sõltumatult mudelist olla keeruline protsess.

Näitena vaatleme mudelit, mis on loodud objektide ruumi põhjal. Olgu mudel selline, mis jagab ruumi kuubikujulisteks osadeks küljepikkusega a. Kui kuubi sees ei asu (ka osaliselt mitte) ühtegi ruumis määratud geomeetrilist keha, siis paigutame selle kuubi keskele graafi tipu. Sellise mudeli koostamine on lihtne. Tema täpsus oleneb kuubi küljepikkusest a. Mida väiksem on a, seda täpsemalt kajastab mudel ruumi. Samas raiskab selline mudel mälu, sest ka suuremad avatud ruumid lõigatakse tükkideks lahti. Võib tekkida probleeme kitsaste ruumidega, millest teed otsides kuupide ebasobiva paigutuse tõttu läbi ei pääse.

2.3 Graafi servade lisamine

Kui graafi tipud on teada, lisame nende vahele servad. Servade lisamiseks on samuti mitmeid võimalusi, mis sõltuvad mudelist ja tippude määramise meetodist. Otsinguks kasutame suunatud graafi, sest kahe punkti vahel liikumisele kuluv energia võib sõltuvalt suunast erineda.

Serva lisamisel tuleb arvesse võtta liikuva agendi võimalusi. Serva tippude koordinaate ja otsinguruumi mudelit kasutades saab agendi parameetrite põhjal omistada servale kaalu – läbimiseks kuluva energia. Otsitava tee optimaalsuse garantiiks ongi parameetritele vastavate kaalude andmine graafi servadele. Võib ka juhtuda, et agent ei ole võimeline antud tippude vahel liikuma (sein on ees, liiga järsk tõus/langus, puudub lennuvõime). Sellisel juhul ei saa graafi antud suunas serva lisada.

Vaatleme näidet, kus objektide ruumi mudel koostatakse järgnevalt: kõik ruumis paiknevad objektid ümbritsetakse risttahukatega, mis mahutavad need objektid võimalikult täpsed. Risttahukate küljed

on paralleelsed vektorruumi baasi vektoritega (telgedega). Graafi tippudeks on risttahukate tipud. Iga tipu juures luuakse serv kõigi tippude juurde, mis on sellest tipust nähtavad (tee peal ei ole ühtegi risttahukat). Kui nüüd agent ei oma lennuvõimet, siis eemaldatakse kõik need servad, mille puhul agent ei saa „jalga maha panna“ ning alles jäävad „nurgast nurgani“ sihid.

3. Puude kasutamine otsinguruumi töötlemisel

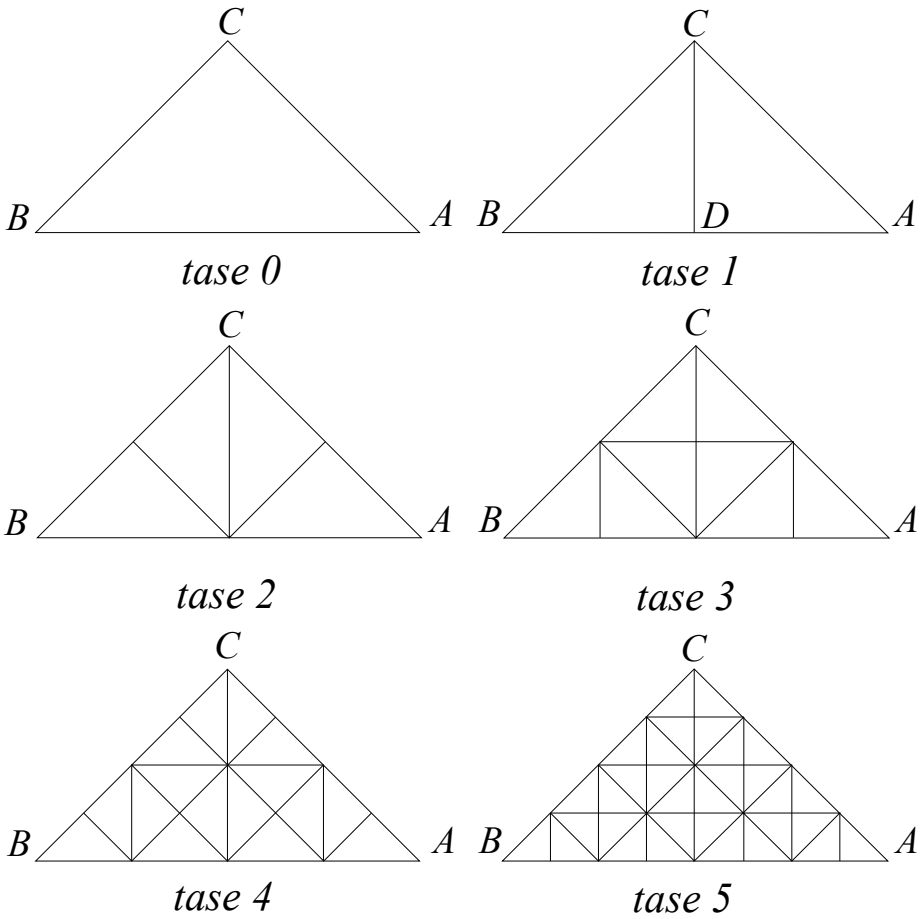
3.1 Kolmnurkade kahendpuu

3.1.1 Kahendpuu struktuur

Kolmnurkade kahendpuu kasutamist maastiku modelleerimisel ja graafi koostamisel on kirjeldatud artiklis [1]. Käesolevas töös on esitatud selle lahenduse parandatud ja täiendatud kuju.

Kolmnurkade kahendpuu on puu, mille lehtedeks on täisnurksed võrdhaarsed kolmnurgad. Andmestruktuuri ja selle rakendamist maastiku visualiseerimisel kirjeldab [2]. Algoritmi realiseerimist käsitleb [3].

Joonisel 3.1 on kujutatud täieliku kahendpuu esimesed 6 taset.

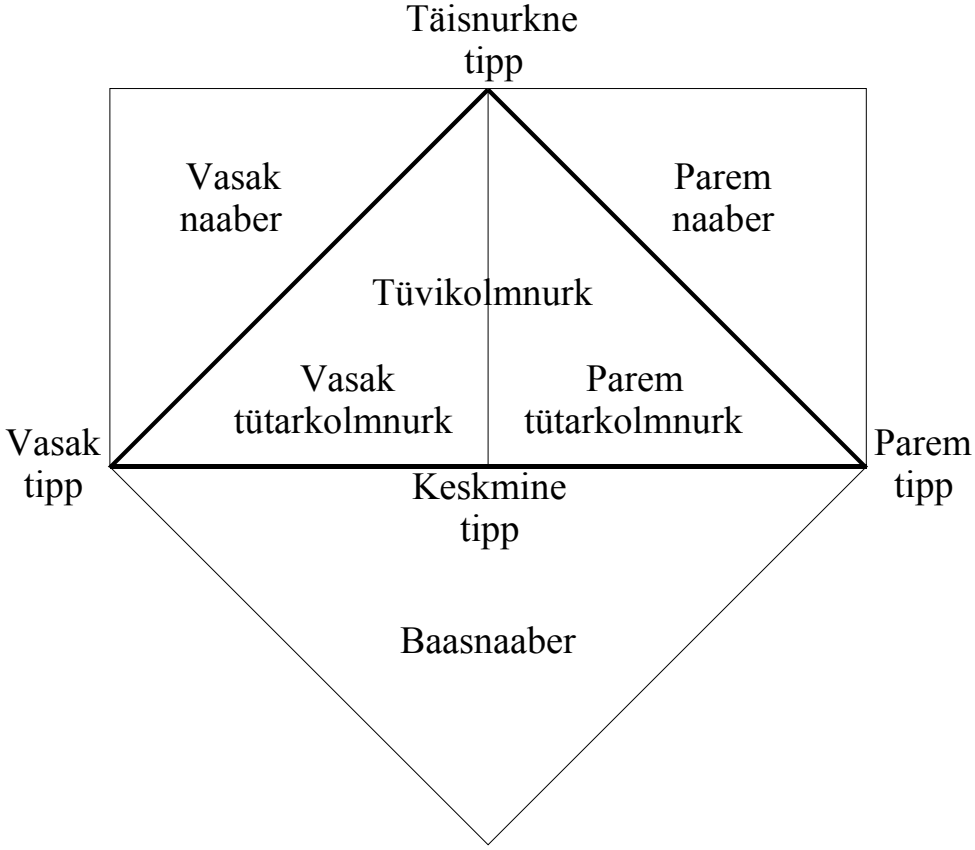


Joonis 3.1: Kolmnurkade kahendpuu astmed 0 - 5

Puu tüvi ehk põhikolmnurk on *ABC*, täisnurkne võrdhaarne kolmnurk kõige madalamal jaotuse astmel. Järgmisel astmel on *ABC* poolitatud sirgega kolmnurga tipu *A* ja kolmnurga aluse *AB*

keskpunkti *D* vahel. *BCD* ja *ADC* on tüvikolmnurga *ABC* tütarkolmnurgad.

Puus on defineeritud ka hulk seoseid. Igal puu kolmnurgal on ka naabrid: tüvikolmnurgaga hüpotenuusi jagav baasnaaber ning vasak ja parem naaber, mille hüpotenuusideks on vastavalt tüvikolmnurga vasak ja parem kaatet.



Joonis 3.2: Suhted kolmnurkade kahendpuus

3.1.2 Mudeli koostamine

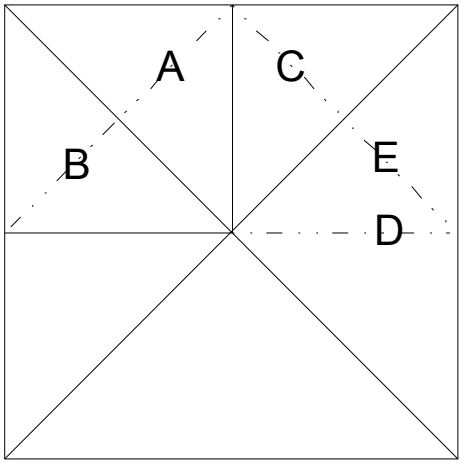
Mudeli koostamine kolmnurkade kahendpuu abil toimub järgmise algoritmi järgi:

1) Loe lähteandmed kõrguskaardina.

2) Jaga maastik ruudukujulisteks tükkideks, mille küljepikkus oleks kahe aste (poolitades jäävad küljed paarisarvulise pikkusega). Maastiku jaotame ruutudeks, sest siis ei tule puud nii suured. Puude töötlemine on rekursiivne ja seega hoiame programmeerides madalamate puude puhul kokku hulga väljakutseid.

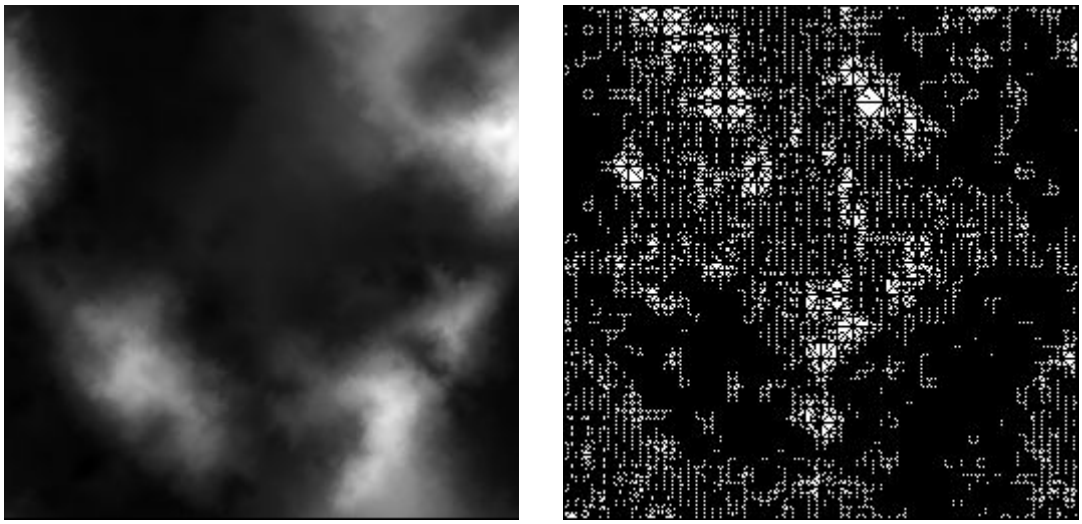
3) Iga ruudu sisse paiguta kaks kolmnurkade kahendpuud nii, et juurkolmnurkade hüpotenuus on ühine. Poolita mõlemad kahendpuud. Määra mõlemale kolmnurgale baasnaaber. Võimalusel määra ka vasak ja parem naaber (äärmistes ruutudes ei ole see alati võimalik).

4) Iga ruudu mõlema puu juure puhul poolita selle alamkolmnurk kui hüpotenuusi keskpunkti ja maastiku kõrguste vahe kõrguskaardi järgi on suurem kui eelnevalt määratud parameeter δ. Korda protsessi rekursiivselt tekkinud kolmnurkade peal. Kolmnurki poolitades tuleb jälgida kolmnurkade kahendpuu suhete säilimist. Ühtlase võre saamiseks tuleb jälgida ka baasnaabrit – kui poolitatav kolmnurk ei moodusta oma baasnaabriga rombi, tuleb baasnaaber poolitada. Ka see protsess peab olema rekursiivne. Näide sellisest „sundpoolitamisest“ on joonisel 3.3. Joonisel on näha, kuidas kolmnurga B poolitamine toob kaasa A, C, D ja E sundpoolitamise.



Joonis 3.3: Rekursiivne sundpoolitamine

Nii koostatud kolmnurkade puu moodustab maastiku kujuga kohaneva võre. Järskude tõusude ja languste modelleerimisel poolitatakse kolmnurki suurema täpsuse saavutamiseks. Tasase pinna puhul kasutatakse selle esitamiseks suuremaid kolmnurki, mida kulub suuruse arvelt vähem. Näide kõrguskaardist ja vastavast kahendpuust on joonisel 3.4.



kõrguskaart (heledam osa on kõrgem) 27708 kolmnurgast koosnev kahendpuu

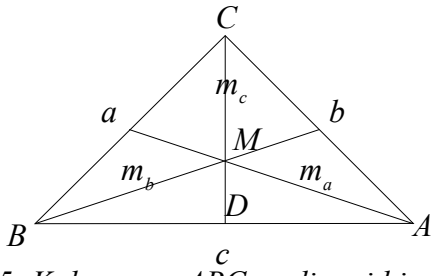
Joonis 3.4: Kõrguskaardi põhjal koostatud kolmnurkade kahendpuu

3.1.3 Graafi tippude märkimine

Graafi tipud märgime maastikule järgnevate reeglite alusel. Graafi tipp lisatakse:

1) Iga kahendpuu leheks oleva kolmnurga kõikidesse tippudesse
2) Iga kahendpuu leheks oleva kolmnurga raskuskeskmesse (mediaanide keskpunkti)

Mediaanide keskpunkt on antud juhul lihtsalt arvutatav, sest kolmnurk on võrdhaarne. Joonisel 3.5 on näidatud kolmnurk *ABC* ja selle raskuskese *M*.



Joonis 3.5: Kolmnurga ABC mediaanid ja raskuskese M

A, *B* ja *C* koordinaate teades saame arvutada punkti *D* (tänu võrdhaarsuse eeldusele lõigu *AB* keskpunkti) koordinaadid:

$$D=\left(\frac{x_A-x_B}{2};\frac{y_A-y_B}{2};\frac{z_A-z_B}{2}\right).$$

Raskuskese *M* asub lõigul *CD*, kusjuures ta jaotab lõigu kaheks - kaks kolmandikku jääb *C* poole ja üks kolmandik *D* poole. Seda teadmist kasutades leiame *M* koordinaadid:

$$M=D+\left(\frac{x_C-x_D}{3};\frac{y_C-y_D}{3};\frac{z_C-z_D}{3}\right).$$

3.1.4 Graafi koostamine

Graaf koostatakse otsinguruumis määratud tippude omavahelisel ühendamisel. Kolmnurkade kahendpuu abil koostatud mudelil ühendatakse graafi tipud kolmnurkade omavahelistest suhetest lähtuvalt.

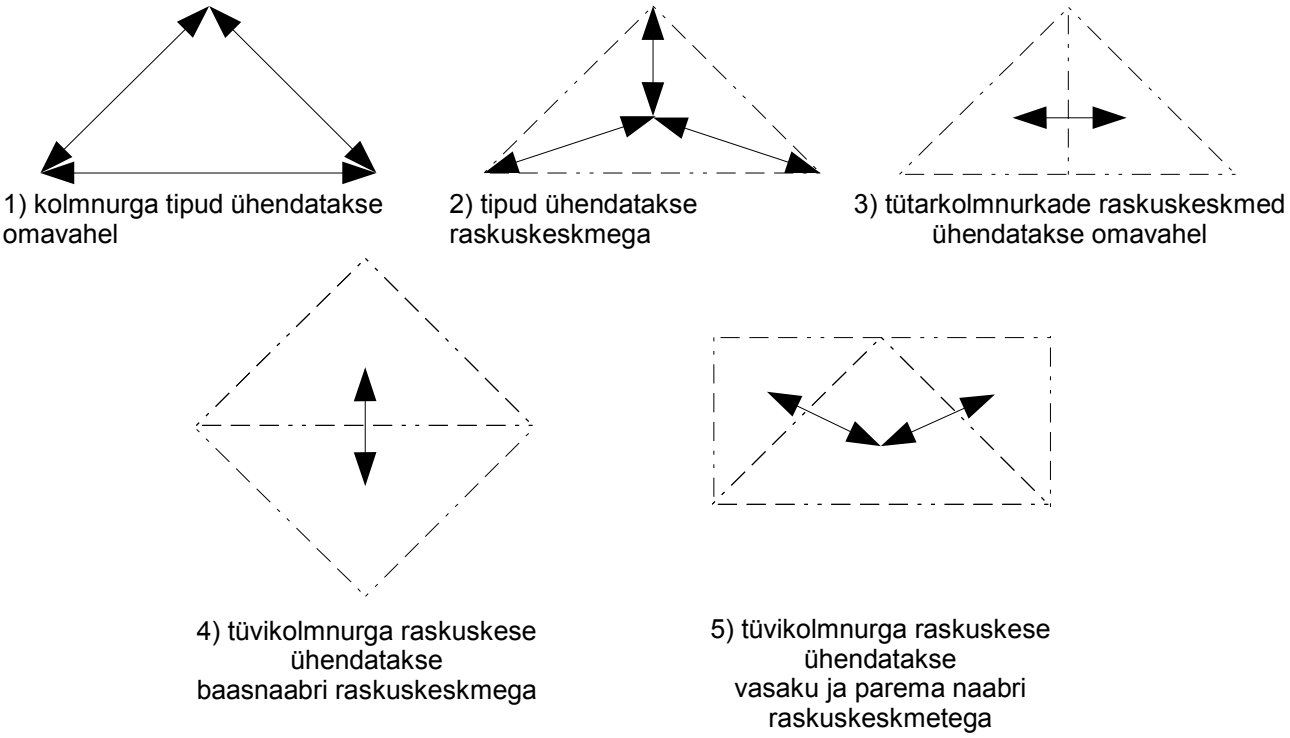
Servad lisatakse järgnevate reeglite alusel:

1) Kolmnurga tipud ühendatakse omavahel
2) Kolmnurga tipud ühendatakse raskuskeskmega
3) Kolmnurga tütarkolmnurkade raskuskeskmed ühendatakse omavahel

4) Tüvikolmnurga raskuskese ühendatakse baasnaabri raskuskeskmega

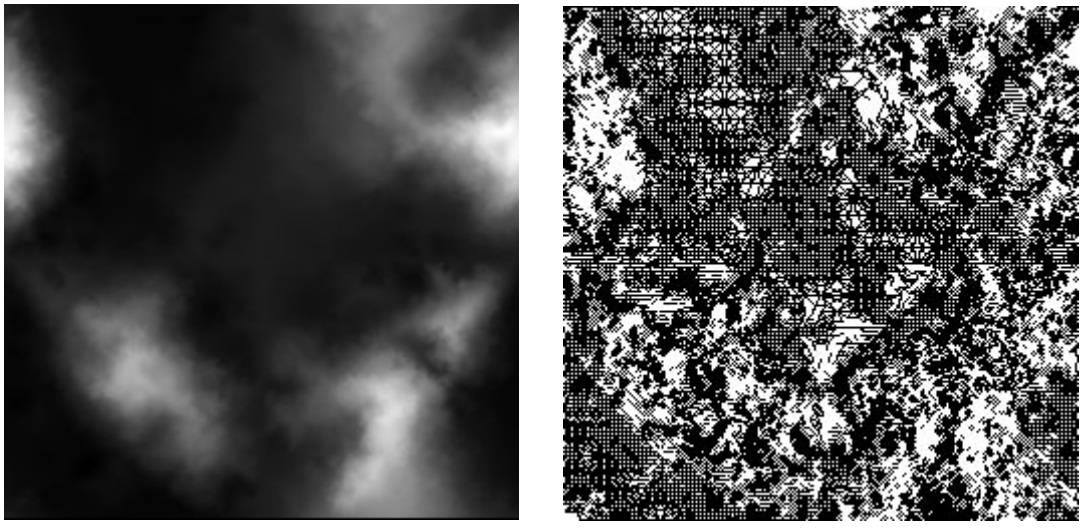
5) Tüvikolmnurga raskuskese ühendatakse vasaku ja parema naaberkolmnurga raskuskeskmetega

Reeglite graafiline esitus on joonisel 3.6.



Joonis 3.6: Graafi tippude ühendamise reeglid

Tulemusena saadaksegi suunatud graaf, millel on võimalik kasutada soovitud otsingumeetodit. Joonis 3.7 on graaf, mis on koostatud joonisel 3.4 kujutatud maastiku põhjal.



kõrguskaart (heledam osa on kõrgem) 371747 teest koosnev teedekaart

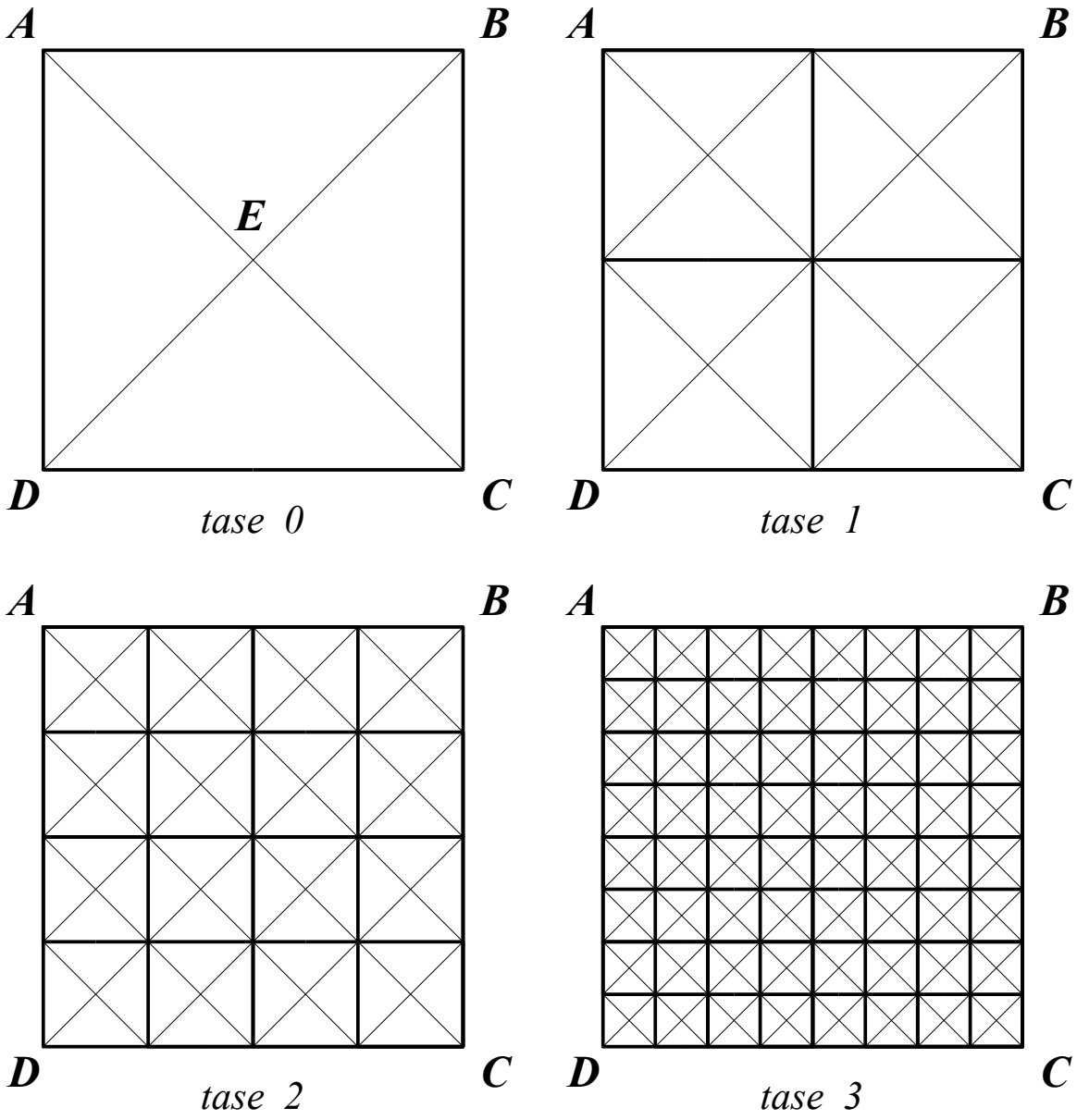
Joonis 3.7: Kõrguskaardi põhjal koostatud teedekaart

3.2 Ruutude neljandpuu

3.2.1 Neljandpuu struktuur

Ruutude neljandpuu andmestruktuuri on üldiselt kirjeldanud Samet [4].

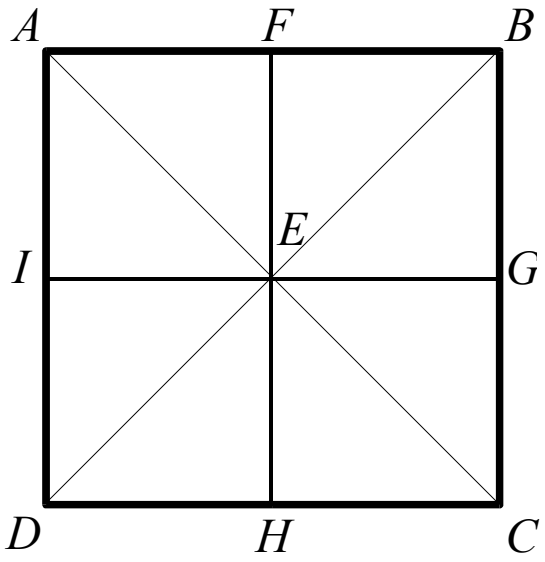
Neljandpuu tippudeks on ruudud, mis jagunevad kolmnurkadeks. Joonisel 3.8 on mudelis kasutatava neljandpuu neli esimest taset täielikul kujul (kõik lehed on maksimaalsel võimalikul sügavusel).



Joonis 3.8: Ruutude neljandpuu neli esimest taset

Esimesel tasemel on üks ruut *ABCD*, mille diagonaalid jagavad selle neljaks täisnurkseks võrdhaarseks kolmnurgaks. Igal järgmisel tasemel on esialgne ruut jagatud neljaks võrdseks alamruuduks, mis on samuti diagonaalide abil neljaks jagatud.

Mudeli loomise algoritmi lihtsaks kirjeldamiseks defineerime neljandpuus järgmised mõisted ja suhted. Suhteid illustreerib joonis 3.9.



Joonis 3.9: Suhted neljandpuus

Nelinurk *ABCD* on tüviruut. *ABE*, *BCE*, *CDE*, *DAE* on tüvikolmnurgad. *AFEI*, *FBGE*, *EGCH* ja *IEHD* on tütarruudud. *AC* ja *BD* on tüviruudu diagonaalid. *FE*, *GE*, *HE* ja *IE* on tüvikolmnurga poolitajad. Kaks sama tüviruudu tüvikolmnurga poolitajat, mis ei asu samal sirgel, eraldavad ühe tütarruudu. Tüviruut on kõigi oma tütarruutude suhtes ülemruut.

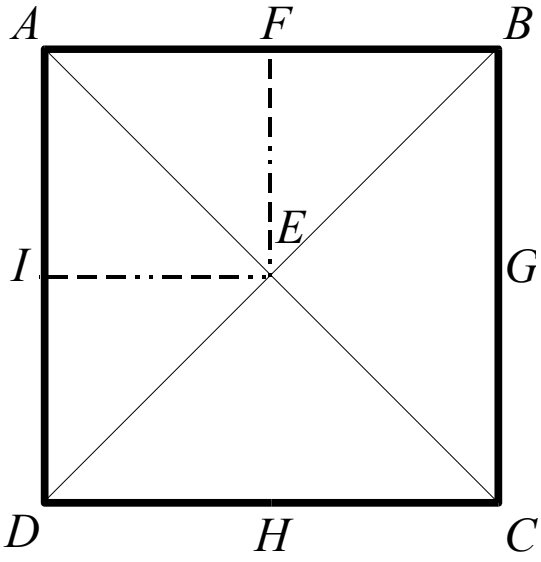
Sama taseme tüviruudud on naabrid, kui neil on ühine serv. Naabreid jaotatakse paiknevuse järgi vasakuks, paremaks, ülemiseks ja alumiseks. Joonisel 3.9 on ruut *AFEI* ruudu *IEHD* ülemine naaber, *IEHD* omakord on *AFEI* alumine naaber. *FBEG* on *AFEI* suhtes parempoolne naaber ja vastupidi, *AFEI* on *FBEG* jaoks vasak naaber.

Sama tüviruudu tüvikolmnurgad on naabrid, kui neil on ühine haar. Vasaku haara jagav naaber on vasak naaber ja paremat haara jagav naaber parem naaber. *DAE* vasakuks naabriks on *ABE* ja paremaks naabriks *CDE*.

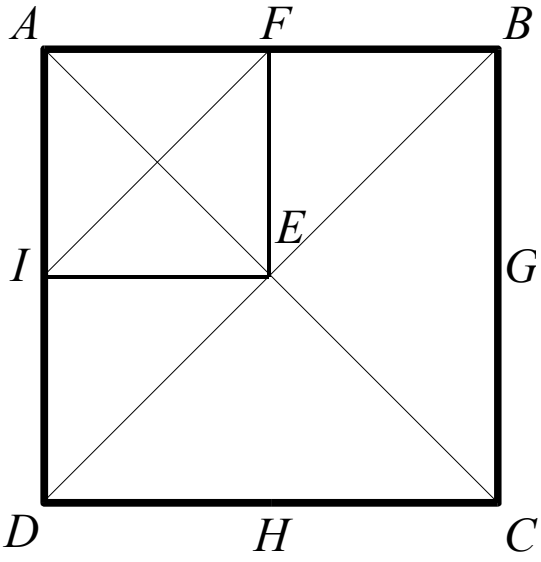
3.2.2 Mudeli koostamine

Ruutude neljandpuu rakendamist maastiku visualiseerimisel on uurinud Lindstrom et al [5]. Maastiku mudeli koostamisel kasutatakse Lindstromi artiklis kirjeldatud meetodeid.

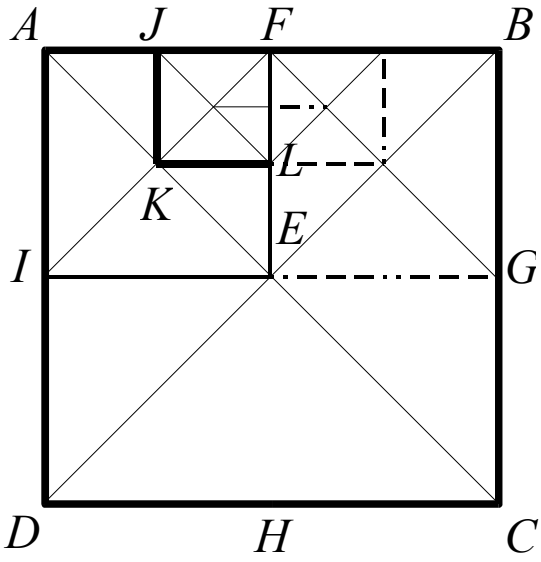
Loodava neljandpuu juureks on ruut, mis katab kogu kõrguskaardi. Ruudu tipud paigutatakse kaardi nurkadesse. Seejärel jaotatakse ruut diagonaalidega tüvikolmnurkadeks.



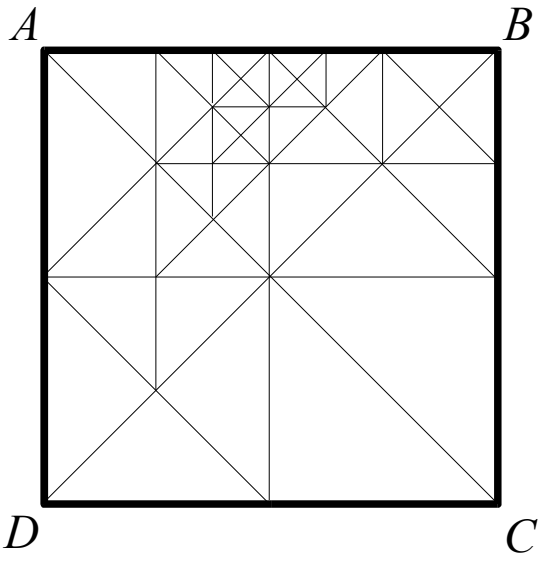
ABE ja *DAE* on poolitatud lõikudega *FE* ja *IE*, mis moodustavad tütarkolmnurga *AFEI.*



Tütarruudus *AFEI* moodustatakse diagonaalidega tütarkolmnurgad



JFLK loomisel tuleb rekursiivselt luua veel kaks alamruutu (katkendjoonega)



Näidis ruutude neljandpuust

Joonis 3.10: Mudeli loomine

Joonisel 3.10 kujutatakse mudeli koostamise etappe ruutude neljandpuu abil. Algoritm mudeli loomiseks on järgmine:

1) Leiame vaadeldava tüviruudu servade keskpunktid. Leiame kõrguskaardi abil nende vertikaalse vahe maastikuga. Kui see on suurem eelnevalt määratud parameetrist δ, poolitame selle serva ja tüvikolmnurga, mille aluseks see on. Leiame selle servaga külgneva naaberruudu ning poolitame

vastava tüvikolmnurga ka seal. Kui naaberruutu pole, siis loome selle ülemtippudes tütarruute, kuni saame luua puuduoleva ruudu.

2) Kui tüviruudu kõik küljed on vajadusel poolitatud, tehakse järgnev kontroll: kui vaadeldavas ruudus on poolitatud kaks tüvikolmnurka, mis on teineteise suhtes naabrid, luuakse tüviruudu tütarruut, mille külgedeks on tüvikolmnurkade poolitajad.

3) Uue tipu puusse lisamisel tuleb paika seada tema suhted teiste külgnevate ruutudega. Kui tal on olemas sama taseme naabreid, siis lisame vastavad seosed. Samuti määrame selle ruudu ülemruudu, milleks on vaadeldav tüviruut. Paneme tähele, et naabrite puudumisel neid looma ei hakata. Kui tüviruudu tüvikolmnurkade tükeldamisel on mõnes naaberruudus tekkinud poolitatud tütarkolmnurki, mis on naabrid, loome naaberruutu uue tütarruudu.

4) Iga uue tütarruudu loomisel tüvi- või naaberruudus peame sellel rakendama algoritmi esimest sammu.

3.2.3 Graafi tippude lisamine

Graafi tipp lisatakse:
1) Iga ruudu tippudesse ja diagonaalide lõikepunkti, vältides korduvaid tippe samas punktis.
2) Iga poolitatud tüvikolmnurga aluse keskpunkti, vältides korduvaid tippe samas punktis.

Korduvate tippude all mõistame geomeetriliselt samas punktis asuvaid tippe. Nende vältimiseks peab enne tipud lisamist kontrollima, kas antud punktis juba tippu pole. Sellel sammul on mudeli koostamisel paika pandud suhete korrektsus väga oluline.

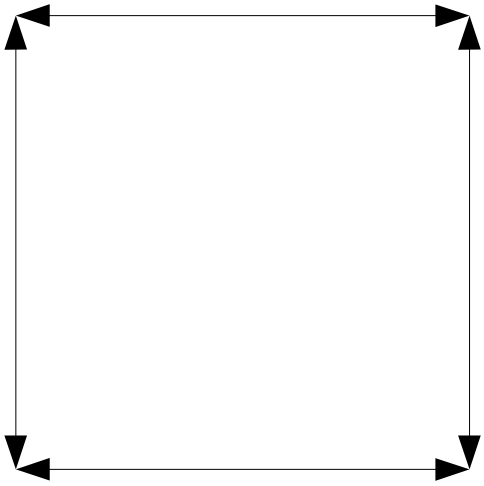
3.2.4 Graafi koostamine

Tipud ühendatakse graafiks järgnevalt:
1 Puu lehtedes olevates ruutudes ühendatakse ühel serval asuvad tipud omavahel.
2 Puu lehtedes olevates ruutudes ühendatakse ruudu tipud diagonaalide lõikepunktiga.
3 Kui tüvikolmnurk ei kuulu (ka osaliselt mitte) tütarruutudesse, siis ühendatakse poolitatud tüvikolmnurga aluse keskpunkt aluse vastastipuga.
4 Pärast lehtede töötlemist täiendatakse nende ülemruute vajalike teedega, et igas ruudus oleksid omavahel teede või teelõikudega ühendatud:

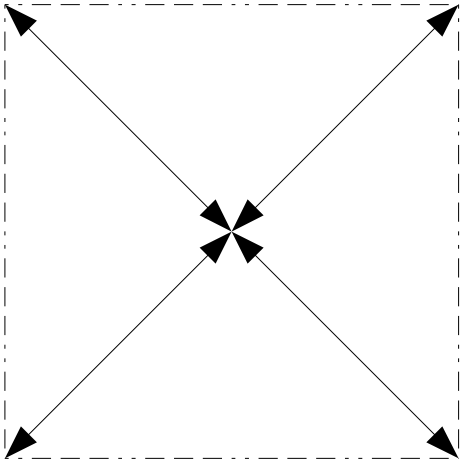
4.1 Ruudu tipud omavahel

4.2 Ruudu tipud ja diagonaalide lõikepunkt

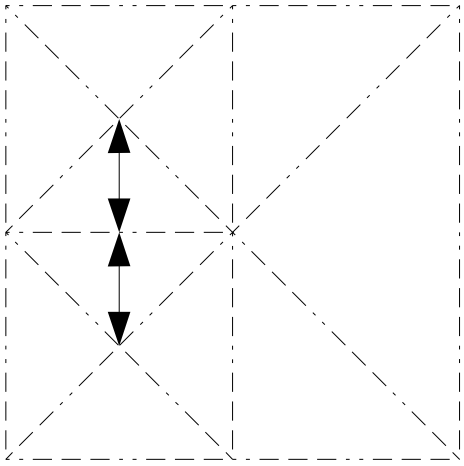
Graafi koostamist on graafiliselt kujutatud joonisel 3.11.



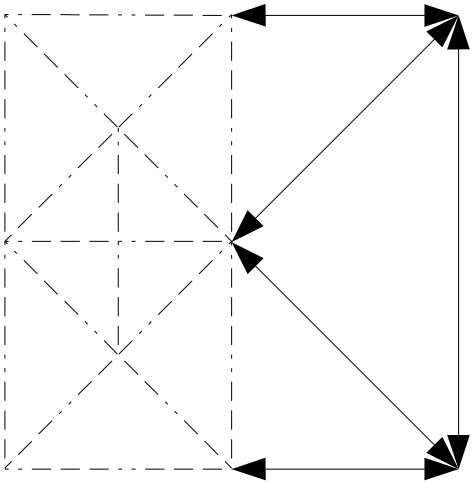
1) Samal serval asuvad ruudu tipud ühendatakse omavahel



2) Ruudu tipud ühendatakse diagonaalide lõikepunktiga



3) Poolitatud tüvikolmnurga aluse keskpunkt ühendatakse vastastipuga



4) Pärast alamruutude töötlemist täiendame nende ülemruudud puuduvate ühendustega

Joonis 3.11: Graafi koostamise etapid

3.3 Serva kaalu arvutamine maastiku mudelil põhineva graafi koostamisel

Otsitavale teele esitatud tingimuste järgimine toimub graafi servadele kaalu määramise teel. Kahe tipu ühendamisel graafis lisatakse nende vahele kaks serva – üks mõlema suuna jaoks. Serva kaal arvutatakse kõrguskaardi ning agendi andmete põhjal. Iga teelõigu kohta koostatakse graafik, millel igale lõigu punktile seatakse vastavusse selle punkti kõrgus kõrguskaardil. Lõigu punktide leidmiseks võib kasutada vabalt valitud täisarvudega töötavat joone algoritmi. Levinud Bresenhami

joonealgoritmi kirjeldab [6]. Kui maastikul on määratud ka erineva läbitavusega piirkonnad, peab graafikule lisama ka informatsiooni teelõigu või selle osade läbitavuse kohta.

Leitud graafiku põhjal arvutab agent enda võimalustest lähtudes serva kaalu. Inimese liikumist modelleeriv agent ei suuda ületada näiteks kolme meetri kõrgust takistust, sõiduk ei pääse läbi tihedast metsast või vajab liikumiseks kattega teed. Kõikide teelõigu punktide läbivaatus teeb kindlaks, kas agent on võimeline seda üldse läbima. Näiteks võivad lõigu otspunktid asuda ligikaudu samal kõrgusel, kuid nende vahel võib olla sein või kuristik.

4. Optimaalse tee leidmine

Lähteandmete töötlemisel saadud graaf on sobiv struktuur teeotsinguks. Otsingumeetodid võima jagada pimeotsinguks ja heuristiliseks otsinguks. Pimeotsing on probleemi lahendamine jõumeetodil, võimaluste proovimisel ei kasutata olemasolevaid teadmisi. Heuristilise otsingu puhul kasutatakse otsuste tegemisel ära teadmisi otsinguruumi kohta. Käesolevas ülesandes on mõistlik kasutada heuristilist otsingut.

Heuristiliseks otsingu peame eelnevalt defineerima hinnangufunktsiooni

$$f(x): V(G) \rightarrow \mathbb{R},$$

mis seab graafi *G* igale tipuga vastavusse reaalarvulise hinnangu selle perspektiivikusele otsingu suhtes. Väiksem väärtus tähistab tippe, mille läbimisel on sihini jõudmine tõenäolisem.

Käesoleva ülesande lahendamisel defineerime hinnangufunktsiooni järgnevalt. Olgu *siht* otsingu lõpptipp, *asukoht* vaadeldav graafi tipp ja *d(x)* funktsioon, mis seab graafi tipuga vastavusse tema kauguse lähtepunktist. Siis leiame tipu hinnangu järgnevalt:

$$f(asukoht) = d(asukoht) + (siht.x - asukoht.x)^2 + (siht.y - asukoht.y)^2 + (siht.z - asukoht.z)^2$$

Heuristilise komponendi valem sarnaneb punktide vahelist otsekauguse arvutamise valemiga, kuid puudub ruutjuur. Juure leidmine on arvutuslikult kulukas protsess ja hinnangul põhinevat tippude järjestust ei muuda. Seega ruutjuurt summast ei leita. Kaugus lähtepunktist on igasse punkti jõudes teada, seega seda eraldi arvutada vaja ei ole.

Enne otsingu algust määrab kasutaja teekonna algus- ja lõpp-punkti. Mõlema punkti jaoks leitakse graafist tipp, mis on sellele kõige lähemal. Nendest tippudest saavad otsingu algus- ja lõpptipp. Seejärel käivitatakse otsingualgoritm. Otsinguks kasutame heuristilist algoritmi A*. A* põhineb laiutiotsingul. Algoritm kasutab tööks avatud ja suletud tippude prioriteetidega järjekorda. Tipud on järjekorras sorteeritud heuristilise funktsiooni järgi. Avatud tippude järjekorras on töö alguses algustipp, suletud tippude järjekord on tühi. Tipu juures hoitakse tema koordinaate, otseste järglaste nimekirja, tagasiviita, kaugust lähtepunktist ning heuristilist komponenti.

Algoritm töötab järgmises tsüklis:

1 Kui avatud tippude järjekord on tühi, teed algus- ja lõpptipu vahel ei leidunud. Lõpetame töö.

2 Eemaldame avatud tippude järjekorrast esimese tipu ja paigutame ta suletud tippude järjekorda. Vaatleme seda tippu kui aktiivset tippu *T*.

3 Kui *T* on otsitav lõpptipp, siis oleme tee leidnud. Täpse teekonna taastame tippude juures olevaid tagasiviitu kasutades.

4 Tähistame *T* vahetute alluvate hulga ***J***. Iga ***J*** elemendi *P* puhul kontrollime, kas ta on juba avatud või suletud tippude järjekorras. Kui *P* ei olnud enne ei avatud ega suletud tippude järjekorras, siis viime läbi järgmised toimingud:
   4.1 Lisame *P* avatud tippude järjekorda.
   4.2 Määrame *P* tagasiviida tipule *T*.
   4.3 Arvutame *P* kauguse lähtepunktist *T* vastava kauguse ning *P* ja *T* vahelise serva kaalu kaudu.
   4.4 Arvutame *P* jaoks heuristilise komponendi väärtuse, kasutades ülalpool antud valemit ning jättes ära summa esimese liidetava (kauguse alguspunktist punkti *P*)

5 Kui *P* oli ühes nendest järjekordadest, siis tuleb kontrollida, kas tema tagasiviit vajab muutmist. See sõltub sellest, kas eelmine teadaolev tee sellesse tippu oli lühem või pikem kui uus. Kui uus on lühem, siis peame muutma *P* tagasiviida tipule *T* ning seejärel rekursiivselt kõik *P*-le tagasi viitavate tippude kaugused ja nii edasi.

6 Sorteerime avatud tippude järjekorra tippude hinnangufunktsiooni järgi kasvavalt. Iga tipu juures arvutatakse selle hinnangufunktsiooni väärtus liites kauguse algtipust ja heuristilise komponendi. Neid hoitakse tippudes eraldi, sest kaugus algtipust võib otsingu käigus muutuda, kui leitakse lühem tee algusesse.

7 Tööd jätkatakse tsükli algusest.

Kui algoritm lõpetab töö punktis 1, siis teed ei leidunud. Kui töö lõpeb punktis 3, siis konstrueerime läbitud tippude jada lõpptipust algtipu suunas tagasiviitade järgi. Saadud jada moodustabki otsitud teekonna. Kui otsinguülesandes on ära märgitud mitu punkti, mida teekond peab läbima, leiame nende vahelised teelõigud eraldi ning ühendame need töö lõpus.

Kokkuvõte

Iga teeotsinguülesanne vajab lahendamiseks mingit otsingustruktuuri. Selle struktuuri täpsusest sõltub otsingutulemuse täpsus ja rakendatavus reaalses maailmas. Lisaks struktuuri täpsusele on oluline ka selle lihtsus ja kasutuskõlblikkus. Keerulise mudeliga töötamine võib olla liiga kulukas ja aeganõudev.

Töös püstitatakse teeotsinguülesanne maastikul ja objektide ruumis mobiilse agendi jaoks ning esitatakse üldine lahendusplaan. Ülesande lahendus jaguneb lähteandmete töötlemiseks ja teeotsinguks. Lähteandmete analüüsil luuakse maastiku või ruumi mudel, mille põhjal koostatakse graaf. Saadud graafil võib kasutada sobivat otsingualgoritmi. Graafi koostamisel võetakse arvesse mobiilse agendi võimalusi ning lisatingimusi läbitava ruumi kohta.

Kirjeldatakse kahte geomeetriliste objektide puudel põhinevat mudelit. Esimene on kolmnurkade kahendpuul põhinev maastiku mudel. Tutvustatakse vastavat andmestruktuuri ning kirjeldatakse selle kasutamist maastiku modelleerimisel. Antud on reeglid saadud mudeli põhjal graafi koostamiseks. Teine mudel põhineb ruutude neljandpuul. Kirjeldatud on andmestruktuuri, maastiku mudeli loomist, graafi tippude märkimist ja nende ühendamist graafiks.

Maastikumudelite jaoks esitatakse lihtne kõrguskaardil põhinev süsteem servade kaalu arvutamiseks, mis on laiendatav erinevatele agentidele. Kirjeldatakse koostatud graafil otsingu läbiviimist algoritmi A* abil. Antakse sobiv heuristiline hinnangufunktsioon ning kirjeldatakse selle kasutamist algoritmis.

Kahendpuu mudeli kasutamist demonstreerib tööle lisatud programm 'Reisija'. Reisija võimaldab graafiliselt jälgida kõrguskaardi lugemist, mudeli koostamist, graafi tippude ja servade lisamist ning viia läbi teeotsingut.

Edasise töö käigus peab uurima ruutude kaheksandpuu kasutusvõimalusi ruumi modelleerimisel ja otsingugraafi koostamisel. Kindlasti tuleb leida veel uusi analüütilisi meetodeid mudelite koostamiseks ning võimalusi olemasolevate parandamiseks. Head otsingualgoritmid, otsinguruumi ja mobiilsete agentide mudelid on reaalsete rakenduste väljatöötamise eeldusteks.

Solving the optimal path problem by using trees of geometrical objects

Term paper

Dan Bogdanov

Abstract

Every search-related problem requires some sort of a structure to perform the search on. This structure along with all the related algorithms determines the quality and usability of the result.

This work presents a generalized path finding problem for a mobile agent on a landscape and in three-dimensional object space. A template for solutions is also provided. Every search problem consists of two steps: processing the input data into a graph and then performing a search on that graph.

Two models are presented, both of which make use of geometrical object trees. The first is a binary triangle tree based method of landscape modelling and graph construction. The second uses quadtrees for the same purpose. Both models are described along with the algorithms for triangulation, graph node selection and interconnection.

A simple method for calculating edge costs in a graph is given. This method is meant for use with a mobile agent moving on a landscape.

The use of A* as a search algorithm is described. The necessary heuristic function is provided with usage notes.

An example application shows the use of the binary triangle tree model. The 'Traveller' program imports heightmaps, constructs the tree and the graph. The A* search algorithm is implemented for immediate testing with the sample data.

Future work should consider octrees' usability for object space modelling. Further research must be done towards creating more analytical models for the search space. In order to create real-world applications, we have to create flexible models for different kinds of mobile agents and also find ways to optimize the search.

Kasutatud kirjandus

[1] D. Bogdanov, D., M. Kapp. Optimaalse tee leidmine kahe punkti vahel etteantud kolmemõõtmelisel maastikul, 2001,
http://dan.bmonde.net/aasta/2001/Optimaalneteemaastikul.html (viimati vaadatud 25. aprill 2004)

[2] M. Duchaineau, M. Wolinski, D. E. Sigeti, M. C.Miller, C. Aldrich, M. B. Mineew-Weinstein. ROAMing Terrain: Real-Time Optimally Adapting Meshes, 1997,
http://www.llnl.gov/graphics/ROAM (viimati vaadatud 25.aprill 2004)

[3] B. Turner. Real-Time Dynamic Level of Detail Terrain Rendering with ROAM, 2000,
http://www.gamasutra.com/features/20000403/turner_01.htm (viimati vaadatud 25. aprill 2004)

[4] H. Samet. The Quadtree and Related Hierarchical Data Structures, ACMComputing Surveys 16 (2), juuni 1984

[5] P. Lindstrom, D. Koller, W. Ribarsky, L. F. Hodges, N. Faust, G. A. Turner. Real-Time, Continuous Level of Detail Rendering of Height Fields, 1996,
http://www.cc.gatech.edu/gvu/people/peter.lindstrom/papers/siggraph96/ (viimati vaadatud 25.aprill 2004)

[6] J. E. Bresenham. IBM Systems Journal 4, lk. 25-30, 1965

Lisa 1: Arvutiprogramm 'Reisija'

Kirjeldus

Programm 'Reisija' realiseerib peatükis 3 kirjeldatud kolmnurkade kahendpuul põhineva otsingusüsteemi. Kõrguskaart loetakse failist ning selle põhjal luuakse maastiku mudel. Puu sügavuse piiramiseks jaotatakse maastik eelnevalt lappideks, milles igaühes on kaks kahendpuud. Tippude määramine ja graafi koostamine toimuvad samuti lappide kaupa. Kui töötlemine on lõppenud, saab kasutaja määrata teekonna algus- ja lõpp-punkti ning programm leiab nende punktide vahelise lühima tee.

Ülevaade programmis kasutatud ja töös kirjeldatud tehnikatest:

1. Lähteandmete laadimine kõrguskaardina
2. Kolmnurkade kahendpuu loomine kõrguskaardi põhjal (jaotis 3.1.2)
3. Graafi tippude määramine (jaotis 3.1.3)
4. Graafi koostamine (jaotised 3.1.4 ja 3.3)
5. Teeotsing (peatükk 4)

Mobiilse agendi mudel on lihtne jalakäija mudel, mis põhineb jaotisel 3.3 ja järgnevate reeglitel:

- jalakäija ei suuda ületada ühe meetri kõrgust takistust (kui graafikul kahe järjestikuse punkti kõrguste vahe on suurem kui üks, seda serva graafi ei lisata)
- ühe meetri läbimine lisab serva kaalule ühe ühiku

Otsing kasutab peatükis 4 kirjeldatud algoritmi A*, mis sisaldab pisemad optimisatsioone kiiruse lisamiseks. Kõige olulisem ajakulu teeotsingul on tagasiviitade korrigeerimine, mis on eksponentsiaalse keerukusega. Lisatud optimisatsioonid püüavadki tagasikorrigeerimise vajadust vältida, kärpides otsingu harusid, kui need on pikemad teadaolevast lühimast teest sihtpunktini.

Realisatsioon

Programm töötab Windowsi platvormil, on kirjutatud keeles C++ ja kompileeritud Microsoft Visual C++ versiooniga 6.0. Kasutatakse graafikateeki Allegro, fontide laadimise ja renderdamise teeki Allegro Font ning kasutajaliidese loomise teeki MASkinG. Programmi lähtekood kompileerub vajalike teekide olemasolul ka teiste kompilaatoritega ning teistel platvormidel.

Windowsi versiooni nõuded süsteemile:

- Operatsioonisüsteem Windows 98/Me/2000/XP
- DirectX versioon 8.1 või uuem
- Vähemalt 500 MHz Intel Pentium või ühilduv protsessor
- Vähemalt 64 MB töömälu
- Videokaart ja monitor mis võimaldavad 800x600 pikslist ekraani.

Keeruliste mudelite puhul jääb sellistest nõuetest väheks. Sadu tuhandeid servi sisaldava graafi koostamiseks kulub ka 2 GHz arvutil minuteid. Soovitatav arvuti muutmälu maht on 256 MB.

Kasutusjuhend

Sisendvorming

Programm loeb kõrguskaarte RAW-vormingus failist. Failis on ridade kaupa järjest ühe- või kahebaidised täisarvud, mis moodustavad kõrguskaardi. Programm toetab kahes suuruses kaarte: 257x257 ja 513x513 ning ühe- ja kahebaidiseid väärtuseid. Faili lugemisel kontrollib programm faili suuruse järgi, kas etteantud andmed on toetatud vormingus.

Graafiline vahend kõrguskaartide koostamiseks on programm nimega Terragen, mis on vabavarana all laetav aadressilt http://www.planetside.co.uk/terragen/. Järgneb lühikirjeldus Terrageni kasutamisest.

1) Laadige alla, installeerige ja käivitage Terragen.
2) Alustage uue projektiga (*'World File' →'New World'*). Valige maastikuvaade (*'View' → 'Landscape'*).
3) Määrake maastiku suurus, vajutades nupule '*Size'* ja seejärel valides 257x257 või 513x513.
4) Genereerige või joonistage maastik (vastavalt *'Generate Terrain'* või *'View / Sculpt...'*).
5) Genereeritud maastiku puhul määrake maastikule kõrguspiir[1]. Selleks avage valik '*Modify...*' ning '*Set Height Range*' järel lahtritesse sisestage 0 ja 255 ning vajutage vastavat nuppu. Sulgege valik.
6) Eksportige maastik faili, vajutades nupule *'Export...'*. Kui maastik on genereeritud, valige meetodiks *'Raw 8 bits'* ja *'Select File and Save...'*. Määrake failile asukoht ja nimi. Kui soovite eksportida 16-bitist kõrguskaarti, kasutage selleks valikut *'Raw 16 bits Intel Byte-Order'*.

1 Agent ei ole võimeline ületama üle ühe meetri kõrgust takistust. Terragen genereerib eksportides maastiku kõrguseid lähtuvalt väärtuspiirist (8 või 16 bitti). 16 biti puhul on kõrgusemuutused liiga järsud, et ühtlase, genereeritud maastiku puhul agent sellel liikuda saaks.

Programmi kasutamine

1. Käivitage programm 'Reisija'. Kõrguskaardi laadimiseks vajutage nupule 'Loe kaart'. Avanevas aknas leidke soovitav fail ning vajutage 'OK'.
2. Järgnevas aknas on võimalik kontrollida valida mudeli täpsust. Kolmnurga poolitamiseks nõutav vahe on täisarv piirides 0 kuni kõrguskaardi suurim väärtus (2^8 - 1või 2^{16} - 1). Keerulise maastiku puhul (näiteks genereeritud maastik) annavad väikesed väärtused küll väga täpse tulemuse, kuid suurendavad töötlemisaega oluliselt. 8-bitise kõrguskaardi puhul on 5-10 sobiv väärtus.
   Samamoodi määrake lapi suurus (vastava ruudu küljepikkus). 32 on toetatud suuruste puhul sobiv väärtus. Eriti täpse mudeli jaoks võib valida 16. Optimaalse vahe ja küljepikkuse valik sõltuvad maastiku iseloomust. Pärast parameetrite seadmist vajutage nupule 'Koosta mudel'. Lähteandmete töötlemise ajal on ekraani alumises ääres võimalik jälgida töö kulgu.
3. Kui kõrguskaart on edukalt laetud, saab ekraani paremas ääres olevate nuppudega kuvada erinevaid mudeli koostamise etappe. Nupud töötavad järgnevalt:
   a) 'TÜHJENDA' – tühjendab ekraani (värvib mustaks).
   b) 'KÕRGUSKAART' – tühjendab ekraani ning joonistab sellele kõrguskaardi. Ka pärast faili laadimist kuvatakse kõrguskaart. Heledamad punktid kujutavad kõrgemaid kohti maastikul.
   c) 'MUDEL' – joonistab ekraanil olevale kujutisele peale rohelist värvi kolmnurkade kahendpuu võre.
   d) 'GRAAFI TIPUD' – joonistab ekraanil olevale kujutisele peale graafi tipud (valged täpid).
   e) 'GRAAF' – joonistab ekraanil olevale kujutisele koostatud graafi servad kollase värviga.
   f) 'OTSI' – pärast sellele nupule vajutamist märkige vasaku hiireklahvi vajutamisega teekonna alguspunkt ning paremaga lõpp-punkt. Seejärel alustab programm otsingut, mida kajastatakse läbitavate servade kujutamisega ekraanil (halli värviga) ning hetketegevuse ning lühima tee pikkuse esitamisega ekraani all ääres. Otsingu lõpul kuvatakse ekraanil valge joonega leitud tee ning all ääres tee pikkus.
   g) 'KIIROTSING' – töötab samamoodi nagu 'OTSING'. Erinevus on selles, et tee leitakse ainult heuristilisele funktsioonile tuginedes. Korrigeerimist ei teha. Otsing on palju kiirem, kuid see-eest ebatäpsem
   h) 'VÄLJU' – väljub programmist.

NB! Kui soovite programmi tööd suvalisel ajal peatada, siis seda võimaldab teha klahvikombinatsioon Control + Alt + End. Seda võimalust saab kasutada näiteks siis, kui andmete töötlemine või otsing kestab liiga kaua ning töö oleks vaja peatada.

Lähtekood

Programm koos lähtekoodiga on tööle lisatud CD-plaadil. Lisatud on programmiga ClassBuilder (http://home.hetnet.nl/~xvenemaj/ClassBuilder.htm) koostatud klassimudel, mida on kasutatud programmikoodi genereerimisel. Alamkaustas '`maastikud`' on mõned näidisfailid, mida saab programmiga kohe kasutada.

Programmi andmemudel on toodud klassidiagrammil (järgmisel leheküljel). Järgneb klasside otstarvete selgitus.

HeightMap – kõrguskaardi klass. Sisaldab faili laadimiseks vajalikke funktsioone ning kõrguskaardi andmeid (pikkust, laiust, sügavust bittides, kaart ise). Kõrguskaarti kasutatakse kahendpuu koostamisel ning graafi tippude ja servade lisamisel.

PatchMap – lappe koondav klass. Maastiku mudel koosneb ruudukujulistest lappidest, mis on klassi *Patch* isendid. Kõik mudeliga töötavad meetodid kutsutakse välja sellel klassil. *PatchMap* kutsub vastavad meetodid välja kõigil lappidel ning koordineerib nendevahelist tegevust.

Patch – maastikulappi esindav klass. Üks maastikulapp sisaldab kahte kahendpuu juurt (*TriTreeNode*). Lapil rakendatavad meetodid käivitatakse mõlema puu peal eraldi. See klass tegeleb ka kahe puu omavahelise ühendamisega.

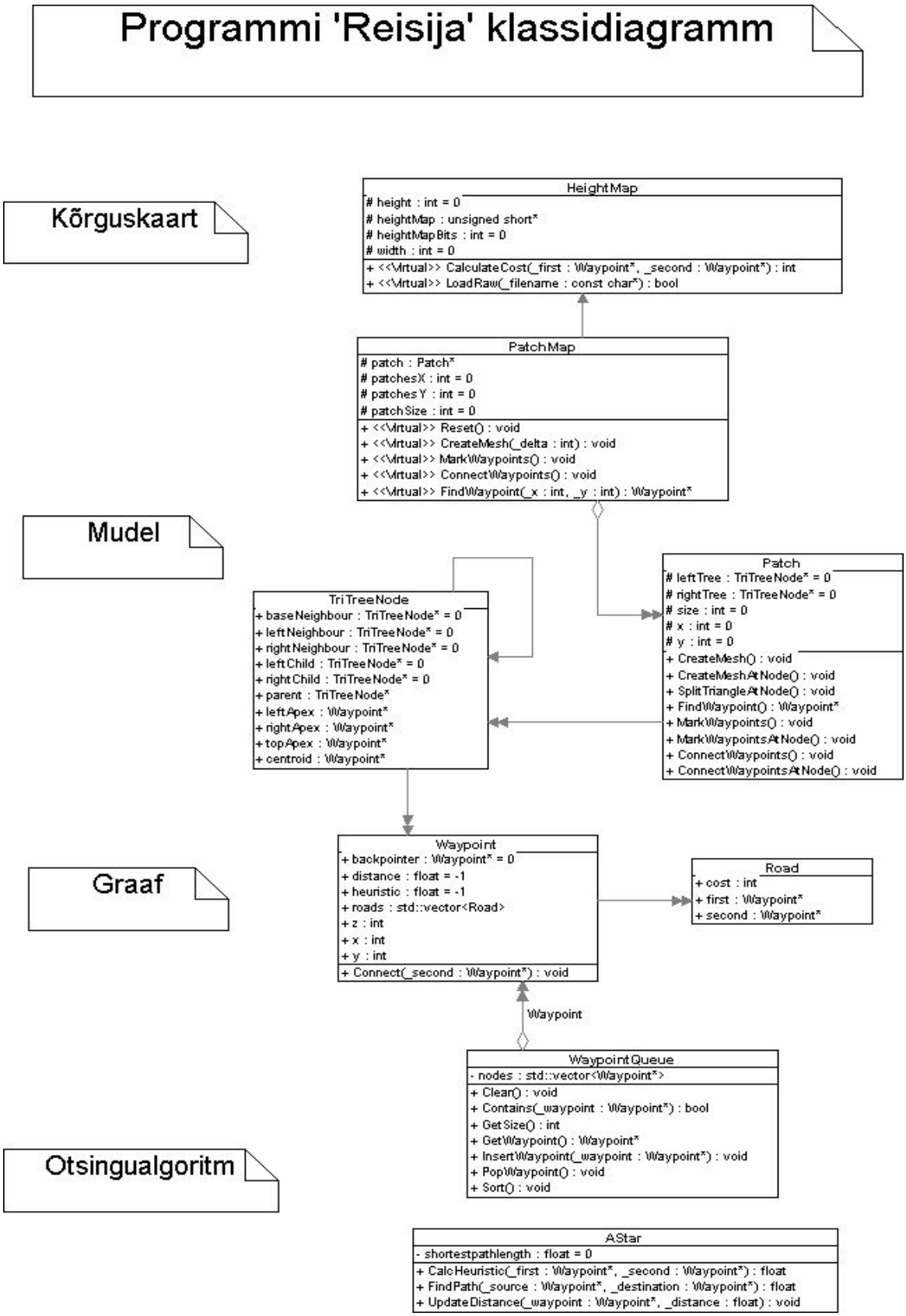
TriTreeNode – kolmnurkade kahendpuu tipp. Sisaldab viiteid ülemtipule, tütarkolmnurkadele, naaberkolmnurkadele ja kõikidele graafi tippudele, mis selle kolmnurgaga seotud on.

Waypoint – graafi tipp. Graafi tipud sisaldavad informatsiooni väljuvate servade (*Road*) kohta ning tipu koordinaate. Otsingu jaoks hoitakse tipu juures ka tema kaugust lähtetipust ning heuristilist hinnangut.

Road – graafi serv. Graafi serval on määratud kaks viita otspunktidele (*Waypoint*) ning kaal.

WaypointQueue – graafi tippude sorteeritav järjekord. Kasutatakse graafiotsingul avatud ja suletud tippude hoidmiseks. Lähtuvalt otsingualgoritmi nõuetest on järjekord sorteeritav tipu heuristilise funktsiooni järgi.

AStar – otsingualgoritmi A* realisatsioon. Leiab kahe tipu (*Waypoint*) vahelise lühima tee.



Programmi ' Reisija' klassidiagramm

Programmi 'Reisija' kompileerimiseks vajalikud teegid on Internetis kättesaadavad järgnevatel aadressidel:

Allegro – http://alleg.sourceforge.net/

Allegro Font – http://www.allegro.cc/resource/resource-view.php?id=29

MASkinG – http://ferisrv5.uni-mb.si/~ma0747/default.asp?page=masking