TARTU ÜLIKOOL
LOODUS- JA TEHNOLOOGIATEADUSKOND
ÖKOLOOGIA JA MAATEADUSTE INSTITUUT
GEOGRAAFIA OSAKOND

MAGISTRITÖÖ
GEOINFORMAATIKAS JA KARTOGRAAFIAS

DELAUNAY TRIANGULATSIOONI RAKENDAMINE DÜNAAMILISTE ATRIBUUTIDE PÕHJAL SAMAJOONTE INTERPOLEERIMISEL JA TULEMUSTE ANIMEERIMINE INTERNETISIRVIJAS

KARL TÕNISSOO

Juhendaja: PhD Raivo Aunap

MSc Margus Tiru

Kaitsmisele lubatud:

Juhendaja:

Instituudi juhataja:

Tartu 2013

SISUKORD

Sissejuhatus 3

1. Samajoonte interpolatsioon 5

1.1. Samajoonte interpolatsioon punktandmetest 5

1.1.2. Interpolatsiooni teoreetilised eeldused 5

1.1.2. Interpolatsiooni tugikonstruktsioonid 5

1.1.3. Samajoonte leidmine tugikonstruktsioonist 6

1.2. Delaunay triangulatsioon 6

1.2.1. Delaunay triangulatsiooni algoritmid 7

1.2.2. Mõned tähtsamad alamülesannete lahendamise algoritmid, mida paljud eeltoodud lahendused kasutavad 13

1.3. Samajoonte silumine 18

1.3.1. Bezier' kõverad 18

2. Veebisirvijates kuvatavate interaktiivsete kaardianimatsioonide loomine 21

2.1. Kaardianimatsioon 21

2.2. Tehnoloogia 22

2.2.1. Arvutuste tegemise koht 22

2.2.2. Andmete hoidmine 23

2.2.3. Erinevad animatsiooniloomise tehnoloogiad kaasaegses internetisirvijas 23

2.3. Ülevaade sarnastest töödest 27

2.3.1. Sarnased uuemad samajooni genereerivad tööd 27

3. Kunstlikult loodud pidevad andmed 29

3.1. Lineaarsed värvigradiendid 29

3.2. Perlin müra 29

3.2.1. Klassikaline müra 29

3.2.2. Simpleks müra 30
4. Internetisirvijas töötava samajoonte generaatori ehitamine 31
4.1. Süsteemi arhitektuuri valikud 31
4.1.1. Süsteemi eesmärgid ja vajadused 31
4.1.2. Kliendipoolsed vs. serveri poolsed arvutused 32
4.1.3. Arendusvahendite valik 32
4.1.4. Interpolatsiooni aluse valik 33
4.2. Süsteemi algoritmide valikud ja ehitamine 34
4.2.1. Delaunay triangulatsiooni algoritmi valik 34
4.2.2. Andmestruktuuri valik ehk topoloogia ehitamine ja CCW testide vähendamine 35
4.2.3. Samajoonte ühendamine 37
4.2.4. Katsetamine gradientidega ja EMHI reaalajaliste andmetega 38
5. Tulemused ja arutelu 41
Kokkuvõte 49
Summary 51
Tänuavaldused 52
Kirjandus 52

SISSEJUHATUS

Seoses arvutite ja interneti leiutamise ja nende inimeste seas laialdasema levikuga, on kartograafidel tekkinud uudsed võimalused inimestega suhtlemiseks. Enam ei pea kasutama staatilisi paberkaarte, vaid saab luua interaktiivseid animeeritud lahendusi (Harrower 2004). Animeeritud kaardid annavad võimaluse vaadata ajalis-ruumilisi andmeid uudse nurga alt. Välja võivad joonistuda mustrid, mida staatiliste kaartidega ei ole lihtne märgata (Harrower, Fabrikant 2008, Lobben 2003, Cauvin et al. 2010c).

Esimesed samajoonte kaardid tehti juba 16ndal sajandil Alonzo de Santa Cruz-i poolt (Cauvin et al. 2010a). Pidevate pindade kujutamise vahendina on need tänapäeval laialdaselt kasutuses. Samajoonte kaartide interpoleerimiseks punktandmetest on välja töötatud väga palju erinevaid lahendusi, kuid millegipärast, kui vaadata internetis erinevaid ilmakaarte, peame endiselt vaatama neid samu punktandmeid. Kui kuskil leidub mõni samajoontega kaart, siis on see tõenäoliselt ette arvutatud joontega. See tähendab interaktiivsuse langemist. Täieliku interaktiivsuse saab tagada vaid reaalajaliste arvutustega (Cauvin et al. 2010c:33).

Käesoleva töö eesmärk on välja selgitada, kas ja kui, siis milliseid meetodeid ja algoritme kasutades oleks võimalik samajoonte animatsioonikiirusel interpoleerimine punktandmetest ja seda kõike otse moodsas internetisirvijas. Üritatakse välja selgitada, millised kaasaegsed veebitehnoloogiad oleksid antud probleemi lahendamiseks kõige sobilikumad. Et veenduda valikute sobilikkuses, realiseeritakse tehtud valikud ka praktikas.

Esmalt proovitakse interpoleerida tavalisi samajoonte pilte ning hinnata nende loomiseks kulunud aega. Kui esimeses etapis saavutatakse rahuldav tulemus, proovitakse samajooni ka animeerida.

Õnnestumisel võiks töö praktilise osa tulemustest olla kasu:

- erinevate teemakohaste, andmetest eraldi seisvate, interaktiivsete veebirakenduste arendamisel;
- *on the-fly* modelleerimisel;

- õppevahendina kasutamisel;
- erinevate pidevate andmete aeg-ridadega tutvumisel ja nende kujutamisel;
- samajoonte animatsioonide loomisel.

1. SAMAJOONTE INTERPOLATSIOON

Samajooned ehk isojooned on pideval pinnal olevad jooned, mis ühendavad vaadeldava nähtuse samade väärtustega punkte. Esimesed samajoonte kaardid tehti ilmselt juba 16. sajandil. Päris kindel võib olla samajoonte kasutamises Halley 1701. aasta töös, kus ta kujutas magnetilise deklinatsiooni samajooni. 1792 kasutasid Du Carla ja Dupain-Triel samajooni maastiku kujutamiseks. Laialdasem kasutus algas sellegipoolest alles 19ndal sajanil. 1817 tegi Humboldt isotermide ehk samatemperatuuriga punkte ühendava kaardi (Cauvin et al. 2010a:347).

1.1. SAMAJOONTE INTERPOLATSIOON PUNKTANDMETEST

1.1.2. INTERPOLATSIOONI TEOREETILISED EELDUSED

Esiteks peab ruumilise interpolatsiooni tegemiseks eeldama alljärgnevate Tobleri hüpoteeside paikapidavust:

1) „Ruum ja ka kujutatav arvuliselt esitatav fenomen on pidevad."

2) „Fenomeni väärtuseid ruumi erinevates punktides saab leida teiste teadaolevate väärtustega punktide alusel."

Interpolatsiooni tegemisel tuleb kasuks Tobleri seaduse järgimine: „Kõik on kõigega seotud, kuid rohkem lähestikku paiknevad asjad on rohkem seotud." (Cauvin et al. 2010a:348)

1.1.2. INTERPOLATSIOONI TUGIKONSTRUKTSIOONID

Interpolatsiooni teostamiseks arvutis kasutatakse põhiliselt kahte erinevat tüüpi tugikonstruktsiooni: triangulatsiooni ehk kolmnurkadest koosnevat võre (*grid*) ja ristkülikutepõhist võre (Lawson 1977).

Ristkülikutepõhise tugiskonstruktsiooni puhul tuleb kõigepealt valida sobiva tiheduse ja suurusega võre, edasi tuleb igale võre sõlmele omistada väärtus. Väärtuse omistamiseks tuleb leida andmestikuga sobiv interpolatsiooni meetod. Valik on väga lai, näiteks kaalutud keskmine, lähim naaber, Kriging, lineaarne interpolatsioon. Samuti peab vastu võtma rida otsuseid: kui suur ala igat võrepunkti mõjutab, mitu punkti

interpolatsiooni kaasata, kuidas tulla toime varieeruva lähteandmestiku tihedusega (Cauvin et al. 2010a:352, Robinson 1978:230).

Triangulatsioonipõhine lähenemine pärineb aegadest, kui samajoonte kaarte tehti käsitsi. Cauvin kirjutab, et triangulatsiooni põhist lähenemist kasutatakse tänapäeval vaid pedagoogilistel eesmärkidel (Cauvin et al. 2010a:352), sellega ei saa paraku nõustuda, sest leidub väga palju praktilise väärtusega töid, kus on valitud just see lähenemine. Tavaliselt kasutatakse Delaunay triangulatsiooni (pikemalt eraldi peatükis 1.2). Triangulatsioonile tuginedes kaob ära vajadus leida võre sõlmedesse väärtused, sest triangulatsioon ise ühendabki orginaal andmepunkte.

Paul Bourke kasutab lähenemist, kus kõigepealt tehakse Delaunay triangulatsioon ja see kantakse üle ruudustikupõhisele võrele. (URL 2)

1.1.3. SAMAJOONTE LEIDMINE TUGIKONSTRUKTSIOONIST

Triangulatsiooni korral toimub samajoonte interpolatsioon piki triangulatsioonis olevate kolmnurkade servasid ehk siis iga serva otspunktide atribuutide alusel, tehakse lineaarne interpolatsioon ehk leitakse kohad, kust samajoon antud serva läbib. Kui kõikidele servadele on samajoonte läbimise kohad interpoleeritud tuleb saadud punktid mingisuguse algoritmi alusel ühendada ja saadakse samajooned (Bourke 1987).

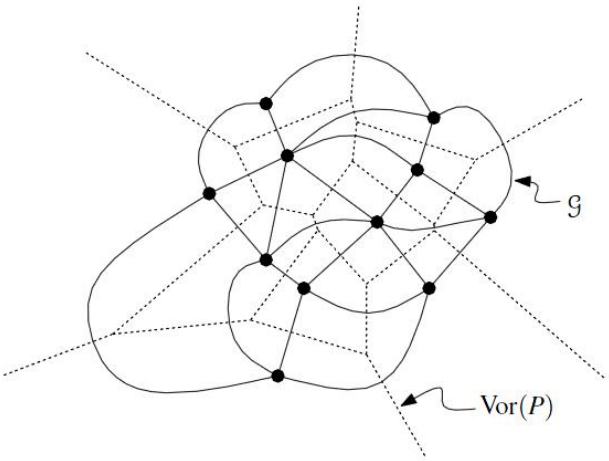
Samajoonte tegemiseks ristkülikutepõhisest võrest tuleb teostada interpolatsioon mööda võre elementide servasid (ristkülikute servasid). Mõned autorid jaotavad ristkülikute võre veel omakorda kolmnurkadeks, kas siis ruudstiku keskele punkti lisamisega või diagonaali pidi kaheks kolmnurgaks tegemisega ja teostavad interpolatsiooni mööda saadud kolmnurkade servasid. Kui servadele on samajoonte läbimise kohad interpoleeritud, saabki need ühendada samajoonteks (Cauvin et al. 2010a:359, Robinson 1978:230).

1.2. DELAUNAY TRIANGULATSIOON

Triangulatsiooniks nimetatakse geomeetrias välja jagamist simpleksiteks. Kahemõõtmelises ruumis on simpleksiteks kolmnurgad. Tasandil oleva punktihulga P vahele on võimalik luua mitmeid erinvaid triangulatsioone. Delaunay triangulatsioonile

vastab olukord, kus mistahes triangulatsioonis oleva kolme punkti poolt moodustatava kolmnurga ümberringjoone sisse ei jää ühtegi teist punkti hulgast P. Delaunay triangulatsioon maksimeerib teda moodustavate kolmnurkade kõigi nurkade miinimum nurga (de Berg et al. 2008).

Punktihulga P Delaunay triangulatsioon on ühtlasi ka antud punktihulga Voronoi diagrammi sirgjoontest koosnev duaalne graaf (joonis 1). Voronoi diagramm koosneb tasandi jaotustest, mis on punktihulga P punktide lähimateks aladeks.



Joonis 1. Punktihulga P Voronoi diagrammi Vor(P) ja selle duaalne graaf G (de Berg et al. 2008).

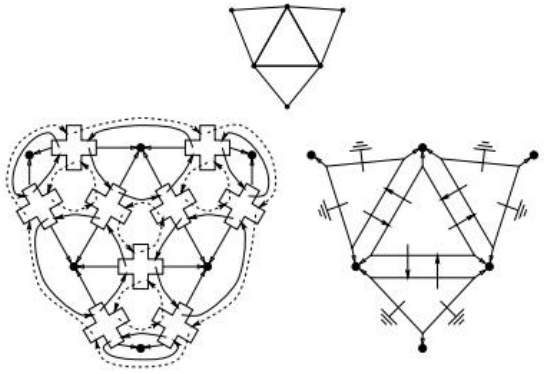
Delaunay triangulatsiooni kirjeldas esmakordset prantsuse juurtega vene matemaatik Boris Delaunay 1934 aastal. Seoses arvutite kasutuselevõtuga on see omandanud väga suure tähtsuse mitmetes eri valdkondades (de Berg et al. 2008).

1.2.1. DELAUNAY TRIANGULATSIOONI ALGORITMID

Matemaatikud ja arvutiteadlased on leiutanud mitmeid algoritme Delaunay triangluatsiooni realiseerimiseks arvutites. Seoses tehnika pideva arenguga (mitmetuumalisus) tehakse veel tänapäevalgi edasiarendusi ja kirjeldatakse uusi olukordi. Siinkohal antakse põgus ülevaate erinevatest Delaunay algoritmidest, rohkem tähelepanu saavad eelnevalt kiirematekks osutunud lahendused. Su ja Drysdale poolt läbi viidud orginaalkoodiga testides olid kolm kiireimat algoritmi Dwyeri jaga ja valluta (*divide and*

conquer), Su ja Drysdale-i *BucketInc* ja Fortune-i *Sweepline*, järeldustes tõdeti, et järjestus võib mõnel teisel süsteemil muutuda (Su, Drysdale 1997).

Triangulatsiooni ehitamisel peab paika panema ka andmestruktuuri. Üks populaarsemaid on Guibas ja Stolfi kirjeldatud "*quad-edge"* struktuur (joonis 2), mis kirjeldab samaaegselt nii Voronoi kui ka Delaunay (Guibas, Stolfi 1985). Seda kasutasid ka Su ja Drysdale oma võrdlustes. (Su, Drysdale 1997) Veel kasutatakse DAG (*Directed Acyclic Graph*) struktuuri, milles säilitatakse kõikide eemaldatud kolmnurkade kohta info ja kõik uued kolmnurgad, mis tulevad vanade asemele omavad seost vanadega. Struktuurist on abi kui otsitakse millisesse kolmnurka punkt sai lisatud (de Berg et al. 2008).

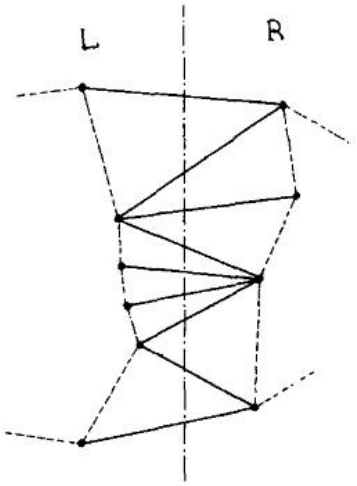


Joonis 2. Üleval on näha triangulatsioon ja all seda kirjeldavad andmestruktuurid. Vasakult paremale vastavalt *"quad-edge"* ja "*triangular"* (Shewchuk 1996).

Shewchuk implementeeris mõlemad andmestruktuurid ja leidis, et "*triangular*" struktuuri (joonis 2) kasutavad algoritmid on umbes kaks korda kiiremad, samas selle implementeerimiseks kirjutatud kood on poole pikem kui "*quad-edge*" puhul (Shewchuk 1996).

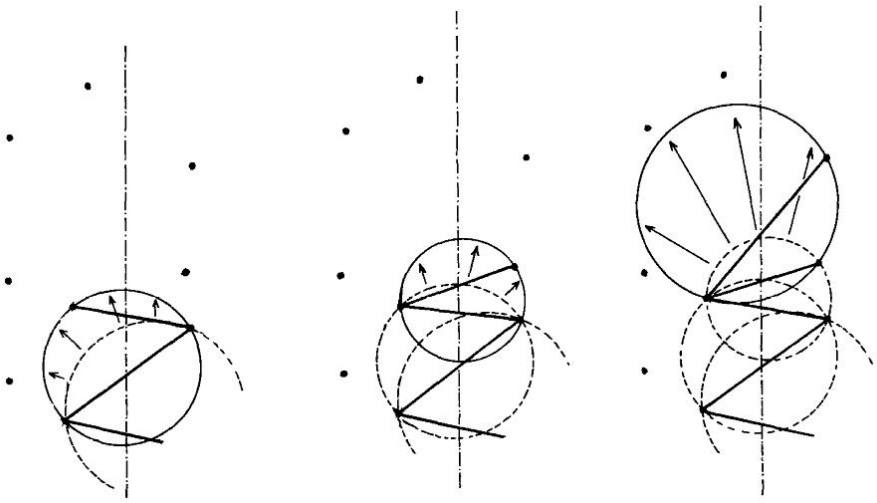
1.2.1.1.Jaga ja valluta *(divide and conquer)*

Jaga ja valluta algoritmi kirjeldasid Guibas ja Stolfi (Guibas, Stolfi 1985). Antud algoritmi keskmiseks keerukushinnanguks on Dweyeri modifikatsioonide abil võimalik saavutada $O(n \times \log(\log(n))$. Halvimal juhul jääks see ka siis $O(n \times \log(n))$ (Dwyer 1987).



Joonis 3. L ja R pooled ning neid lahutav x teljega risti olev sirge (Guibas, Stolfi 1985).

Guibas ja Stolfi sordivad kõigepealt punktid x-telje järgi. Edasi kui on kuni 3 punkti, siis tehakse kohe triangulatsioon, vastasel juhul jaotatakse punktid x-teljega risti oleva sirge abil kahte enam vähem võrdse suurusega rühma L (*left*) ja R (*right*) (joonis 3). Rühmadeks jaotamist korratakse rekursiivselt kuni kõigist rühmadest tehakse Delaunay triangulatsioonid (Guibas, Stolfi 1985).



Joonis 4. L ja R poolte ühendamisel liigub poolte vahelt läbi (alt üles) ümberringjoone kontroll, mis aitab kindlaks määrata millsed L-R servad on legaalsed (Guibas, Stolfi 1985).

Rühmade ühendamisel tuleb kustutada mõned L-L servad ja mõned R-R servad, kuid kunagi ei ole vaja teha uusi L-L ja R-R servasid ehk kõik uued servad peaks olema L-R servad. Servade ühendamine toimub y-teljega samas suunas ja järjekorras, mille määrab L ja R lahutav siht (servade järjekord tuleneb sihi ületamise järjekorrast). Ühendamine algab mõlema poole alumise ühise puutuja leidmisega (mis on ühtlasi esimene ühendus). Edasi liigutakse ümberringjoonte leidmisega (joonis 4) alt üles, kustutades mittesobivaid (L-L ja R-R) servasid ja ühendades sobivaid L-R servasid (kui ümberringjoone sisse ei jää punkte), kuni jõutakse L ja R poolte ülemise ühise puutujani (Guibas, Stolfi 1985).

Dwyer muutis Guibas-Stolfi algoritmi jaotades lähtepunktid ristkülikutest koosnevasse võresse. Kõigepealt tehakse igas ruudus triangulatsioon, seejärel tuleb ühendada paari kaupa samades ridades olevaid võre osasid, kuni kogu rida saab ühendatud. Lõpuks ühendatakse kõik read omavahel samamoodi (Dwyer 1987).

Kattajainen ja Koppinen kasutavad sarnast lähenemist nagu Dwyer, ainult et ühendused tehakse *"quad-tree"* struktuuri järgides (puu struktuur milles iga vanem omab 4 last). Nende endi tehtud võrdluste põhjal on ka tulemused Dwyeri algoritmiga sarnased (Katajainen, Koppinen 1988).

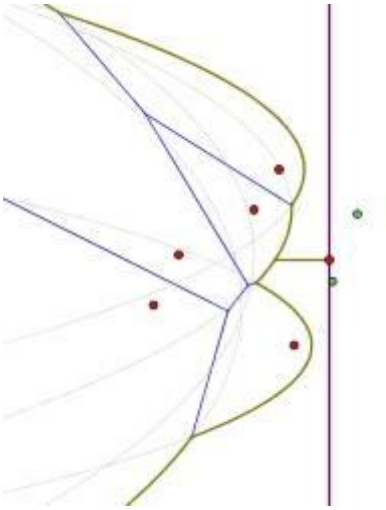
1.2.1.2. Kingituse pakkimie*(gift wrapping)*

Kõigepealt leitakse üks Delaunay tingimustele vastav kolmnurk ja siis lisatakse iga järgnev Delaunay tingimustele vastav kolmnurk (ühe kaupa) olemasoleva struktuuri külge. Sisuliselt otsitakse iga viimati lisatud kolmnurga servale punkti, millega ühendamisel tekiks selline kolmnurk, mille ümberringjoone sisse ei jääks ükski teine punkt. Esialgne serv valitakse punktipilve servast. Edasi valitakse mingisuguse kindla kriteeriumi (näiteks kaugus servast) alusel esialgne kontrollitav punkt. Selleks võib kasutada tuttavat ristkülikutele tuginevat võre ja spiraalotsingut. Edasi kontrollitakse ainult selliseid punkte, mis jäävad antud servaga kontrollitud punktide ümberringjoone raadiuste sisse. Antud algoritmi kiirus sõltub õigete kandidaatpunktide leidmise kiirusest. Kiiruse poolest jääb see parimatele lahendustele alla (Su, Drysdale 1997).

1.2.1.3. Pühkimise joon *(sweep line)*

Antud algoritmi kutsutakse tihti selle looja Steven Fortune järgi Fortune-i algoritmiks. Algne algoritm tehti Voronoi diagrammide loomiseks. Voronoi diagrammidelt on lihtne üle minna Delaunay triangulatsioonile. Kasutatakse kahte joont, mis läbivad punktivälja. Oletame, et jooned läbivad välja vasakult paremale. Üks on pühkimise joon (*sweep line*) ja teine on rannajoon (*beachline*) (Fortune 1986).

Pühkimise joone mõistet kasutatakse arvutusgeomeertrias tihti erinevate eukleidilises ruumis paiknevate probleemide lahendamiseks. Rannajoon tekib paraboolidest, mis näitavad punktidest sama kaugel asuvaid alasid. Pühkimise joon on vertikaalne sirge. Sellest vasakule jäävad punktid on lisatud Voronoi struktuuri ja paremale jäävad punktid veel ootavad lisamist (joonis 5) (Fortune 1986).



Joonis 5. Pühkimise joon (lilla) ja rannajoon (roheline) liiguvad vasakult paremale ning moodustuvad Voronoi servad (sinine) (URL 1).

Algoritmis eristatakse kahte tüüpi sündmuseid. Esimene on punkti sündmus. Iga kord kui pühkimise joon kohtab uut punkti, siis tehakse rannajoonele horisontaalne projektsioon millest saab uus parabool, mis näitab nii lisatud punktist kui ka pühkimise joonest sama kaugele jäävaid alasid. Teine tähtis moment on ringjoone sündmus. Kui parabooli kaar kahaneb punktiks ja kaob rannajoonest, sellel hetkel võib kindel olla, et antud ümberringjoone sisse ei saa tulla ühtegi punkti. Voronoi jaoks on viimane olukord

verteksi asukohaks. Paraboolide kokkupuutepunktidest moodustuvad Voronoi servad (Fortune 1986).

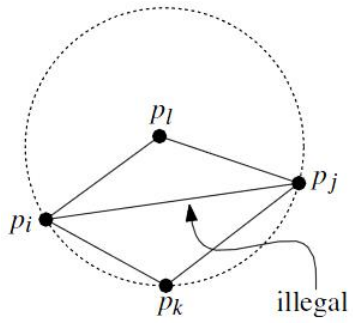
1.2.1.4. Kumer kere *(Convex hull)*

Eraldi on veel kumera kere (või kumera kesta) tüüpi algoritmid, mis reaalselt lahendavad d-mõõtmelise probleemi d+1-mõõtmelisena, leides punktide kumera kere. Näiteks kolmemõõtmelises ruumis leitud kumer kere vastab kahemõõtmelises ruumis Delaunay triangulatsioonile (kui õige nurga alt vaadata). Ka ümberringjoone testi üks seletustest lähtub samast (joonis 7), kumera kere puhul oleks kõik kolmnurgad paraboloidil ja kogu asja projektsioon xy-tasandile oleks Delaunay triangulatsioon. Parimad kumera kere leidmise algoritmid jäävad Su ja Drysdale järgi vähemalt kahemõõtmelises ruumis teistele kiiruse poolest alla (Su, Drysdale 1997).

1.2.1.5. Juurde lisav *(incremental)*

Juurde lisavat algoritmi peetakse paljude autorite poolt kõige lihtsamini implementeeritavaks. Tõenäoliselt on tegu kõige laialdasemalt kasutatud Delaunay algoritmiga (Žalik, Kolingerová 2003).

Idee seisneb punktide ükshaaval triangulatsiooni lisamises. Peale iga uue punkti lisamist muudetakse olemasolevat triangulatsiooni, kuni see vastab Delaunay tingimustele. Lawson kasutas peale punkti lisamist kuskile kolmnurka mittesobivate servade pöördeid, mille tulemusel muutavad need sobivateks (joonis 6) (Lawson 1977).



Joonis 6. Lawsoni ringi kriteerium. Kui kolme punkti järgi teha ümberringjoon, saab serva legaalsuse üle otsustada vaadates kuhu jääb neljas punkt. Kui neljas jääb ringi sisse (pl) on legaalne serv temaga ühendatud (pk-pl) (de Berg et al. 2008).

Tavaliselt alustatakse suure kolmnurga konstrueerimisega. See peab olema nii suur, et ükski tema sisse pandava kolmnurga ümberringjoon ei sisaldaks selle kolmnurga punkte (de Berg et al. 2008). Pärast punki lisamist kuskile kolmnurka tekib kolm uut kolmnurka. Seejärel tuleb kontrollida, kas uute kolmnurkade välisservad (algse kolmnurga servad) on legaalsed, ehk kas nende ümberringjoontesse jääb ka nende naaberkolmnurga punkt. Kui jääb tuleb teha servade pööre ja tekkinud uued välisservad kontrollida. (Lawson 1977).

Bowyer ja Watson (mõlemad pakkusid sama lahenduse samas väljaandes eraldi artiklites) kustutasid kõik kolmnurgad, millede ümberringjoontes lisatud punkt oli, ja seejärel ühendasid järele jäänud naabri kaotanud servade tipud uude punkti (Bowyer 1981, Watson 1981).

Juurde lisavat tüüpi algoritmi kõige aeglasemad osad on ümberringjoone test (peatükk 1.2.2.1) ja kolmnurga leidmine (millesse punkt sisestati) (peatükk 1.2.2.2). Kolmnurga leidmist saab parandada, kui integreerida Bentley, Weide ja Yao poolt kirjeldatud lähima naabri leidmise algoritm ning Guibas ja Stolfi poolt kirjeldatud kolmnurga otsingu algoritm. Su ja Drysdale suutsid selle täiendusega teha juurde lisavast algoritmist nende testitud põhiliste algoritmide hulgast kiiruselt teise algoritmi, nende katsetes jäi see alla vaid Dwyeri jaga ja valluta algoritmile (Su, Drysdale 1997).

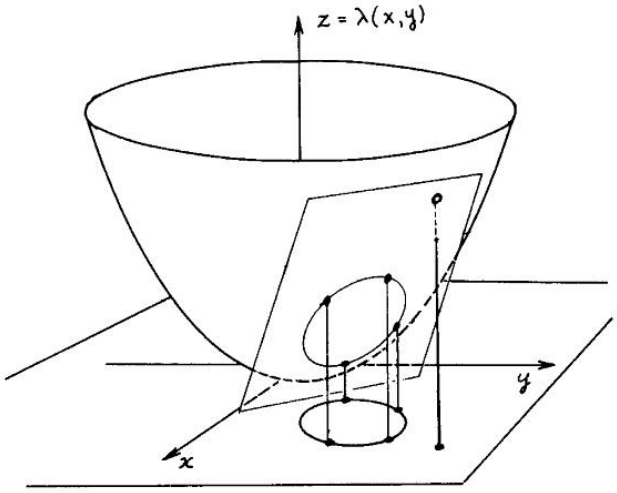
1.2.2. MÕNED TÄHTSAMAD ALAMÜLESANNETE LAHENDAMISE ALGORITMID, MIDA PALJUD EELTOODUD LAHENDUSED KASUTAVAD

Triangulatsiooni loomisel tuleb lahendada mitmeid alamülesandeid ja oluline on ka nende puhul leida parimad lahendused. Enamus algoritmidest kasutab alamülesannete lahendamiseks Guibas ja Stolfi poolt kirjeldatud ümberringjoone kontrolli ja CCW testi (peatükk 1.2.2.2).

1.2.2.1. Serva legaalsuse kontroll

Antud testi võiks ka nimetada ümberringjoones asumise testiks. Selle tulemusel saab teada, kas vaadeldav punkt asub vaadeldava kolmnurga ümberringjoone sees. Kõige otstarbekam on kasutada determinandi hindamist. Test eeldab, et kolmnurga punktid a b c on vastupäeva (punktide järjekord). Kui punktide järjekord muutub siis muutub ka testi tulemus (Guibas, Stolfi 1985).

Determinandi hindamise saab lahti mõttestada järgnevalt. Kui meil on paraboloid P, mis on määratud valemiga $z = x^2 = y^2$ ja mittevertikaalne tasand H, mis on määratud valemiga $z = \alpha x + \beta y + \gamma$, siis saame, et P ja H ühisosa projektsioon tasandile O_{xy} on $x^2 + y^2 = \alpha x + \beta y + \gamma$, tegemist on ringiga.



Joonis 7. Kolmnurga punktide poolt määratud tasandi ja paraboloidi ühisosa projektsioon x y-tasandile on ring (Guibas, Stolfi 1985).

Punkti $p = (p_x; p_y)$ saab panna paraboloidile $p = (p_x; p_y; p_x^2 + p_y^2)$ Kui nüüd määrata tasand H kolmnurga punktide a b c poolt, mis on pandud paraboloidile nagu punkt p, saame determinanti hinnates teada, kas punkt d on kõrgemal kui tasand H (ümberringjoonest väljas) või madalamal kui tasand H (ümberringjoone sees) (joonis 7) (Guibas, Stolfi 1985).

$$\begin{vmatrix} 1 & a_x & a_y & a_x^2 + a_y^2 \\ 1 & b_x & b_y & b_x^2 + b_y^2 \\ 1 & c_x & c_y & c_x^2 + c_y^2 \\ 1 & d_x & d_y & d_x^2 + d_y^2 \end{vmatrix} > 0$$

Kui determinandi väärtus on suurem kui 0, siis on vaadeldav punkt d kolmnurga ümberringjoonest väljaspool.

$$\begin{vmatrix} 1 & a_x & a_y & a_x^2 + a_y^2 \\ 1 & b_x & b_y & b_x^2 + b_y^2 \\ 1 & c_x & c_y & c_x^2 + c_y^2 \\ 1 & d_x & d_y & d_x^2 + d_y^2 \end{vmatrix} = 0$$

Kui determinandi väärtus on 0, siis on vaadeldav punkt d kolmnurga ümberringjoone peal.

$$\begin{vmatrix} 1 & a_x & a_y & a_x^2 + a_y^2 \\ 1 & b_x & b_y & b_x^2 + b_y^2 \\ 1 & c_x & c_y & c_x^2 + c_y^2 \\ 1 & d_x & d_y & d_x^2 + d_y^2 \end{vmatrix} < 0$$

Kui determinandi väärtus on väiksem kui 0, siis on vaadeldav punkt d kolmnurga ümberringjoone sees (Guibas, Stolfi 1985).

1.2.2.2. Kolmnurgas olemise kontroll ja CCW (*counter-clockwise*) test

Kolmnurgas olemise kontrolli kasutatakse *incremental* tüüpi Delaunay algoritmides, et leida, millisesse kolmnurka punkt lisati. Samuti kirjeldan siinkohal CCW testi, mis näitab kas kolmnurga punktid on päri või vastupäeva ja seda kasutatakse paljudes Delaunay algoritmides vahetult enne serva legaalsuse kontrolli.

1)Guibas-Stolfi *locate* rutiin

Punkti asumine kolmnurgas tehakse kindlaks vaadates igat kolmnurga külge eraldi. Kui on punktid A, B ja C siis saab vaadata mis pool on punkt C sirgest AB. Selleks tuleb lahendada determinant ehk **CCW test**.

$$\begin{vmatrix} X_A & Y_A & 1 \\ X_B & Y_B & 1 \\ X_c & Y_c & 1 \end{vmatrix}$$

Kui lahend on suurem kui 0, siis on kolmnurk vastupäeva, kui väiksem kui 0, siis päripäeva. Juhul kui punktid on ühel sirgel, tuleb lahendiks 0. Punkt on kolmnurgas, kui ta on kõigist külgedest samal poolel. Kui järjest kolmnurga külgi kontrollida ja mõne puhul selgub, et

punkt ei ole selles kolmnurgas, tuleb võtta selle serva naaberkolmnurk ja korrata eelnevat, kuni leitakse õige kolmnurk (Guibas, Stolfi 1985).

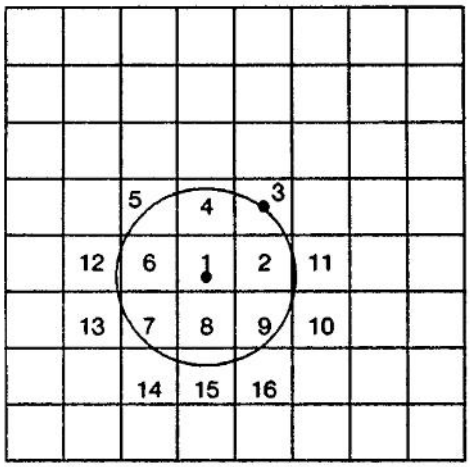
2)Ämbrite süsteem (*bucketing*)

Su- Drysdale tegid eelnevalt kirjeldatud Guibas- Stolfi *locate* rutiinile ühe lihtsa täienduse, mille abil saab kogu algoritmi veelgi paremaks. Selleks tuleb katta kogu vaadeldav ala ristkülikutest koosneva võrega (*grid*). Iga uue punkti lisamisel saab punkti koordinaate ümardades kiiresti teada, kuhu ristkülikusse/ämbrisse see langeb. Iga ämber jätab meelde temasse viimati lisatud punkti. Kolmnurga otsimiseks käivitatakse Guibas-Stolfi *locate* rutiin, kuid seda ei tehta suvalisest servast lähtudes, vaid kasutades punkti saanud ämbri või tema lähiümbruses oleva ämbri punktist saadud serva. Niimoodi toimides hakatakse kolmnurka otsima sellele (kolmnurgale) oluliselt lähemalt ja see leitakse mõne kontrolli tagajärjel sõltuvalt ämbrite struktuuri tihedusest. Kui ämbris ei ole ühtegi punkti, hakatakse sellest lähtuvalt spiraalselt otsima lähimat ämbrit kus oleks mõni punkt (Su, Drysdale 1997).

1.2.2.3. Spiraalotsing

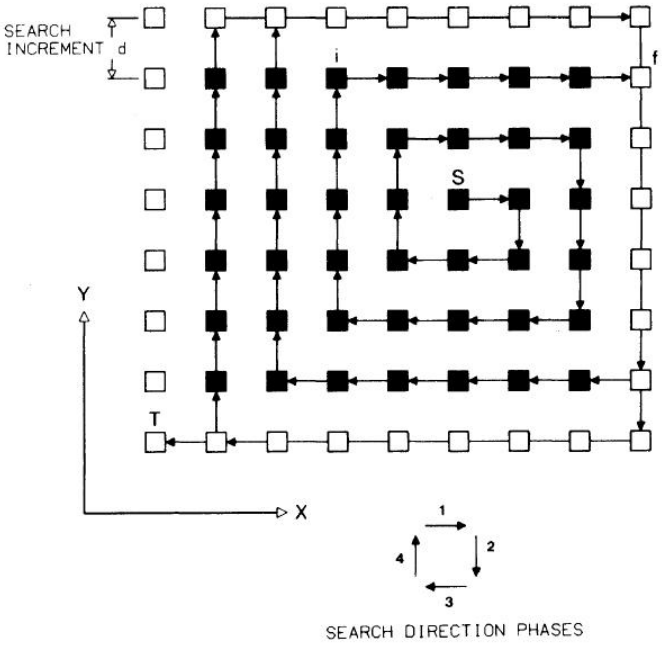
Spiraalotsingu üldpõhimõte on, et alustatakse mingisugusest võre elemendist ja siis hakatakse spiraalselt temast eemalduma, käies läbi vastavaid võre elemente. Otsing peab lõppema, kui sobiv element leitakse.

Bentley, Weide ja Yao pakkusid välja omapärase spiraalotsingu, mis liigub küll spiraalselt, kuid otsib vaid võre elementidest, mis jäävad otsingu algpunktist tõmmatud ringi või lõikuvad sellega (joonis 8)(Bentley et al. 1980).



Joonis 8. Spiraalotsing, mis alustab väikseima numbriga ruudust ja liigub spiraalselt läbi ruutude, mis jäävad ringi või lõikuvad sellega (Bentley et al. 1980).

Hall pakkus välja lahenduse, mis arvestab võre (*grid*) mõõtmetega (joonis 9) (Hall 1982).



Joonis 9. Päripäeva kulgev spiraalotsing, mis arvestab ka võre mõõtmetega (Hall 1982).

1.2.2.4. Punkti kustutamine triangulatsioonist

Devillers pakkus välja meetodi, mis on modifikatsioon Helleri (vigasest) algoritmist ning teeb sellest atraktiivse lahenduse (Devillers 1998). Mostafavi kirjeldab 4 sammulise

punkti kustutamise algoritmi, mis tema sõnul on lihtsam kui Devillers-i pakutud, kuid muutub suuremate naabrite arvuga aeglasemaks. (Mostafavi et al. 2003).

1.3. SAMAJOONTE SILUMINE

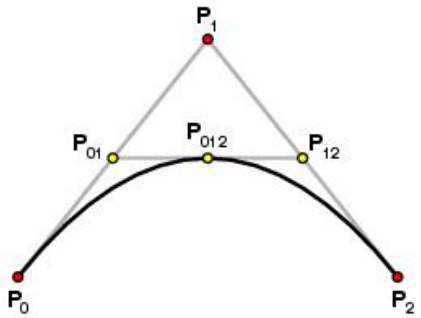
Visuaalselt parema tulemuse saamiseks võib samajooni peale ühendamist siluda. Järgnevat toon ära silumise võimalused Bezier' kõveratega. Neid sai ka töö empiirilises osas proovitud. Lisaks Bezier' kõveratele kasutatakse arvutigraafikas ka *B-spline* kõveraid, mis on küll paindlikumad, kuid vajavad rohkem arvutusi ning on raskemini mõistetavad (Andersson 2003).

1.3.1. BEZIER' KÕVERAD

Üsnagi populaarne on silumine Bezier' kõverate abil. Bezier' kõverad said oma nime prantsuse autotootja Renault inseneri Pierre Bezier järgi, kes neid 1960ndatel laialdaselt populariseeris ja kasutas. Matemaatiliselt on tegu Bernsteini polünoomidega, mis olid juba olemas 1912 aastal. Bezier' kõverad on atraktiivsed arvutigraafikas, sest neid saab suhteliselt kiiresti leida ning joone kuju on hästi kontrollitav. Üheks populaarsemaks viisiks Bezier' kõverate leidmisel on De Casteljau algoritm, mille töötas välja samanimeline mees, kes oli Bezier' kaasaegne ja töötas Citroeni jaoks. Algoritm jagab punktide ja kontrollpunktide vahelisi joonte segmente rekursiivselt pooleks, kuni saavutatakse piisavalt sujuv kõver (Farouki 2012).

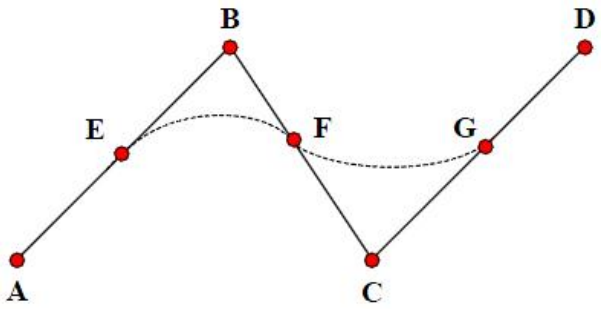
1.3.1.1. *Quadratic* Bezier' kõverad

Quadratic kõverate puhul määratakse joone kuju kahe joone punkti ja ühe kontrollpunktiga (joonis 10).



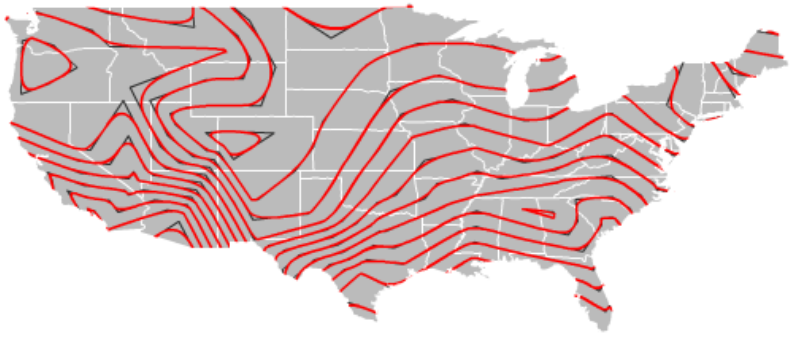
Joonis 10. Ühe kontrollpunktiga - P_1 quadratic Bezier' kõver punktide P_0 ja P_2 vahel (URL 11).

Isojoone silumiseks *quadratic* kõveratega, tuleb leida kahe järjestikuse serva keskkohad ja need ühendada quadratic kõveraga nii, et kontrollpunkt jääb servade ühisesse punkti (joonis 11). *Quadratic* kõverate plussideks on lihtsus, suhteliselt kiire arvutatavus (Rui et. al. 2010).



Joonis 11. *Quadratic* Bezier' kõveratega silumine. Punktide E ja F kontrollpunktiks on B ning punktide F ja G kontrollpunktiks on C. E, F ja G asuvad vastavalt AB, BC ja CD keskel (Rui et al. 2010).

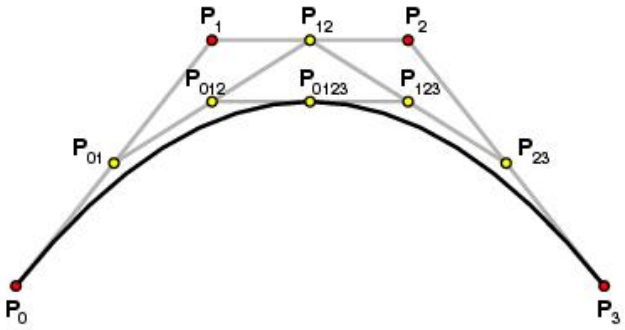
Zachary Forest Johnson tõi välja, et sellise meetodiga siludes ei läbi samajoon interpoleeritud punkte (joonis 12) (URL 12).



Joonis 12. USA samajoonte kaart. Sirgete tipud (mustaga) on ainsad kohad, kuhu samajoonte punktid on interpoleeritud, kuid *quadratic* Bezier' kõveratega siludes samajooned ei läbi neid punkte (URL 12).

1.3.1.2 *Cubic* Bezier' kõverad

Cubic kõvera puhul määravad joone kuju 4 punkti. 2 joone punkti ja 2 kontrollpunki (joonis 13).



Joonis 13.Kahe kontrollpunktiga - P_1 ja P_2 *cubic* Bezier' kõver punktide P_0 ja P_3 vahel (URL 11).

Cubic kõveratega silumisel on kontrollpunktide asukohtade leidmine mõnevõrra keerulisem. Kõigepealt tuleb leida servade keskkohad, seejärel leida punkt naaberservade vahel. Otsitava punkti K asukoht on määratud antud servade pikkuste suhtega servade keskkohtade vahelisel lõigul. Kui esialgsed servad on S1 ja S2 ning otsitav punkt on serva S1 keskkohast kaugusel d1 ning serva S2 keskkohast kaugusel d2, siis S1/S2=d1/d2. Edasi "nihutatakse" keskkohtade vaheline lõik sellisel viisil, et tema orientatsioon ei muutu, aga punkt K asub S1 ja S2 ühises punktis. Sedasi toimides on kontrollpunktide asukohad nüüd nihutatud lõikude otspunktides. Kui juba kontrollpunktide asukohad on määratud, tuleb joonistada *cubic* kõverad. *Cubic* kõverate puhul on veel võimalik kasutusele võtta koefitsent, mis määrab, kui siledad või teravad kõverad tekivad. Koefitsent määraks kontrollpunktide kaugust nihutatud kõveral S1 ja S2 ühispunktist. Kui kontrollpunktid on üksteisele lähemal, tekib teravamate nurkadega kujutis (URL 4).

Cubic kõverate peamiseks plussiks võrreldes *quadratic* kõveratega on see, et silutud samajoon läbib interpoleeritud punkte (URL 12). Negatiivseks aspektideks on suurem arvutusmahukus (Farouki 2012).

2. VEEBISIRVIJATES KUVATAVATE INTERAKTIIVSETE KAARDIANIMATSIOONIDE LOOMINE

Antud peatükis antakse ülevaate erinevatest tehnoloogilistest tööriistadest, millega on võimalik interaktiivseid animatsioone veebisirvijates luua. Samuti tutvustatakse lühidalt animatsioonide ja täpsemalt kartograafiliste animatsioonide teooriat.

2.1. KAARDIANIMATSIOON

Tajutav, sujuv animatsioon algab kuskil 24st kaadrist sekundis, edaspidi *fps* (*frames per second*). Seda kasutab ka filmitööstus. Sellise kaadrisagedusega ei taju kasutaja enam staatilisi pilte. Mida rohkem kaadreid sekundis näidata, seda sujuvam on animatsioon. Miinimum piiri, mille juures saab veel tinglikult animatsioonist rääkida on kuskil 5fps juures (Harrower, Fabrikant 2008).

Animeeritud kaardid annavad võimaluse vaadata ajalis-ruumilisi andmeid uudse nurga alt. Välja võivad joonistuda mustrid, mida staatiliste kaartidega ei ole lihtne märgata. Animeeritud kaardid sobivad hästi näitamaks geograafiliste fenomenide ajalist muutust ja erinevaid protsesse (Harrower, Fabrikant 2008, Lobben 2003, Cauvin et al. 2010c).

Harroweri järgi on animeeritud kaardid ajaloo vältel teinud läbi muutuse, mis üha rohkem kaasab kasutajat. Kui algselt said kasutajad animatsiooni lihtsalt vaadata, siis hiljem oli juba võimalik kontrollida esitust. Edasi lisandus võimalus kontrollida välimust ja lõpuks kaasaegsete animatsioonide puhul on kasutajal võimalik olla ise autoriks (Harrower 2004).

Lobben klassifitseeris kaardianimatsioone vastavalt aja, ruumi ja fenomeni muutumisele nelja rühma. Kui eeltoodud näitajatest muutub ainult aeg, on tegu aeg-rea (*time-series*) animatsiooniga. Teise rühma moodustavad animatsioonid, kus muutub ainult ruum (*areal animation*), siia alla lähevad näiteks erinevad läbilennud kolmemõõtmelistel kaartidel. Kui ruum ei muutu, aga aeg ja fenomen võivad muutuda, on tegu temaatilise kaardianimatsiooniga. Aeg võib olla ka muutumatu ja ainult fenomen muutub. Neljanda klassi moodustavad protsesse kujutavad animatsioonid, millede puhul võivad muutuda kõik kolm aspekti, erandina võib olla ruum staatiline (Lobben 2003).

Konstrueerimise alusel jaotatakse kartograafilised animatsioonid kolmeks. Esiteks kaadri põhised, kus kõik kaadrid konstrueeritakse eraldi ja siis pannakse kokku ja näidatakse järjest. Sellist lähenemist peetakse väga töömahukaks. Teiseks animatsioonid, mis kasutavad objektide alg- ja lõppasukohta või atribuutide olekut (*keyframes*) ning vahepealsed kaadrid arvutatakse (*tweening*). Sellistel kaartidel on tavaliselt taustaks staatilised pildid, millede peal liigutatakse teisi objekte, vastavalt vajadusele peidetakse ja näidatakse erinevaid objekte. Kolmandaks algoritmilised animatsioonid, kus kõik kaadrid arvutatakse eraldi välja, sellistele animatsioonidele saab lisada täieliku interaktiivsuse vastavalt vajadustele (Cauvin et al. 2010c:52).

Lisaks tavamõõtkavale on animeeritud kaartidel ka ajaline mõõtkava, mis näitab nähtuse kulgemise aja suhet animatsiooni kestvusesega. Lisaks on neil tempo, mis näitab kui kiiresti kujutatav nähtus ajaühikus muutub ja ajaline resulutsioon (vähim ajaline jaotus) (Harrower, Fabrikant 2008).

Ajalise mõõtme lisandumine teeb kasutajal kujutatu jälgimise keerukamaks (Harrower, Fabrikant 2008). Fabrikant on uurinud, kuidas kasutaja animatsiooni tajub. Tähtis on, kuidas toimub kasutaja teadmiste konstruktsioon, kui ta kaarti kasutab. Harrower ja Fabrikant ei soovita väga pikkade animatsioonide kasutamist. Kasutajal on raske üle minuti pikkustest animeeritud kaartidest kasulikku infot ammutada (Harrower, Fabrikant 2008). Tversky tõi välja, et animatsioonist on kasu pigem pidevalt (mitte diskreetselt) muutuvate nähtuste näitamisel. Samuti rõhutas ta interaktiivsuse tähtsust, kasutajal peab olema võimalus animatsiooni kulgu juhtida (Tversky et al. 2002).

2.2. TEHNOLOOGIA

2.2.1. ARVUTUSTE TEGEMISE KOHT

Iga veebileht paikneb kuskil arvutis, mida kutsutakse serveriks. Kui kasutaja küsib serverist oma arvutiga veebilehte, siis server peab andma küsitud aadressile vastava sisu. Kuvatava veebilehe arvutusi saab teha nii serveris (*server-side*), mis lehte näitab, kui ka kliendi arvutis, millega kasutaja lehte vaatab (*client-side*). Serverid on tavaliselt võimsad arvutid ja neil on võimalus kasutada kiireid serveri poolseid keeli (Brinzarea et al. 2006).

Igal veebilehe puhul peavad arendajad tegema valikuid, mida peaks arvutama serveris ja mida kliendi arvutis. Serveri poolsete arvutuste kasutamine eeldab, et server suudab kõik potensiaalsed kasutajad ära teenindada (kõigile arvutused ära teha). Serveri poolsete keelte kasutamine aitab ka turvaliselt piirata kasutajate ligipääsu erinevatele andmebaasidele ja lehe osadele (Brinzarea et al. 2006).

Kliendipoolseid arvutusi tehakse veebisirvijas. Üldiselt on need aeglasemad (kui serveripoolsed arvutused) ja sõltuvad ka kliendi arvuti võimekusest. Kliendipoolseid arvutusi kasutatakse kliendi käitumisele reageerimiseks (interaktiivsus). Vastavalt kliendi käitumisele (nuppude vajutamine) saab kliendipoolsete keelte abil serveri poolseid skritpte (teenuseid) tööle lasta ja saadud vastusega omakorda midagi peale hakata (Brinzarea et al. 2006).

Seoses üha suureneva graafikakiirenduse toetuse tulemisega internetisirvijatele, soovitatakse keerulist graafikat üha enam arvutada kliendi arvutis. Sellise lähenemisega saab ära hoida aeglast andmevahetust serveriga (URL 17).

2.2.2. ANDMETE HOIDMINE

Lihtsamad veebilehed hoiavad oma andmeid lehe struktuuris või eraldi failides, mis on serveris neile kättesaadavad. Keerukamatel juhtudel luuakse eraldi andmebaas, milles vajaminevaid andmeid hoitakse ja millele saab päringuid teha. Moodsas veebitehnoloogias ehitatakse rakendused võimalikult dünaamiliselt. Rakendus saab uue sisu, kui vahetada andmeid. Andmete rakendustest eraldi hoidmine võimaldab mugavalt kasutada erinevaid väliseid andmeid. Populaarseks on saanud erinevate andmehaldajate poolt loodud andmeteenused, mis on mingil kokkulepitud vormis ja millest rakendused saavad neid küsida ning vastavalt sellele lehe sisu muuta.

2.2.3. ERINEVAD ANIMATSIOONILOOMISE TEHNOLOOGIAD KAASAEGSES INTERNETISIRVIJAS

Eelkõige tutvustab antud peatükk praktilises osas kasutatud tehnoloogiaid ja argumente, mida oleks vaja teada, et praktilisest osas tehtud valikuid mõista.

2.2.3.1. Flash/Flex, Java, Silverlight ja HTML5 (*Rich Internet application-RIA*)

Nii Flash/Flex-i(edaspidi Flash), Silverlighti, Java kui ka HTML5 tehnoloogiatega on võimalik luua interaktiivseid veebirakendusi.

Plugin on lisaprogramm mille kasutaja peab paigaldama oma arvutisse. See aitab internetisirvijal näidata sisu, mille näitamist internetisirvija muidu teha ei oskaks. Flash vajab asjade kuvamiseks Flash Playerit. Java käivitamiseks peab olema installitud Java plugin (Brinzarea et al. 2006). Silverlighti kasutamiseks peab samuti olema installitud vastav Microsoft Silverlight plugin. Internetisirvijad ei luba vananenud plugin programme vaikimisi jooksutada, kasutaja peab neid eraldi lubama. 50% kodulehtedelt lahkumisi pannakse plugin-ide installimise vajaduse alla (URL 7). HTML5 on ainus eeltoodud tehnoloogiatest, mis vajab lihtsalt moodsat internetisirvijat.

Samuti on probleeme Flashi kodulehtede jooksutamisega nutiseadmetes (kui peaks selline vajadus tekkima), sest Adobe (tehnoloogia omanik) ei tooda alates 15 august 2012 neile ise Flash Playerit. Ka Microsoft Silverlight-i plugin ei tööta nutiseadmetes.Normaalsel viisil ei ole võimalik nutiseadmetes vaadata ka *Java applet*-e. Kuna HTML5 jookseb otse internetisirvijas, saab seda ka nutiseadmetes vaadata.

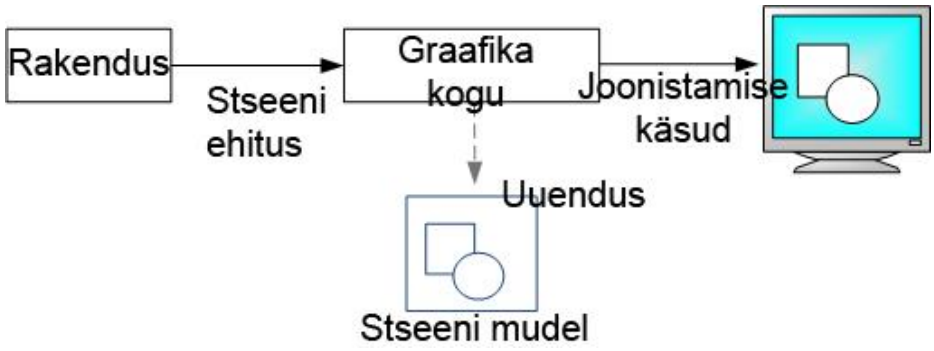
HTML5 ei ole veel valmis, kuid stabiilset versiooni lubatakse aastaks 2014. Lisaks tehakse brauseris istuvat JavaScripti mootorit järjest paremaks ja vahe kiiremate tehnoloogiatega on järjest väiksem (URL 7). Kasutajate poolt tehtud tehnoloogiate võrdlused on näidanud, et HTML5 hakkab Flashile järele jõudma. Kui 2010 oli Flashi tehnoloogia kiiruse poolest kindlalt üle, siis 2012 tehtud testides olid mõned testid juba HTML5 kasuks (URL 5).

HTML5 näol on tegemist lahtise standardiga, see ei kuulu otseselt ühelegi firmale või internetisirvijale. Hetke veebitööstus soosib HTML5 arendamist, suured firmad teevad sellel alal koostööd ning tulevik on selle tehnoloogia päralt.

2.2.3.2. HTML5 canvas ja SVG

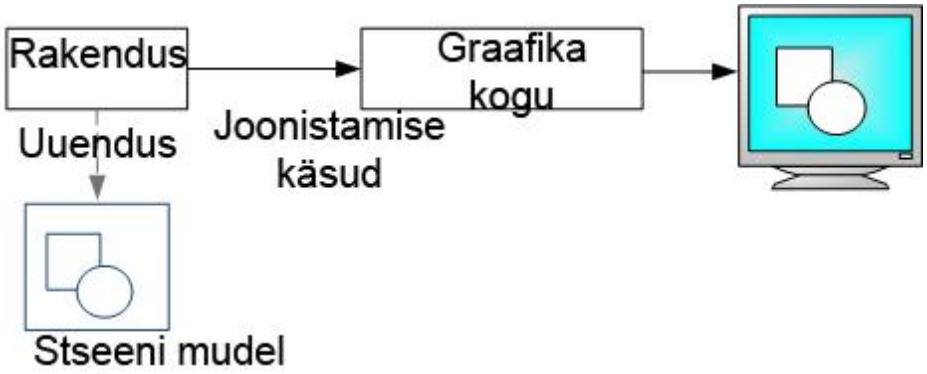
Mõlemad tehnoloogiad kuuluvad HTML5 standari juurde. Mõlema tehnoloogiaga on võimalik saavutada väliselt sama. HTML5 standardis saab animatsioone luua ka WebGL (peamiselt 3D jaoks) ja CSS3 (*Cascading Style Sheets* - HTML lehtede kujundamiseks kasutatav tehnoloogia) abil, praktilises osas lahendatava ülesande osas ei annaks need tehnoloogiad praegu midagi juurde ja seetõttu neid pikemalt ei käsitle (URL 7, URL 6).

SVG (*Scalable Vector Graphics*) kujutab endast skaleeritavat vektor graafikat. HTML5s järgib SVG niiöelda *retained-mode* mudelit (joonis 14), mis tähendab, et kõik renderdatavad objektid säilitatakse arvuti mälus stseeni mudelis. SVG elemente hoitakse lehe DOM (*document object model*) strukuuris ja nende atribuute on võimalik peale esmarenderdamist muuta. See põhjustab elemendi uuesti renderdamise. Muudetakse skriptide ja CSS abil (URL 7, URL 6).



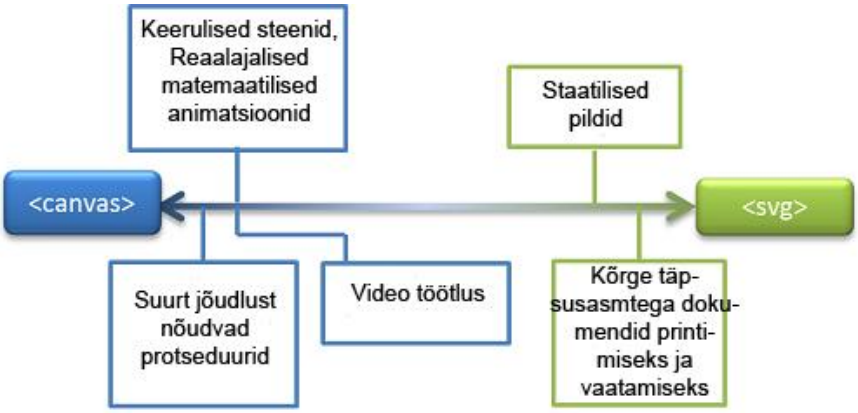
Joonis 14. *Retained-mode* mudeli puhul hoitakse stseeni seis mälus (URL 6).

Canvas on HTML element, millele saab joonistada. Canvas järgib *immediate mode* mudelit (joonis 15) ehk canvas-ile joonistatud asju ei saa hiljem muuta, peale renderdamist kaob seos objektiga. Muutuste kajastamiseks tuleb uus asi vana asemele joonistada. Muudetakse ainult skriptide abil (URL 7, URL 6).



Joonis 15. *Immediate mode* mudeli puhul ei hoita stseeni eraldi mälus (URL 6).

Canvasi renderdamise kiirust mõjutab põhiliselt selle suurus. Mida suurem canvas, seda rohkem pikseleid tuleb joonistada. SVG muutub üha aeglasemaks iga lisanduva elemendiga, sest kõik hoitakse dokumendis eraldi objektidena (URL 7, URL 6).



Joonis 16. Keerulised stseenid ja real-ajalised matemaatilised animatsioonid sobivad tavaliselt paremini canvas elemendiga lahendamiseks (URL 7).

Canvas sobib paremini keeruliste arvutuslike stseenide animeerimiseks (joonis 16). SVG sobib rohkem täpsete jooniste jaoks. Animatsiooni vahendina sobib SVG pigem objektide parameetrite animeerimiseks (näiteks ringi raadiuse muutumine). SVG eeliseks on lihtsamini implementeeritav interaktiivsus, mis tuleneb DOM (*Document Object Model*) eventide kasutamise võimalikkusest. Canvas-i pilt on raster. Sellest tingituna ei saa seda skaleerida ja hiire sündmuste kuulamine on keerukam kui SVG puhul (URL 7, URL 6).

2.2.3.3. Animatsioon HTML5 canvas elemendil

Canvas elemendil toimuva animatsiooni juhtimiseks kasutatakse JavaScripti. Animatsiooni tegemiseks javascripti abil on mitmeid võimalusi. Saab kasutada "*setInterval*" ja "*setTimeout*" ja "*requestAnimationFrame*" funktsioone. Nii "*setTimeout*" kui ka "*setInterval*" võimaldavad teatud aja tagant mingeid protsesse käima lükata. Mõlemad funktsioonid ei tea aga midagi internetisirvija olukorrast hetkel, kui tegevust nõutakse, see võib põhjustada internetisirvija ülekoormamist (URL 18).

Kõige targem lahendus on kasutada uudset Firefoxi poolt välja käidud "*requestAnimationFrame*" funktsiooni. Antud funktsioon suhtleb internetisirvijaga, tegevus lükatakse käima ainult siis, kui brauser on selleks valmis. Animatsiooni ei tehta, kui kasutaja ekraanil ei ole parasjagu canvas elementi, näiteks kui ta on kerinud lehe animatsiooni juurest ära. Selline lähenemine aitab ka säästa akut (kui on tegu akuga

seadmega). Taolise lähenemisega kaasneb animatsioon, mis on maksimaalselt 60kaadrit sekundis. Tegu on mõistliku piiriga, sest enamus moodsate arvutite ekraanide pildi uuendamise sagedus on (*refresh rate*) 60Hz (URL 18).

2.3. ÜLEVAADE SARNASTEST TÖÖDEST

Sarnaseid töid on tehtud erinevate samajoonte kaartide genereerimiseks. Samuti leidub samajoonte animatsioone, kus kaadrid on ette arvutatud. Siiski ei ole autori kätte jõudnud ühtegi tööd, kus proovitaks samajooni andmepunktidest lennult interpoleerida ja samal ajal animeerida rääkimata selle tegemisest internetisirvijas kasutades HTML5-te.

2.3.1. SARNASED UUEMAD SAMAJOONI GENEREERIVAD TÖÖD

YiHong ja YongJiang tegid töö samajoonte genereerimisest tuginedes Delaunay triangulatsioonile. Samajoonte silumiseks kasutati *quadratic B- spline* meetodit. Nende töö juures oli huvitava aspektina toodud ka algoritm samajoonte vaheliste alade värvimiseks (YiHong, YongJiang 2010). Xiao-ping Rui töörühm tegi samajooned kasutades Delaunay triangulatsiooni ja silumiseks kasutasid nad *quadratic* Bezier' kõveraid. Nende töö erilisus seisneb piiravate joonte ja alade lisamises triangulatsiooni (Rui et al. 2010). Jia-Hui, Lan-Fang ja De-Tang kasutasid samuti sarnast lähenemist interpoleerimaks samajooni erinevate gaasi ja nafta maardlate näitajate visualiseerimise tarbeks. Silumiseks kasutati *cubic* Bezier' kõveraid (Chen et al. 2010). Zhao, Liu, Li ja Xing testisid sarnast metoodikat kontuuride leidmiseks, kasutades pihuarvuteid (arvutava seadmena) (Zhao et al. 2012).

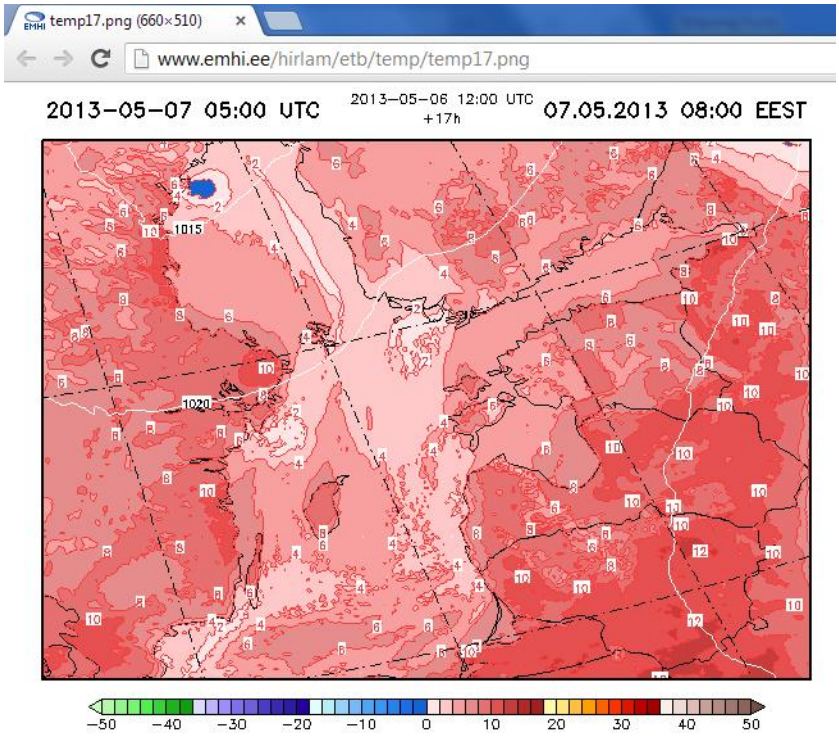
Razafindrazaka ehitas Delaunay triangulatsioonile toetudes üles maastiku generaatori. Töö lõpptulemus oli küll kolmemõõtmelise pseudomaastiku generaator, kuid ülesehituselt on tegu sarnase tööga, sest see toetub *incremental* tüüpi Delaunay triangulatsioonile. Käesoleva töö praktilisel poolel tehtud koodi saaks kerge vaevaga viia sarnase tulemuseni (Razafindrazaka 2009).

Zachary Forest Johnson on lahendanud sarnaseid probleeme oma blogis, kus ta tegi miniatuurse USA samajoonte kaardi kasutades sarnaseid meetodeid (Delaunay triangulatsioonil põhinev samajoonte genereerimine) ja Flash (joonis 12). Samuti tegi ta OpenLayers JavaScripti koodikogule täienduse, millega on võimalik samajooni luua OpenLayers kaartidele. Positiivne on, et tema töö kood on kõigile vabalt kättesaadav.

OpenLayersi kaartidele loodud samajooned on SVG vormis. Võimalus on muuta samajoonte intervalli ja jämedust (URL 8).

Vincent Woo on teinud JavaScripti abil samajoonte genereerimise programmi, mis toetub ruudustiku põhisele grid-ile ja kasutab canvas elementi. Andmeid hoitakse eraldi failides. Võimalus on muuta samajoonte intervalli (URL 9).

Ette arvutatud kaadritega animatsiooni võib näha näiteks EMHI (Eesti Meteoroloogia ja Hüdroloogia Instituut) Hirlam mudelites. Iga kaader on tegelikult PNG formaadis pilt ja nende vahetust juhitakse JavaScripti abil (joonis 17).



Joonis 17. Internetisirvija aadressiribalt on näha ,et EMHI HIRLAM mudeli animatsiooni kaader on ette arvutatud PNG formaadis pilt (URL 3).

Robert Roth tegi Flash-iga samajoonte visualiseerimise ja õpetamise vahendi. Tema töö on mõeldud pigem samajoone kaardi koostajale abiks valikute tegemisel. Saab proovida erinevaid interpoleerimise võtteid, värve jämedust ja palju muud mis mõjutab samajoonte kaardi väljanägemist. Kõik jooned on ette arvutatud (Roth. et al. 2006).

3. KUNSTLIKULT LOODUD PIDEVAD ANDMED

Selleks, et töö praktilises osas oleks võimalik samajooni genereerida on vaja kirjeldada mõned pidevate pindade loomise võimalused.

3.1. LINEAARSED VÄRVIGRADIENDID

Gradient on suuruse muutumise määr ühe pikkusühiku kohta. Värvigradient on arvutigraafikas sujuv üleminek ühelt värvilt teisele. Levinud on nii radiaalsed värvigradiendid, mis tekitakse ellipsi abil, kui ka lineaarsed gradiendid. Radiaalse gradiendi puhul on üks äärmusvärv ellipsi keskel ja teine selle välisservas. Vahepealsed värvid arvutatakse lineaarse interpolatsiooni teel. Lineaarse gradiendi puhul leitakse värvitoonid samuti lineaarselt interpoleerides kahe äärmuspunkti vahel (URL 13).

3.2. PERLIN MÜRA

Tegu on pseudojuhusliku (näib juhuslikuna) mürafunktsiooniga, mida kasutatakse ühtsete visuaalsete parameetritega juhuslike naturaalselt paistvate tekstuuride loomiseks. Perlin müra (*Perlin noise*) on oma nime saanud Ken Perlini järgi, kes sai selle leiutamise eest tehnilise saavutuse oskari. Algselt arendas Perlin oma müra funktsiooni protseduuriliste loomuliku välimusega tekstruuride loomiseks Holliwoodi tarbeks 1980ndate esimesel poolel. Tänapäeval kasutatakse tema loodud mürafunktsiooni väga laialdaselt nii filmitööstuses kui ka kõikjal mujal arvutigraafikas, kus on vaja luua protseduurilisi loomuliku välimusega tekstuure. Perlin näitas ka, kuidas kolmemõõtmelist müra saab kasutada kahemõõtmelise müra muutuste näitamiseks ajas, kui asendada üks mõõde ajaga. (URL 10).

3.2.1. KLASSIKALINE MÜRA

Klassikalise müra loomiseks luuakse võre (*grid*). Igasse võre punkti paigutatakse lineaarne juhusliku suunaga gradient. Et leida müra väärtus punktis P, tuleb võtta teda ümbritsevad võre punktid ja interpoleerida nende põhjal P väärtus. Võre punktide arv sõltub ruumi mõõtmetest valemi 2n järgi, kus n on ruumi mõõtmete arv. Näiteks kahemõõtmelises ruumis tuleb gradiendi väärtus punktis P leida 4 ümbritseva punkti pealt. Edasi tehakse punkti P asukohast lähtuvalt kaalutud interpolatsioon kasutades sobilikku S kõverat (nagu bilineaarne interpolatsioon) (Perlin 1985, URL 10).

3.2.2. SIMPLEKS MÜRA

Simpleks müra (*simplex noise*) abil on võimalik saavutada sama mis klassikalise müraga, aga kiiremini (Perlini katsetel umbes 10 protsenti kiirem). Mida rohkem mõõtmeid, seda kiirem peaks simpleks müra olema (võrreldes klassikalisega). Selle saavutamiseks tegi Perlin mõned üldistused, näiteks varem oli kahemõõtmelise ruumi müra leidmiseks ruut ja 3 mõõtmelise ruumi puhul kuup, siis nüüd kahemõõtmelises ruumis on tegu kolmnurgaga (simpleks) ja kolmemõõtmelises tetraeedriga. Veel tegi ta täienduse pinna väärtuste arvutamise viisi. Näiteks kahemõõtmelise võrdkülgsetest kolmnurkadest koosneva võre puhul mõjutavad, mingis võre elemendis oleva punkti P väärtust ainult teda ümbritsevad kolm punkti (punktide mõjuraadius on niimoodi planeeritud), selline lähenemine aitas vähendada korrutustehete arvu iga punkti leidmisel (Perlin 2002).

4. INTERNETISIRVIJAS TÖÖTAVA SAMAJOONTE GENERAATORI EHITAMINE

Eelnevate peatükkidega kirjeldasin erinevaid teiste autorite poolt loodud vahendeid. Käesolevas töö osas valin välja sobivaima lahenduskäigu samajoonte generaatori ehitamiseks internetisirvijas. Analüüsi tulemusel peab leidma sobiva tehnoloogia ja algoritmid töö peamise eesmärgi lahendamiseks, milleks on samajoonte animatsioonikiirusel interpoleerimine ja kuvamine brauseris.

4.1. SÜSTEEMI ARHITEKTUURI VALIKUD

4.1.1. SÜSTEEMI EESMÄRGID JA VAJADUSED

Selleks, et teha arhitektuuriliselt parimad valikud, peab kaardistama täpsemad planeeritava süsteemi eesmärgid ja vajadused.

1) Töö tulemust peab olema võimalikult kerge kasutada erinevate samajoonte genereermise ja animeerimisega seotud rakenduste arendamiseks. Põhirõhk on animatsioonil ja reaalajalise pildi genereerimisel internetisirvijates.

2) Samajoonte genereerimine võiks toimuda võimalikult kiiresti. Tuleb leida võimalikult kiired algoritmid, et oleks võimalik saavutada lennult animeerimist.

3) Samajoonte generaator peab hakkama saama andmepunktide kadumise või juurde lisandumisega animeeritava perioodi jooksul. Vajadus tuleneb otseselt reaalajaliste andmete kasutamise võimalikkusest.

4) Süsteem peab toimima võimalikult paljudes masinates. Eelistatud on lahendused, mis töötavad otse internetisirvijas, ilma lisavahendeid paigaldamata.

5) Süsteem ei tohi enda koduks olevat serverit koormata. Võimalikult palju arvutusi võiks teha kliendi (vaataja) arvutis.

6) Samajoonte väliseid visuaalseid parameetreid (paksus, värvus) peab saama animatsiooni käigus muuta.

7) Töö tulemusel tekkinud kood võiks olla korralikult struktureeritud ja kommenteeritud, et selle erinevaid osasid saaks võimalikult kerge vaevaga välja vahetada ja edasi arendada.

8) Töös tekkinud kood võiks olla võimalikult sõltumatu teistest koodikogudest.

9) Töö tulemust võiks olla hea integreerida teistesse rakendustesse.

4.1.2. KLIENDIPOOLSED VS. SERVERI POOLSED ARVUTUSED

Otsustasin kasutada võimalikult palju kliendi arvuti poolseid arvutusi. See tagaks sõltumatuse serverist, sest juhul kui teha rakendus, mida vaadatakse väga paljude inimeste poolt korraga, ei teki serveril raskusi kõigi vaatajate vajaduste rahuldamisega.

Lisaks räägib kliendipoolsete arvutuste kasuks animatsiooni kaadrite lennult arvutamise vajadus, sest serverist tulemuste küsimine ning saatmine ja nende õigete asjadega sidumine võtaks raskesti ennustatava lisaaja (sõltub ka andmemahtudest ja ühenduse kiirusest).

4.1.3. ARENDUSVAHENDITE VALIK

Antud alapeatüki eesmärk on põhjendada HTML5 kasutust. Reaalselt piirasid autori oskused valiku kahe suure tehnoloogia vahel Flash/Flex + ActionScript 3 (edaspidi Flash) ja HTML + JavaScript (edaspidi HTML5). Alternatiivina saaks veel ilmselt kasutada Java-t, mis on üsna sarnane ActionScript 3-le ja millega on autoril olnud põgusaid kokkupuuteid. Viimased aastad on autor eeltoodud variantidest kõige rohkem tegelenud HTML5 tehnoloogiaga ja tunneb ennast kõige efektiivsemalt just selle abil arendamisel. Lähtudes peatükis 2.2 toodud aspektidest leian, et HTML5 on antud probleemi lahendamiseks kõige perspektiivikam valik ja töö saab realiseeritud selle abil.

SVG ja canvas elemendi võrdluses valin canvas elemendi (võrdlus peatükis 2.2.3.2.). Kaks põhilist argumenti mis antud probleemi käsitlemisel välistavad SVG kasutamise:

1)DOM struktuuri muutmine ehk siis elementide eemladamine ja uute lisamine igas kaadris oleks aeglasem, kui canvasi uuendamine, sest objekte on suhteliselt palju.

2)Samajoonte animatsiooni kaadrid tuleb nagunii eraldi välja arvutada, seega konkreetsete joonte seoseid ei ole vaja mälus hoida.

Valik on ka kooskõlas argumendiga, et keerulised reaal-ajalised matemaatilised animatsioonid tuleks teha canvas elemendil (joonis 16).

4.1.4. INTERPOLATSIOONI ALUSE VALIK

Ristkülikutel põhineva võre puhul peab iga kord peale andmepunktide muutumist tegema interpolatsiooni, et saaks need üle kanda võre sõlmedele ja alles siis saab teha samajoonte läbimise kohtade leidmise interpolatsiooni. Triangulatsiooni puhul seda sammu ei ole ja seega saab kohe teostada samajoonte läbimise kohtade leidmise interpolatsiooni. Isojoonte ühendamine võiks toimuda mõlema lähenemise puhul enam-vähem sama kaua.

Punktide lisandumine ja eemaldumine mõjutab rohkem triangulatsiooni kiirust, sest see mõjutab otseselt alusstruktuuri. Ristkülikutel põhineva võre puhul sellest olulist lisaajakulu ei tuleks. Sobilike algoritmide puhul on võimalik punktide lisamine triangulatsiooni teha piisavalt kiireks, seetõttu ei ole antud argument nii oluline.

Ristkülikutel põhineva võre puhul oleks eeliseks laiem valik interpolatsioonimeetodeid, millest tulenevalt võiks andmeid tundes saavutada paremaid tulemusi. Siiski peaks ka arvestama interpolatsioonimeetodite erinevat kiirust. Punktide valik ja interpolatsioon peavad olema piisavalt kiired ja juba see kaotaks võimaluse kasutada paljusid võimalikke meetodeid. Triangulatsiooni puhul toimub esmane interpolatsiooni kriteeriumi valik (punktid mille vahel interpoleeritakse) juba selle loomisega. Triangulatsiooni loomine toimub valdavalt animatsiooni väliselt, välja arvatud lisatavad ja eemaldatavad punktid. Triangulatsiooni olemasolul saab juba teostada väga kiire lineaarse interpolatsiooni (igas animatsiooni kaadris).

Vektori põhiste maastikumudelite arvutis kujutamiseks kasutakse TIN (*Triangular Irregular Network*) mudelit (Cauvin et al. 2010b:330). Lisaboonusena saab triangulatsioonilt kerge vaevaga minna kolmemõõtmelisele maastiku mudelile, kui tõsta triangulatsioonipunktid atribuutide järgi kolmandasse mõõtmesse. Rakenduste arendamisel on antud potensiaali omamine positiivne.

Delaunay triangulatsiooni põhine lähenemine interpolatsiooniks sobib eriti hästi ebaühtlase tihedusega andmepunktide puhul, juhtudel kui mõnest piirkonnast on väga palju punkte ja mõnest teisest suhteliselt vähe. Kohad, kus on rohkem andmeid, saavad mõistagi interpoleeritud täpsemini (URL 2).

Leian, et käesoleva reaalajalise ülesande lahendamiseks sobib Delaunay triangulatsiooni põhine võre paremini kui ristkülikute põhine. Minu valik on siinkohal puhtalt analüüsi tulemus. Tulevikus tuleks sobivate algoritmide korral kindlasti proovida ka teist lähenemist (praktikas), antud töö maht seda ei hõlma. Valiku kõige tähtsamaks kriteeriumiks on animatsioonist lähtuvalt probleemi lahendamise kiirus.

4.2. SÜSTEEMI ALGORITMIDE VALIKUD JA EHITAMINE

4.2.1. DELAUNAY TRIANGULATSIOONI ALGORITMI VALIK

Eelnevalt sai põgusalt kirjeldatud erinevaid Delaunay triangulatsiooni algoritme (peatükk 1.2). Kõige kiiremaks peetakse Dwyeri algoritmi, kuid Fortune pühkimisjoone ning Su ja Drysdale poolt tehtud juurde lisav algoritm ei jää palju alla (Su, Drysdale 1997). Kõige mõistlikum on valik teha Su ja Drysdale poolt tehtud töö põhjal, sest uuemat usaldusväärset võrdlust ei ole ning ära toodud kolm algoritmi saavutasid kõik väga häid tulemusi.

Dwyeri jaga ja valluta algoritm oli küll kõige kiirem, kuid selle ja ka Fortune algoritmide miinusteks on dünaamilisuse puudumine. Uue punkti lisamisel peaks uuesti arvutama kogu triangulatsiooni. Su ja Drysdale-i juurde lisav algoritm ei jäänud Dwyeri omale üleliia palju alla ja samal ajal saab seda lennult muuta, ilma et peaks kogu suurt triangulatsiooni uuesti arvutama. Lisaks peetakse selle implementeerimist ja sellest aru saamist lihtsamaks, igasugune lihtsustus tuleb autori meelest kasuks. Samuti näen antud algoritmis potensiaali topoloogia ehitamiseks otse triangulatsiooni loomisel.

Kokkuvõttes valin antud probleemi raames Su ja Drysdale poolt kirjeldatud kiire juurde lisava (*incremental*) algoritmi. Dwyeri algoritmi poole pöördumine oleks ilmselt potentsiaalikas liikuvate punktide puhul, sest siis tuleks igas animatsiooni kaadris kogu triangulatsioon nagunii uuesti arvutada.

4.2.2. ANDMESTRUKTUURI VALIK EHK TOPOLOOGIA EHITAMINE JA CCW TESTIDE VÄHENDAMINE

Otsustasin luua enda andmestruktuuri, mis ei pruugi küll olla arvuti mälu kasutuse osas kõige optimaalsem, aga annab programmeerijale rohkem vabadust. Su ja Drysdale tõdesid oma erinevaid algoritme võrdlevas töös, et paljud algoritmid oleks ilmselt kiiremad, kui ei kasutataks täielikult Guibas ja Stolfi poolt kirjeldatud "*quad-edge*" andmestruktuuri, sest selle haldamine on kulukas (Su, Drysdale 1997).

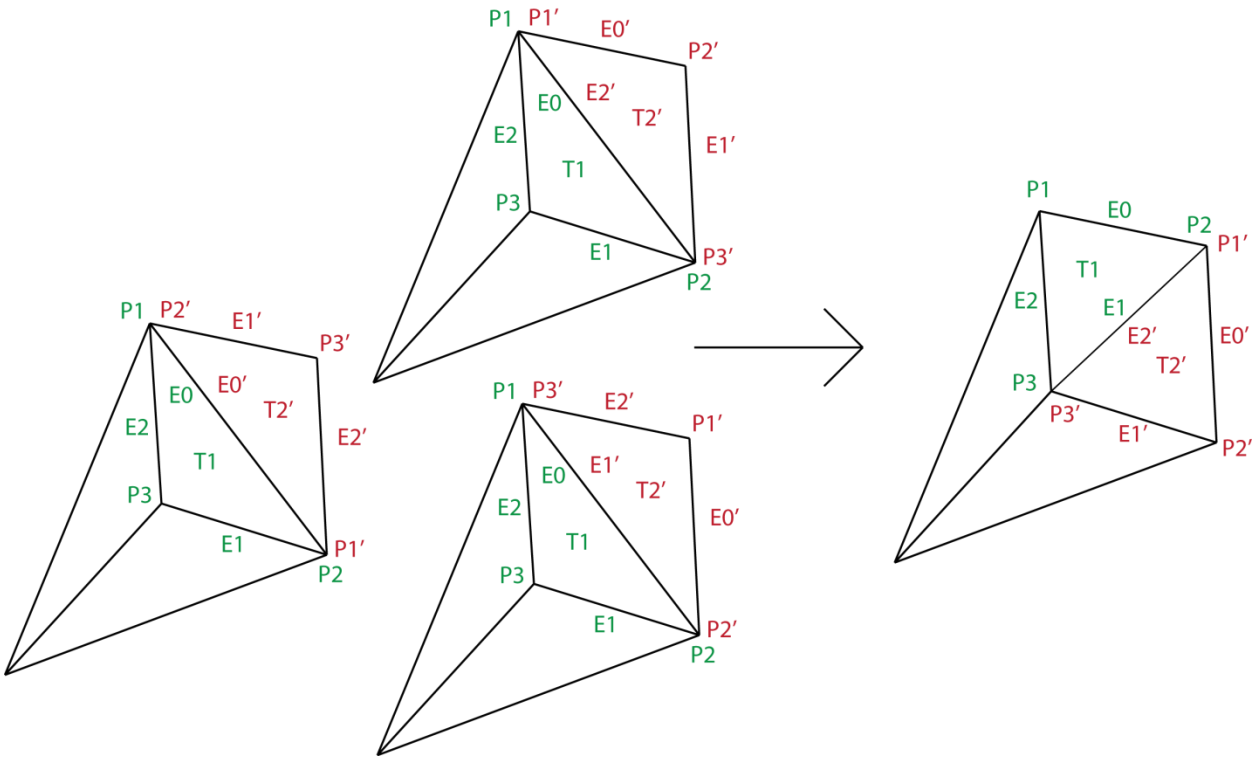
Ehituskividena kasutasin punkte, servasid ja kolmnurki. Kõik ehituskivid paigutasin eraldi jadadesse. Järgnevalt kirjeldatud viidete all mõtlen viiteid asukohtadele jadades. Kolmnurgad ja servad omavad viiteid neid moodustavatele punktidele. Kolmnurgad omavad ka viiteid neid moodustavatele servadele. Servad omavad ka viiteid neist mõlemale poole jäävatele kolmnurkadele. Sisuliselt kogu loodav süsteem rajaneb viidetele algsetes jadades ja kõik läbiviidavad operatsioonid vahetavad neid viiteid.

Selline lähenemine on võimalikult programmeerija sõbralik, sest reaalselt teavad kõik ehituskivid kõike, mida neile saaks teada anda. Lisaks tagab viidetega opereerimine taolise käitumise juures minimaalse arvuti mälu hõivamise: algsed andmepunktid võtavad nagunii oma ruumi, kõik loodavad kolmnurgad ja servad on sisuliselt viited algsetele punktidele.

Selleks, et triangulatsiooni oleks lihtsam uuendada ja ka samajooni oleks lihtsam ehitada, oleks hea, kui triangulatsiooni elemendid teaksid oma naabreid. Selle tagamiseks tuleb jälgida, et uute servade lisandumisel ning vanade servade kustutamisel või pööramisel salvestataks neile õigete naaberkolmnurkade viited.

Naabrussuhete haldamiseks töötasin välja lahenduse, mis aitab ühtlasi vältida CCW testi (vt. peatükk 1.2.2.2) tegemist vahetult enne serva legaalsuse kontrollimist. Serva legaalsuse kontrolliks on vaja teada kolmnurga orientatsiooni (punktide järjekorda) ja tavaliselt tehakse selle jaoks CCW test või siis saadakse orientatsioon andmestruktuurist. Pakun välja lahenduse, kus kõikide kolmnurkade orientatsioonid salvestatakse nii, et neile saab kohe ümberringjoone testi rakendada (ilma CCW testi kasutamata). Selleks, et lahendus töötaks, tuleb käivitada protsessid, mis omistavad osalevatele kolmnurkadele ja servadele uued viited (jooni 18).

Kolmnurk, millesse punkt lisati, muutub kolmeks kolmnurgaks, millest igaüks salvestatakse kohe vajalikus orientatsioonis. Algse kolmnurga naabrid on nüüd igaüks kolme uue kolmnurga naabrid. Naaberkolmnurk võib esialgse suhtes olla orienteeritud kolme moodi. Orientatsiooni saab paika panna, kui võrrelda P1 punkti viite väärtust naaber kolmnurga P1', P2', P3' viidete väärtustega (joonis 18). Kui orientatsioon on teada, siis saab juba täpselt määratleda, mis viiteid tuleb vahetada.



Joonis 18. Serva pööramisel serva ja kolmnurga siseste viidete vahetumine, mis tagab kolmnurga punktide kindla järjekorra vahetult enne ümberringjoone testi. P- punkt, E- serv ja T- kolmnurk.

Esimese punkti lisamiseks koostasin suure kolmnurga (nagu teised autorid seda teevad (de Berg et al. 2008, Guibas, Stolfi 1985)), mis on nii suur, et tema sisse pandavad pisikesed kolmnurgad oma ümberringjoontega selle punktideni ei ulatu. Kõik suure kolmnurgaga seotud interpolatsioonis mitte kasutatavad kolmnurgad tähistasin servade viidetes negatiivse indeksiga, see aitas aru saada kus triangulatsiooni kasulik osa lõppeb.

4.2.3. SAMAJOONTE ÜHENDAMINE

Samajoonte ühendamise algoritmi tegin vastavalt olemasolevale struktuurile ja selle võimalustele. Algoritmi tulemusel saab kolmnurga servadele interpoleeritud samade väärtustega punktid ühendatud samajoonteks. Koodis hoidsin samajooni eraldi objektidena, millede atribuutideks olid atribuudi väärtus (samajoone väärtus) ja teda moodustavate punktide jada . Lisaks märkisin iga samajoone puhul ära kas tegu on lahtise või kinnise samajoonega, et hiljem oleks neid lihtsam renderdada.

Peale samajoonte ühendamist proovisin silumiseks *cubic* ja *quadratic* Bezier' kõveraid, kasutades peatükis 1.3 ära toodud algoritme. Kui silumisest üldse loobuda, siis saaks samajooned näiliselt ühendada kolmnurga kaupa, igas kolmnurgas ühendad vastavad atribuutide paarid, aga naabritega ei ühenda ja see oleks oluliselt kiirem, sest lisaks kiiremale "ühendamisele" jääb ära ka silumisalgoritmi poolt tehtavad arvutused. Kui andmeid on piisavalt tihedalt, siis võib selline variant mõningatel juhtudel paremini sobida.

4.2.3.1. Kolmnurga tipu probleem

Samajoonte ühendamise aluseks on kolmnurgad. Kõigepealt interpoleeritakse kolmnurga servadele punktid, kust isojooned neid läbivad. Kui interpoleeritud punkt langeb kokku kolmnurga tipuga, siis on sama tipp ka paljude teiste kolmnurkade tipp ja ka nende tippudesse läheb sama punkt. Sellised koha puhul on väga keeruline kiiresti otsustada, kuidas samajoon peaks antud kohta läbima ja head lahendust oli raske leida. Kuna samajoonte ühendamine peab olema väga kiire, võtsin kasutusele lihtsaima ja kiireima lahenduse, milleks on väga väikese arvu juurde liitmine antud punkti atribuudile enne interpolatsiooni. Selline teguviis lükkab jooned tipust ära servadele. Kui liidetud arv on piisavalt väike, siis visuaalselt on kõik korras ja joon läbib ikkagi punkti. Selline lähenemine kõrvaldab samamoodi ka teise võimaliku keerulise olukorra, kus kõik kolm sama kolmnurga tippu võivad saada sama atribuudi väärtuse.

4.2.3.2. Tavaline ühendamise algoritm

Algoritm on ehitatud nii, et punktide atribuutandmete muutumisel tuleb interpoleerida triangulatsioonis olevate servadele samajoonte läbimise kohad. Selleks tuleb kõik servad, kus toimus andmemuutus, läbi käia ja teha atribuutide vahel lineaarne interpolatsioon. Interpolatsiooni tulemusel tekkinud punktid salvestasin eraldi jadasse ja

servadele lisasin viited sinna jadasse. Samuti iga interpoleeritud punkt omab viidet oma servale servade jadas. Servad omavad viiteid kolmnurkadele, mille osadeks nad on ja kolmnurgad omakorda omavad viiteid oma servadele.

Joonte ühendamiseks võtsin kõigepealt interpoleeritud punktide jada esimese punkti serva ja vastavalt esimese punkti atribuudile liikusin naabrussuhete abil läbi triangulatsiooni servadesse, kus on sama atribuut. Iga läbitud interpoleeritud punkti eemaldasin üldisest interpoleeritud punktide jadast, et ei toimuks kordust. Samal ajal lisasin punkti samajoonele. Iga samajoone puhul tuli meelde jätta esimese serva indeks. Kui lõpuks jõuti algservani tagasi, siis on tegemist kinnise tsirkulaarse samajoonega. Kui jõuti triangulatsiooni välisservani, siis pöörduti tagasi esimese serva juurde ja liiguti teises suunas, kuni jõuti ka sealt triangulatsiooni välisservani. Mõlema suuna punktide jadad tuleb ühendada, kuid üks tuleb keerata tagurpidi järjestusse, et hiljem oleks samajoone punktid õiges järjekorras. Edasi võetakse üldisest interpoleeritud punktide jadast järgmine alles jäänud punkt ja ühendatakse selle atribuudi samajoon ja nii edasi kuni üldine interpoleeritud punktide jada saab tühjaks.

4.2.4. KATSETAMINE GRADIENTIDEGA JA EMHI REAALAJALISTE ANDMETEGA

Programmi tekitatud samajoonte hindamiseks on vaja mingisuguseid referentsandmeid. Otsustasin need ise luua. Selleks kasutasin HTML5 töövahendeid (ka JQuery). Kõigepealt tegin lihtsamaid must-valgeid gradient pilte. Reaalse muutuva maastiku emuleerimiseks kasutasin kolmedimensionaalset Perlini simpleks müra, mille ühe dimensiooni asendasin ajaga. Mõlemal juhul toimisin samamoodi, iga piksel saab väärtuse vahemikus 0-255. 0 tähendab, et piksel on must ja 255, et valge. Genereeritud piltidelt eraldasin juhuslikult valitud asukohtadest piksli väärtused (0-255). Nende väärtuste põhjal genereerisin loodud koodi abil samajooned. Kindlasti saaks saavutada paremaid tulemusi, kui spetsiaalselt valida maastikul punkte, kust proove võtta.

Perlin müra realisatsiooni JavaScriptis leidsin internetist (URL 14). Seda oli mõistagi vaja modifitseerida. Testide läbiviimiseks tegin lahenduse, milles canvas element haaras endale maksimaalse akna suuruse. Selline teguviis aitas paremini mõista kuidas mõjutab canvas elemendi ulatus samajoonte genereerimist, sest erineva suurusega canvasi

saamiseks piisab akna suuruse muutmisest. Kogu canvas elemendi täitsin Perlin müraga.

Animatsiooni testimiseks asendasin kolmemõõtmelise Perlin müra ühe mõõtmetest ajaga. Ettearvutatud kaadrid (pildilt juhuslikest kohtadest võetud punktid) pandi JavaScripti abil HMTL lehe struktuuri nii, et iga kaader oleks eristatav. Animatsiooni käivitamisel ühendati igas kaadris x ja y koordinaatidega punktidele atribuudid puhtalt jada järjekorra alusel. Ainsad üleliigsed operatsioonid on HTML lehest võetud tekstist jada tegemine, selle numbriteks tegemine ja vastavalt soovitud samajoonte tiheduse saavutamiseks määratud numbriga läbi jagamine.

Lisaks proovisin samajooni interpoleerida EMHI poolt pakutavate, XML formaadis reaalajaliste numbrilisel kujul olevate pidevate nähtuste andmetega (õhu temperatuur, õhurõhk), mis uuenevad iga tund (joonis 19) (URL 15). Ilmajaamade koordinaadid olid samuti kättesaadavad EMHI kodulehelt (URL 16). Nende põhjal koostasin GeoJSON formaadis faili (joonis 20), millesse sai lisatud iga ilmajaama "*wmocode*" väli, et oleks võimalik eri allikate sidumine. Kogu esialgne samajoonte koodi arendamine toimus nende andemetega, iga tund automaatselt muutuv andmestik aitas vigu leida.

```
▼<observations timestamp="1368964737">
 ▼<station>
    <name>Tallinn (Harku)</name>
    <wmocode>26038</wmocode>
    <phenomenon>Variable clouds</phenomenon>
    <visibility>23.0</visibility>
    <precipitations>0</precipitations>
    <airpressure>1013.1</airpressure>
    <relativehumidity>67</relativehumidity>
    <airtemperature>22.7</airtemperature>
    <winddirection>359</winddirection>
    <windspeed>2.5</windspeed>
    <windspeedmax>4.3</windspeedmax>
  </station>
 ▶<station>...</station>
```

Joonis 19. Iga tund uuenev EMHI andmeteenus on XML formaadis (URL 15).

Joonis 20. Prooviks koostatud Eesti ilmajaamade GeoJSON formaadis faili struktuuri näidis. Juhin tähelepanu, et antud fail on puhtalt katsetamiseks, struktuur ei ole läbi mõeldud.

Programmi testimiseks kasutati Apache PHP serverit (versioon pole oluline, sest arvutused toimuvad kõik internetisirvijas) ja Google Chrome internetisirvijat versioon 26.0.1410.64 m. Tegu on suhteliselt vana arvutiga, 2008 kokku pandud, hiljem mõned komponendid vahetatud.

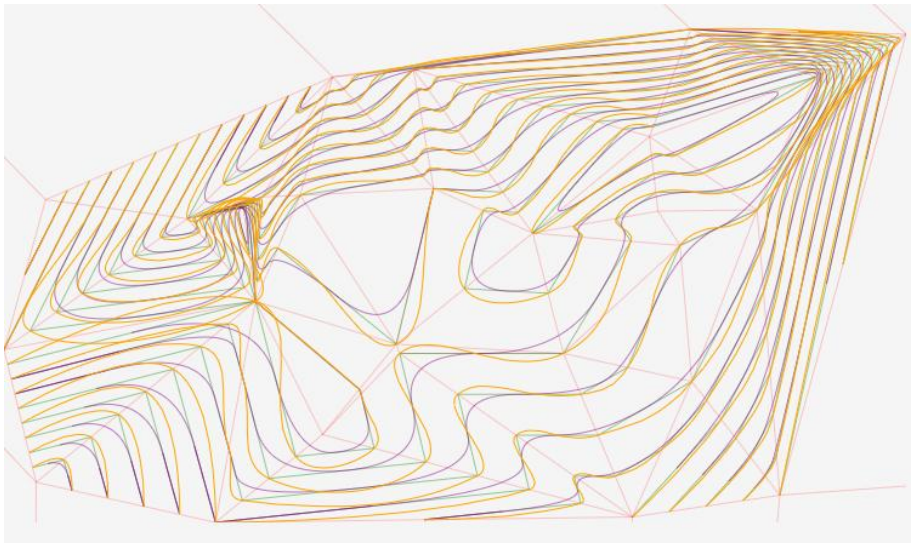
Arvuti tähtsamad parameetrid:

protsessor	Intel Core 2 Quad Q9300 2,5GHz
emaplaat	Gygabyte G41M- Combo
graafikakaart	GeForce GTX 550 Ti
RAM	DDR3 2*4GB 1333 MHz (*over clocked*)
OS	Windows 7 -64bit

5. TULEMUSED JA ARUTELU

Antud töös realiseeriti juurde lisavat (*incremental*) tüüpi Delaunay triangulatsiooni kood. Lisaks loodi samajoonte ühendamise kood. Veel realiseeriti samajoonte silumise kood nii *quadratic* kui ka *cubic* Bezier' kõveratega. Selgus, et *Cubic* Bezier' kõverad võivad omavahel sõlme minna, *quadratic* Bezier' kõveratega seda ei juhtu (joonis 21). *Cubic* Bezier' kõverad läbivad interpoleeritud andmepunkte, aga *quadratic* Bezier' kõverad mitte. Viimast täheldas ka Zachary Forest Johnson (URL 12). Tavalise kaardi puhul on see ilmselt probleem, kui samajoon interpoleeritud punktist niivõrd kaugele jääb. Animeeritud kaardi puhul, kui tegu on vaid ühe kaadriga ja nähtuse üldine iseloomulik muutumine antakse edasi, ei oma selline kõrvalekalle piisavalt tähtsust. Kasutatud *quadratic* Bezier' kõverate arvutamise kiirus on parem, kui *cubic* Bezier' kõveratel. Samuti sobib nendega silumise meetod hästi triangulatsiooni iseloomuga, mis ei lase neil sõlme minna.

Cubic kõverate sõlme minemist saaks vähendada, kui kasutada koefitsenti, mille abil saaks neid teravamaks teha. Mõningatel juhtudel aitaks samajoonte tiheduse muutmine. Kõike eelnevat arvesse võttes leiti, et praegusel kujul *cubic* Bezier' kõverad ei sobi.



Joonis 21. Samajoonte silumine Eesti ilmajaamade näitel. Oranž - *cubic* Bezier', lilla- *quadratic* Bezier', roheline- silumata joon.

Gradiendile paigutati punktid juhuslikult. Punktide atribuutide alusel genereeritud samajooned siluti *quadratic* Bezier' kõveratega. Mida rohkem andmepunkte, seda täpsemini iseloomustavad interpoleeritud samajooned gradient maastikku. Kui võrrelda 10 ja 100 punktiga proove, saab aru, et ka punktide paiknemine on oluline (joonised 22-27). Servaaladele tekivad pikad kiitsakad kolmnurgad, mis ei sobi eriti hästi interpolatsiooniks.

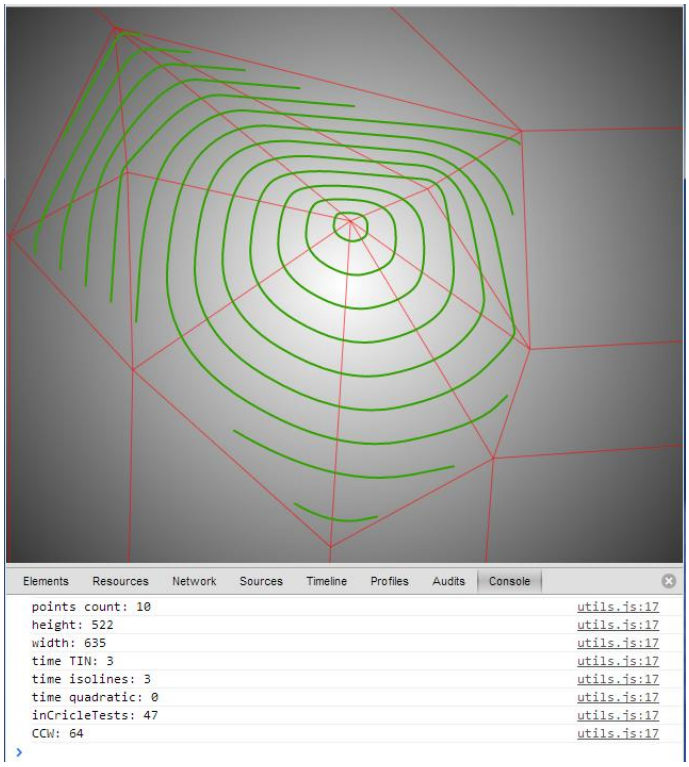
Kolmnurga tipuprobleemi lahendamine väikese arvu juurde liitmisega tekitab väiksema arvu punktide korral teravaid nurkasid (joonis 24 keskel), seda saaks leevendada kui kontrollida joonel liiga lähestikku asuvaid punkte ja asendada need ühe punktiga.

Radiaalgradientide katse näitab, et CCW ja ümberringjoone test kasvavad samas rütmis punktide arvuga (tabel 1). 1000 punkti peal on toimunud aeglustumine samajoonte ühendamise, silumise ja Delaunay konstrueerimisega. Tuleb meeles pidada, et antud mõõtmised ei ole paljude katsete keskmised, see võis olla ühekordne anomaalia. Silumisele kulub ka kõige suurema arvu punktide korral üllatavalt vähe aega.

CCW testide arvu ja punktide arvu suhe on kuskil 7,4 (tabel 1). Antud tulemus on väga huvitav, sest Su ja Drysdale algoritm, mis on ka käesoleva töö aluseks, teeb kuskil 19 CCW testi iga lisatud punkti kohta (ca 10 kolmnurga otsimisel ja 9 triangulatsiooni uuendamisel) (Su, Drysdale 1997). Ümberringjoone teste (iga lisatud punkti kohta) tegi Su ja Drysdale algoritm veidi alla 9, see klapib ka antud töö tulemustega (tabel 1).

Tabel 1. Radiaalgradiendilt samajoonte konstrueerimine 10-1000000 punktiga.

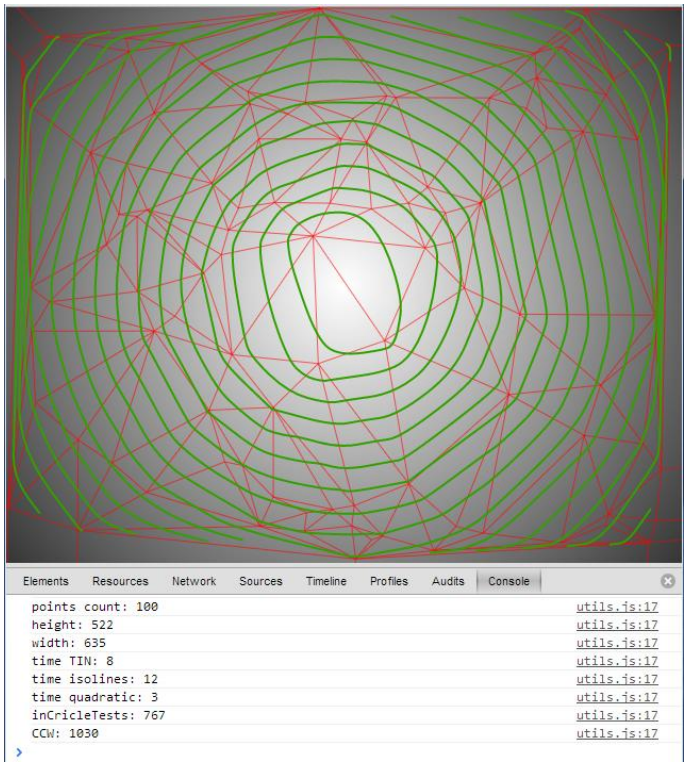
	Arv		Konstrueerimise aeg ms		
Punktid	Ümberringjoone test	CCW test	Samajooned	Silumine	Delaunay
10	47	64	3	0	3
100	767	1030	12	3	8
1000	8924	7957	541	47	495
10000	89488	74971	144	7	144
100000	899632	744162	560	15	1317
1000000	8993712	7424055	5949	63	17991



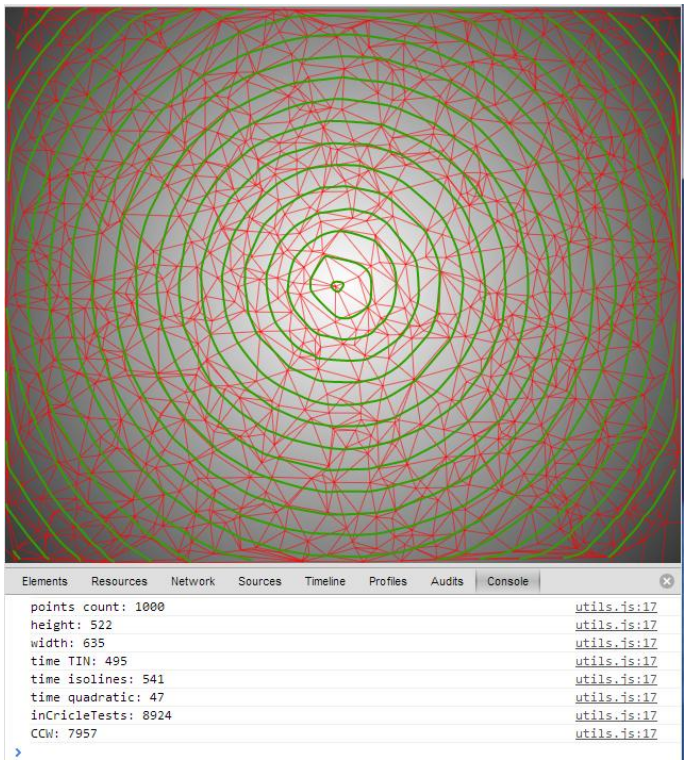
Joonis 22. Samajooned arvutatud 10 punkti põhjal, tasutaks radiaalgradient.

Peab tõdema, et peatükis 4.2.2 kirjeldatud algoritmi ja kasutusel oleva andmestruktuuriga õnnestus CCW testide arvu vähendada (võrreldes Su ja Drysdale orginaal algoritmiga (Su, Drysdale 1997)). Võib oletada, et vahe tuli *quad-edge* andmestruktuurist loobimise tõttu. Varasemalt on Shewchuk saanud oluliselt paremaid tulemusi kasutades tavalist kolmnurkadel põhinevat struktuuri (Shewchuk 1996) Täpsemate võrdluste jaoks peaks proovima kõiki algoritme võrdsetes tingimustes (samas keeles, samadel andmetel ja samal arvutil).

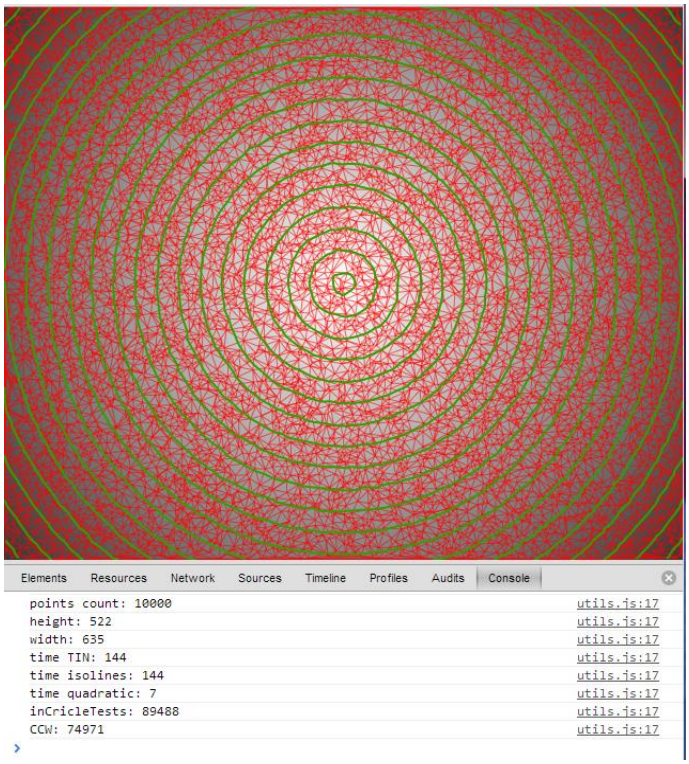
Testid staatiliste kaadrite arvutamisega (joonised 22-27) andsid lootust, et animatsioon on võimalik (tabel 1). Nendest võib järeldada, et arvutis millel teste jooksutati, tasuks animatsiooni üritada alla 10000 punktiga. 1000 punkti peal oli test küll väga aeglane, kuid ainuüksi silumisele kulunud aeg näitab, et midagi oli väga valesti, kas siis arvutiga või juhtus midagi koodis.



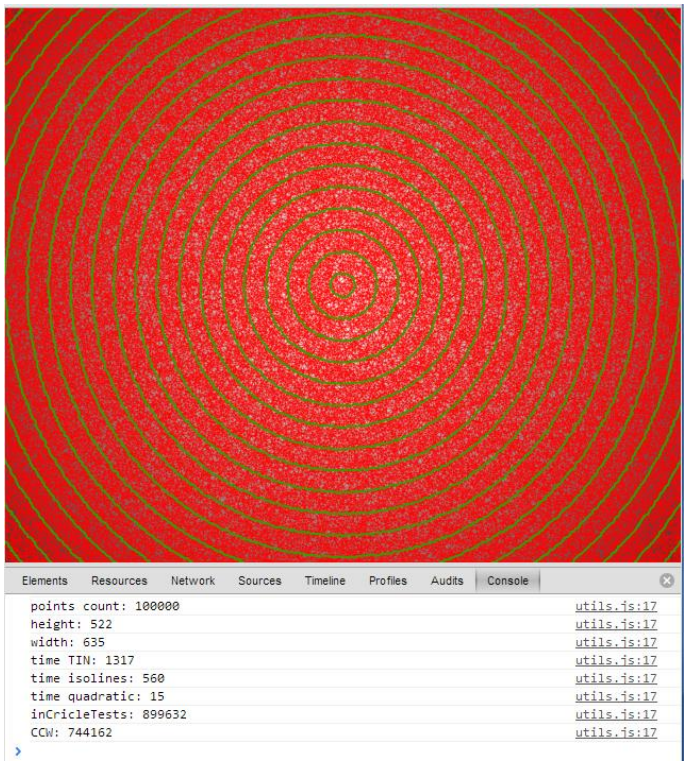
Joonis 23. Samajooned arvutatud 100 punkti põhjal, tasutaks radiaalgradient.



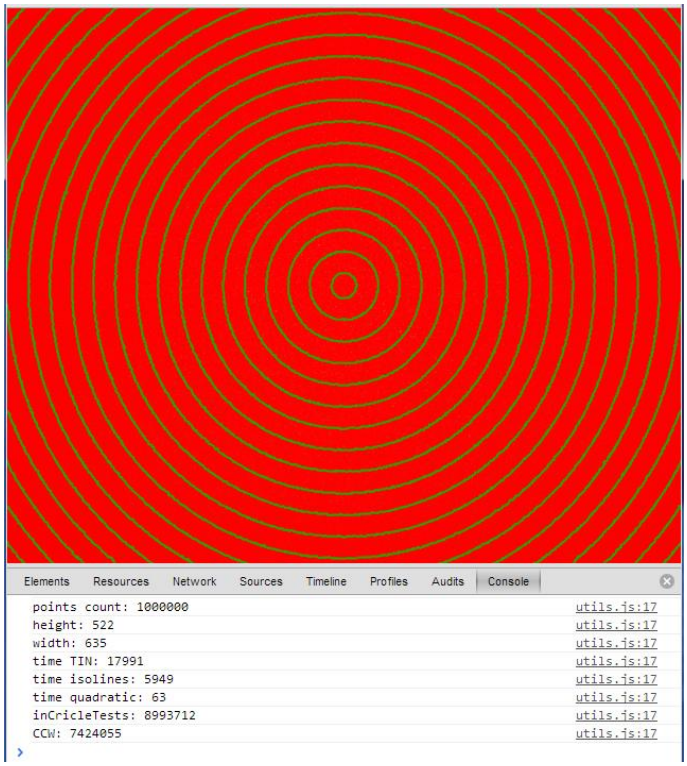
Joonis 24. Samajooned arvutatud 1000 punkti põhjal, tasutaks radiaalgradient.



Joonis 25. Samajooned arvutatud 10000 punkti põhjal, tasutaks radiaalgradient.



Joonis 26. Samajooned arvutatud 100000 punkti põhjal, tasutaks radiaalgradient.

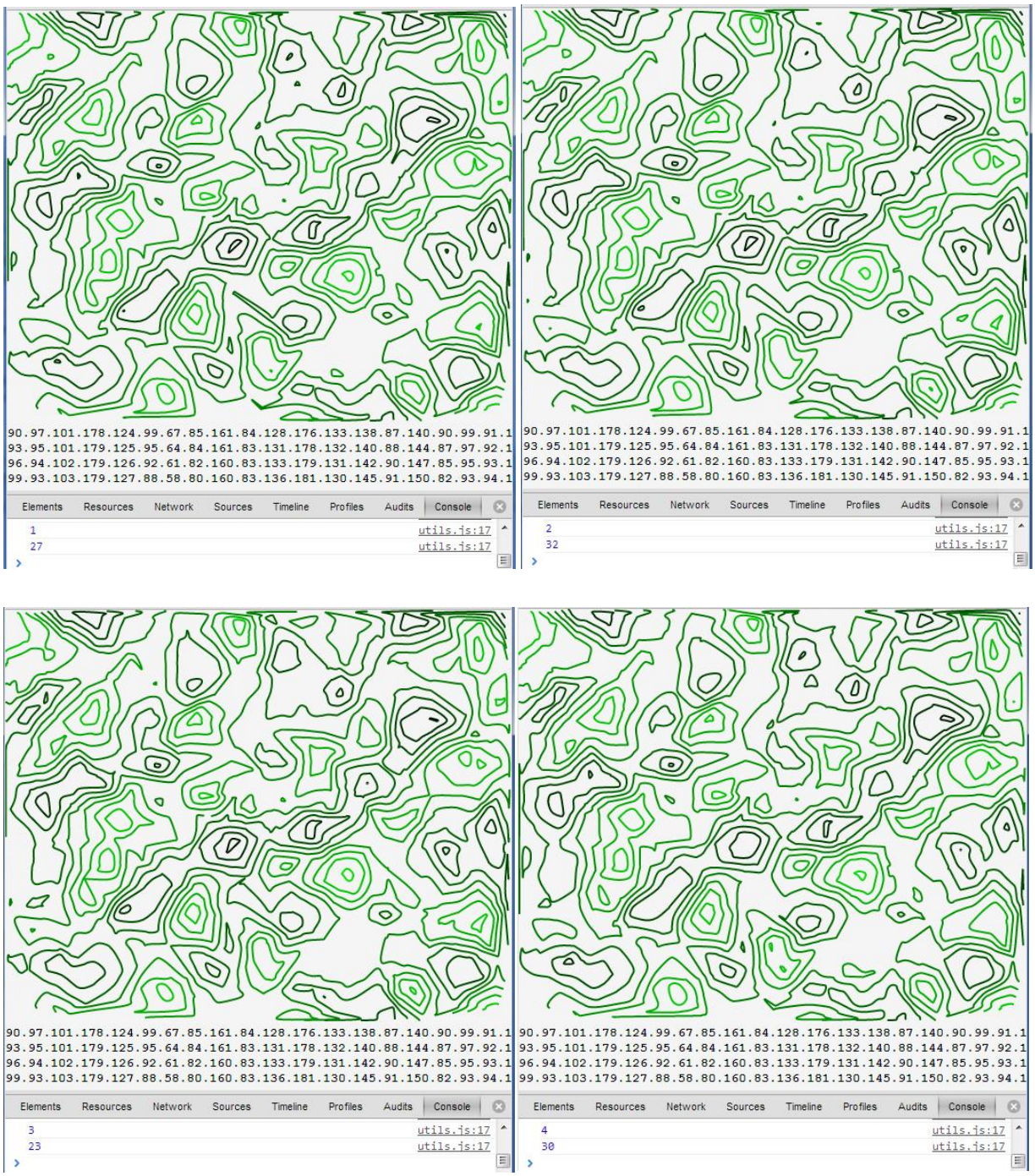


Joonis 27. Samajooned arvutatud 1000000 punkti põhjal, tasutaks radiaalgradient.

Eesti ilmajaamade reaalajaliste andmetega samajoonte koodi arendades täheldati, et andmed tulid üle võrgu liiga aeglaselt. Reaalajaliste andmete kuvamine on võimalik, kuid animeerimiseks võivad üle võrgu võetavad andmed liiga aeglaseks jääda.

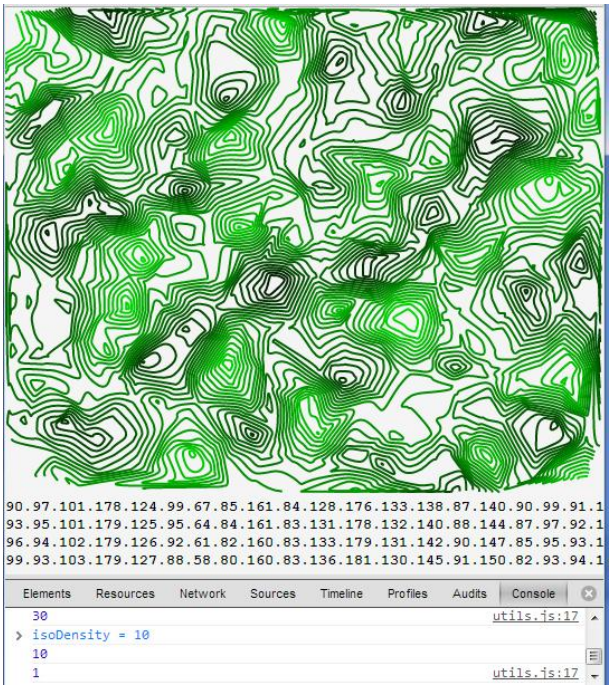
Perlin simpleks müra abil testitud samajoonte animatsioonid näitasid, et päris vigadevaba veel samajoonte kood ei ole, mõnikord võib märgata kuidas mõni samajoon ei ole ühendatud (joonis 28. kaader 1 keskel tumedate joonte kohal). Animatsiooni sujuvus sõltub kaadri arvutusmahukusest. Näidises (joonis 28) kulus iga kaadri arvutamiseks 23-30 millisekundit. Kui lähtuda 5 kaadrit sekundis nõudest, siis võib igat kaadrit arvutada 200 ms. 24 kaadrit sekundis (filmitööstus) saavutaks, kui iga arvutus kestaks ca 42 ms. Seega võib samajoonte animatsiooni loomise lugeda õnnestunuks.

Kasutajate kaasamine on kaasaegsete kaardianimatsioonide puhul väga tähtis. Antud lähenemine jätab arendajale väga head võimalused kasutaja kaasamiseks. Animatsiooni juhtimise saab anda täielikult kasutaja kätte.



Joonis 28. Nelja kaadriga samajoonte animatsioon. Vasakul üleval kaader 1, paremal üleval kaader 2, vasakul all kaader 3 ja paremal all kaader 4. Animatsiooni andmed on näha samajoonte all. Kaadrite arvutamiseks kulunud ajad vastavalt 27, 32, 23, 30 ms. Ala mõõtmed laius 535 pikslit ja kõrgus 508 pikslit. Punkte 1000.

Saab ehitada lahendusi, kus kasutajal on võimalik vaadata nii tervikanimatsiooni, kasutades tavapäraseid (*play*, *pause*, *stop)* kontrolle kui ka üksikuid kaadreid, mille võiks ühendada näiteks liuguri (*slider)* külge, mis näitaks animatsiooni käiku. Samuti saaks kasutaja kontrollida, mis suunas animatsioon jookseks. Lisaks saab ehitada lahendusi, kus kasutaja saab olla ise animatsiooni autor kujundades samajoonte värvi, tihedust (joonis 29), jämedust ja ka silumismeetodeid. Saaks ka ehitada lahenduse, kus vaataja kasutab enda andmeid (kui need on sobilikul kujul). Seda kõike võimaldab andmete hoidmine rakendusest eraldi ja kaadrite lennult arvutamine.



Joonis 29. Samajoonte tiheduse lennult muutmine (sama mis joonis 28 kaader 1).

KOKKUVÕTE

Töö üheks peamiseks eesmärgiks oli välja selgitada, kuidas internetisirvijas samajooni animatsioonikiirusel interpoleerida. Antud eemärgi täitmiseks valiti samajoonte interpoleerimise aluseks Delaunay triangulatsiooni.

Triangulatsiooni realiseerimiseks valiti dünaamilise iseloomuga Su ja Drysdale poolt kirjeldatud juurde lisavat tüüpi (*incremental*) algoritm. Andmestruktuuri paika panemisel loobuti populaarsest *quad-edge* struktuuri kasutamisest ning kasutati tavalist kolmnurkade põhist lähenemist. Huvitava kõrvaltulemusena vähendas lähenemine Su ja Drysdale algoritmi CCW testide arvu, antud tulemust oleks vaja põhjalikumalt edasi uurida.

Samajoonte ühendamisel lähtuti olemasolevast andmestruktuurist. Ühendatud samajooned siluti kasutades *quadratic* Bezier' kõveraid. Prooviti veel cubic Bezier' kõveratega silumist, kuid sellega kaasnesid sõlmede tekkimine ja suurem arvutusmaht. Samajoonte loomisel kasutati kunstlikult loodud gradient pindasid ja Perlin simpleks müra. Samuti katsetati samajoonte genereerimist Eesti ilmajaamades mõõdetud EMHI poolt pakutavate reaalajaliste andmetega. Katsed staatiliste andmetega andsid lootust animatsiooni võimalikkusest.

Perlin simpleks müra kasutades, täideti töö üks peamisi eesmärke, samajoonte animeerimine. Asendades kolmemõõtmelises Perlin simpleksmüras ühe mõõtme ajaga, tekitati ajas muutuv pidev pind. Antud pinnalt atribuutandmeid lugedes sai võimalikuks sujuva samajoonte animatsiooni loomine.

Animatsiooni reaalajalisus tagab kasutajale maksimaalse kontrolli samajoonte välimuse üle. Saab muuta nii joonte tihedust, värvi kui ka jämedust. Loodud kooditeeki on võimalik kasutada samajoonte animatsioonide ja reaalajaliste samajoonte visualiseeringute ehitamiseks. Samuti on võimalik teha maksimaalselt kasutajasõbralik, interaktiivne animatsiooni juhtimisliides, mille abil saaks ka vaadata üksikuid kaadreid.

Kogu süsteem tugines kliendipoolsetele arvutustele ja HTML5 canvas elemendile, kasutades JavaScript programmeerimise keelt. Ainus kood, mida autor ise ei kirjutanud, oli Perlin müra loomiseks saadud JavaScripti klass. Töö autor ei ole veel leidnud ühtegi tööd,

kus tehtaks animatsioonikiirusel samajoonte interpolatsiooni internetisirvijas, kasutades HTML5 canvas elementi.

SUMMARY

Interpolation and Animation of Isolines Inside a Browser Using Delaunay Triangulation and Dynamic Data

Main goal of this paper was to find a way to interpolate and animate isolines inside a modern webbrowser in real time.

Delaunay triangulation was chosen as an interpolation grid. From among many Delaunay triangulation algorithms incremental type was chosen and the one described by Su and Drysdale. Famous quad-edge datastructure was abandoned and simple triangular one was used instead. Interesting byproduct of this was reduction of CCW tests required by original algorithm. The only CCW tests used were the ones required by Guibas & Stolfi locate rutine.

Connecting isolines was straightforward. Cubic and quadratic Bezier curves were tried for smoothing the isolines. Cubic ones were mixing with each other and their calculations took more time, therefore quadratic curves were chosen.

Isoline construction was tested upon self-made gradient surfaces. Also some tests were carried out with real-time weather data of Estonian. Animation of isolines was tested against Ken Perlin's simplex noise surface. Results showed that that the goal of interpolating and displaying isolines in animation speed can be achived.

Methods explored can lead to building data- driven visualizations. Visual properties like thickness, density and color of isolines can be changed on the go. Also it should be possible to build applications where user has highest level of control over which frames and in which order are played.

All the calculations were done client-side using HTML5 canvas element and JavaScript. All the code except Perlin noise class was made by the author of this paper.

TÄNUAVALDUSED

Kõigepealt sooviksin tänada oma juhendajaid Raivo Aunapit ja Margus Tiru. Lisaks tänan Erki Saluveeri, kes oli valmis vajadusel töö probleeme arutama. Eriline tänu läheb veel Mattias Linnapile ja Aare Puussaarele, kes olid abiks olulise allika hankimisel.

KIRJANDUS

Andersson, F., 2003. Bezier and B-spline Technology. Magistritöö, Umea Ülikool.

Bentley, J. L., Weide, B. W., Yao, A.C, 1980. Optimal expected time algorithms for the closest point problems. ACM Transactions on Mathematical Software, Vol. 6. pp. 563-580.

Bourke, P.D., 1987. A Contouring Subroutine. BYTE Magazine (June).

Bowyer, A., 1981. Computing Dirichlet tessellations. The Computer Journal. Vol. 24. pp. 162-166.

Brinzarea, B., Darie, C., Cherecheş-Toşa, F., Bucica, M, 2006. AJAX and PHP: Building Responsive Web Applications. Packt Publishing Ltd, Birmingham, 284 pp.

Cauvin, C., Escobar, F., Serradj, A., 2010a. Thematic Cartography Volume 1: Thematic Cartography and Transformations. ISTE Ltd, London, 496pp.

Cauvin, C., Escobar, F., Serradj, A., 2010b. Thematic Cartography Volume 2: Cartography and the Impact of the Quantitative Revolution. ISTE Ltd, London, 432pp.

Cauvin, C., Escobar, F., Serradj, A., 2010c. Thematic Cartography Volume 3: New Approaches in Thematic Cartography. ISTE Ltd, London, 320pp.

Chen, J., Dong, L., Lu, D., 2010. Reasearch of Isoline Generation in Well Testing Based on PEBI Grids. 3rd Joint International Conference on Modelling and Simulation. pp.48-54.

de Berg, M., Cheong, O., van Kreveld., Overmars, M., 2008. Computational Geometry: Algorithms and Applications. Springer-Verlag, Berlin, 386pp.

Devillers, O., 1998. On Deletion in Delaunay triangulation. 15th Annual ACM Symposium on Computational Geometry. pp. 181–188.

Dwyer, R.A., 1987. A Faster Divide-and-Conquer Algorithm for Constructing Delaunay Triangulations. Algorithmica. Vol. 2. pp. 137-151

Farouki, R.T., 2012. The Bernstein polynomial basis: A centennial retrospective. Computer Aided Geometric Design. Vol. 29. pp. 379- 419.

Fortune, S., 1986. A sweepline algorithm for voronoi diagrams. SCG '86 Proceedings of the second annual symposium on Computational geometry. pp. 313-322

Guibas, L.J., Stolfi, J., 1985. Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. ACM Transactions on Graphics. Vol. 4. pp. 74-123.

Hall, R.W., 1982. Efficient Spiral Search in Bounded Spaces. IEEE Transactions on Pattern Analysis and Machine Intelligence. Vol. 4. pp. 208-215.

Harrower, M., Fabrikant, S., 2008. The role of map animation in geographic visualization. In: Geographic Visualization: Concepts, Tools and Applications. Dodge, M., Turner, M., McDerby M. (Editors), Wiley and Sons. pp. 49-65.

Harrower, M., 2004. A Look at the History and Future of Animated Maps. Cartographica. Vol. 39. pp. 33-42.

Katajainen, J., Koppinen, M., 1988. Constructing Delaunay triangulations by merging buckets in quadtree order. Fundamenta Informaticae. Vol. 11. pp. 275–288.

Lawson, C. L., 1977. Software for C1 Surface Interpolation. In J. R. Price (Editor), Mathematical Software III. Academic Press, New York, pp. 161–194.

Lobben, A., 2003. Classification and Application of Cartographic Animation. The Professional Geographer. Vol. 55. pp.318 – 328.

Mostafavi, M. A., Gold, C., Dakowicz, M., 2003. Delete and insert operations in Voronoi/Delaunay methods and applications. Computers & Geosciences. Vol. 29. pp. 523-530

Perlin, K., 2002. Improving noise. ACM Transactions on Graphics. Vol. 21. pp. 681–682.

Perlin, K., 1985. An Image Synthesizer. SIGGRAPH '85 Proceedings of the 12th annual conference on Computer graphics and interactive techniques. Vol. 19. pp. 287-296.

Razafindrazaka, F.H., 2009.Delaunay Triangulation Algorithm and Application to Terrain Generation. African Institute for Mathematical Sciences.

Robinson, A., Sale, R., Morrison, J., 1978. Elements of Cartography. John Wiley & Sons, New York, 448pp.

Roth, R., Harrower, M., Burt, J.E, 2006. Isoline Engine: A Digital Assistant for Surface Mapping. ICA Commission on Visualization and Virtual Environments Workshop. Vancouver.

Rui, X.P., Song, X.F., Ju, Y.W., 2010. An isoline rendering method considering of constrained conditions. IGARSS. pp. 4007-4010.

Shewchuk, J.R., 1996. Triangle: engineering a 2D quality mesh generator and Delaunay triangulator. First Workshop on Applied Computational Geometry, Association for Computing Machinery. Philadelphia, Pennsylvania, pp. 124–133.

Su, P., Drysdale, R.L.S., 1997. A Comparison of Sequential Delaunay Triangulation Algorithms. Computational Geometry. Vol. 7. pp. 361–385.

Zhao, L., Liu, S., Li, J., Xing, H, 2012. The Contour Drawing Automatically Based on PDA Environment pp. 184 - 187.

Žalik, B., Kolingerová, I., 2003. An incremental construction algorithm for Delaunay triangulation using the nearest-point paradigm. International Journal of Geographical Information Science. Vol. 17. pp.119-138.

Tversky, B., Morrisony, J.B., Betrancourt, M., 2002. Animation: can it facilitate?. International Journal of Human-Computer Studies. Vol. 57. pp. 247–262.

Watson, D.F., 1981. Computing the n-dimensional Delaunay tessellation with Application to Voronoi polytopes. The Computer Journal. Vol.24. pp. 167-172.

YiHong, W., YongJiang, L., 2010. An isoline generating algorithm based on Delaunay. 2010 2nd International Conference on Computer Engineering and Technology. Vol. 7. pp. 173-176.

URL 1, 14.05.2013: http://www.ams.org/samplings/feature-column/fcarc-voronoi

URL 2, 14.05.2013: http://paulbourke.net/papers/triangulate/

URL 3, 07.05.2013: http://www.emhi.ee/index.php?ide=19,394,423,424

URL 4, 16.05.2013: http://www.antigrain.com/research/bezier_interpolation/index.html

URL 5, 16.05.2013: http://pacoup.com/2011/02/03/flash-vs-html5-performance/

URL 6, 16.05.2013: http://msdn.microsoft.com/en-us/library/windows/desktop/ff684178(v=vs.85).aspx

URL 7, 16.05.2013: http://msdn.microsoft.com/en-us/library/ie/gg193983(v=vs.85).aspx#Introduction

URL 8, 16.05.2013: http://indiemaps.com/blog/2012/04/introducing-openlayers-symbology/

URL 9, 16.05.2013: http://vincentwoo.com/js/isolines.html

URL 10, 16.05.2013: http://www.noisemachine.com/talk1/index.html

URL 11, 16.05.2013: http://www.ams.org/samplings/feature-column/fcarc-bezier#2

URL 12, 16.05.2013: http://indiemaps.com/blog/2008/06/isolining-package-for-actionscript-3/

URL 13, 18.05.2013: http://www.w3.org/TR/2012/CR-css3-images-20120417/

URL 14, 18.05.2013: http://www.sjeiti.com/perlin-noise-versus-simplex-noise-in-javascript-final-comparison/

URL 15, 19.05.2013: http://www.emhi.ee/ilma_andmed/xml/observations.php

URL 16, 19.05.2013: http://www.emhi.ee/

URL 17, 19.05.2013: http://msdn.microsoft.com/en-us/hh562071.aspx

URL 18, 19.05.2013: http://creativejs.com/resources/requestanimationframe/

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina_____Karl Tõnissoo_____

(sünnikuupäev:_____8.09.1987_____)

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose
Delaunay triangulatsiooni rakendamine dünaamiliste atribuutide põhjal samajoonte interpoleerimisel ja tulemuste animeerimine internetisirvijas _____,

mille juhendajad on _____ Raivo Aunap, Margus Tiru_____,

1.1. reprodutseerimiseks säilitamise ja üldsusele kättesaadavaks tegemise eesmärgil, sealhulgas digitaalarhiivi DSpace-is lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
1.2. üldsusele kättesaadavaks tegemiseks Tartu Ülikooli veebikeskkonna kaudu, sealhulgas digitaalarhiivi DSpace´i kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.

2. olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile.

3. kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest tulenevaid õigusi.

Tartus, **20.05.2013**