Joonas Nieminen

Avoimen datan visualisointi verkkoselaimessa

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikka

Insinöörityö

11.3.2014

Metropolia

Tekijä(t) Otsikko Sivumäärä Aika	Joonas Nieminen Avoimen datan visualisointi verkkoselaimessa 47 sivua + 1 liite 11.3.2014
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja(t)	Lehtori Vesa Ollikainen Yliopettaja Erja Nikunen

Tässä insinöörityössä toteutettiin Suomen seutukuntien avointa dataa visualisoiva kartogrammi. Toteuttamista varten koostettiin ensin suunnitteluperusteita, joilla datavisualisaatioista saadaan mahdollisimman hyödyllisiä. Visualisaatio toteutettiin verkkoselaimella katseltavaksi ja tätä varten luotiin katsaus modernien selainten kykyihin sekä työkaluihin, joita voidaan käyttää niille ohjelmistoja toteutettaessa.

Työn alussa esitellään datavisualisaation perusteet. Tämän jälkeen koostetaan ensin visualisaatioiden graafisen suunnittelun periaatteita, sekä huomioita ihmisen näkökyvystä ja tiedonkäsittelystä kuten siitä, miten ihminen aistii värejä ja hahmoja. Lisäksi kerätään tärkeimpiä seikkoja visualisaatiota suunniteltaessa verkkoon, modernien verkkoselainten kyvyistä sekä siitä, millainen käyttöliittymä niille soveltuu.

Näiden periaatteiden nojalla kuvataan toteutettavan visualisaation suunnitelma ja esitellään kartogrammin konsepti, johon suunnitelma perustuu. Tämän jälkeen käydään läpi tarvittavan avoimen seutukuntadatan koosto ja esitellään D3-visualisointikirjasto, jolla ohjelma toteutetaan. Syntynyt visualisaatio käydään läpi ja lopuksi arvioidaan kehitysprosessia sekä sen tuloksia.

Työn tuloksena syntyi Suomen seutukuntadataa visualisoiva epäjatkuva kartogrammi, joka yritettiin ensin rakentaa ilman varsinaista karttadataa. Tämän kokeilun tulokset eivät olleet täysin tyydyttäviä, joten kartogrammin seutukunnille lisättiin sijaintidataa. Näin syntynyt visualisaatio arvioitiin riittävän hyväksi täyttämään työn tavoite.

Avainsanat	Datan visualisointi, HTML5, D3

Author(s) Title Number of Pages Date	Joonas Nieminen Visualization Of Open Data In Web Browsers 47 pages + 1 appendix 11 March 2014
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor(s)	Vesa Ollikainen, Senior Lecturer Erna Nikunen, Principal Lecturer

The aim of the study was to develop a cartogram regionally visualizing open data in Finland. To guide the process, design principles about building maximally useful visualizations were first compiled. The visualization was developed to be viewed with a web browser, so an overview of the abilities of modern browsers and the tools used to develop software for them was include in the study.

The thesis begins by introducing the fundamentals of data visualization. Then the principles of the graphic design of visualizations are covered, as well as considerations about the visual system and human data processing capabilities. Also some of the most important points about designing visualizations for the web and the capabilities of browsers are introduced, and the user interfaces best suited for their use are briefly evaluated.

Using the collected principles the design of the visualization to be developed is then outlined, and the concept of a cartogram is introduced. After this the process of collecting the open sub-region data is described, and the basics of the D3 visualization library which is used to implement the visualization are explained. The completed data visualization is then presented, and the development process and its results are reviewed.

The result of this thesis was a non-contiguous cartogram that visualizes data about the sub-regions of Finland. The construction of the cartogram without map data was first attempted, but the result was deemed not to be sufficient. Placement data was then added to the cartogram, and the resulting visualization was concluded to meet the goals of the study.

Keywords	Data visualization, HTML5, D3

Sisällys

Lyhenteet

CSS	Cascading Styles Sheets. Kieli dokumentin esityksen kuvaamiseen.
CSV	Comma-separated Values. Tiedostomuoto tabulaarisen datan tallentamiseen tekstinä.
D3	Data-Driven Documents. JavaScript-kielinen kirjasto datavisualisaatioiden rakentamiseen verkkoselaimessa.
DOM	Document Object Model. Tapa esittää ja muokata HTML- ja XML-dokumenttien olioita.
HTML	HyperText Markup Language. Kieli verkkoselaimessa esitettävien dokumenttien kuvaamiseen.
HTML5	Edellisen uusin versio, johon luetaan usein mukaan myös siihen liittyvät tekniikat.
HTTP	Hypertext Transfer Protocol. Sovellusprotokolla hypertekstin siirtämiseen.
MV*	Model View *. Kokoelma sovellusarkkitehtuurimalleja käyttöliittymän erottamiseen sovelluslogiikasta.
JSON	JavaScript Object Notation. Formaatti datan esittämiseen avain-arvopareina.
SVG	Scalable Vector Graphics. Formaatti kaksiulotteisen vektorigrafiikan merkitsemiseen.

1 Johdanto

Eduskunnan tulevaisuusvaliokunta julkaisi 2.10.2013 selvityksen sadasta tärkeimmästä teknologisesta ratkaisusta ja niiden merkityksestä Suomelle. Tärkeimmäksi niistä selvitys valitsi avoimen datan ja niin kutsutun ”big datan”. [1.] Avoimen datan idea on jakaa organisaatioiden keräämää dataa kenen tahansa käytettäväksi ja yhdisteltäväksi, ja ”big data” taas on niin suuria datamääriä, ettei niitä voida tavallisten menetelmien avulla käsitellä. Tästä datasta voi löytyä odottamattomia oivalluksia mille tahansa elämänalueelle.

Nopein käyttöliittymä tutkia ja tulkita kyseisenlaista dataa on tarkastella jonkinlaista visuaalista esitystä siitä. Ihmisen aivot ovat supertietokone hahmojen etsimiseen tiedosta, ja datan visualisointi on työkalu esittää data sen omaksuttavimmassa muodossa. Tarkoituksenmukaisen ja hyödyllisen visualisaation rakentaminen ei kuitenkaan ole itsestään selvää vaan vaatii huomioita graafisesta tyylistä ja ihmisen havaintokyvystä. Työn ensimmäisenä tavoitteena onkin kerätä mahdollisimman kattavasti näihin perustuvia visualisaatioiden suunnitteluperusteita, joita voidaan hyödyntää paitsi datavisualisaation suunnittelussa ja arvioinnissa myös lukuisissa muissa tarkoituksissa, kuten käyttöliittymien suunnittelussa ja graafisessa viestinnässä.

Luonteva alusta datavisualisaatiolle on moderni verkkoselain. Uusimmat selaimet kykenevät piirtämään reaaliaikaista kaksi- ja jopa kolmiulotteista grafiikkaa ilman selaimeen asennettavia liitännäisiä. Selainsovelluksena toteutettu visualisaatio on näin suurelle yleisölle yhtä helposti saavutettavissa ja jaettavissa kuin mikä tahansa muukin verkkosivusto. Visualisaatiolla on myös mahdollisuus olla koko ajan yhteydessä uusimpaan dataan selaimen tiedonsiirto-ominaisuuksien avulla. Selainten toimintojen ja suorituskyvyn kehittyessä on syntynyt jako verkkodokumentin ja selaimessa toimivan sovelluksen välille, jolloin myös kehitystyökalujen määrä, monimutkaisuus ja valinnanvaikeus ovat nousseet nopeasti.

Näiden kolmen – datan visualisoinnin, avoimen datan ja modernien verkkoselainten – yhteennivoutumasta syntyi tämän insinöörityön aihe. Nämä kolme aihealuetta yhdistetään, ja tavoitteena onkin rakentaa avoimesta datasta visualisaatio selaimelle sopivien työkalujen avulla. Visualisaatio on eräänlainen abstrakti kartta Suomen seutukunnista ja niiden väestönkehityksestä. Työssä koostettua teoriaa käytetään tämän tapausesimerkkinä rakennetun niin kutsutun kartogrammin kriittiseen arviointiin.

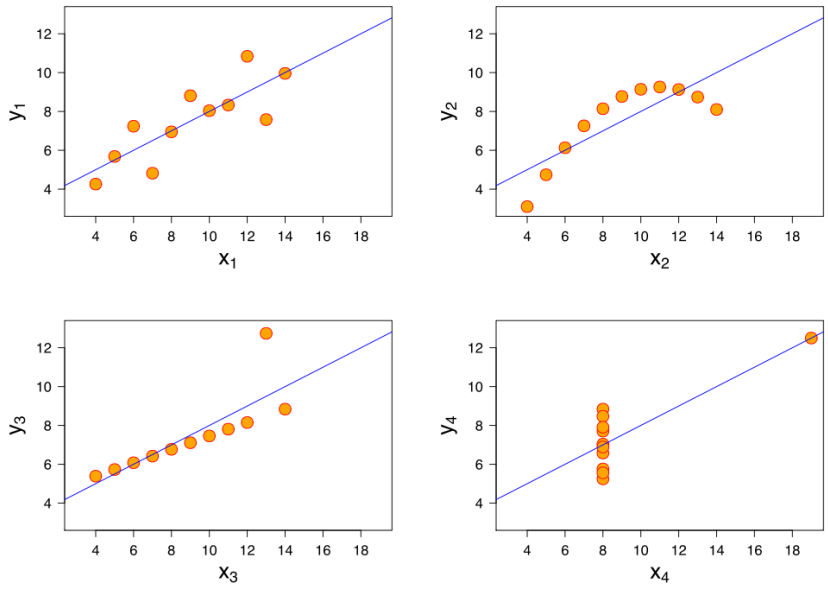
2 Datan visualisointi

2.1 Lähtökohdat

Datan visualisoinnilla tai datagrafiikalla tarkoitetaan mitattujen suureiden visuaalista esittämistä abstrakteilla kuvilla, jotka yhdistelevät pisteitä, viivoja, koordinaattijärjestelmiä, numeroita, symboleja, sanoja, varjostuksia ja värejä. Erilaisia perustavanlaatuisia datagrafiikoita ovat datakartat, aikasarjat, tila-aikaselostukset ja yhteyksiä kuvaavat grafiikat. [2, s. 9, 15.] Tässä luvussa koostetaan teoriaa datavisualisoinnista ja ihmisen tiedonkäsittelyn ominaisuuksista, jota voidaan hyödyntää tarkoituksenmukaisten ja tehokkaiden visualisaatioiden suunnittelussa.

Visualisaatio on parhaimmillaan väline, joka auttaa järkeilemään kvantitatiivisesta tiedosta, ja koska ihminen hankkii näköaistinsa avulla tietoa nopeammin kuin muilla aisteilla yhteensä, se on nopein tiedonvälityskanava aivoihin. Hyvin suunniteltu datagrafiikka on tällöin tehokkain tapa kuvata, tutkia ja tiivistää usein erittäin suuria datajoukkoja, ja tavoitteena onkin optimoida esitys aivojen tiedon- ja hahmonetsintäkyvyille helpoimmin omaksuttavaan muotoon. [2, s. 9; 3, s. 2.] Koostetut suunnitteluperusteet pyrkivät kuvaamaan mahdollisimman paljon graafisista ominaisuuksista ja ihmisen tiedonkäsittelystä, joita visualisaatioiden optimoinnissa tarvitaan.

Visualisaatiosta voi ymmärtää valtavia määriä dataa välittömästi, ja se mahdollistaa odottamattomien ominaisuuksien tai hahmojen havaitsemisen. Nämä odottamattomat havainnot voivat toimia uusien oivallusten pohjana. Toisaalta datan virheet ja häiriöt, jotka syntyvät esimerkiksi datan keruumenetelmistä, ovat oikeanlaisesta esityksestä heti nähtävissä. Visualisaatio helpottaa datan pienen ja suuren mittakaavan tulkintaa, joka voi olla erityisen arvokasta ominaisuuksia yhdistävien hahmojen tunnistamisessa. Visualisaatio tukee myös uusien hypoteesien muodostamista datasta. [3, s. 3–4.] Kuvan 1 niin kutsutussa Anscomben nelikossa kaikilla neljällä datajoukolla on sama keskiarvo, varianssi ja korrelaatio, mutta visuaalisesta esityksestä näkee heti, miten erilaisia datajoukot ovat. Kuvassa 2 on John Snow’n kartta vuoden 1854 kolerakuolemista, ja sen avulla todettiin koleran levittäjäksi kartan keskellä oleva Broad Streetin vesipumppu, jonka sulkemisen jälkeen tartunnat loppuivat. Tämä auttoi myös kumoamaan teorian hajusta tautien levittäjänä.



Kuva 1. Anscomben nelikko [4].

Kuva 2. Kartta kolerakuolemista [2, s. 24].

Mikäli visualisaation suunnittelu perustetaan ihmisen näköaistiin ja kognitiivisiin kykyihin, se käyttää universaalia kieltä – kuten matematiikkaa – eikä näin ole sidoksissa mihinkään yksittäiseen kieleen tai kulttuuriin. Samat periaatteet pätevät moneen muuhunkin muotoilun ja suunnittelun alaan. [2, s. 9.] Kasvamme kaikki lähtökohtaisesti samanlaisessa maailmassa: kohteilla on pinta, jolla on tekstuuri ja väri, eivätkä ne katoa sattumanvaraisesti. Valo kulkee suorassa linjassa ja heittää varjoja, heijastuu pinnoilta tietyllä tavalla, ja painovoima vaikuttaa kappaleisiin. Opimme aistimaan samalla tavoin jatkuvia ääriviivoja ja erilaisia muotoja. Kehitämme perimämme ja ympäristömme takia samanlaiset visuaaliset järjestelmät, ja mikäli visualisaatio perustetaan näihin keskeisiin kognitiivisiin ominaisuuksiin, siitä on todennäköisesti suurin hyöty. [3, s. 7–11.]

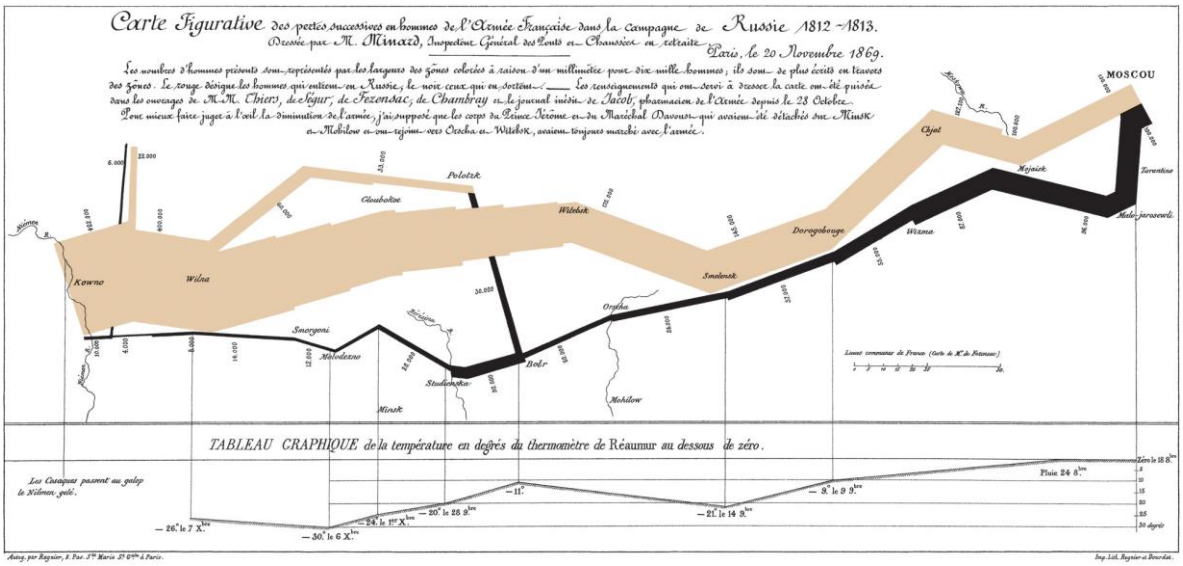
2.2 Datavisualisaatioiden graafinen tyyli

Ensimmäinen hyödyllisen visualisaation tavoite on yksinkertaisesti näyttää data. Toisaalta mikäli data on riittämätöntä, tai selitettävä malli perusteeton, ei sen visuaalisella esittämisellä ole mitään arvoa. Grafiikan ilmaisuvoimaa tuhlaa myös pelkkä muutaman numeron koristelu, ja sitä voi olla turha myöskään käyttää esittämään yksinkertaisia lineaarisia vaihteluja, kuten suoraa x-y-koordinaatistossa, joka yleensä on parempi ilmaista yhdellä tai kahdella numerolla. Visualisaation tulisi saada katselija pohtimaan sen substanssia – ei käytettyjä keinoja tai graafista suunnittelua ja tuotantoa eikä vääristää datan sisältöä. Grafiikka viestii parhaiten suuria ja monimutkaisia datajoukkoja, ja sen tarkoituksena on niiden monimutkaisuuden paljastaminen ja selkeyttäminen.

Tavoitteena on näyttää paljon numeroita pienessä tilassa ja tehdä suuret datajoukot ymmärrettäviksi. Datasta tulisi saada yleiskuva, mutta siitä pitäisi selvitä pienet yksityiskohdat. Silmää tulisi auttaa vertailemaan datan eri osia. Esityksen täytyy palvella selkeää tarkoitusta kuten datan selitystä tai tutkimista, ja olla läheisessä yhteydessä datajoukon tilastolliseen ja sanalliseen kuvaukseen. Hyvässä visualisaatiossa kohtaavat substanssi, tilastotiede ja muotoilu, ja se kommunikoi monimutkaisia ideoita selkeästi, tarkasti ja tehokkaasti. Paras mahdollinen visualisaatio antaa sen tutkijalle eniten ideoita lyhyessä ajassa, vähimmällä määrällä graafisia elementtejä ja pienimmässä mahdollisessa tilassa. [2, s. 13–51.]

Kuvassa 3 on vuonna 1869 julkaistu esitys Napoleonin katastrofaalisesta vuoden 1812 kampanjasta Venäjällä. Ruskea vuo kuvaa armeijan kokoa hyökkäyksessä, ja siitä ilmenee armeijan koko eri kohdissa ja se, miten armeija koko ajan pienentyy. Musta vuo kuvaa armeijan perääntymistä. Vuosta voidaan näin lukea myös armeijan paikka tiet-

tyinä aikoina, sekä se, mihin suuntaan se liikkuu ja armeijan jakautumiset. Alhaalla on vielä kuvaaja lämpötilan vaihteluista, ja vuosta voidaan lukea lämpötilan ja armeijan kutistumisen suhteesta. Visualisaatiossa on kuvattu yhteensä seitsemän muuttujaa ja sitä pidetään yleisesti erittäin onnistuneena.



Kuva 3. Napoleonin 1812 hyökkäys Venäjälle [5].

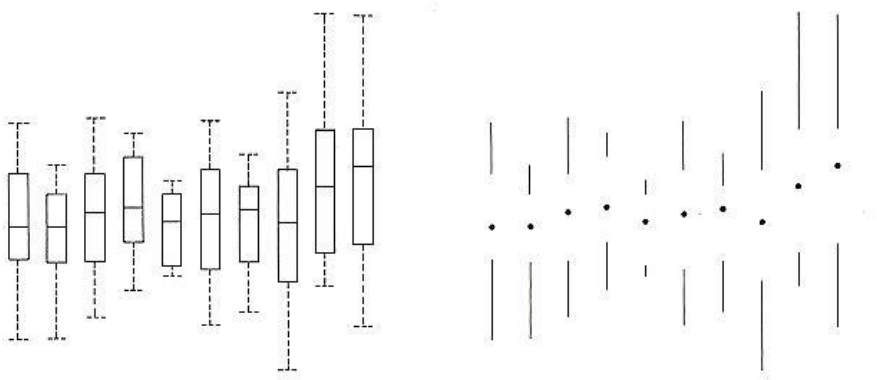
Keskeinen ongelma datagrafiikkaa suunniteltaessa on se, miten saada visuaalinen esitys yhdenmukaiseksi numeerisen arvon kanssa, eli saada esityksestä vääristämätön. Ihminen ei pysty tarkasti tulkitsemaan ja vertailemaan pelkkien graafisten elementtien ominaisuuksia. Jopa pelkän viivan pituuden arvioiminen riippuu aina kontekstista ja jopa siitä, mitä muut ovat viivasta sanoneet. Kuitenkin periaatteena esityksen pinnan fyysisen koon tulee aina olla suoraan verrannollinen esitettyyn suureeseen. Grafiikan voidaankin katsoa valehtelevan grafiikan muutoksen ja datan muutoksen suhteen verran. [2, s. 56–57.]

Toinen huomionarvoinen seikka on muodon ja datan vaihtelut ja se, miten suunnittelun tulisi pysyä yhdenmukaisena koko esityksessä. Visualisoinnin tulisikin esittää datan vaihtelut eikä vaihteluja suunnittelussa. Mikäli esityksen yhdestä osasta tehdyt oletukset eivät päde toisessa, voidaan sen katsoa taas valehtelevan. Tästä seuraa myös, että yksiulotteista dataa ei tule esittää pinta-alalla, ja toisaalta yleisempi sääntö, että esittävän muuttujan ulottuvuuksien ei tulisi ylittää datamuuttujan ulottuvuuksia. [2, s. 60–71.]

Grafiikka ei saa myöskään lainata dataa ilman kontekstia. Kaiken kvantitaviisen järkeilyn perustana on vertailu, ja mikäli esitys ohittaa vertailun mahdollistavan kontekstin, se

on harhaanjohtavaa. Esimerkiksi jos viivadiagrammi esittää vain yhden kohdan datasta, ei ilman kontekstia voida päätellä, onko esitetty kohta osa jaksollista vaihtelua. [2, s. 74–75.]

Periaate, joka voi olla hyödyllinen datagrafiikan arvioinnissa, on ”datamuste” (data-ink) [2, s. 93]. Se kuvaa sitä osaa esityksestä, jota ei voida pyyhkiä pois menettämättä datainformaatioita. Toisin sanoen se on grafiikan ydin eli ne toisteettomat elementit, jotka on aseteltu kuvaamaan vaihteluja esitetyissä numeroissa. Harkinnanarvoinen tavoite onkin maksimoida datamusteen (tai pikselien) osuus visualisaatiosta. Mikäli jokin elementti ei ole datamustetta tai mikäli se on toisteinen, sen voi luultavasti pyyhkiä pois. [2, s. 93–96.] Kuvassa 4 pyyhitään laatikkokuvaajasta pois kaikki dataa kantamattomat pikselit. Lopputulos viestii yhtä paljon informaatiota, ja huomio kiinnittyy siksi enemmän dataan. Toisaalta mikäli graafiset elementit, jotka eivät suoraan ole datamustetta, voidaan perustella jotenkin, ei pyyhkiminen ehkä ole välttämätöntä. Laatikkokuvaajassa perusteena voitaisiin ehkä käyttää esityksen vakiintuneisuutta, mutta tyypillisesti, mikäli huomio kiinnittyy riisutummassa esityksessä helpommin dataan, se on luultavasti lähtökohtaisesti parempi.

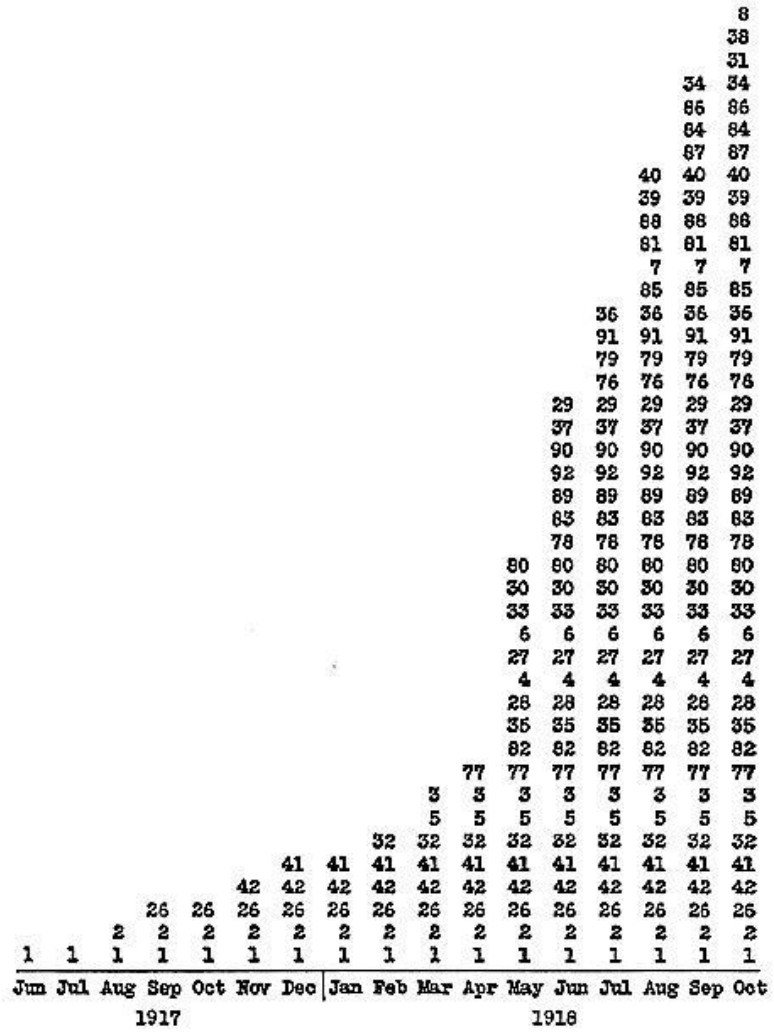


Kuva 4. Laatikkokuvaajan datamusteen maksimointi [2, s. 125].

Toinen datamusteeseen liittyvä periaate on termi ”kaavioromu” (chartjunk) [2, s. 108]. Kaavioromua ovat muun muassa kaikenlaiset koristeet ja elementit, jotka eivät kerro tarkastelijalle mitään uutta. Pahimmasta päästä ovat optisia illuusioita aiheuttavat toisteiset viivakuviot, joilla on vaikkapa koristeltu pylväsdiagrammeja. Tällaisia kuvioita saattaa myös seurata pikselöityneestä grafiikasta, joten visualisointien tulisi yleensä olla pehmennettyjä [3, s. 64]. Esityksen ruudukko saattaa olla erittäin häiritsevä, mikäli se on turhan suurikontrastinen tai räikeä. Ruudukon tulisi olla hillitty ja parhaimmillaan vain vihjattu. Huomion ei tulisi kiinnittyä pinnallisiin elementteihin vaan kiinnostavaan dataan, ja visualisaatioista ei saada kiinnostavaa vain lisäämällä ornamentteja. [2, s.

107–120.]

Tavoitteellisesti graafisten elementtien tulisi usein palvella useaa tarkoitusta pelkän data-arvon esittämisen lisäksi. Jos tämä saavutetaan, viestivät visualisaatiot tehokkaammin useampia monimutkaisempia muuttujia. Pylväsdiagrammi voidaan esimerkiksi koostaa itse havaintoarvoista, jolloin esitys näyttää sekä arvot että vertailee niiden lukumäärää eri pylväiden pituuksien perusteella, kuten kuvassa 5. Toinen tapa hyödyntää elementtejä usealla tavalla on käyttää dataan perustuvia selitteitä. Esimerkiksi pistediagrammin reunat voivat koostua pisteiden arvoista, tai reunaviiva voi ilmaista pisteiden vaihteluväliä. [2, s. 140–149.]

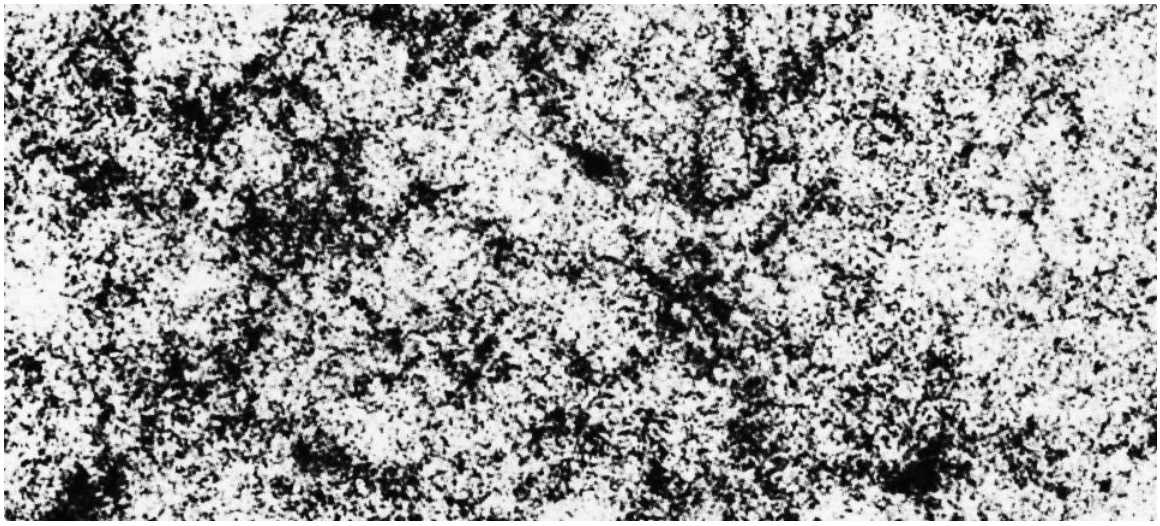


Kuva 5. Yhdysvaltalaiset divisioonat Ranskassa. Esityksestä ilmenee divisioonien lukumäärä kunakin kuukautena, divisioonien tunnus sekä sijoituksen kesto. [2, s. 141.]

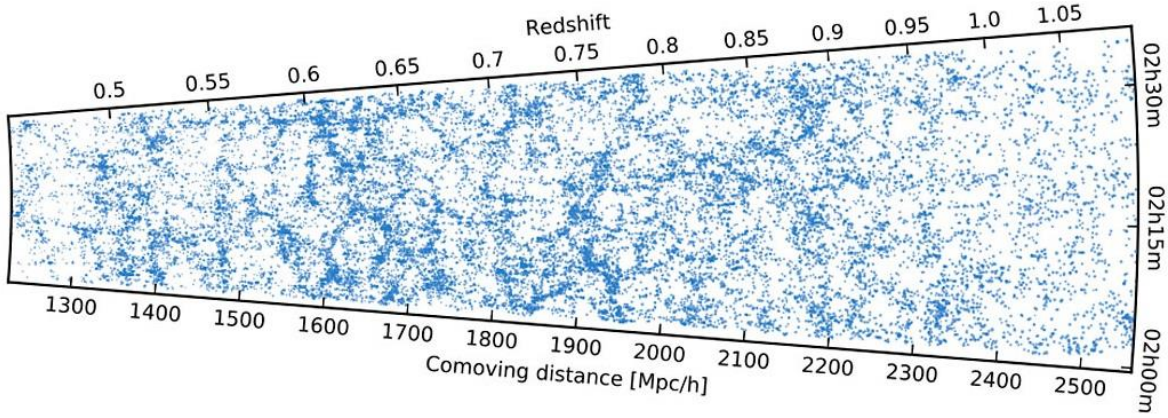
Informaatiota voidaan järjestellä useaan kerrokseen luomalla useita katselusyvyyksiä ja -kerroksia. Grafiikka voidaan suunnitella tarjoamaan ainakin kolme katselusyvyyttä.

Ensin kaukaa nähty kokonaisvaltainen kuva, joka koostuu pienten rakenteiden yhdistymisestä, toiseksi läheltä tarkasteltu hienojakoinen datan rakenne ja kolmanneksi siitä, mitä grafiikan takana on, eli mitä grafiikka vihjaa. Tällainen katseluarkkitehtuurin analyysi auttaa järjestämään monimutkaista tietoa hierarkkisesti. [2, s. 154–159.]

Grafiikka on ehkäpä paradoksaalisesti sitä luettavampaa, mitä yksityiskohtaisempaa se on, ja visualisaation sekavuus on suunnittelun virhe – ei datan ominaisuus. Tulkitsemisen yksinkertaisuus nousee monimutkaisen informaation huolellisesta järjestelystä [6, s. 37, 51.] Visualisaation laajemmasta yleiskuvasta saa mahdollisuuden valita, miten kuvan yksityiskohtia lajittelee ja vertailee, samalla tavalla kuin maisemaa katsottaessa. Yksityiskohdissa visualisaation tahti tiivistyy, hidastuu ja muuttuu henkilökohtaisemmaksi. Tällaiset mikro- ja makrokatsantokannat ovat universaaleja ja perustuvat ihmisen tiedonkäsittelyn ominaisuuksiin. [6, s. 38.] Kuvissa 6 ja 7 galaksien muodostama verkkomainen rakenne paljastuu, kun tarkastellaan makroskaalan kuvaa niiden jakautumisesta.



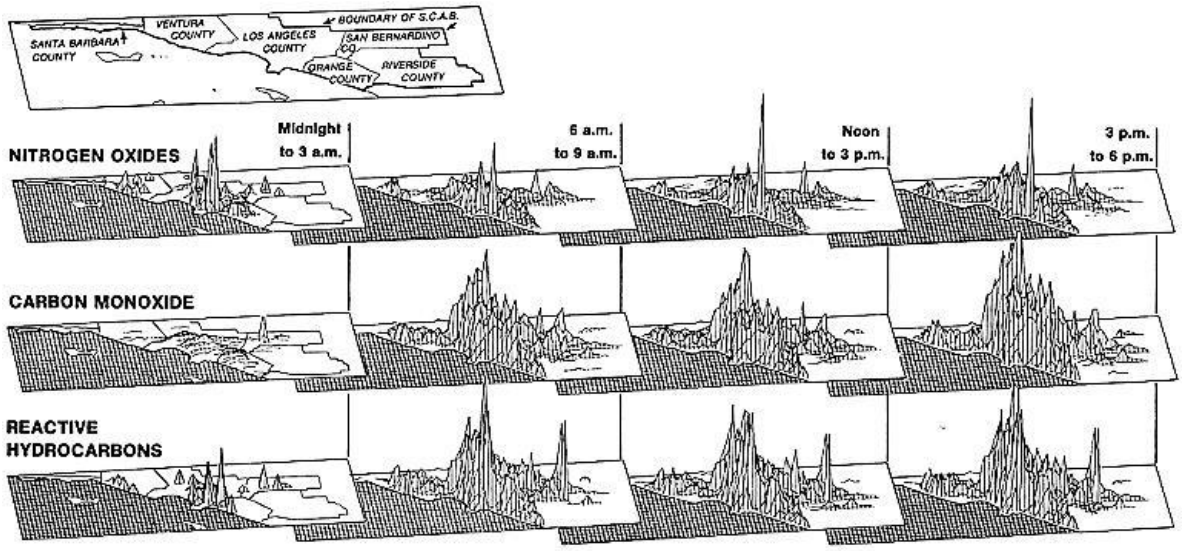
Kuva 6. Galaksien muodostamia rakenteita pohjoisella taivaalla. Rajattu. [1, s. 27.]



Kuva 7. Leikkaus maailmankaikkeuden rakenteesta [7].

Datan määrän suhdetta esityksen pinta-alaan eli datatiheyttä voi myös olla hyödyllistä arvioida. Matalan datatiheyden esitys saa tarkastelijan heti epäilemään sen uskottavuutta, kuten mitä esityksestä on jätetty pois tai onko data valikoitu epärehellisesti. Korkeatiheyksisessä esityksessä tarkastelijalla on valta muokata ja valita dataa itselleen sopivaan muotoon, eikä esitys ole rajoittunut toimittajan tai pinnallisten elementtien lisääjän määräämäksi. Visualisaation tulisikin esittää suuria datajoukkoja suurella tiheydellä, eikä päinvastoin. Yleensä datavisualisaatioita voi pienentää reilusti [2, s. 161, 166], ja ellei data suoraan ehdota jotain muotoa visualisaatiolle, sen tulisi olla horisontaalinen (noin 50 % leveämpi kuin korkeampi, sillä silmämme on tottunut etsimään eroja horisontista) [2, s. 186, 190] ja selittävän muuttujan sijaita horisontaalisella sivulla. [2, s. 186–187.]

Esimerkki korkean datatiheyden visualisoinnista on rakentaa sarja pieniä itsenäisiä kuvia, joissa jokaisessa näytetään sama yhdistelmä muuttujia ja joita indeksoidaan muutoksilla toisessa muuttujassa. Näin syntyy sarjakuvaa tai elokuvan kuvaruutuja muistuttava esitys, joka havainnollistaa useaa muuttujaa helposti vertailtavassa ja luettavassa muodossa. Tällaista esitystä kutsutaan ”pieniksi monikerroiksi” (small multiples), ja ne muodostavat selostavan jatkumon, joka kuvaa vaihteluja muuttujien suhteessa indeksimuuttujan saadessa uusia arvoja. [2, s. 168–169.] Ne kiinnittävät huomion automaattisesti datan vaihteluihin eivätkä vaihteluihin sen kehyksissä [6, s. 67]. Mikäli visualisaatio olisi pelkkä animaatio, ei ruutuja voisi vapaasti vertailla, ja suurin osa informaatiosta olisi koko ajan piilotettu, jolloin sitä jouduttaisiin esityksen sijaan pitämään muistissa. Piilotettuja osuuksia vasten ei myöskään voisi testata visuaalisia hypoteeseja. [3, s. 337–339.] Pieniä monikertoja hyödyntävä visualisaatio on kuvassa 8.



Kuva 8. Los Angelesin ilman sisältämät molekyylit ajan suhteen. Tutkittu paikka on usean moottoritien risteämä. [2, s. 42.]

Visualisaatio on käytännöllistä integroida osaksi tekstiä ja käyttää selitteitä tai lyhyitä viestejä tarkastelijan apuna. Kun grafiikka on osa tekstiä, ei silmän tarvitse vaellella tekstuaalisen ja graafisen materiaalin välillä, kuten tavallisessa teknisessä kirjoituksessa (esimerkiksi tässä). Tekstiä pidetään grafiikassa aina datamusteena. Esitystä ei ole järkevää pilkkoa useaan osioon, vaan kaiken tarpeellisen tiedon voi esittää yhdessä. Tekstin tulisi silti vain selittää, miten grafiikkaa luetaan (jos se on tarpeen), eikä selittää, mitä grafiikka sisältää, sillä esitys itsessään jo toivottavasti viestii kaiken tarpeellisen, ja sen uudelleen selittäminen tekstissä olisi toisteista. [2, s. 180–182.]

2.3 Visuaalinen havaintokyky

Visuaalinen havaintokyky voidaan karkeasti jakaa kolmeen vaiheeseen. Ensimmäisessä vaiheessa näkymästä erotetaan nopeasti ilman varsinaista tietoista osallistumista rinnakkain prosessoiden kartta sen matalan tason ominaisuuksista, niiden väristä, tekstuurista ja liikeradoista. Tämä kartta on siirtymävaihe ymmärrettäessä elementtien merkittävyyttä esityksessä, ja mikäli tietoa halutaan ymmärrettävän nopeasti, se tulee esittää niin, että nämä rinnakkaiset prosessit huomaavat sen helposti. [3, s. 20–21.]

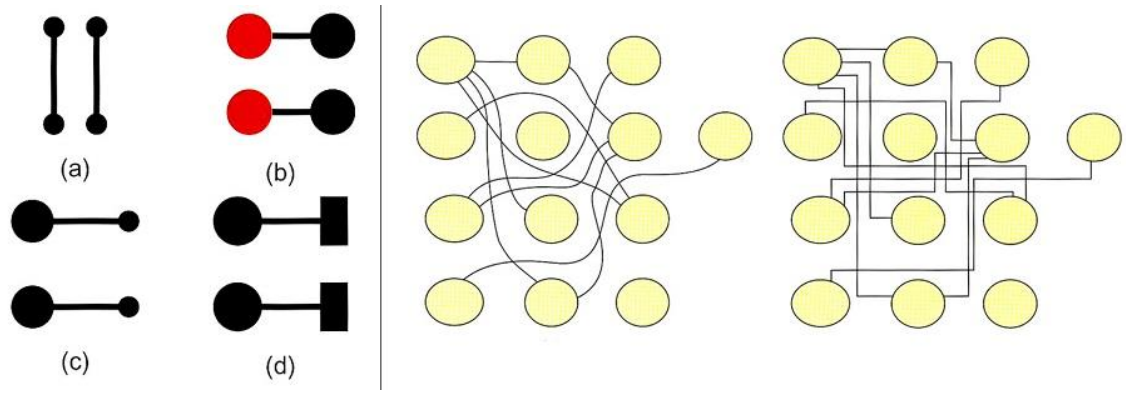
Toisessa vaiheessa nopeat aktiiviset prosessit jakavat näkökentän alueisiin ja yksinkertaisiin hahmoihin kuten jatkuviin ääriviivoihin, samanvärisiin alueisiin ja yhtenevästi teksturoituihin alueisiin. Liikehahmot tunnistetaan myös tässä vaiheessa. Toisen vaiheen prosessointi on hitaampaa sarjaprosessointia. Hahmontunnistusvaihe on erittäin

joustava, ja siihen vaikuttavat ensimmäisen vaiheen suuri määrä ominaisuuskarttadataa ja tarkkaavaisuuden määräämät visuaaliset haut. Haut voivat olla epämääräisiä. Saatamme esimerkiksi etsiä erilaisia rakenteita tai poikkeuksia sääntöihin. Toisaalta haut voivat olla tarkkoja, kuten positiivinen suuntaus kuvaajassa. Hakuja voidaan helpottaa tekemällä visualisaatiosta niin kompakti kuin mahdollista. [3, s. 21,141.]

Kolmas vaihe on visuaalinen työmuisti, jossa kohteita pidetään tarkkaavaisuuden määräämänä. Kolmannessa vaiheessa rakennetaan haut, joihin visuaaliset etsintästrategiat voivat vastata. Työmuistissa voidaan pitää vain muutamaa kohdetta kerrallaan, ja ne on rakennettu hahmoista, jotka voivat vastata hakuihin, ja pitkäaikaisesta muistista haetusta tehtävään liittyvästä tiedosta. Esimerkiksi jos etsimme tiekartasta reittiä, etsii visuaalinen haku yhdistettyjä punaisia ääriviivoja, jotka merkitsevät tietä kahden kaupunkia merkitsevän symbolin välillä. [3, s. 22.]

Näiden kolmen vaiheen lisäksi havaintokykyyn vaikuttaa huomiokyky, joka aina uuden tiedon saapuessa säätää näitä kolmea vaihetta ylhäältäpäin odotusten ja arvokkaaksi arvioidun sisällön mukaan. Myös ensimmäisen vaiheen kohdekartat koostetaan huomiokyvyn säätämien herkkyyksien mukaan. Toisessa vaiheessa huomiokyky etsii koko ajan uusia huomiota vaativia kohteita. [3, s. 22.]

Tärkeä kognitiivinen työkalu visualisaatioiden lukemisessa ja suunnittelussa on hahmontunnistus. Helppo ja käyttökelpoinen hahmontunnistuslaki on läheisyys. Mikäli symbolit liittyvät toisiinsa, ne on perusteltua asettaa toistensa läheisyyteen – erityisesti tekstiselitteet. Läheisesti sijoitellut elementit myös säästävät visuaalisia hakuja ja helpottavat tiedon omaksumista. Läheisyyttä, väriä ja muotoa voimakkaampi hahmontunnistuslaki on virheettömyys. Mikäli elementit ovat kytketty toisiinsa viivalla tai nauhalla ymmärretään niiden välillä olevan suhde. Suhde on helpompi hahmottaa, mikäli elementit ovat jatkuvia, eivätkä sisällä nopeita suunnanvaihtoja kuten suoria kulmia. Symmetria voi olla voimakas hahmontunnistusmenetelmä, mikäli kohteet ovat toistensa peilikuvia, melko pieniä ja järjestetty pystyyn tai vaakaan. Toinen läheisyyttä voimakkaampi hahmolaki on sulkeuma ja jaettu alue. Ääriviivalla suljettu alue nähdään yleensä hahmona, ja tilaa jaetaan sen sisä- ja ulkopuolelle. Sisälle jäävä alue mielletään jaettuna alueena. Huomioon tulee ottaa myös se, miten jotkin graafiset elementit nähdään taustana ja toiset hahmoina taustan edessä. Hahmon erottumiseen taustasta vaikuttavat kaikki edellä mainitut hahmolait, ja lisäksi pienet kohteet suurempien edessä nähdään yleensä hahmoina. [3, s. 181–190.] Kuvassa 9 on esimerkit kahdesta hahmolaista.

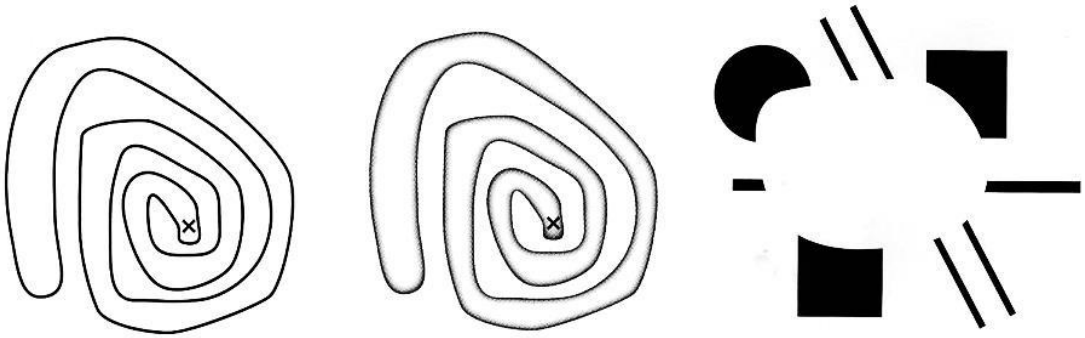


Kuva 9. Vasemmalla toisiinsa kytketyt havaitaan vahvemmin toisiinsa liittyvinä kuin a) läheiset b) saman väriset c) saman kokoiset ja d) saman muotoiset kohteet. Oikealla taas esimerkki miten pehmeät kytkökset ovat helpompi hahmottaa kuin kulmikkaat. [3, s. 184.]

Hahmontunnistusta voidaan helpottaa jakamalla visualisaation elementit eri visuaalisille kanaville. Eri kanavia ovat suuntaus, koko, tekstuuri, väri, stereoskopinen syvyys ja liike. Tämän takia väri ja muoto ovat prosessoitavissa erikseen ja helposti erotettavissa, eikä toisen avulla ilmaistu tieto häiritse toista. Eri tavoin prosessoiduille visuaalisille ominaisuuksille muodostuvat myös eri kohdekartat, ja niitä on nopeampi etsiä. Esimerkiksi punaiset pisteet on helppo etsiä mustien neliöiden seasta mutta hankalaa, mikäli punaisia pisteitä etsitään mustien ja punaisten neliöiden seasta. [3, s. 143–159.]

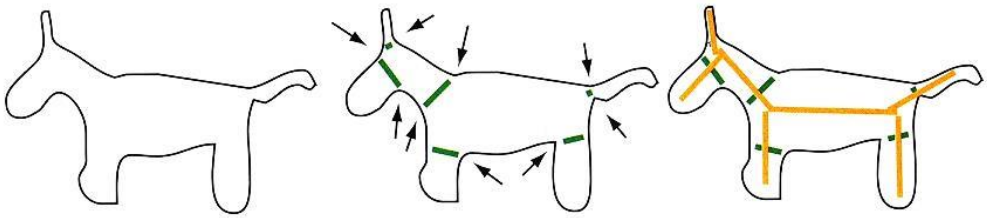
Mikäli elementtejä ei jaeta toisistaan tarpeeksi erottuville kanaville, voi saman kanavan elementtien vuorovaikutuksesta syntyä informaatiota sisältämättömiä hahmoja. Odottamattomia hahmoja voi syntyä myös visualisaation ja hallintakäyttöliittymän vuorovaikutuksesta. Esimerkiksi kahden yhdensuuntaisen mustan viivan väliin syntyy kolmas valkoinen viiva, joka ei viesti mitään, mutta kiinnittää huomiota. [6, s. 53, 61.]

Aivot ovat erityisen herkät pitkille jatkuville reunoille kahden alueen välissä, ja nämä alueet ovatkin näin erittäin kriittisiä visualisaatioissa. Reunan voi määritellä viiva, raja kahden alueen välissä, liikehahmo tai raja tekstuurissa. Reunoja voidaan myös nähdä, vaikka niitä ei olisikaan, kuten kuvassa 9 oikealla. Esimerkiksi vektorikentän esittämisessä on hahmojen erottamisen kannalta hyödyllistä, mikäli vektoriviivat muodostavat jatkuvia reunoja ja ääriviivoja. [3, s. 191–198.] Monimutkaisten reunojen havaitsemista voidaan tehostaa varjostamalla reunojen sisä- tai ulkopuolet liukuvalla tummennuksella, kuten kuvassa 10 keskellä. Suljettu muoto erottuu taustasta huomattavasti helpommin tällä menetelmällä. Toisaalta myös muodon erottumista taustasta voidaan tehostaa värittämälle sen reunat ikään kuin hohtaviksi. [3, s. 76–77.]



Kuva 10. Keskimmäisestä kuvasta on paljon helpompi nähdä, onko x ääriviivojen sisä- vai ulkopuolella, kuin vasemmasta. Oikealla pyöreän kappaleen ääriviiva muodostuu vaikka sellaista ei kuvassa ole. [3, s. 77, 192.]

Kohteen rakenteen tunnistaminen perustuu nykykäsityksen mukaan siluetteihin ja geoneihin. Visuaalisesta informaatiosta etsitään ensin reunat, sitten komponenttien akselit, solmupisteet ja massakimpaleet. Seuraavaksi tunnistetaan geonit eli keskeiset kolmiulotteiset elementit kuten kartiot, sylinterit ja laatikot. Tämä perusteella tutkitaan, miten geonit liittyvät toisiinsa, ja lopuksi kohde tunnistetaan. Siluetin tunnistus on erityisen tärkeä osa rakenteen tunnistusta ja selittää esimerkiksi kykyämme tulkita ääriviivapiirustuksia. Siluetista voidaan tunnistaa ääriviivojen ja etenkin koveruuden perusteella kohteen rakenteellinen muoto. Siluetin koveruudesta voidaan päätellä myös kohteen komponentit ja se, miten ne ovat kiinnittyneet toisiinsa, kuten kuvassa 11 esitetään. Geoni- ja siluettiteorioista seuraa, että joitain yksinkertaistettuja esityksiä on helpompi tulkita kuin alkuperäisiä kuvia. Viivapiirros saattaa esimerkiksi olla nopeampi tulkita kuin valokuva ja geoneista rakennettu diagrammi helpompi muistaa kuin perinteinen laatikkodiagrammi. [3, s. 299–306.]

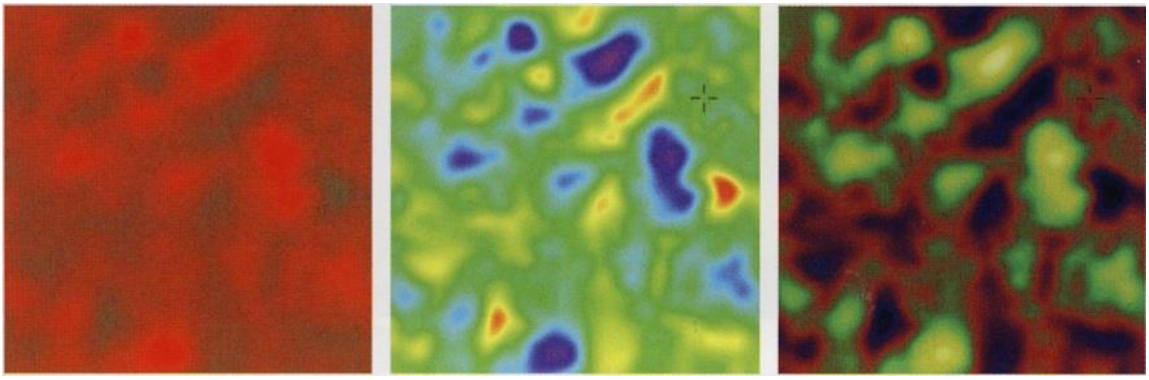


Kuva 11. Yksi tapa tunnistaa siluetista kohteen rakenne. Ensin kuvasta etsitään koveruudet, ja niiden perusteella rakenteen runko. [3, s. 302.]

Liikehahmojen tunnistamista ei tunneta yhtä hyvin kuin staattisten, mutta olemme niille erittäin herkkiä, etenkin suhteelliselle liikkeelle. Rajoja ja ääriviivoja voidaan havaita pelkän liikkeen perustella. Liike onkin erittäin alikäytetty visualisaation menetelmä. Liikettä voidaan pitää visuaalisen kohteen ominaisuutena samalla tavoin kuin väriä, ko-

koa ja asemaa. Mikäli data koodataan liikkeen vaiheeseen, se on tyypillisesti helpoimmin luettavissa. Liike voi paljastaa ryppäitä moniulotteisesta data-avaruudesta, ja aivot jakavat usein liikkuvat kohteet hierarkkisiin ryhmiin. Herkin ihmisen havaintokyky on noin 0,5–4 cm sekunnissa eteneville kohteille, katseltuna normaalilta näyttöetäisyydeltä. Tärkeä liikehahmo on kausaalinen liikkeen aiheuttaminen, esimerkiksi mikäli kahden kohteen törmätessä törmäyksen vastaanottaja liikkuu 70 millisekunnin kuluessa, hahmotetaan sen liike törmääjän aiheuttamaksi. Liikkeeseen saatetaan myös lukea inhimillisiä tunteita. [3, s. 229–236.]

Värien käyttäminen visualisaatioissa on monimutkainen osa-alue, josta tässä voidaan esittää vain muutama hyödyllinen periaate. Ensinnäkin värit ovat erittäin hyödyllisiä kohteiden erotteluun ja nimeämiseen, mutta huonoja muodon, yksityiskohtien tai avaruudellisen asettelun viestimiseen. Selvää on että värikylläisiä värejä tulisi käyttää vain pienten symbolien tai viivojen koodaamiseen ja himmeämpiä värejä (tai esimerkiksi pastellisävyjä) suurilla alueilla. Pienillä värikoodatuilla symboleilla tulee kuitenkin olla tarpeeksi suuri luminanssikontrasti ja kromaattinen ero taustaan. Toinen vaihtoehto on käyttää ääriviivoja. Mikäli symboleita koodataan väreillä, ei pitäisi käyttää kymmentä väriä enempää, sillä sitä suuremmat määrät eivät ole enää nopeasti havaittavissa. Väristä on myös hankala lukea arvoa luotettavasti, ja esimerkiksi datakarttaa piirrettäessä on datan muodot helpoin erottaa pelkän luminanssin avulla harmaan sävyistä koostetusta kartasta. Mikäli kartasta pitää lukea arvoja värisävyn perusteella, sopivinta on käyttää spiraalivärisekvenssiä, joka kiertää eri värisävyjen läpi, ja nousee koko ajan luminanssissa. Jos värisekvenssi etenee keltainen-sininen-suunnassa, se on myös värisokeiden luettavissa. Kuvassa 12 tällaista sekvenssiä on käytetty oikealla. On myös huomionarvoista, että värien käyttö saattaa vaihdella kulttuurista toiseen. [3, s. 95–131.] Hyvä strategia värien valintaan visualisaatiota suunniteltaessa on käyttää harmonista ja vaaleaa luonnosta löytyvää väripalettia [6, s. 90].



Kuva 12. Sama data ensin saturaatiolla, sitten spektrillä ja kolmanneksi spiraalisekvenssillä ilmaistuna. Oikeanpuoleisesta on helppo erottaa datan muoto, arvot sekä maksimi ja minimi [3, s. 131].

Perinteisissä kaksiulotteisissa visualisoinneissa on selvät etunsa, ja voimakkaimmat hahmonetsintämekanismit toimivat kahdessa ulottuvuudessa. Tyypillisesti kolmiulotteinen visualisaatio voidaan aina esittää kahdessa ulottuvuudessa. Kolmiulotteisiin visualisointeihin pätevät omat monimutkaiset sääntönsä, joita tässä on mahdollista vain sivuta. Kolmiulotteisen avaruuden hahmottamisen pohjana ovat niin sanotut syvyysvihjeet. Tärkeimpiä niistä ovat lineaarinen perspektiivi, koon ja tekstuurin kaltevuus, varjostukset ja tarkennuksen syvyys. Toisaalta myös liikkeestä ja stereoskopisesta syvyydestä voidaan päätellä kohteen paikka. Kolmiulotteista visuaalisointia suunniteltaessa tulisi syvyysvihjeitä antaa paras mahdollinen yhdistelmä. Pintojen muotoja on helpompi hahmottaa, mikäli ne ovat teksturoituja esimerkiksi ruudukolla ja varjostettuja. Valaistuksen havainnollisuus tulee myös ottaa huomioon. Kappaleiden muoto on helpoin hahmottaa kun joko valonlähde tai kappale on liikuteltavissa ja valon voi säätää heijastumaan kriittisistä kohdista. [3, s. 239–291.]

2.4 Verkon olosuhteet

Verkkoon suunniteltu visualisaatio toimii parhaillaan käyttöliittymänä ihmisen hahmontunnistus- ja päätöksentekojärjestelmän sekä internetin valtavien tietovarantojen ja laskentakyvyn välillä [3, s. 2]. Mikäli se toteutetaan moderneille, eli käytännössä HTML5-standardin [8] suurilta osin toteuttaville selaimille, se on riippumaton käyttöjärjestelmästä ja selauslaitteesta. Tällöin käyttäjän ei tarvitse asentaa selaimeensa minkäänlaisia liitännäisiä ja voi vain navigoida oikeaan osoitteeseen. Visualisaatio on näin myös helppo jakaa eteenpäin. Tässä työssä toteutettava visualisaatio rakennetaan moderneilla selaimilla katseltavaksi, ja sitä varten on syytä esitellä joitakin verkkosovel-

lusten suunnittelun pääpiirteitä.

Eri laitteilla ja selaimilla tarkasteltuna ei datavisualisaatioita kuitenkaan voi suunnitella vain yhdelle näyttökoolle tai resoluutiolle, vaan sitä on usein voitava tarkastella niin älypuhelimen pienellä kuin pöytäkoneen suuremmalla näytöllä. Näyttölaitteet yleisesti rajoittavat tarkkuutta verrattuna esimerkiksi painettuun visualisaatioon. Selaimelle ohjelmoitaessa voidaan esitys kuitenkin tehdä suurennettavaksi ja tarkennettavaksi, mikä mahdollistaa useat eri katselusyvyydet. Selaimessa käyttäjällä on mahdollisuus valita ja personoida esitettävä data.

Selaimessa visualisaatiolla on mahdollisuus hyödyntää selaimen tiedonsiirto-ominaisuuksia, joten visualisaatio voi olla jatkuvasti yhteydessä uusimpaan datajoukkoon ja päivittää sen välittömästi näkyviin. Tarkastelijan on myös yleensä mahdollista tutkia selaimessa visualisaation lähdekoodia ja sen käyttämiä datajoukkoja.

Suunniteltaessa datavisualisaatiota selaimella käytettäväksi on hyvä ottaa huomioon muutama verkkosuunnittelun periaate. Mikäli sovellukseen tarvitsee syöttää jotain, tulee syötön tapahtua samasta käyttöliittymäelementistä kuin tulosteen. Esimerkiksi taulukkoa voi editoida suoraan sen riveillä eikä erillisestä syötekentästä. Näin ei synny keinotekoista jakoa syötön ja tulosteen välille. Käytännössä kuitenkin visualisaatiossa usein joudutaan parametrien säädöt ja muut työkalut erottamaan esityksestä. Elementtien olisi hyvä olla myös suoraan valittavissa. Raahaaminen ja tiputtaminen ovat hyvä esimerkki tällaisesta suorasta vuorovaikutuksesta. Verkkosovelluksessa on mahdollista käyttää kontekstuaalisia käyttöliittymäelementtejä, mikä on erittäin hyödyllistä visualisaatioita suunniteltaessa. Datasymbolien vieressä voi olla työkaluja niiden manipulointiin, ja symbolia painettaessa, tai vietäessä osoitin sen päälle, siitä voidaan esittää lisätietoa tai esimerkiksi sen tarkat arvot datajoukossa. [9, s. 1-74, 79–81.]

Verkkosovelluksessa on helpompi navigoida, mikäli toiminnot eivät aiheuta sivulta pois siirtymistä. Toiselle sivulle navigoinnin sijaan voidaan sivun päälle vaikkapa avata pienempi ikkuna tai dialogi. Sivulla voidaan käyttää myös erilaisia upotteita, jotka paljastavat lisätyökaluja tai -informaatiota. Upotteet eivät peitä osaa sivusta kuten dialogit ja pienet ikkunat. Sisältöä voidaan jakaa eri välilehtiin, mikä myös poistaa turhaa sivujen välillä navigointia. [9, s. 105–136.] Visualisaatio voidaan asettaa karttasovellusten kaltaiseen panoroitavaan ja tarkennettavaan elementtiin, jolloin käyttäjä voi vapaasti seikkailla sen kaksiulotteisella pinnalla. Tällöin uloimmalla katselutasolla voidaan näyttää erittäin suuria määriä dataa ja tarjota yleiskuva datasta, kuten luvussa 2.2.1 esitettiin. [9, s. 149–155.]

Mahdollisten toimintojen tulisi antaa jonkinlainen kutsu käyttämään niitä. Kutsu voidaan saada aikaan johdattelevalla tekstillä tai esimerkiksi erottuvalla värityksellä. Osoittimen siirtäminen elementin päälle voi myös tuottaa dynaamisen kutsun, kuten pienen animaation, tai tuoda esiin tekstuaalisen vihjeen sen käytöstä. Huomionarvoista on, että käyttöliittymää kosketuslaitteella käytettäessä ei voida hyödyntää osoittimen kohdistamisen tapahtumia. Interaktion tarjoavien elementtien tulisi jollakin tavalla ilmaista käyttömahdollisuutensa, eikä tähän ole yksiselitteistä tai universaalia ratkaisua. [9, s. 181–213.]

Mikäli visualisaatiossa tai sitä ympäröivässä käyttöliittymässä tapahtuu tilan muutoksia, ne tulee esittää siirtymien avulla. Elementtejä voidaan esimerkiksi värittää, himmentää ja kirkastaa tilan muuttuessa tai huomiota ohjata himmentämällä kaikki paitsi haluttu huomion kohde. Mikäli elementti ei ole käytössä tai siihen ladataan sisältöä, se voidaan värittää himmeäksi. Tilan muutoksia voidaan esittää myös animaatioilla. Animaatiot ovat käyttökelpoisimpia osoittamaan sisällön poistumista, näkyviin tuloa sekä kohteen paikan tai ympäristön muutosta. [9, 217–221, 228–231.]

3 Selain kehitysalustana

3.1 Modernien selainten kyvyt

Työssä esimerkkinä toteutettava visualisaatio rakennetaan katseltavaksi moderneille verkkoselaimilla. Eri valmistajien selaimet ovat kuitenkin harmillisen monissa tekniikoissa ja standardeissa epäyhteensopivia, ja teknologioiden yhteensovittaminen eri selainten välillä vaatii usein ylimääräistä työtä. Tässä luvussa tutkitaan ensin hieman sitä mihin modernit selaimet kykenevät, sekä lyhyesti millaisilla työkaluilla ohjelmointityötä niille tehdään ja mitä kehitykseen sisältyy. Kyvyistä ja työkaluista valitaan visualisaation toteuttamiseen vain tarpeelliset, mutta ne on kuitenkin hyvä esitellä yleiskatsauksella.

Moderneilla selaimilla tarkoitetaan tässä yhteydessä HTML5-standardin ja siihen liittyvät tekniikat suurelta osin toteuttavia selaimia, joista käytännössä suosituimpia ovat Google Chrome, Mozilla Firefox, Safari, Internet Explorerin versiot 9:stä eteenpäin ja Opera. Termiä ”HTML5” käytetään usein kuvaamaan varsinaisen standardin lisäksi kaikkia muita siihen liittyviä tekniikoita kuten CSS3-tyylittelyominaisuuksia ja selaimen JavaScript-ohjelmointirajapintoja. Selaintekniikka on kuitenkin nopeasti liikkuva ja mo-

nimutkainen kohde, jota määrittelevät useat eri standardit sekä selainvalmistajien omat tekniikat. Tekstissä yritetään tasapainoilla uuden ja vakiintuneen tekniikan välillä, joten yleistykset ja lievät epätarkkuudet ovat väistämättömiä. Toinen selvennystä vaativa asia on verkkosovellusten terminologia. Selainsovelluksella tarkoitetaan tässä rikasta modernissa verkkoselaimessa toimivaa sovellusta, joka saattaa – ei kuitenkaan välttämättä – kommunikoida palvelimen kanssa. HTML5 ja siihen liittyvät tekniikat ovat mahdollistaneet itsenäisemmät asiakkaan puolella selaimessa suoritettavat sovellukset, joissa palvelimen rooli keskittyy datan varastoimiseen ja jakamiseen.

Selainsovellus kykenee varastoimaan ja käsittelemään tietoa usealla eri tavalla ja tekniikalla. Asiakaspuolen paikallisen tietokantaperusteisen tallennuksen tarjoavat Chrome, Safari ja Opera WebSQL-rajapinnalla, joka tuo sovelluksen käytettäväksi relaatiotietokannan. WebSQL:n tulevaisuus on kuitenkin epäselvä. Yleisemmin tuettu ja käytetty rajapinta on Web Storage, johon tietoa varastoidaan nimi-arvoparina. Web Storage jakautuu session- ja localStorage -rajapintoihin, joista ensimmäinen on istuntokohtainen ja toinen istuntojen välillä toimiva. Kolmas tietokantarajapinta on IndexedDB, joka on Web Storagen kaltainen mutta indeksoitu nimi-arvoparitietokanta. Sovellus voi toki myös tallentaa asiakasselaimeen dataa perinteisten keksien avulla. File Access -rajapinta taas antaa sovellukselle kyvyn tiedostosyötteiden käsittelyyn. [10.]

Selain kykenee useisiin tiedonsiirtomenetelmiin. Kyky asynkronisiin hypertekstin siirtorajapinnan (HTTP) -kutsuihin oli jo kauan ennen HTML5:tä, mutta se on yhä tärkeämpi keino selainsovelluksissa välittää sisältöä ilman sivulatauksia. HTML5-standardiin sisältyvät palvelimen lähettämät tapahtumat (Server-sent Event) mahdollistavat palvelimen tapahtumien vastaanottamisen selainsovelluksessa. WebSocket-protokolla ja -ohjelmointirajapinta mahdollistavat kevyen ja tehokkaan TCP-yhteyden käyttämisen suoraan selaimesta, joka helpottaa esimerkiksi reaaliaikaisten pelien tai keskusteluohjelmien rakentamista. [10.]

Uudet multimediaominaisuudet ovat toinen huomattava osa-alue. Selain kykenee ilman liitännäisiä toistamaan ja manipuloimaan ääntä sekä videota Audio Data- ja Timed Track -rajapinnoilla. Audio Data -rajapinta mahdollistaa myös äänen ohjelmallisen tuottamisen. Vielä merkittävämmin pystyvät selainsovellukset tuottamaan pikseligrafiikkaa Canvas-elementillä, vektorigrafiikkaa SVG-elementillä ja monipuolista grafiikkaa uusilla tyylittelyominaisuuksilla (”Cascading stylesheets” (CSS) ja sen 3. versio). Reaaliaikaista laitteistokiihdytettyä kolmiulotteista grafiikkaa pystytään tuottamaan WebGL-rajapinnalla. [10.]

Rajapintoja ja tekniikoita on lukuisia muitakin, esimerkiksi mobiilisovelluksia varten

suunnitellut paikkatieto-, laitteen orientaatio-, akkuteho- ja kosketustapahtumarajapinnat. Alussa mainituista selaimista kaikki muut paitsi Internet Explorer toteuttavat myös Web Workers -rinnakkaisohjelmointirajapinnan, jonka avulla selain voi jakaa skriptejä suoritettavaksi samanaikaisesti usealla suoritinytimellä. Kaikkia yllä mainittuja ominaisuuksia ja rajapintoja voidaan hyödyntää rikkaampien ja interaktiivisempien visualisaatioiden toteuttamisessa. [10.]

3.2 Kehitystyökalut

Verkon ohjelmointikieli on JavaScript. Työssä toteutettavan visualisaation ohjelmoimiseen tarvitaan JavaScriptiä, ja tässä luvussa esitellään siksi sen pääpiirteet. Kaikki modernit verkkoselaimet pöytätietokoneilla, pelikonsoleissa, tableteissa ja älypuhelimissa sisältävät JavaScript-tulkin (jota kutsutaan myös moottoriksi), ja Javascript onkin näin maailman käytetyin ohjelmointikieli. Selaimessa HTML määrittelee verkkosivun sisällön, CSS sen esitystavan ja JavaScript sen toiminnallisuuden ja käyttäytymisen. JavaScript on hieman epäonnisesti nimetty: sillä ei ole mitään tekemistä Javan kanssa, ja se on jo kauan sitten kasvanut skriptikielitaustastaan vakaaksi ja tehokkaasti yleiskäyttöiseksi ohjelmointikieleksi. [11, s. 1.]

JavaScript on korkean tason dynaamisesti tyypitetty tulkittu ohjelmointikieli, joka soveltuu oliopohjaiseen ohjelmointiin sen prototyyppiperustaisten olioiden avulla. Prototyyppi on käytännössä nimi-arvo-assosiaatiotaulu, joiden arvot ovat olioiden data-arvoja tai metodeja. Kielessä ei ole varsinaisia luokkia, mutta luokkien kaikki toiminta voidaan toteuttaa niin halutessa JavaScriptin olioilla ja niiden prototyypeillä. Kieli tukee myös funktionaalista ohjelmointia luontevasti anonyymien ja mahdollisten ajonaikaisesti määriteltävien funktioiden avulla. Funktioita voidaan antaa argumenteiksi toisille funktioille, niitä voidaan palauttaa funktioista ja käsitellä kuten mitä tahansa muuttujia. [11, s.1, 115, 163.]

JavaScriptin kielen ydin määrittelee minimaalisen rajapinnan tekstin, tietorakenteiden, päivämäärien, säännöllisten lausekkeiden ynnä muun käsittelyyn, mutta syöttö ja tulostus sekä kehittyneemmät ominaisuudet kuten tiedonsiirto, tallennus ja grafiikka ovat isäntäympäristön vastuulla. Verkkoselaimessa tulokohta kaikille sen toiminnoille on globaali nimiavaruus, eli Window-olio, jonka alla kaikki määritellyt toiminnot ja muuttujat oletusarvoisesti ovat. Window-oliolla on allansa esimerkiksi Document-olio, joka edustaa rajapintaa varsinaiseen ”verkkosivuun” ja tarjoaa työkalut sen käsittelyyn. JavaScript ajetaan aina dokumentin kontekstissa, ja se liitetään HTML-dokumenttiin sen

<script>-elementteihin joko omina tiedostoinaan tai suoraan dokumentin sekaan. [11, s. 2, 307, 311.]

Verkkosivustot jaetaan yleensä kahteen kategoriaan. Ensimmäinen kategoria on hypertekstiä sisältävät verkkodokumentit, joiden esitys on melko staattinen. Näissä JavaScriptillä päästään käsiksi Document-olioon ja sen sisältämiin Element-oliohin, ja dokumentin rakennetta ja esitystä voidaan näin dynaamisesti muokata. JavaScriptillä voidaan myös antaa dokumentille toiminnallisuutta rekisteröimällä tapahtumakäsittelijöitä. Dokumenteissa JavaScriptin tulisi keskittyä parantamaan selauskokemusta hienovaraisesti, eikä dokumentin tulisi riippua siitä. Sisältöä voidaan animoida ja lisätä siihen visuaalisia efektejä, tarjota mahdollisuus järjestää sisältöä haluttuun muotoon, tai piilottaa halutessa näkyvistä.

Toinen verkkosivustojen kategoria ovat verkkosovellukset. Näissä hyödynnetään dokumenttien jakelun lisäksi kaikkia selaimen tarjoamia ominaisuuksia, joita esiteltiin edellisessä luvussa. JavaScriptin ja selaimen avulla voidaan näin luoda natiivisti suoritettavien sovelluksien toiminnallisuutta lähestyviä selainsovelluksia. Tällaiset sovellukset ovat täysin riippuvaisia JavaScriptistä, eikä niiden voidakaan olettaa toimivan ilman sitä. [11, s. 310, 311.]

Document Object Model eli DOM on selaimen tarjoama dokumentin rakennetta ja sen operaatioita kuvaava rajapinta. DOM on perustavanlaatuinen rajapinta selaimelle ohjelmoitaessa, mutta jokainen selain toteuttaa rajapinnan mahdollisesti omalla tavallaan, eikä sama JavaScript-koodi välttämättä toimi halutusti kaikilla selaimilla. DOM on muutenkin monella tapaa epäonnistunut, ja tästä on perinteisesti seurannut tyytymättömyys koko JavaScriptiä kohtaan. Tätä helpottamaan on ohjelmoitu useita kirjastoja, jotka DOM:in käsittelyn lisäksi auttavat esimerkiksi AJAX-ohjelmoinnissa, johon liittyy samanlaisia selainten välisiä epäyhteensopivuuksia. Nämä kirjastot sisältävät usein muitakin hyödyllisiä työkaluja selainsovellusten ohjelmointiin, kuten animointia ja tapahtumankäsittelijoitä yksinkertaistavia toimintoja.

Modernit selaimet tarjoavat nykyään myös monenlaisia kehitystyökaluja verkkosovellusten kehittämiseen, kuten interaktiivisen konsolin, debuggerin sekä työkaluja olioiden ja tietorakenteiden prototyyppien tarkasteluun. Kehitystyökalut auttavat myös dokumenttien tyylityksen ja rakenteen kehittämisessä erilaisin tarkastustyökalujen avulla sekä auttavat tarkastelemaan dokumentin tekemää verkkoliikennettä. Työkaluista voi usein myös tarkastella dokumentin haun ja dokumentin itse tekemien HTTP-kutsujen ominaisuuksia erittäin yksityiskohtaisesti sekä tutkia dokumentin piirron ajoituksia.

Verkkodokumenttien monimutkaistuessa ja niiden muuttuessa verkkosovelluksiksi JavaScript-lähdekoodi kasvaa ja monimutkaistuu nopeasti, ja tätä helpottamaan on kehitetty useita erilaisia ratkaisuita. Koodia rakenteistetaan ja järjestellään usein jonkin suunnittelumallin mukaan, ja luonteva malli selainsovelluksille on Model-View-Controller-malli ja sen johdannaiset. Mallin toteuttamista helpottamaan on ohjelmoitu lukematon joukko nk. MV*-kirjastoja. Kirjastot tyypillisesti tarjoavat tavan jakaa ohjelma malleihin, näkymiin ja ohjaimiin. Mallit sisältävät ohjelmat tiedon ja datan, näkymät tyypillisesti käyttöliittymän (eli varsinaisen dokumentin), joka tarkkailee malleja, ja ohjain käsittelee syötteet ja päivittää malleja. Nämä kirjastot sisältävät usein myös työkaluja datan välitykseen palvelimen ja sovelluksen välillä. [12.] Usein näiden suunnittelumallien lisänä ja tukena käytetään asynkronisia moduulimäärittelyjä, eli lähdekoodi jaetaan moduuleihin, jotka asynkronisesti noudetaan niitä tarvittaessa. Tarkoitukseen on tehty useita erilaisia malleja ja kirjastoja, mutta toimintoa ei ole selaimissa vielä vakiona.

Toinen näkökulma selainsovellusten kehitykseen on ohjelmoida sovellukset jollakin toisella kielellä toteutetulla kehitysympäristöllä ja kääntää ne JavaScriptiksi ja verkkodokumentiksi. Tämänkaltaiset kehitysympäristöt abstraktoivat selainten väliset erot ja selaimen toiminnot ja rajapinnat sekä pyrkivät tekemään kehityksestä virtaviivaisempaa ja sovelluksista suorituskykyisempiä. Toisaalta kaikki JavaScriptiksi käännettävät kielet eivät ota yhtä holistista kantaa kehitykseen vaan pyrkivät parantamaan pelkän JavaScriptin kirjoituskokemusta toteuttamalla kääntäjän. Kääntäjiä on useille olemassa oleville kielille, ja lisäksi on kehitetty monia uusia kieliä, jotka kääntyvät JavaScriptiksi. Kaikki ottavat kantaa erilaisiin koettuihin puutteisiin JavaScriptissä, kuten luokattomiin oliohin, ja tarjoavat perinteisemmän oliomallin. Toiset taas pyrkivät tekemään helpommaksi vaikkapa funktionaalisen ohjelmoinnin, johon JavaScript soveltuu hyvin, ja toiset yrittävät vain yksinkertaistaa JavaScriptin syntaksia.

Selainten avoimen lähdekoodin JavaScript-moottorien kypsyessä saatiin toteutettua myös avoimen lähdekoodin palvelintoteutuksia JavaScriptille. Näistä tunnetuin ja suosituin on varmasti Node.js. Sen varaan on sittemmin rakennettu palvelinsovellusten lisäksi erittäin laaja skaala kaikenlaisia selainsovellusten kehitystyökaluja. Node.js:n avulla on toteutettu niin paketinhallintajärjestelmä kuin lukuisia kehityksen automatisointityökaluja sekä esimerkiksi syntaksitarkastajia ja koodin pakkaajia, jotka kaikki jaellaan Node.js:n pakentinhallinnalla. Useat JavaScriptiä tuottavat kääntäjät ovat myös toteutettu Node.js:llä. Näiden työkalujen avulla JavaScript-kehittäjä voi rakentaa oman natiivisti suoritettavan kehitysympäristön ja työnkulun.

Sovellusten monimutkaistuessa tarvitaan yleensä myös jonkinlainen testauskirjasto, ja

valinnanvaraa on tässäkin erittäin runsaasti. Toiset testauskehykset toimivat natiivisti, esimerkiksi yllä mainitulla Node.js:llä tai jonkinlaisella päättömällä selaimella, ja toiset toimivat selaimessa, jossa testauksen tulosta voi tarkastella sen tuottamalta verkkosivulta.

3.3 Avoin data

Työssä toteutettava visualisaatio pyritään toteuttamaan avoimia datavarantoja hyödyntäen, visualisoimalla selaimessa kaikille avointa dataa. Käytettyä dataa voi toivottavasti visualisaation tarkastelun jälkeen siirtyä tutkimaan helposti verkkoselaimella, jolle visualisaatiokin on rakennettu. Näin verkossa toisaalla sijaitseva data on esitetty visualisaatiossa toisella tavalla, joka toivottavasti tuo jonkinlaista lisäarvoa datan ymmärtämiseen.

Internet on jokapäiväinen ja kaikkialla läsnä, joten idea jakaa pelkkien dokumenttien lisäksi dataa on kerännyt suosiota ja noussut jonkinlaiseksi ilmiöksi. Visualisaatioiden suunnittelijoille avoimet datavarannot tarjoavat mahdollisuuden tuoda dataa suuren yleisön hyödynnettäväksi luomalla sille esityksiä ja tarjoamalla käyttöliittymän tähän jalostamattomana hyödyttömään resurssiin. Visualisaatio on nopein ja helpoin menetelmä tutkia ja yhdistellä avointa dataa.

Avoimeksi dataksi tai laajemmin avoimiksi tietovarannoiksi voidaan lukea sellaiset saavutettavat aineistot, jotka täyttävät Open Knowledge Foundationin ehdot vapaasta uudelleenjakelusta ja -käytöstä, teknisten rajoitteiden puutteista sekä käyttöalueen rajoittamattomuudesta. Käytännössä avointa on sellainen data, jota voidaan vapaasti jaella ja käyttää, ainoana mahdollisena rajoitteena tekijän mainitseminen tai datan muuttamattomana jakaminen. [13.]

Avoin data voi olla hyödyksi lukemattoman monilla elämän alueilla. Mikäli kyse on valtioiden datasta, sitä voidaan käyttää esimerkiksi läpinäkyvyyden, demokraattisen hallinnon ja osallistumisen parantamiseen tai toisaalta auttaa valtiojohtoja toimimaan tehokkaammin. Avoin data voi auttaa uusien tuotteiden, palvelujen ja innovaatioiden luomisessa sekä tuottaa uutta tietoa eri aineistojen yhdistelmistä. Useat tutkimukset ovat arvioineet avoimen datan arvoksi pelkästään EU:ssa useita kymmeniä miljardeja euroja. [14.] Huomattavia esimerkkejä avoimen datan jakelijoista ovat Wikipedia, Euroopan unionin Open Data Portal sekä Yhdysvaltojen ja Ison Britannian hallintojen datanavausprojektit data.gov ja data.gov.uk. Dataa jakelevat lukuisat muutkin valtiot ja

kaupungit. Avointa tieteellistä dataa taas jakavat monet yliopistot. Suomessa avoimen datan jakelussa on kunnostautunut Tilastokeskus ja monet muut viranomaiset kuten Maanmittauslaitos ja Museovirasto. Tässä työssä datalähteenä käytetään sopivan ohjelmointirajapinnan puutteen takia Tilastokeskusta ja Wikipedia-sivuja.

Käyttökelpoisimmillaan avointa dataa jaellaan linkitettynä datana tai HTTP:n avulla toimivalla ohjelmointirajapinnalla, jolloin se on suoraan esimerkiksi verkkoselaimen käytettävissä ja linkitettävissä. Monissa tapauksissa data on kuitenkin yhä jonkinlaisena raakana taulukkodatatiedostona. Linkitetyssä datassa käytetään verkkoa linkitettyjen hypertekstidokumenttien lisäksi jakamaan koneluettavaa ja toisiinsa linkitettyä Resource Description Framework (RDF) -mallin mukaista dataa. RDF-aineistoihin voidaan tehdä hakuja ja muodostaa niistä yhdistelmiä tähän tarkoitetulla kyselykielellä. [15.]

4 Visualisaation suunnittelu

4.1 Kartogrammi

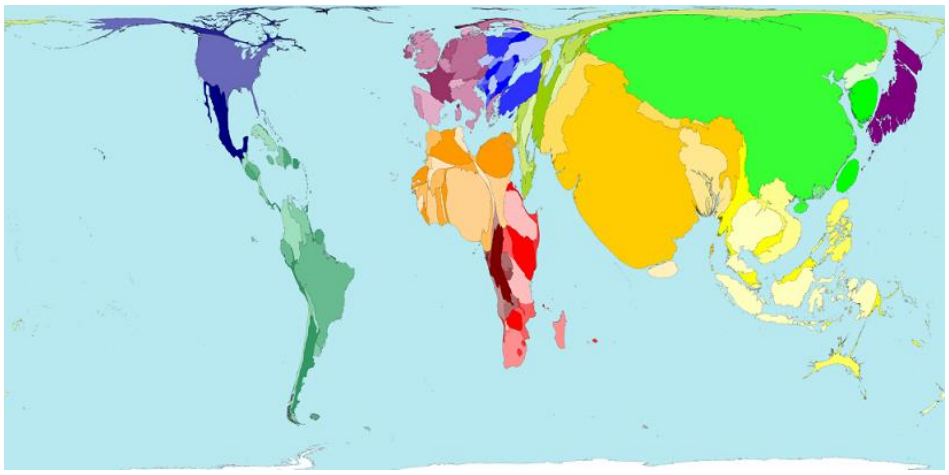
Työn tavoite on suunnitteluperusteiden koostamisen lisäksi ohjelmoida tapausesimerkkinä toimiva visualisaatio. Edellisissä luvuissa on kerätty kaikki tarvittava visualisaation suunnittelua varten sekä esitelty työkalut, joilla toteutus tapahtuu. Seuraavat luvut kuvaavat toteutettavan visualisaation suunnittelun sekä miten ja millä työkaluilla kehitystyö tapahtui.

Esityksen pohjaidea on niin kutsuttu kartogrammi, joka tässä luvussa esitellään ensin yleisellä tasolla, ja lopuksi siitä johdetaan suunnitelma toteutettavasta visualisaatiosta, joka on hieman pidemmälle viety ja abstraktimpi kartogrammi. Tämän toteutuksessa kokeillaan perinteisestä poikkeavaa menetelmää, jossa kartogrammi rakennetaan ilman varsinaista karttadataa pelkkien kartan alueiden vierekkäisyyksien avulla. Kartan alueet vetävät toisiaan puoleensa simuloitujen vetovoimien avulla.

Kartoissa on yleensä suora yhteys maailman ja kartan osien välillä. Maailman loputtomista muodoista vain muutama voidaan valita esitettäviksi kartalla, ja kun kartasta yritetään tehdä tarkka miniatyyri kohteestaan, syntyy välttämättä vääristymiä, mitä ei kuitenkaan pidetä kovin tärkeänä seikkana karttoja piirrettäessä. Mikäli kartta piirretään tarkoituksella jonkin muuttujan vääristämäksi, sitä kutsutaan kartogrammiksi. Kartogrammissa kartan alueiden kokoa (tai joskus etäisyyttä ja muotoa) muutetaan sen arvon mukaan tarkastellussa datajoukossa, tyypillisenä esimerkkinä kunkin alueen asu-

kasluvun mukaan. Tavalliset kartat voidaan näin nähdä pinta-alakartogrammeina, sillä niissä alueet piirretään suhteessa toisiinsa niiden kattaman alan mukaan.

Kartogrammeja voi piirtää moneen tarkoitukseen. Mikäli alan määrittävänä muuttujana on esimerkiksi asukkaiden vauraus, muuttuu maailmankartta dramaattiseksi esitykseksi maailman tilasta. Asukasluvun mukaan piirretty maailmankartta taas saattaisi toimia oikeudenmukaisempana päätöksentekovälineenä. Kartogrammi voi myös toimia opastekarttana kuten metrokartat, joissa maantieteellisen paikan esittäminen on vähemmän tärkeä kuin reittiverkoston luettavuus mutta joista paikka on kuitenkin usein pääteltävissä. Kartogrammilla on helppo esittää monia konsepteja, jotka tavallisella kartalla olisivat hankalia. [16, s. 4-5.] Esimerkki kartogrammista on kuvassa 13. Tässä maiden pinta-aloja on muuteltu niiden asukkaiden työmatkojen kestoilla.

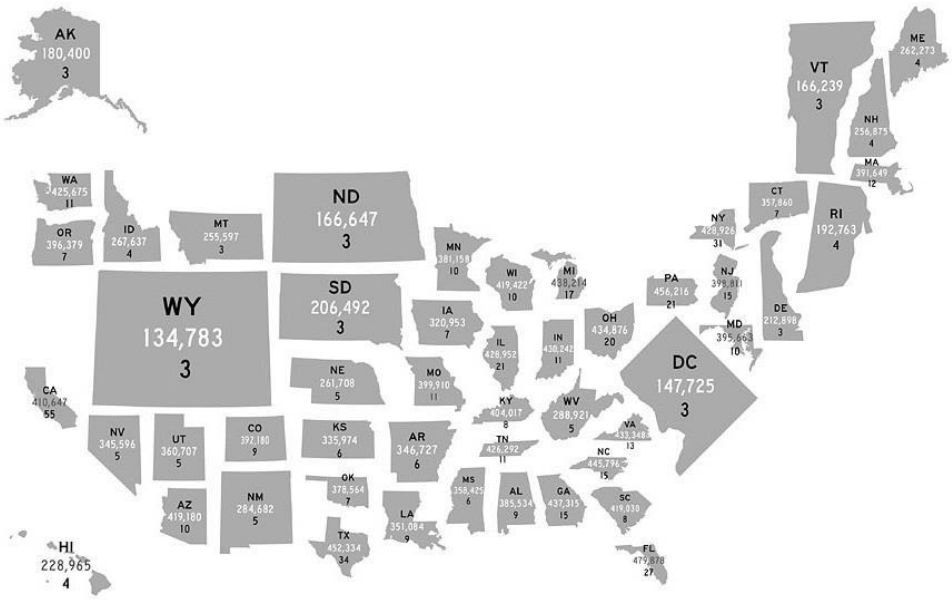


Kuva 13. Kartogrammi työmatkojen kestoista maittan [17].

Karttojen tekijät ovat aina joutuneet valitsemaan, millä tavoin he vääristävät maapalloa esittääkseen haluamansa informaation. Projektiota, eli pallon tai ellipsoidin pituus- ja leveyspiirien muunnosta tasoon, valitessaan piirtäjä ottaa tietyn näkökulman, joka aina liioittelee jotain osuutta kartasta ja joutuu näin tekemään kompromisseja eri tavoitteiden välillä. Tästä seuraa useita erilaisia kuvia maapallosta, jotka aina muokkaavat sitä, millä tavoin näemme maailman. Kartogrammeissa vääristymää ei nähdä kompromissina, vaan sillä viestitään dataa. [16, s. 9.]

Kartogrammia, joka yrittää säilyttää karttapiirteiden rajojen oikeellisuuden kutsutaan jatkuvaksi (”contiguous”), ja niiden piirtämiseen on kehitetty useita tietokonealgoritmeja. Tekniikat saattavat esimerkiksi yrittää säilyttää piirteiden muodon ja kulman toisiinsa nähden mahdollisimman tarkkana. Kartogrammi ei välttämättä yritä säilyttää rajoja, joten sitä kutsutaan epäjatkuvaksi (”non-contiguous”). Tällöin alueet eivät pysy kiinni

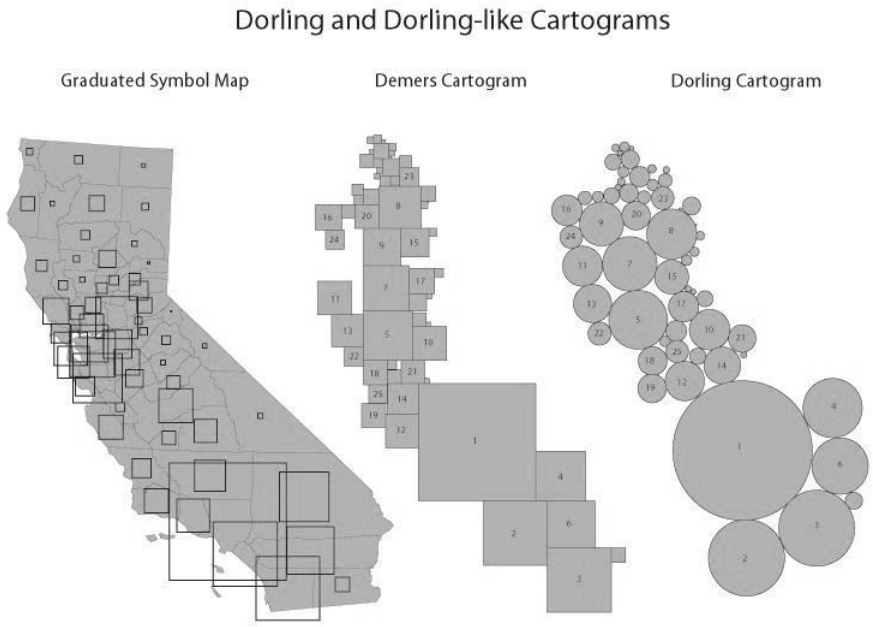
toistensa rajoissa, vaan niiden väliin syntyy tyhjää tilaa. [16, s. 32.] Kuvassa 14 on epäjatkuva kartogrammi Yhdysvalloista, joka kuvaa, kuinka montaa äänestäjää valitut edustajat edustavat.



Kuva 14. Epäjatkuva kartogrammi vaalituloksista [18].

Oikeat matemaattiset ominaisuudet säilyttävä kartogrammi ei kuitenkaan välttämättä ole käyttökelpoisin vaihtoehto kartografisiin tarkoituksiin. Mikäli alueiden rajojen muodolla ei ole merkitystä, tai niiden halutaan olevan mahdollisimman selkeitä, voidaan alueet piirtää yksinkertaisina muotoina. Ympyrää käytettäessä alueen muotona kutsutaan kartogrammia ympyräkartogrammiksi tai Dorlingin kartogrammiksi, ja se on epäjatkuva. Toinen käytetty vaihtoehto ovat neliöt, joista on esimerkki kuvassa 15. Ympyräkartogrammissa alueympyröihin kohdistetaan simuloituja voimia, jolloin ne eivät voi peittää toisiaan ja toisaalta jättävät väliinsä mahdollisimman vähän tyhjää tilaa. Alueet hylkivät eräänlaisen painovoimamallin mukaisesti ympyröitä, jotka ovat niiden päällä, mutta vetävät puoleensa ympyröitä, joiden kanssa ne jakavat rajaa. Näin syntyneessä kartassa ovat maantieteelliset naapurit luonnollisesti lähellä toisiaan, ja joitakin ulkorajojen muotoja säilyy. [16, s. 32.]

Dorlingin kartogrammi näyttää selkeästi miten muuttujan arvot ovat jakaantuneet, sillä yksinkertaisten muotojen kokoja on huomattavasti helpompi vertailla keskenään kuin muodon säilyttävissä kartoissa. Mikäli kartan alueiden määrä kasvaa, vähentyvät topologian rikkovat piirteet ja kartta on lähempänä alkuperäistä. [16, s. 34.] Esimerkki Dorlingin kartogrammista on kuvassa 15 oikealla.



Kuva 15. Dorlingin kartogrammi (oikealla), ja sen kaltaisia kartogrammeja [19].

4.2 Suomen seutukuntadatan visualisointi

Työssä toteutettava visualisaatio on epäjatkuva ympyräkartogrammi Suomen seutukunnista, jotka ovat muutaman kunnan muodostamia aluekokonaisuuksia, jotka liittyvät toisiinsa kuntien välisen yhteistyön ja työssäkäynnin perusteella. Visualisaatiossa Suomen seutukunnat esitetään ympyröinä, joiden alaa muutellaan seutukuntaa vastaavan data-arvon mukaan,ja kytketään toisiinsa viivalla. Viiva kuvaa seutukuntien vierekkäisyyksiä. Näin syntyvästä visualisaatiosta näkee nopeasti väestön jakauman eri puolille maata, väestönkehityksen trendit ja toisaalta poikkeavat havainnot.

Visualisaatiossa kokeillaan myös, soveltuuko syklitön suuntaamaton verkko Dorlingin kartogrammin tietorakenteeksi. Tämä verkko G on järjestetty pari (V, E), jossa

- V on verkon solmujoukko – eli Suomen seutukuntien joukko – joissa kaikilla alkioilla on pinta-ala, nimi ja mahdollista lisädataa
- E on verkon kaaret, eli joukko kaksialkioisia V:n aitoja osajoukkoja jotka ilmaisevat seutukuntien naapuruussuhdetta (jaettua rajaa). Kaariin voidaan kuitenkin tarvittaessa lisätä ominaisuus ”etäisyys”.
- Verkossa ei sallita silmukoita: siksi E:n alkion seutukunnat eivät saa olla samat.

Osa Suomen seutukunnista (Ahvenanmaalla) on saaria, joten etäisyysarvoa tarvitaan joissain tapauksissa niitä yhdistäville kaarille. Verkon rakenne ei määrää sen esitysta-

paa, joten visualisaation toteutuksessa solmut asetetaan aina mahdollisimman lähelle toisiaan mahdollisimman lyhyellä kaarella, käyttäen Dorlingin kuvaamien kaltaisia simuloituja voimia [16, s. 32]. Esitys saattaa vaatia myös muita korjauksia. Suomen topologinen muoto häviää joiltakin osin, esimerkiksi pohjoisessa. Mikäli ympyrään mahtuu, siinä näytetään seutukunnan nimi ja datan arvo.

Kartta rakennettaessa tällä menetelmällä ei tarvita mitään varsinaista paikkadataa: kunkin seutukunnan ”sijainti” ilmoitetaan vain epäsuorasti sen suhteilla muihin. Menetelmä toimii parhaiten eheillä muodoilla, ja etenkin Suomen eteläinen puolikas on erittäin yksinkertainen muoto. Mikäli verkkoon perustuva visualisaatio on täysin lukukelvoton, se joudutaan mahdollisesti toteuttamaan Dorlingin kuvaamalla sentroidimenetelmällä [16, s. 33] tai käyttämällä muita improvisoituja korjaavia keinoja.

Sovellus antaa mahdollisuuden valita eri suureita visualisoitavaksi, joten tapa, jolla data-arvo muunnetaan ympyrän säteeksi, on kriittinen. Esityksen tulee olla yhdenmukainen eri suureiden välillä, eli säteen vaihtelun tulee aina kuvata saman suurusluokan vaihtelua datassa. Eri datajoukkojen kaikki arvot normalisoidaan esitysvaiheessa alan minimin ja maksimin (jotka voivat riippua ohjelman ajoympäristön näyttötarkkuudesta) määrittämälle välille, ja ala A lasketaan siksi yhtälöllä

$$A = A_{min} + \frac{(d - d_{min})(A_{max} - A_{min})}{(d_{max} - d_{min})}$$

d on normalisoitava data-arvo

A on pinta-alan minimi

A_{max} pinta-alan maksimi

d_{min} on sen datajoukon alin arvo johon d kuuluu

d_{max} datajoukon maksimi.

Visualisaation tarkoitus on edustaa muualla verkossa olevia resursseja havainnollisessa visuaalisessa muodossa. Jokaisesta seutukunnasta on Wikipediassa oma artikkeli, jossa esitetään sama väestödata kuin rakennettavassa esityksessä. Visualisaatio kokoaa seitsemänkymmenen seutukunnan väestödatan yhteen esitykseen, jossa näkee makrotason esityksen niiden kaikkien väestömääristä, ja seutuja voi vertailla heti keskenään. Yksittäinen artikkeli näyttää väestönkehityksen yhdessä seutukunnassa, mutta visualisaatio esittää ne suhteessa toisiinsa ja paljastaa maanlaajuiset trendit helpommin.

5 Kehitysprosessi

5.1 Datan haalinta

Kuten edellisessä luvussa mainittiin, edustaa visualisaatio verkossa muualla sijaitsevia resursseja ja dataa, eli tässä tapauksessa Wikipedia-artikkeleita Suomen seutukunnista ja niistä löytyvää väestönkehitysdataa. Ihannetapauksessa data haettaisiinkin sovelluksen suorituksen aikana suoraan Wikipediasta, ja Wikipedialla onkin eräänlainen rajapinta artikkeleiden noutamiseen. Rajapinnasta kuitenkin saa vain raa'an artikkelin Wikipedian käyttämässä Wikitext-merkintämuodossa, ja tämä täytyisi käsitellä monimutkaisesti, jotta se olisi käyttökelpoista. Wikiteksti ei aina edes sisällä kaikkea artikkelin dataa, vaan se voi sijaita muualla.

Jokaisen seutukunnan artikkelissa on kuvaus sen väestön kehityksestä vuosina 1980-2010, viiden vuoden välein. Data ei ole ohjelmallisesti Wikipediasta saatavissa, mutta visualisaatio voi silti edustaa artikkeleita – data vain joudutaan haalimaan muualta. Tässä tapauksessa samasta lähteestä kuin Wikipediassa on käytetty, eli Tilastokeskuksesta. Tilastokeskuksella on verkkokäyttöliittymä, jonka avulla dataa voi hakea sen tietokannoista [20]. Tästä käyttöliittymästä haettiin CSV-tiedostona sama data jonka seutukuntien artikkelit sisältävät, eli kuntien väkiluvut vuosilta 1980-2010. Dataan otettiin lisäksi mukaan kuntien pinta-alat vertailua varten. Tilastokeskuksen data on kuntakohtaista, joten Tilastokeskuksen verkkosivuilta tarvittiin lisäksi luokitusavaimet kunnille ja seutukunnille, eli tieto, mitä kuntia kukin seutukunta sisältää. Tämä löytyi sarkaineroteltuna tekstitiedostona. [21.] Lisäksi Wikipediasta kopioitiin vielä lista seutukunnista Wikipedia-artikkelien nimeämiskäytäntöjen takia, jotta visualisaatiosta voidaan helposti linkittää artikkeleihin. Tilastokeskuksesta haettiin myös CSV-tiedosto elokuun 2013 ennakkoväkiluvuista.

Nämä neljä aineistoa koostettiin yhteen Excelissä. Luokitusavainten avulla seutukunnille laskettiin väestösumma jokaiselle vuodelle sen sisältämistä kunnista sekä kuntien yhteenlaskettu pinta-ala seutukunnan pinta-alaksi. Turha data siivottiin pois. Näin syntyi 70 verkon solmun joukko, jossa kaikilla on

- seutukunnan numeerinen tunnus Tilastokeskuksen luokitusavaimesta saatuna, jolla solmuun on helppo viitata
- seutukunnan nimi
- seutukunnan pinta-ala

- väestömäärä vuosilta 1980-2010 viiden vuoden välein
- väestömäärä vuoden 2013 elokuussa
- seutukunnan nimi Wikipedian käyttämässä muodossa.

Data muunnettiin vielä JavaScript Object Notation- eli JSON-muotoon, joka koostuu avain-arvoparioliosta, ja ne merkitään aaltosulkujen väliin. Kulmasulut merkitsevät listaa. Avain on aina lainausmerkeissä vasemmalla, ja arvo kaksoispisteen jälkeen oikealla. Data näyttää tältä:

```
{"nodes":  [
            {”sid”: 63, "seutukunta": "Etelä-Pirkanmaa", "pinta-ala": 1191.8,
            "1980":  45005,  "1985":  44507,  "1990":  43901,  "1995":  43049,
            "2000": 42141, "2005": 42395, "2010": 43191, "2013": 43361, "url-
            form": "Etelä-Pirkanmaan seutukunta"},
            {”sid”: 53, "seutukunta": "Forssa", "pinta-ala": 1482.3, "1980":
            35647, "1985": 36656, "1990": 36653, "1995": 36800, "2000": 35866,
            "2005": 35455, "2010": 35286, "2013": 34753, "urlform": "Forssan
            seutukunta"},
            { ”sid”: 175, "seutukunta": "Haapavesi-Siikalatva", "pinta-ala":
            4162.82,  "1980":  17296,  "1985":  17978,  "1990":  18151,  "1995":
            18093, "2000": 17040, "2005": 16202, "2010": 15230, "2013": 14753,
            "urlform": "Haapaveden-Siikalatvan seutukunta"},
             // ...
],
```

Verkon kaaria eli seutukuntien vierekkäisyyksiä varten ei ollut mitään automaattista menetelmää, joten ne tehtiin käsin kartan avulla. Jokainen kaari on pari lähteestä ja kohteesta, solmujen numeerisen tunnuksen mukaan, ja kaaria on yhteensä 168. Kaaren on pitänyt ilmoittaa vain yhteen suuntaan, eikä lähde- ja kohdeparin järjestyksellä ole merkitystä. Data on samassa JSON-tiedostossa solmujen kanssa ja näyttää tältä, jatkuen suoraan ylemmästä esimerkistä:

```
”links”: [
            {”source”: 196, ”target”: 197},
            {”source”: 196, ”target”: 193},
            {”source”: 196, ”target”: 191},
            // ...
]}
```

Ohjelmointirajapinnan puutteen vuoksi saattaa Wikipedia-artikkeleiden ja visualisaation

data olla jonkin ajan kuluttua toisistaan poikkeavaa, mutta tälle ei voi mitään. Uusinta väestötietoa ei myöskään haeta mistään automaattisesti, ja ellei sitä käsin lisätä verkon dataan, se on ikuisesti elokuu 2013. Mikäli se olisi mahdollista, visualisaatio olisi arvokkaampi jos se käyttäisi dynaamista dataa. Sopivat rajapinnat ovat kuitenkin vielä erittäin harvinaisia, eikä niiden rakentamiselle ole juurikaan motivaatiota. Visualisaatio kuitenkin edustaa näkemystä tällaisesta internetistä, vaikka kulissien takana data on vielä staattista.

5.2 D3.js-visualisointikirjasto

Visualisaation toteuttamiseen valittiin visualisointikirjasto Data-Driven Documents, jota kutsutaan yleensä d3.js:ksi, tai pelkästään D3:ksi. D3 on JavaScript-kirjasto, jonka avulla verkkodokumentteja voi manipuloida jonkin datajoukon perusteella. Datan voi näin visualisoida suoraan verkkodokumenttien HTML- ja SVG-elementtien avulla ja tyylitellen sen CSS:llä. D3 tarjoaa työkalut datan sitomiseen suoraan dokumentin elementteihin ja suorittaa sen jälkeen datan mukaisia muunnoksia dokumenttiin. Esimerkiksi pylväsdiagrammin voisi rakentaa sitomalla dataa HTML-taulukkoon, jolloin sarakkeita luotaisiin datan alkioiden verran ja niiden leveyttä tai pituutta muutettaisiin datan arvon mukaan. [22.]

Tätä samaa periaatetta voidaan käyttää rakentamaan lähes minkälaisia visualisaatioita tahansa; D3 ei yritäkään tarjota jokaista mahdollista ominaisuutta vaan työkalut ominaisuuksien koostamiseen, yksinkertaisten rakennuspalikoiden avulla. Näin kirjasto auttaa käyttämään CSS3:n, HTML5:n ja SVG:n ominaisuuksia niiden täysien kykyjen mukaisesti, pysyy avoimena sekä toisaalta erittäin kevyenä ja pystyy käsittelemään suurempia datasettejä. [22.]

Ensimmäinen D3:n perustyökalu on dokumentin elementtien valitseminen eli ”selection”. D3:n valintatyökalut helpottavat dokumentin muokkaamista, joka selaimen DOM-rajapinnalla on erittäin työlästä ja monisanaista ja vaatii paljon manuaalista dokumentin läpikäyntiä. D3 antaa deklaratiivisen tavan valita dokumentin osia joko yksittäin, tai valiten kaikki halutun kaltaiset elementit. Elementtejä voi valita tyypin, nimen ja CSS-luokan tai ominaisuuden perusteella. Valituille elementeille voi sen jälkeen kohdistaa lukuisia operaatioita, kuten asettaa niiden ominaisuuksia tai tyylittelyjä, rekisteröidä tapahtumankäsittelijöitä, lisätä tai poistaa elementtejä, tai muuttaa niideen teksti- tai HTML-sisältöä. DOM:iin pääsee kuitenkin aina myös suoraan käsiksi, sillä kukin valinta on vain lista sen elementtejä. [22.]

D3 antaa asettaa elementtien ominaisuuksia dynaamisesti. Tyylit, attribuutit ja muut elementin ominaisuudet voidaan asettaa funktioiksi, joilla haluttu arvo lasketaan, pelkän vakion sijaan. Usein tämä funktio perustuu käsiteltävään datajoukkoon. Data on D3:ssa aina lista arvoja. Kun valittuihin elementteihin sidotaan dataa valinnan ”data”-metodilla, ja ominaisuudelle annetaan funktio jolla se lasketaan, välitetään kuhunkin elementtiin sidottu data-arvo indeksin perusteella tälle funktiolle argumenttina. Tähän perustuu esimerkiksi ensimmäisessä kappaleessa mainittu pylväsdiagrammin rakentaminen, ja dynaamiset ominaisuudet ovat D3:n toinen perustyökalu. [22.]

Datan arvojen mukainen määrä valittuja elementtejä luodaan tai poistetaan valinnan ”enter”- ja ”exit”-valitsimilla. Kun data on sidottu valintaan, paritetaan jokainen elementti data-arvon kanssa. Mikäli elementtejä on vähemmän kuin dataa, muodostavat ylimääräiset elementit ”enter”-valinnan, joka voidaan esittää lisäämällä siihen elementtejä. Mikäli elementtejä ei siis ole yhtään, niitä luodaan data-arvojen määrän verran. Valinnan ”data”-metodia voidaan käyttää päivittämään sisältöä jos DOM-elementit ovat jo olemassa, ja ”exit”-metodilla poistaa olemassaolevia elementtejä, joita on enemmän kuin datassa arvoja. Valintojen metodien ketjuttaminen on tyypillinen tapa käyttää D3:ta. Alla oleva esimerkki tekisi kuusi dataan sidottua kappale-elementtiä ja asettaisi niiden tekstisisällön ja fonttikoon datan mukaan. [22.]

```
d3.select("body").selectAll("p")
    .data([4, 8, 15, 16, 23, 42])
  .enter().append("p")
   .style("font-size", function(d) { return d + "px"; });
    .text(function(d) { return "Olen numero " + d + "!"; });
```

D3 ei itsessään määrää esitystapaa millään tavalla, vaan se riippuu selaimen tarjoamista tyylittelyominaisuuksista. Uudet CSS-ominaisuudet voidaan siis ottaa heti käyttöön, eikä se vaadi kirjastoon minkäänlaisia muutoksia. D3:lla toteutettuja visualisaatioita on näin myös helppo debugata selaimen sisäänrakennetulla tarkastimella, kun elementit, joita D3 manipuloi, ovat niitä, joita selain ymmärtää natiivisti. [22.]

D3:n transitio-ominaisuudet antavat operaattorien välittömän soveltamisen sijaan työkalut niiden interpoloimiseen ajan kuluessa. Esimerkiksi valintojen atribuuttien ja tyylien vaihtumisen voi näin animoida. Elementin värin voi vaikkapa animoida muuttumaan sulavasti valkoisesta mustaksi tai sen koon isommaksi. Interpolaattoreita voi myös

muokata ja laajentaa tukemaan monimutkaisempia ominaisuuksia ja tietorakenteita. [22.]

Näiden perustyökalujen lisäksi D3 tarjoaa laajan valikoiman muita työkaluja. Listojen käsittelyyn on esimerkiksi lukuisia erilaisia metodeja, jotka auttavat datan kanssa työskentelyssä. Ulkoisia resursseja kuten dataa voi ladata ja käsitellä useissa eri formaateissa. SVG-grafiikan piirtämiseen on apufunktioita ja esimerkiksi ajan käsittelyyn suuri joukko funktioita. Karttadatan kuten polkujen ja projektioiden käsittelyä varten on todella laaja ja yksityiskohtainen kirjasto. Geometriakirjastolla voidaan luoda esimerkiksi Voronoin soluasetelmia tai quadtree-jakoja. [23.]

Layout-kirjasto antaa valmiit työkalut erilaisten sommitelmien tekemiseen, kuten hierarkioita, histogrammeja, piirakoita, pinoja, puita, ja toteutettavaa visualisaatiota varten erittäin hyödyllisesti voimasommitelman. Voimasommitelma on joustava verkko, jonka solmujen sijaintia muutetaan erilaisten simuloitujen fyysisten voimien perusteella. Solmuilla voi olla esimerkiksi hylkivä voima, jolloin solmut haluavat poispäin toisistaan, puoleensa vetävä voima, jolloin solmut haluavat lähelle toisiaan, ja painovoima, joka vetää solmuja alueen keskikohtaan päin, ja ne pysyvät näkyvillä. Lisäksi voidaan käyttää useita muitakin voimia kuten kitkaa tai luoda omia rajoituksia simulaatioon. [23.]

Edellä on esitelty kaikki tarvittavat työkalut joiden avulla seuraavassa kappaleessa toteutettu visualisaatio voidaan toteuttaa ja ymmärtää. D3 tarjoaa kaikki visualisaatiossa tarvittavat ominaisuudet eikä mitään muita kirjastoja tai sovelluskehyksiä tarvittu. D3 hoitaa datan lataamisen, joten kehitystyökaluja kuvaavassa luvussa esitellyt MV*-sovelluskehykset ovat tarpeettomia, etenkin kun data on vain staattinen tiedosto eikä sitä ladata ulkoiselta palvelimelta, jolloin MV* antaisi apukeinoja sen hallintaan. Lisäksi toteutettu ohjelma on niin pieni, ettei esimerkiksi moduulilataajille ollut mitään tarvetta. Mikäli kehityksessä olisi käytetty jotain JavaScriptiksi käännettävää kieltä, olisi kadotettu suora yhteys selaimen konsolin ja tarkastintyökalujen sekä lähdekoodin väliltä. D3 ja selaimen rajapinnat ja kehittäjätyökalut osoittautuivat erittäin kykeneväksi yhdistelmäksi.

5.3 Tuotetun ohjelman rakenne

Tuotetun ohjelman pohja on yksinkertainen HTML-dokumentti, sen tyylittelytiedosto, yksi tiedostollinen JavaScriptiä ja datatiedosto JSON-formaatissa. HTML-dokumenttiin kirjoitettiin valmiiksi minimaalinen käyttöliittymä datasetin vaihtamiselle ja pohja visuali-

saatiossa käytetylle selitteelle, joka näytetään visualisaation seutukuntia painettaessa. Lisäksi dokumenttiin kirjoitettiin pieni johdantoteksti visualisaatiolle. Visualisaation ja muun dokumentin sisällön tyylittely on yhdessä CSS-tiedostossa.

Varsinainen ohjelma sijaitsee yhdessä JavaScript-tiedostossa. Ohjelmassa luodaan ensin visualisaatiota varten SVG-elementti ja lisätään se dokumenttiin. Tämän jälkeen alustetaan voimasommitelma, joka asetetaan raahattavaksi hiirellä tai sormella. Varsinainen suoritus alkaa lataamalla D3:n json-funktiolla datatiedosto eli visualisoitava verkko. Verkko tallennetaan hieman käsiteltynä: verkon linkkien tulee voimasommitelmassa olla indeksejä solmujen listasta, kun ne datassa ovat seutukuntien tunnistenumeroita, joten nämä numerot vaihdetaan solmulistan indekseiksi iteroimalla solmulistaa.

Ladatun verkon arvot normalisoidaan luvussa 5.2 esitellyn yhtälön mukaisesti. Minimisäteeksi asetettiin 3 ja minimipinta-alaksi siten $\pi 3^2$. Maksimisäteeksi asetettiin 80, josta laskettiin samalla tavalla pinta-alan maksimi. Normalisointifunktio rakennetaan välittämällä ”makeNormalizer”-funktiolle joukko solmuja sekä niiden muuttujien nimet, joiden arvojen perusteella normalisointi tehdään. Funktio palauttaa normalisointifunktion, josta arvolla kutsumalla saa normalisoidun arvon ”makeNormalizerille” annetussa joukossa. Funktio palauttaa piirrettävien ympyröiden säteitä, jotka on normalisoitu ympyrän pinta-alaa vastaamaan. Ohjelmassa on yksinkertaisempaa välittää ympäriinsä sädettä kuin pinta-alaa, koska SVG:n ympyräelementtien koko annetaan säteenä, ja törmäystarkastus tarvitsee ympyröiden sädettä estääkseen päällekkäisyydet. Koska visualisaatiossa aina ensin näytetään seutukuntien pinta-ala, normalisoidaan ensin sen mukaan, ja normalisoitu arvo liitetään kullekin solmulle ”radius”-muuttujaksi. Kun ensimmäinen visualisoitava muuttuja on normalisoitu, voidaan alustaa visualisaation graafinen esitys, joka tapahtuu ”init”-funktiossa.

Alustusfunktiossa valitaan ensin D3:n valitsimella SVG-elementistä kaikki ”link”-luokkaiset elementit (joita ei ole) ja sidotaan niihin verkon kaaret. Nämä elementit tuodaan sitten näkyviin D3:n ”enter”-funktiolla ”line”-elementteinä. Seuraavaksi samalla tavalla luodaan ja sidotaan verkon solmut ”g”-ryhmäelementteinä. Näihin ryhmiin lisätään ensin ympyrä, jonka säteeksi asetetaan ryhmään sidotun datan ”radius”-muuttujan palauttava funktio. Ryhmään lisätään myös tekstielementti, jonka sisällöksi asetetaan ryhmän datan ”seutukunta”-muuttujasta (joka on seutukunnan nimi Wikipediassa) ympyrään mahtuvan pätkän palauttava funktio.

Voimasommitelman solmuiksi asetetaan datan solmut ja kaariksi datan kaaret, minkä jälkeen simulaatioita ruvetaan ajamaan. Joka simulaatioaskeleella kutsutaan layoutin

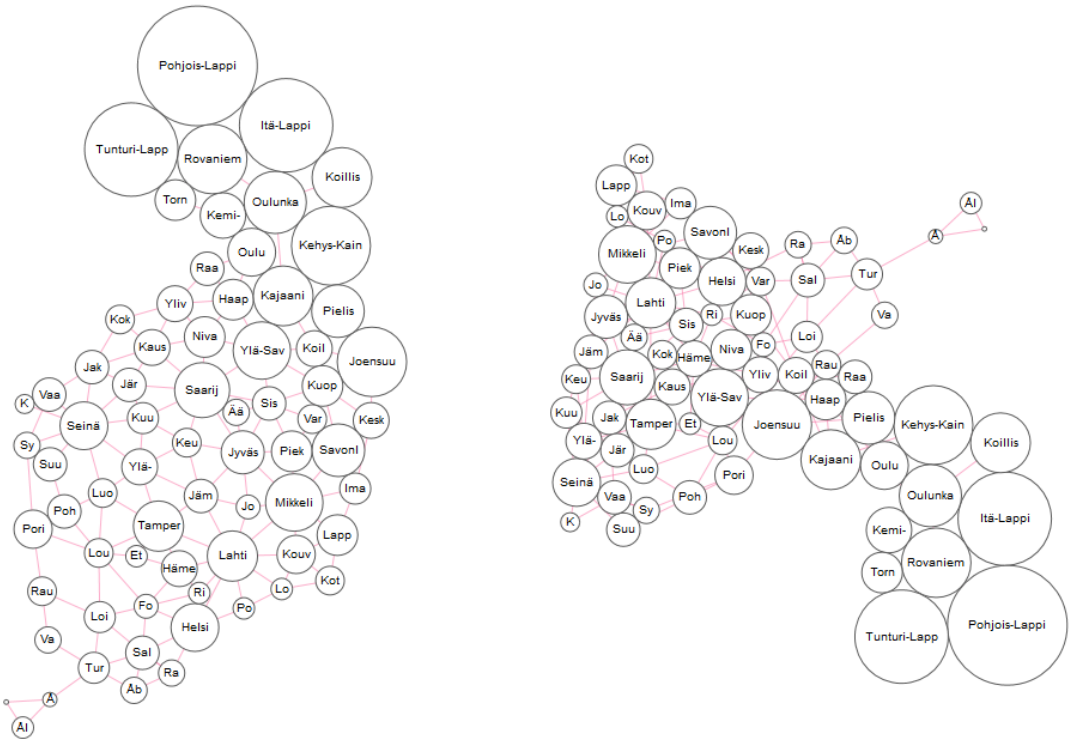
”tick”-funktiota, jossa jokaisen elementin koordinaatit asetetaan simulaation laskemiksi koordinaateiksi. Jokaisen solmun koordinaatteihin vaikuttaa myös törmäyksiä laskeva funktio, joka saatiin D3:n API-sivuilta.

Datasetin valitsemiseen tarkoitettuihin käyttöliittymäelementteihin asetetaan tapahtumankuuntelijat, jotka kutsuvat ”redraw”-funktiota. Funktiossa solmujen ”radius”-muuttujaan normalisoidaan käyttöliittymästä valittu arvo, minkä jälkeen kaikki ”circle”-elementit valitaan, ja niiden koko animoidaan vastaamaan datan arvoa. Myös kaikki tekstielementit päivitetään sopimaan kuhunkin ympyrään.

SVG:n ”circle”-elementteihin eli seutukuntiin liitetään tapahtumakuuntelija, joka kutsuu ”showTooltip”-funktiota sidotulla datalla. Funktio valitsee piilotetun selite-elementin, ja asettaa sen sisällön vastaamaan dataa. Selite näytetään vain, mikäli voimasimulaation jäähdytystekijä on tarpeeksi pieni, eli mikäli esitys ei ole suuressa liikkeessä. Funktio myös siirtää selitteen seutukunnan kohdalle. SVG-elementtiin liitetty tapahtumakuuntelija piilottaa selitteen, kun tyhjää kohtaa painetaan.

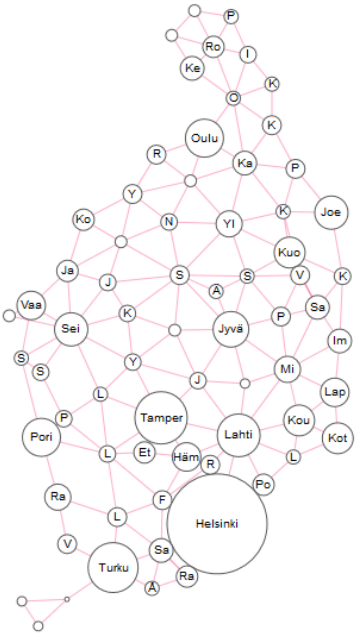
5.4 Visualisaatio ja käyttöliittymä

Luvussa 5.2 kerrottiin, että visualisaatiossa kokeillaan menetelmää rakentaa kartta ilman mitään varsinaista karttadataa, kun jokaisen seutukunnan paikan määräävät vain sen naapuruussuhteet muihin. Toteutetussa visualisaatiossa se tapahtuu asettamalla voimasommitelman ”charge”-voima negatiiviseksi, jolloin solmut hylkivät toisiaan. Arvoksi asetettiin -130. Kuvassa 16 on tällä menetelmällä aikaan saatu visualisaatio, hyvässä ja huonossa tapauksessa.



Kuva 16. Sentroidittoman visualisaation hyvä ja huono tapaus.

Vasemmassa versiossa verkkoa on ravisteltu hiirellä kiinni tarttumalla, ja aseteltu raahaamalla parempaan asentoon. Oikealla on verkko sellaisena kuin se koskemattomana piirtyi. Verkko on piirtynyt kieroon, peilikuvana ja ylösalaisin. Verkko muoto on suurilta osin epädeterministinen. Verkon saa ravistelemalla ja raahaamalla aukeamaan ”takuista”, jotka syntyvät, kun ympyröitä jää loukkuun muiden väliin eikä verkko pääse aukeamaan optimimuotoonsa. Peilikuvalle ei voi mitään, ja se oli ongelma jota luvun 5.2 suunnitelmassa ei otettu huomioon. Kuvassa 17 on vielä kuva verkosta, kun datasetti on vaihdettu vuoden 2013 asukasluvuksi.



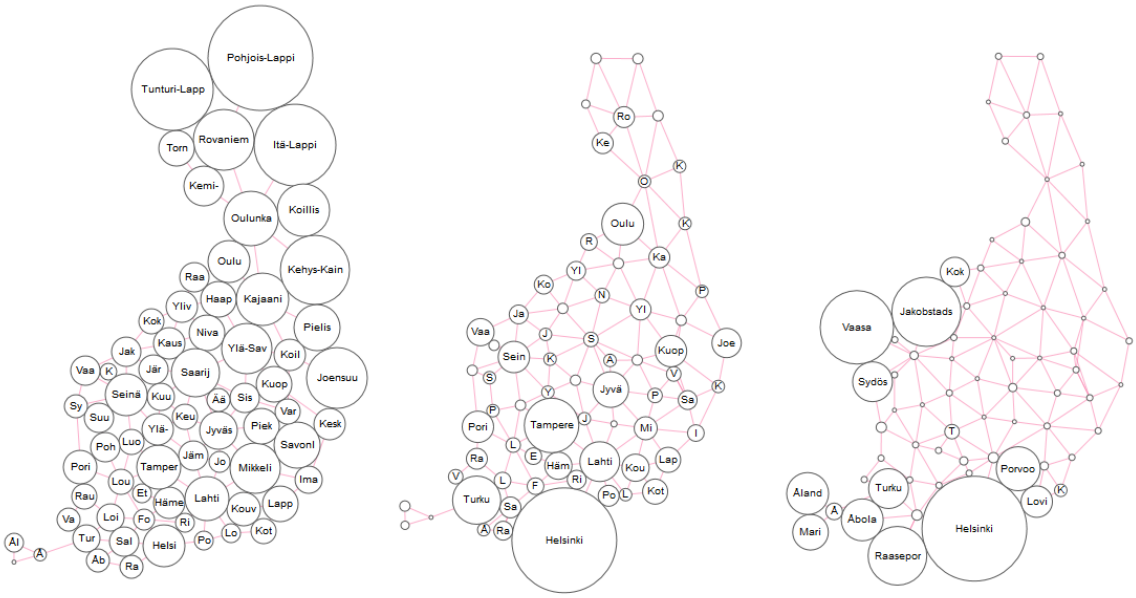
Kuva 17. Vuoden 2013 asukasluku

Peilikuvaksi piirtymistä ei voi ratkaista millään helpolla tavalla tällä tavalla piirretyssä verkossa. Tämän takia visualisaatiosta päätettiin rakentaa toinen versio, jossa seutukuntien paikkaan vaikuttaa voimasommitelman omien voimien sekä törmäysvoiman lisäksi voima, joka ohjaa seutukunta kohti niiden sentroidia kartalla. Sentroididata koostettiin tuomalla seutukuntakartta selaimeen ja liittämällä siihen tapahtumakuuntelija, joka kirjaa ylös klikkauksien xy-koordinatit. Näin klikkaamalla kunkin seutukunnan summittaista sentroidia saadaan ylös sentroidin koordinaatit, ja kun kuvan koko tiedetään, voidaan koordinaatit muuttaa koon ja koordinaatin suhteeksi, jolloin niitä voidaan käyttää kaikissa saman kuvasuhteen kuvissa sentroideina. Tämä sentroididata lisättiin alkuperäiseen verkkoon, ja se näyttää tältä:

```
"centroids": [
            {"sid":197, "x":0.53, "y":0.18},
            {"sid":196, "x":0.44, "y":0.19},
            {"sid":194, "x":0.56, "y":0.28},
            // ...
]}
```

Voimalayoutin ”tick”-funktioon lisättiin siis simulaation sisäisten voimien ja törmäystarkistuksen lisäksi kolmas rajoitus, joka siirtää solmuja kohti kunkin sentroidia. Siirto skaalataan vakiolla, joka määrää miten voimakkaasti solmu haluaa sentroidiinsa. Tilas-

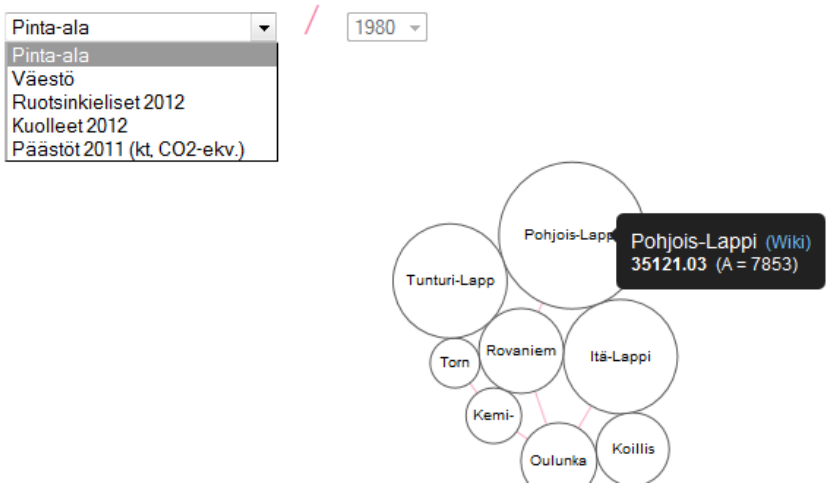
tokeskuksesta löytyi myös lisää seutukuntadataa, ja datajoukkoihin lisättiin vaihtelun saamiseksi ruotsinkielisten määrä vuonna 2012, kuolleiden määrä vuonna 2012 ja päästöt vuonna 2011. Kun tämä uusi voima lisättiin visualisaation, saatiin kuvan 18 mukainen visualisaatio.



Kuva 18. Visualisaatio käyttäen sentroideja. Vasemmalla pinta-ala, keskellä väestö vuonnta 1985 ja oikealla ruotsinkielisten määrä vuonna 2012.

Näin syntynyt verkko vastaa paljon tarkemmin oikeaa karttaa, ja verkon piirtyminen on deterministisempää. Seutukunnat saattavat kuitenkin päätyä väärään paikkaan muiden vangitsemana, mutta tarttumalla kiinni virheellisesti sijoitettuun ympyrään ja raahaamalla sitä hieman verkko selkenee helposti. Hylkimisvoimaksi asetettiin vielä 30, jolloin solmut vetävät toisiaan hieman puoleensa. Tämä ratkaisu on itse asiassa hieman lähempänä Dorlingin alkuperäistä kartogrammia, tosin tässä visualisaatiossa seutukuntien välillä on melko paljon tyhjää tilaa.

Visualisaation käyttöliittymä on yksinkertaisuudessaan kaksi pudotusvalikkoa, joista valitaan datajoukko ja datajoukon alijoukko. Näiden lisäksi seutukuntaa painettaessa aukeaa sen kohdalle selite, joka kertoo seutukunnan nimen, antaa linkin seutukunnan Wikipedia-sivulle, sekä kertoo datan tarkan arvon ja ympyrän pinta-alan. Kun ensimmäisestä pudotusvalikosta valitaan esitettäväksi pinta-ala, poistetaan toinen käytöstä, mutta pienenä kauneusvirheenä jää sen ensimmäinen arvo näkyviin, ja voi saada kuvan että esitetään sen vuoden dataa. Kuvassa 19 näkyvät kaikki käyttöliittymäelementit avattuina.



Kuva 19. Käyttöliittymäelementit

6 Päätelmiä

6.1 Ohjelmasta

Ohjelmasta tuli lopulta joka tavalla pienempi kuin työtä suunnitellessa ja aloittaessa ajateltiin. Luvussa 3.2 esiteltiin ja nimettiin useita erilaisia kehitystyökaluja, JavaScriptiksi käännettäviä kieliä, kirjastoja ja sovelluskehyksiä, mutta mitään näistä ei nähty aiheelliseksi hyödyntää toteutetussa ohjelmassa. Suunnitelman tarkentuessa ja selkeytyessä osoittautui, miten pieni ohjelmasta tulisi, ja siitä karisi pois paljon tarpeettomaksi osoittautunutta kehitystä monimutkaistavaa tekniikkaa. Työn fokus siirtyi aiottua enemmän kehitystyökaluista ja tekniikoista visuaaliseen suunnitteluun.

Harmillista oli verkon yli toimivan rajanpinnan puute, jolla data olisi voitua hakea ajonaikaisesti, mikä työtä aloitettaessa oli suunnitelmana. Data jouduttiin siksi haalimaan käsin, ja suunnitelmana ollut muualla sijaitsevien verkon resurssien esittäminen havainnollisemmassa muodossa jäi siksi aika kuihtuneeksi vertauskuvaksi. Rajapinnan puute toisaalta jätti projektista pois sen käyttöä abstrahoivien sovelluskehysten ja kirjastojen hyödyntämisen, sillä niille ei ollut minkäänlaista käyttöä.

Dataa koostettaessa pyrittiin sitä erheellisesti räätälöimään tällöin vielä toteuttamatonta ohjelmaa varten. Yksi suora ongelma, joka tästä seurasi, oli miten seutukunnassa pinta-ala ja eri vuosien väestömäärät olivat datassa samalla tasolla. Pinta-ala nimettiin

solmun ominaisuudeksi ”pinta-ala”, ja kunkin vuoden väkiluku vain ”2005” ja niin edelleen. Tämä aiheutti ongelmia dataa normalisoidessa: eri vuosien väestömäärät pitää normalisoida yhtenä datajoukkona, jotta kunkin vuoden maksimi ei ole maksimisäteellä piirretty ympyrä, vaan suurin ympyrä on kaikkien vuosien suurin. Mikäli normalisointia ei tehtäisi kaikilta vuosilta, olisi väestömäärää esitettäessä Helsinki joka vuonna saman kokoinen, vaikka datassa se oikeasti kasvaa joka vuosi. Normalisaatiofunktiota käytettäessä jouduttiin siksi kovakoodaamaan vuodet yhdeksi joukoksi, eikä datan valintavalikkoja saatu rakennettua dynaamisesti, vaan ne kovakoodattiin HTML-dokumenttiin. Lisäksi valikoista tuli hieman erikoiset, kun vuoden voi valita vain väestön kohdalla, mutta vuosi 1980 jää kummittelemaan valinnaksi.

Ohjelman varsinainen visualisaatio-osa eli SVG-grafiikka valmistui D3:n avulla todella lyhyessä ajassa, ja kaikki toimi ensi yrittämällä. Paljon enemmän aikaa kului kaikkeen oheistoiminnallisuuteen, kuten käyttöliittymäelementteihin ja datan käsittelyyn ja normalisointiin. Kun D3:n toimintalogiikan valitsimineen ja ”enter”-funktioneen ymmärtää, on visualisaatioiden kehittäminen sillä erittäin nopeaa ja luontevaa, ja sen tarjoama ohjelmointirajapinta on todella viimeistelty. Se paljastaa ja vahvistaa selaimen ominaisuuksien todelliset kykyjä eikä rajoita ohjelmoijaa millään tavalla, vaan on aina sitä tehokkaampi työkalu, mitä etevämpi sen käyttäjä on.

Hyvä idea jatkokehitykselle olisi korjata ensin dataa sen verran, että data-arvo ja vuosi ovat eri tasoilla. Tämä helpottaisi uuden datan lisäämistä, valitsimien dynaamista rakentamista sekä yksinkertaistaisi normalisaattorifunktioiden rakentamista.

6.2 Työkaluista

Suuri apu ohjelmoinnissa oli selaimen kehittäjätyökalut, etenkin JavaScript-konsoli ja elementtien tarkastintyökalu. Kehitystyö eteni lähes interaktiivisesti: kun lähdekoodiin lisättiin funktioita tai muuttujia tai D3:lla tehtiin valintoja, voitiin niiden sisältöä ja vaikutuksia aina tutkia suoraan konsolilta. Tietorakenteiden sisältöä pääsee tutkimaan tulostamatta mitään, ja funktioita voi kutsua suoraan konsolilta. Tämä on yksi selainkehityksen parhaita puolia, ja mikäli käytettäisiin vaikkapa jotain JavaScriptiksi käännettävää kieltä, hukattaisiin suora yhteys lähdekoodin ja selaimen konsolin väliltä.

Konsolilta pystyttiin myös kokeilemaan D3:n funktioita, kun visualisaatio oli piirretty. Kun kirjasto on ladattu dokumentin mukana, voidaan toki myös sen funktioita kutsua konsolilta. Voitiin esimerkiksi kokeilla kohdistaa visualisaation erilaisia siirtymiä tai

luonnostella ohjelman ominaisuuksia kirjoittamatta niitä lähdekoodiin. Ohjelma voitaisiin rakentaa kokonaan interaktiivisesti konsolilta ja jollain näppärällä menetelmällä kirjoittaa tiedostoon.

Toisaalta jos kirjoitetussa ohjelmassa on jokin syntaktisesti oikea bugi, ja se välittää D3:lle jotain vääränlaista, voi kirjaston antama virhe olla täysin hyödytön, eikä ongelmaa löydä helposti. Dynaaminen tyypitys on tässä mielessä mahdollista hämmennystä aiheuttava. Virheen tuottavan rivin voi kuitenkin etsiä selaimen debuggerilla, ja tätä jouduttiin kehityksessä kerran käyttämään, kun D3:lle annettiin vahingossa argumenttina funktion sijaan funktion nimi, ja tämä tuotti erittäin hanakan virheen.

JavaScript on monella tavoin erittäin miellyttävä ohjelmointikieli. Funktionaaliset ominaisuudet kuten sulkeumat ja ajonaikaisesti rakennettavat funktiot ovat todella ilahduttavia toimintoja. Lisäksi prototyyppioliot ovat näppärä tietorakenne hajautustauluna, ja niitä on usein todella luontevaa käyttää vaikkapa paluuarvoina, kun olion voi luoda suoraan lyhyellä lausekkeella ”`{ }`”, samalla tavoin kuin listaliteraalia ”`[ ]`”, ja sen osoittaminen on erittäin helppoa pisteoperaattorilla. Varsinaiset hajautustaulut ovat kuitenkin joukkojen ohella tulossa uuteen JavaScriptin versioon, jolloin oliot eivät joudu väärinkäytettäviksi. Toisaalta hajautustaulun käyttö on monisanaisempaa.

JavaScriptissä on myös mukavan vähän syntaksia. Vaikka asiat joskus toimivat hieman erikoisesti, ne toimivat kuitenkin vähällä syntaksilla. Palasia, joista ohjelma koostuu, voi olla todella vähän, joten niistä on helppo järkeillä, eikä mikään ole moniselitteistä normaaleissa olosuhteissa. Monenlaisia erikoisuuksia kuitenkin on, kuten mahdollisesti hämmentävä ”this”-muuttuja. Kuitenkin kun periaatteet kuten funktionäkyvyyden ja sulkeumat omaksuu, kaikki toimii järkeenkäyvällä tavalla.

Harmillisimpia puutteita on kuitenkin ainakin nimiavaruuksien puute, jota usein paikataan olioiden ja nimettömien funktioiden avulla. Kun koodilohko ympäröidään heti kutsuttavalla anonyymillä funktiolla (”`(function() { /*koodi*/ })();`”), sen sisältämiä muuttujia ja funktioita ole määritelty sen ulkopuolella. Tätä ongelmaa ratkaisevat asynkroniset moduulilataajat, mutta yhden tiedoston ohjelmassa sillä ei juuri ole virkaa. Kun sovellukset kasvavat, se on kuitenkin ehdottomasti paikallaan. Nimiavaruudetkin korjaantuvat uusissa JavaScript-versioissa.

Suurin turhautumisen aihe oli tässäkin kehitysprosessissa ehdottomasti selaimen surkea DOM-rajapinta. Dokumentin sisällön läpikäynti ja muuttaminen ilman mitään apukeinoja on erittäin työlästä ja monisanaista. D3:n valitsimet kuitenkin helpottivat kehitystä, eikä DOM:iin tarvinnut koskea juuri ollenkaan.

6.3 Visualisaatiosta

Alkuperäinen näkemys työtä suunniteltaessa ja aloiteltaessa oli jollakin tavoin jaetusta Suomesta tehty kartogrammi, jonka osasten koko määritetään kunkin osan data-arvon mukaan. Visualisaation suunniteltiin käyttävän dataa hakemalla se lennosta jostain muualta verkosta, jonkin rajapinnan avulla. Visualisaation suunniteltiin aina tulevan myös tästä syystä selaimelle, ja sen suunniteltiin edustavan visuaalisesti muualla verkossa sijaitsevia resursseja. Työn alussa asetettiin tavoite rakentaa avoimesta datasta visualisaatio selaimelle käyttäen moderneja työkaluja. Luvussa 5.2 kerrottiin myös, miten visualisaatiossa kokeillaan, onnistuuko karttaverkon rakentaminen ja esittäminen ilman varsinaista karttadataa. Visualisaatiota ruvettiin toteuttamaan tältä pohjalta.

Dorlingin menetelmän sentroidien sijaan visualisaatiota kokeiltiin toteuttaa ensin käyttämällä vain voimasommitelman hylkimisvoimaa, ja tulokset esiteltiin luvussa 7.2. Tuloksessa oli positiivisia ja negatiivisia puolia. Kartasta tuli mahdollisella pienellä nykimisellä ja järjestelyllä välillä erittäin näyttävä, ja jollain tapaa tämä kokeilu osoittautui onnistuneeksi ja saattaisi olla jatkokehityksen arvoinen. Mikäli verkosta tulisi hieman varmemmin parhaana tapauksena kuvatun kaltainen, sitä voitaisiin pitää onnistuneena. Yksinkertaisella muodolla menetelmä toimisi paremmin: Pahimmat takut tulevat esimerkiksi pinta-alaa esitettäessä pohjoisessa, jossa seutukunnat ovat alaltaan suurimpia, eikä verkko järjesty lähellekkään ideaalista. Koska verkko usein joutuu purkamaan takuista, ja etenkin koska se välillä piirtyy peilikuvaksi, päätettiin visualisaatiosta tehdä toinen versio, jossa seutukunnat haluavat päästä sentroideihinsa. Tässä menetetään kuitenkin jotain, kun esitys tavallaan kovakoodataan sentroidien avulla.

Sentroiditon versio on toisaalta sympaattinen, kun se tarvitsee käyttäjän apua järjestyäkseen. Sotkeutunut verkko voi ensimmäistä kertaa visualisaatiota katselevalle olla todella epäselvä, ja sentroidillinen versio on siksi paljon käyttäjäystävällisempi. Toisessakin versiossa seutukunnat voivat tarvita hieman apua päästäkseen sentroidiinsa, mutta kun verkko on interaktiivinen, pystyy mahdollisia virheitä järjestämään käsin, toisin kuin Dorlingin kuvaamassa menetelmässä.

Siirtelemällä seutukuntia ympäriinsä pystyy tutkimaan, mitä ne syrjäyttävät paikoiltaan, ja siirtämään muiden sentroidiinsa pääsyä estäviä seutukuntia pois tieltä. Sentroideja käytettäessä karttaa on huomattavasti havainnollisempi lukea, ja vaikka jokin kartan osa sisältäisi pientä dataa, on kaarista muodostuvien kartan ääriviivojen avulla silti helppo tulkita sijainteja. Tämä on erityisen selvää tarkasteltaessa vaikkapa ruotsinkielisten määrää, jossa maan ääriviivat pohjoisessa ovat luettavissa, vaikka muoto toisaal-

ta on täysin vääristynyt. Mikäli seutukunnat vain kasattaisiin mahdollisimman kompaktisti, kuten Dorlingin menetelmässä, olisi karttaa haastavampi lukea, sillä muoto ei erojen ollessa suuria säilyisi millään tavalla. Kartogrammiin kuuluu aina oletus, ettei kartan osasten maantieteellisen sijainnin oikeellisuudesta ole mitään takeita. Karttaa on kuitenkin todella paljon helpompi lukea, mikäli seutukunnat ovat edes suunnilleen oikeilla paikoillaan.

Jotta kaikki seutukunnat saataisiin piirrettyä, määritettiin niille minimipinta-ala. Tämä kuitenkin vääristää esitystä, etenkin pienimmillä ympyröillä. Esityksen voidaan katsoa siis valehtelevan luvun 2.2 esittelemällä tavalla. Toinen vääristyksen lähde on pyöristys pinta-alaa laskettaessa, sillä se voi olla vain kokonaisluku. Suuremmilla ympyröillä datan ja pinta-alan suhteet ovat hyvin lähellä toisiaan, esimerkiksi ruotsinkielisten lukumäärässä Helsingin ja Turun datan suhde on 6.984..., ja pinta-alojen suhde 6.925..., joten esitys valehtelee tässä 0.05 yksikön verran. Pienillä ympyröillä valhe kasvaa melkoiseksi: samassa esityksessä Helsinkiä ja Itä-Lappia vertaillessa valhe on n. 5007. Jotta suurimmat ja pienimmät saataisiin molemmat näkyviin ja klikattaviksi, täytyy tässä valita joko tarkkuus tai havainnollisuus.

Luvussa 2.2 esiteltiin datamusteen konsepti eli se, miten kaikki muste (tai tässä tapauksessa pikselit), jotka eivät viesti dataa ovat luultavasti turhia. Tämä pidettiin mielessä visualisaatioita tyyliteltäessä. Kunkin seutukunnan ala viestitään sen kehän avulla, eikä muuta lukukelpoista keinoa keksitty. Periaatteessa kehän sijaan voisi piirtää ympyrän säteen, mutta näin syntyvä visualisaation ei toimisi karttana millään tavalla. Vähäpikselisempää keinoa kuin kehä ei luultavasti ole. Myös ympyrän ala voitaisiin piirtää, mutta kehän piirtäminen viestii saman verran dataa vähemmillä pikseleillä. Verkon kaaret viestivät dataa eli sen, että solmut ovat toistensa naapureita. Kaaren viiva siis on datamustetta. Lisäksi solmun teksti on tietenkin datamustetta, sillä se nimeää seutukunnan. Muita graafisia elementtejä ei ole, ellei selitettä lueta osaksi visualisaatiota. Siinäkin on pelkkää datamustetta eli tekstiä. Tumma tausta auttaa erottamaan sen taustasta, ja vasemmassa reunassa oleva kolmio yhdistää sen seutukuntaan eli viestii dataa.

Toinen luvussa 2.2 esitelty suunnitteluperiaate oli kaavioromu. Mitään sellaista ei visualisaatiossa nähdäkseni ole. Toisteista tietoa on selitteen teksti, mikäli se näkyy kokonaan ympyrään piirrettynä, mutta tämä on luultavasti epärelevanttia. Kaaren viivan väri taas erottaa viivan ympyrästä; mikäli se olisi myös musta, olisi karttaa hankalampi lukea, sillä ympyröiden kehät ja kaariviivat sulkisivat alueita, jotka hämmentäisivät silmää. Kun ne ovat tarpeeksi erivärisiä, näin ei käy.

Väestön määrä on yksiulotteinen arvo, mutta kartassa se on esitetty kaksiulotteisesti

pinta-alalla, vastoin luvun 2.2 periaatetta. Toisaalta taas pinta-alaan yhdistyy seutukunnan spatiaalinen sijainti, ja sillä viestitään aina kartan osaa. Ehkäpä on siksi perusteltua esittää arvo ympyränä, sillä se kuvaa aina kaksiulotteista aluetta kartalla. Tämä kuitenkin on miettimisen arvoista. Ympyrä viestiikin oikeastaan kahta muuttujaa: dataarvoa ja seutukunnan sijaintia kartalla ja suhteessa muihin. Ympyrästä on hankala kuitenkaan tarkasti sanoa, miten paljon se on suurempi tai pienempi kuin toinen.

Olisi erittäin mielenkiintoista, jos dataa olisi usealla tavalla jakautuvina joukkoina. Suurin osa datasta on kuitenkin väestömäärästä riippuvaa, eli siellä missä on eniten ihmisiä, kuolee myös eniten. Visualisaation suunniteltiin edustavan verkon muita resursseja, eikä ole selvää, miten selkeästi ympyrän edustavat Wikipedian sivua seutukunnasta. Selite linkittää sinne, ja se sisältää datan eri tavalla esitettynä, mutta yhteys ei välttämättä ole täysin selvä. Mikäli data haettaisiin suoraan Wikipediasta olisi yhteys paljon selvempi. Tämä ei kuitenkaan ollut millään yksinkertaisti mahdollista, ja tämä suunnittelun seikka saattaakin jäädä löyhäksi, joten siihen oli ehkä turha yrittää siksi sitoutua. Lopussa tästä poikettiinkin jonkun verran, ja dataa lisättiin muutama muuttuja. Etenkin ruotsinkielisten määrästä syntyvä kartta on oikein onnistunut esimerkki kartogrammista visualisaation välineenä.

Vuosissa eteneminen järjestyksessä ei tällä skaalalla ole myöskään yhtä havainnollista kuin suunnitteluvaiheessa ajateltiin, eikä välttämättä juuri viesti paljoakaan lisädataa. Kuitenkin vuosien kelaamisella voi juuri ja juuri nähdä, miten suuret kaupungit kasvavat ja pienemmät seutukunnat pienenevät entisestään. Miellyttävä efekti tulee myös pinta-alasta siirryttäessä väkimäärään, kun painopiste siirtyy pinta-alaa dominoivasta pohjoisesta täysin etelän suuriin kaupunkeihin. Tämä tuskin kuitenkaan on mitenkään kovin kohahduttava ilmiö. Ruotsinkielisten lukumäärää tarkasteltaessa saadaan vaihtelua väkimäärästä riippuvien muuttujien lisäksi, jolloin painopiste siirtyy rannikolle. Tuotettu visualisaatio kuitenkin toteuttaa ne tavoitteet, jotka sille työtä aloittaessa asetettiin, ja on suunnitellun mukainen.

7 Yhteenveto

Tässä ohjelmistotekniikan insinöörityössä toteutettiin avointa Suomen seutukuntadataa kartogrammina esittävä datavisualisaatio verkkoselaimelle käyttäen d3.js-datavisualisointikirjastoa. Visualisaatio pyrittiin suunnittelemaan koostettujen datavisualisaatioiden suunnitteluperiaatteiden mukaisesti, ja suunnitelun onnistuneisuutta

arvioitiin lopuksi näiden periaatteiden avulla.

Työ aloitettiin määrittelemällä ensin, mitä datan visualisoinnilla tarkoitetaan ja mihin sitä tarvitaan. Tämän jälkeen koostettiin seikkoja, joita mahdollisimman hyödyllisen visualisaation suunnittelussa tulee ottaa huomioon. Nämä seikat jakautuivat visualisaation graafiseen suunnitteluun sekä ihmisen tiedonkäsittelyn ja näkökyvyn ominaisuuksiin, joita visualisaatiossa tulee ottaa huomioon. Seuraavaksi esiteltiin vielä seikkoja, joita visualisaatiota verkkoselaimella suunniteltaessa tulisi ottaa huomioon.

Tämän jälkeen luotiin yhteenveto HTML5-standardin toteuttavien modernien selainten kyvyistä, ja rajapinnoista, joilla niitä käytetään. Kykyjen lisäksi koostettiin pieni perehdytys JavaScriptistä selaimen ohjelmointikielenä sekä työkaluihin, joita selaimelle kehitettäessä voidaan hyödyntää.

Seuraavaksi pohdittiin, mitä on avoin data, mitä hyötyä siitä on ja miten sitä voidaan hyödyntää verkossa sijaitsevien datavisualisaatioissa, mikäli se on saatavissa verkon yli jonkin rajapinnan avulla. Lisäksi esiteltiin, miten avoimen datan hyödyllistä käyttöä on sen visualisointi, jolloin tämä mahdollisesti vaikeasti tulkittava tai laaja data saadaan kaikkien ulottuville havainnollisessa muodossa.

Luvussa 5 luotiin ensin pohja työssä toteutettavan visualisaation suunnittelemiselle esittelemällä kartogrammit, joissa kartan osien kokoa muutetaan kuhunkin alueeseen liittyvän data-arvon avulla. Luvussa esiteltiin erilaiset kartogrammit sekä toteutetussa visualisaatiossa käytetty ympyräkartogrammi, jossa jokaista aluetta kuvataan ympyrällä. Tämä otettiin lähtökohdaksi Suomen seutukuntien visualisoinnille, jossa jokainen seutukunta on kartogrammin ympyrä, ja seutukuntien data on sama datajoukko, joka löytyy kunkin seutukunnan Wikipediasivulta. Lisäksi esiteltiin suunnitelma kuvata kartogrammi suuntaamattomana verkkona, jossa verkon solmut ovat seutukuntia ja niiden naapuruussuhteet verkon kaaret. Tällöin kartan esittämiseen ei tarvittaisi mitään karttadataa, mikäli solmut hylkivät toisiaan.

Visualisaation suunnittelun jälkeen kuvattiin varsinainen toteutusprosessi. Ensin kerrotaan, miten visualisoitava data haalittiin, ja sen jälkeen esitellään käytetty visualisointikirjasto D3. Ohjelmasta tuli lopulta käytetyn kirjaston ansiosta hyvin yksinkertainen verkkodokumentti. Toteutuksen kuvauksessa esitellään yksityiskohtaisesti miten JavaScript-ohjelman suoritus etenee ja miten koodi on jäsennelty. Lisäksi esitellään aikaan saatu D3:n voimasommitelmaan vahvasti perustuva visualisaatio sekä minimalistinen käyttöliittymä, jolla sitä käytetään. Kokeiltu sentroiditon versio onnistui suhteellisen hyvin, mutta sen puutteiden takia päätettiin toteutusta jatkaa hieman pidemmälle,

ja verkon solmuille määritettiin sentroidikoordinaatit, joihin ne haluavat päästä. Lisäksi solmuille annettiin muutama data-arvo Wikipedian sisältämien arvojen lisäksi. Lisäämällä sentroidivoima ja uusi data saatiin visualisaatiosta tyydyttävä.

Visualisaatiota arvioitiin lopuksi alussa koostettujen suunnitteluperiaatteiden avulla, ja suurilta osin visualisaatiota voidaan pitää onnistuneena, ja tavoite luettiin täytetyksi. Projekti tarkensi alkupuolellaan näkemystä visuaalisen suunnittelun datalähtöisyydestä, ja samalla kaiken ohjelmistokehityksen luonteesta datan prosessointina, ja toisaalta käyttöliittymistä datan esittäjinä. Koostettu teoria antoi paljon ajattelemisen aihetta kaikesta visuaalisesta suunnittelusta, ja monta vinkkiä ja huomiota jäi mieleen. Erityisti iskostui mieleen huomio, miten kaikkien käyttöliittymän (joka interaktiivinen visualisaatio myös on) elementtien tulisi lähtökohtaisesti viestiä dataa ja miten graafisilla elementeillä tulee siten olla syntaksi, eli niiden pitää tarkoittaa jotain.

Toteutusvaiheessa projekti perehdytti erittäin hyödylliseksi ja voimakkaaksi osoittautuneeseen D3-visualisointikirjastoon ja antoi oikean projektin ja hyvän syyn harjoitella sen käyttämistä. Tärkeä opetus oli myös miten dataa ei kannata missään nimessä räätälöidä mitään toteutumatonta suunnitelmaa vasten, vaan jäsennellä data mahdollisimman luontevasti, eikä yrittää pakottaa sitä mihinkään keksittyyn muottiin. Tässä korostui myös arvojen merkitys kaikessa ohjelmoinnissa. Arvoja on yksinkertaista siirrellä ohjelman sisällä, sekä ohjelmien välillä erilaisten rajapintojen ja tiedonsiirtokanavien välillä, eikä arvoja kannata kääriä mihinkään turhaan.

Vaikka asiat luulee osaavansa, tarkentuu kirjoittamalla niistä kuva väkisinkin, kun osaaminen on jollakin tavalla viestittävä eteenpäin. Toisaalta myös kävi ilmi, että projektin edetessä suunnitelma voi muuttua radikaalisti, ja vision konkretisoituessa ja tullessa eloon se ei ikinä vastaa täysin mielessä ollutta kuvaa. Jotkin pikkujutuiksi mielletyt seikat voivat osoittautua suuremmiksi, ja suuret suunnitelmat saattavat oikeasti olla paljon mielikuvia pienempiä.

Lähteet

1 Suomen sata uutta mahdollisuutta: radikaalit teknologiset ratkaisut. 2013. Eduskunnan tulevaisuusvaliokunta. Verkkodokumentti. <http://www.eduskunta.fi/triphome/bin/thw.cgi/trip?${APPL}=erekj&${BASE}=erekj&${THWIDS}=0.16/1380824356_459162&${TRIPPIFE}=PDF.pdf>. Luettu 2.10.2013.

2 Tufte, Edward R. 2001. The Visual Display of Quantitative Information. Connecticut: Graphics Press.

3 Ware, Colin. 2013. Information Visualization: Perception for Design. 3rd ed. Massachusetts: Morgan Kauffman.

4 <https://en.wikipedia.org/wiki/File:Anscombe%27s_quartet_3.svg>. Verkkodokumentti. Luettu 9.10.2013.

5 <https://upload.wikimedia.org/wikipedia/commons/2/29/Minard.png>. Verkkodokumentti. Luettu 9.10.2013.

6 Tufte, Edward R. 1990. Envisioning Information. Connecticut: Graphics Press.

7 Huge Map of the Distant Universe Reaches Halfway Point. Verkkodokumentti. <http://www.eso.org/public/announcements/ann13022/>. Luettu 9.10.2013.

8 HTML5: A vocabulary and associated APIs for HTML and XHTML. Verkkodokumentti. W3C. <http://www.w3.org/TR/html5/>. Päivitetty 6.8.2013. Luettu 24.9.2013.

9 Scott, Bill & Neil, Theresa. 2009. Designing Web Interfaces. California: O'Reilly Media.

10 HTML5 Rocks. Verkkodokumentti. Google. <http://www.html5rocks.com/en/>. Luettu 25.9.2013.

11 Flanagan, David. 2011. JavaScript: The Definitive Guide. 6th ed. California: O'Reilly Media.

12 Osmani, Addy. Journey Through The JavaScript MVC Jungle. Verkkodokumentti. <http://coding.smashingmagazine.com/2012/07/27/journey-through-the-javascript-mvc-jungle/>. Luettu 9.10.2013.

13 Avointen tietovarantojen määritelmä. Verkkodokumentti. Open Knowledge Foundation. <http://opendefinition.org/okd/suomi/>. Luettu 24.9.2013.

14 What is Open Data? Verkkodokumentti. <http://opendatahandbook.org/en/what-is-open-data/index.html>. Luettu 24.9.2013.

15 Berners-Lee, Tim. Linked Data. Verkkodokumentti. <http://www.w3.org/DesignIssues/LinkedData.html>. Päivitetty 18.6.2009. Luettu 24.9.2013.

16 Dorling, Daniel. 1996. Area Cartograms: Their Use and Creation. Verkkodokumentti. Concepts and Techniques in Modern Geography. <http://qmrg.org.uk/files/2008/11/59-area-cartograms.pdf>. Luettu 22.9.2013.

17 Commuting Time. Verkkodokumentti. <http://www.worldmapper.org/display.php?selected=141>. Luettu 9.10.2013.

18 Op-Chart: How Much Is Your Vote Worth? Verkkodokumentti. <http://www.nytimes.com/interactive/2008/11/02/opinion/20081102_OPCHART.html?_r=1&>. Luettu 9.10.2013.

19 Cartogram Types. Verkkodokumentti. <http://www.ncgia.ucsb.edu/projects/Cartogram_Central/types.html>. Luettu 9.10.2013.

20 Tilastokeskuksen PX-Web-tietokannat. Verkkodokumentti. <http://pxweb2.stat.fi/database/StatFin/databasetree_fi.asp>. Luettu 9.10.2013.

21 Kunnat 2013 / Seutukunnat luokitusavain. Verkkodokumentti. <http://www.stat.fi/meta/luokitukset/kunta/001-2013/luokitusavain_skunta_teksti.txt>. Luettu 9.10.2013.

22 Data-Driven Documents. Verkkodokumentti. <http://d3js.org/>. Luettu 26.12.2013.

23 API Reference. Verkkodokumentti. <https://github.com/mbostock/d3/wiki/API-Reference/>. Luettu 26.12.2013

Tuotetun JavaScript-ohjelman lähdekoodi

```
// svg-element size
var size = 840;

// Maximum and minimum radii and areas for the SVG circles
var rmax = 90,
    rmin = 3,
    Amin = Math.PI*rmin*rmin,
    Amax = Math.PI*rmax*rmax;

// The loaded data graph in memory as a JS object.
var graph = {};

// Visualization graph's links and nodes
// Globals because of the tick-function
var link,
    node;

// Normalizer functions
var normalizers = {};

var svg = d3.select("body").append("svg")
    .attr("width", size)
    .attr("height", size)
    .on("click", function(){
        if (d3.event.target.localName == "svg") {
            d3.select("#tooltip").classed("hidden", true);
        }
    });

var force = d3.layout.force()
    .friction(0.9)
    .size([size, size])
    .charge(-130)
    .gravity(0.1)
    .on("tick", tick);


d3.json("verkko.json", function(error, data) {
    graph.nodes = data.nodes,
    graph.centroids = data.centroids,
    graph.links = data.links.map(function(l) {
        // Links must be indices of the node array.
        for (var i = 0; graph.nodes[i].sid != l.source || i < l.length; i++);
        for (var j = 0; graph.nodes[j].sid != l.target || j < l.length; j++);
        return {source: i, target: j};
    });
```

```
    normalizers.vaesto = makeNormalizer(graph.nodes,
        ["1980","1985","1990","1995","2000","2005","2010","2013"]
    );

    ["kuolleet2012", "paastot2011", "ruotsinkieliset2012", "pinta-ala"]
        .forEach(function(s) {
            normalizers[s] = makeNormalizer(graph.nodes, [s]);
    });

    graph.nodes.forEach(function(node) {
        node.radius = normalizers["pinta-ala"](node["pinta-ala"]);
    });

    init();
});

function init() {
    force.nodes(graph.nodes)
         .links(graph.links)
         .start();

    link = svg.selectAll(".link")
        .data(graph.links)
        .enter().append("line")
            .attr("class", "link");

    node = svg.selectAll(".node")
        .data(graph.nodes)
        .enter().append("g")
            .attr("class", "node")
            .on("click", function(d) {showtip(d);})
            .call(force.drag);

    node.append("circle")
        .attr("r", function(d) {return d.radius;})

    node.append("text")
        .attr("dy", ".3em")
        .style("text-anchor", "middle")
        .text(function(d) {return d.seutukunta.substring(0, (d.radius/3)-1); });

    d3.select("#selectset")
      .select("option")
      .property("selected", true);
    d3.select("#selectyear")
      .property("disabled", true)
      .select("option")
      .property("selected", true);
```

```
}

function showtip(d) {
    if (force.alpha() < .05) {
        d3.select("#tooltip")
          .style("left", d.x+d.radius/2 + "px")
          .style("top",  d.y+d3.select("#controls").property("clientHeight")-15 + "px")
          .classed("hidden", false);

        d3.select("#seutu")
          .text(d.seutukunta);
        d3.select("#value")
          .text(d[resolveSelectedSet()]);
        d3.select("#radius")
          .text("(A = " + (Math.PI*d.radius*d.radius).toFixed(2) + ")");
        d3.select("#wiki")
          .property("href",
"https://fi.wikipedia.org/wiki/"+encodeURIComponent(d.urlform));
     }
}

function hidetip() {
    d3.select("#tooltip")
      .classed("hidden", true);
}

function resolveSelectedSet() {
    var e = d3.select("#selectyear");
    return e.property("disabled")
        ? d3.select("#selectset").property("value")
        : e.property("value");
}

function redraw(set) {
    graph.nodes.forEach(function(node) {
        if (parseInt(set)) {
            node.radius = normalizers.vaesto(node[set]);
        } else {
            node.radius = normalizers[set](node[set]);
        }
    });

    svg.selectAll("circle")
        .transition()
        .duration(500)
        .attr("r", function(d) {return d.radius;})

    svg.selectAll("text")
        .text(function(d) {return d.seutukunta.substring(0, (d.radius/3)-1); });
```

```
    force.start();
}

function makeNormalizer(nodeset, keyset) {
    var data = [],
        min, max;

    nodeset.forEach(function(node){
        keyset.forEach(function(key){
            data.push(node[key]);
        });
    });

    min = d3.min(data);
    max = d3.max(data);

    return function(val) {
        return Math.sqrt(Amin + ((val - min)*(Amax - Amin))/(max - min))/Math.PI;
    }
}

function tick(e) {
    node.each(centroidForce(.6 * e.alpha))
        .each(collide(.5))
        .attr("transform", function(d) { return "translate(" + d.x + "," + d.y + ")";
});

    link.attr("x1", function(d) { return d.source.x; })
        .attr("y1", function(d) { return d.source.y; })
        .attr("x2", function(d) { return d.target.x; })
        .attr("y2", function(d) { return d.target.y; });
}

function centroidForce(alpha) {
    return function(o) {
        graph.centroids.forEach(function(ce) {
            if (ce.sid == o.sid) {
                o.x += (ce.x*size - o.x) * alpha;
                o.y += ((ce.y - 0.1)*size - o.y) * alpha;
            }
        });
    }
}

// From the D3 API pages.
function collide(alpha) {
    var quadtree = d3.geom.quadtree(graph.nodes);
    return function(d) {
```

```
        var   r = d.radius + 16,
            nx1 = d.x - r,
            nx2 = d.x + r,
            ny1 = d.y - r,
            ny2 = d.y + r;
        quadtree.visit(function(quad, x1, y1, x2, y2) {
            if (quad.point && (quad.point !== d)) {
                var x = d.x - quad.point.x,
                    y = d.y - quad.point.y,
                    l = Math.sqrt(x * x + y * y),
                    r = d.radius + quad.point.radius;
                if (l < r) {
                    l = (l - r) / l * alpha;
                    d.x -= x *= l;
                    d.y -= y *= l;
                    quad.point.x += x;
                    quad.point.y += y;
                }
            }
            return x1 > nx2
                || x2 < nx1
                || y1 > ny2
                || y2 < ny1;
        });
    };
}

// The selection menu event handlers.
d3.select("#selectset").on("change", function() {
    hidetip();

    if (this.value != "vaesto") {
        d3.select("#selectyear")
          .property("disabled", true)
          .select("option")
          .property("selected", true)

        redraw(this.value);
    } else {
        d3.select("#selectyear")
          .select("option")
          .property("selected", true)

        d3.select("#selectyear")
          .property("disabled", false);

        redraw(d3.select("#selectyear").property("options")[0].value);
    }
});
```

```
d3.select("#selectyear").on("change", function() {
    hidetip();
    redraw(this.value);
});
```