Kart på nett. Hvem, hva, hvordan?

Atle Frenvik Sveen

Atle Frenvik Sveen: Web maps. Who, what, how?

KART OG PLAN, Vol. 74, pp. 174–180, POB 5003, NO-1432 Ås, ISSN 0047-3278

Web maps have become ubiquitous on the web. There are a number of dedicated web map pages, classified and «catalog» sites that rely heavily on maps, and governmental and public agencies use maps extensively to share their data. Maps are also becoming a key component in new communities such as «social media» pages, «niche» services, and even web journalism [1]. While use of maps on the web has a long history, one can argue that the widespread use of web maps that shares a familiar look, feel and technology on the web today represents something new. While maps now are a common component of the web, much like embedded videos and attractive design, our understanding of how they work and operate is limited. The first web map clients were developed in the early 90s, but many web maps were restricted to static images or based on browser plugins. The launch of Google Maps in 2005 and the rise of JavaScript and single page apps (SPAs) around 2010 gave rise to a new surge in the development of web map applications. While these applications may seem complex, their core functions are rather simple. JavaScript is used to monitor mouse movements, send HTTP requests to a WMS- or tile-server and manipulate the DOM to give the user an impression of a dynamic map. Browser limitations used to mean that raster data such as WMS and tile schemes were the only feasible solution for displaying map data in a browser. The situation is different now. Clever tricks, better JavaScript engines and new HTML5 concepts like Canvas and WebGL allow a web map client to be based entirely on vector data. Depending on one's budget, level of technical understanding and other constraints, many web mapping clients are available.

Keywords: web mapping, JavaScript, map clients

Atle Frenvik Sveen, M.Tech. Software developer, Bouvet ASA, Kjøpmannsgata 35, NO-7011 Trondheim. E-mail*:* atle@frenviksveen.net

Introduksjon

Kart har blitt allemannseie på weben: det finnes en rekke dedikerte kart-nettsider, annonsesider er storforbrukere av kart og offentlige institusjoner bruker i stor grad kartløsninger for å kommunisere med innbyggere og for å dele data. I tillegg til dette ser vi at kart begynner å bli en nøkkelkomponent i nettsamfunn, sosiale medier, nisjetjenester og journalistikk på nett [1].

Kart er ikke noe nytt på web, men den utstrakte bruken av kart, på en lett gjenkjennelig måte, kan sies å representere noe nytt. Selv om kart nå er en integrert del av web-sider, på samme måte som video og nytenkende design, er det få av de som bruker dem som forstår hvordan de er bygd opp og fungerer. Dette gjelder spesielt programmerere som jobber med å lage web-sider, som i mange tilfeller ikke kjenner til konsepter som projeksjoner, koordinatsystemer eller kartografi. På den annen side finner man GIS-spesialistene, som har inngående kjennskap til projeksjoner, men som ikke vet hva som ligger i begreper som «HTTP requester», DOM-manipulasjon, AJAX og andre prinsipper som ligger til grunn for moderne web-løsninger, inkludert kartløsninger på web.

I denne artikkelen vil jeg prøve å gi et overblikk over tingenes tilstand hva angår webkart-klienter. Dette inkluderer en diskusjon om hva er webkart-klient er, historien bak disse og hvordan de fungerer. I tillegg presenteres en oversikt over eksisterende webkart-klienter, både proprietære og fri programvare. Målet er at både GIS-eksperten og den jevne web-programmereren skal lære noe om kart på nett.

Historikk

Bruk av kart på web er ikke noe nytt fenomen i web-ens relativt korte historie. Allere-

de på 90-tallet ble de første forsøkene på å lage interaktive webkart gjort, av aktører som Xerox PARC, universitetet i Berkley, US Census Bureau og den kommersielle aktøren MapQuest [2]. Tabell 1 gir en oversikt over disse tidlige prosjektene.

Tabell 1: En oversikt over tidlige webkart-prosjekter.

Navn	Utvikler	År
Xerox PARC Map Viewer	Xerox PARC	1993
GRASSLinks	Universitetet i Berkley	1995
MapQuest	MapQuest	1996
TIGER Map Server	US Census Bureau	1997

Selv om disse forsøkene prøvde å lage webkart med en viss grad av interaktivitet var normen for webkart på dette tidspunktet mer eller mindre statiske bilder av kart. Disse bildene ble vanligvis presentert på websidene på samme måte som andre bilder, uten noen slags kontekst eller interaktivitet. Mot slutten av 90-tallet kom det flere såkalte webkart-servere som prøvde å gjøre noe med dette, slik som UMN Mapserver, ESRI ArcIMS og GeoServer [3], men disse ble ikke tatt i bruk av den gjennomsnittlige web-utvikleren, og standarden var fortsatt statiske bilder.

Først i 2005 kom det store gjennombruddet for webkart, da Google lanserte sin tjeneste Google Maps [4]. Ikke lenge etter kom nettsiden HousingMaps.com, som *kombinerte* boligannonser fra Craigslist med karttjenesten til Google og lagde det som kalles den første webkart *mashup*en. Mashup er et begrep fra musikkindustrien som brukes om musikk basert på flere andre verk mikset sammen. Det ble tatt i bruk for å beskrive tjenester som HousingMaps.com, der data fra *flere forskjellige* kilder kombineres for å lage nye produkter som kanskje ingen av de opprinnelige dataeierne hadde tenkt på. Suksessen til denne første mashupen, og de andre som fulgte i kjølvannet gjorde at Google snart lanserte et åpent API for Google Maps. Dette gjorde at webutviklere kunne, enkelt og lovlig, inkludere et tilpasset kart fra Google Maps i sine web-løsninger [5]. Lanseringen av dette APIet bidro til å akselerere den teknologiske utviklingen innenfor kart på web, en utvikling vi fortsatt ser i dag.

Kartgrensesnittet Google Maps lanserte og gjorde populært ble kalt «slippy maps», og kjennetegnes ved at det presenterer brukeren for et kart som kan panoreres ved å «ta tak i det» og dra det ved bruk av musepekeren, og zoome inn og ut ved hjelp av scrollhjulet. Begrepet «slippy map» brukes i dag sjelden, av den grunn at nesten alle web-kart er såkalt «slippy maps». Etter lanseringen av Google Maps fulgte aktører som Yahoo!, Nokia og Microsoft etter og lanserte karttjenester som tilbød lignende funksjonalitet: et API for å lage tilpassede kartklienter med et innebygd bakgrunnskart med data for hele verden basert på «kartfliser» (eller «tiles») levert som bilder over HTTP-protokollen [6].

I 2006, etter den første «Where 2.0»-konferansen og suksessen til Google Maps, utviklet og lanserte MetaCarta den første versjonen av OpenLayers. Dette var den første «slippy map»-kartklienten lansert med en fri-programvarelisens. Med unntak av det integrerte bakgrunnskartet tilbød OpenLayers tilsvarende funksjonalitet som Google Maps [7]. Samtidig fortsatte ESRI videreutviklingen av ArcIMS: versjon 3.0 ble lansert i 2000 og tilbød en «HTML Viewer». Først i 2011 ble ArcIMS erstattet av ArcGIS 10.1, som tilbød APIer i Javascript, Flex og Silverlight [8]. De siste årene har en annen kartklient med en fri programvarelisens blitt populær: Leaflet ble lansert i 2011 basert på teknologi fra CloudMade, et selskap som selger «kartfliser» basert på OpenStreetMap-data [9]. Teamet bak OpenLayers er også i gang med å lansere OpenLayers 3, en helt ny kartklient som sikter på å dra nytte av mulighetene ny teknologi gir.

Hvordan fungerer en webkart-klient?

I denne sammenhengen tenker vi på en webkart-klient som koden som sørger for å vise

et kart på en nettside samt lar brukeren interagere med kartet. Flere forskjellige teknologier og programmeringsspråk kan, og har blitt brukt til dette, men i dag bruker nesten alle webkart-klienter JavaScript. Dette på tross av at teknologier som Java Applets, QuickTime, Adobe Flash og Flex samt Microsoft ActiveX og Silverlight har vært eller er enklere å jobbe med, og kan gi flere muligheter og høyere hastigheter. Hvorfor har da JavaScript blitt nærmest enerådende? JavaScript er den eneste teknologien som ikke krever tredjeparts-tillegg, såkalte plugins til nettleseren, for å fungere. Hvis man bruker en teknologi som er avhengig av plugins vil man ikke følge web-standarder, og man vil kunne oppleve en reduksjon i antall potensielle brukere. En annen årsak er at plugins kan føre med seg sikkerhetsproblemer.

JavaScript skiller seg fra slike plugins ved at alle moderne nettlesere kommer med en innebygget «motor» for å tolke og eksekvere JavaScript-kode. Dermed vil alle nettlesere være i stand til å vise et webkart skrevet i JavaScript uten av brukeren må installere noen tillegg. I det siste har flere og flere moderne web-applikasjoner blitt utviklet som såkalte «enkelt-side-applikasjoner» («single page apps», SPA) som fungerer på samme måte som en desktop-applikasjon. Google Drive (tidligere Docs) og Gmail er gode eksempler på slike SPAer. Slike løsninger krever bedre ytelse fra JavaScript-motorene i nettleserne [10]. På bakgrunn av dette har nettleserprodusentene fokusert mye på nettopp dette, og man har fått en situasjon der man konkurrerer om økt ytelse i JavaScript-motorene [11], noe som kommer sluttbrukeren til gode og legger til rette for mer funksjonsrike og bedre webkart.

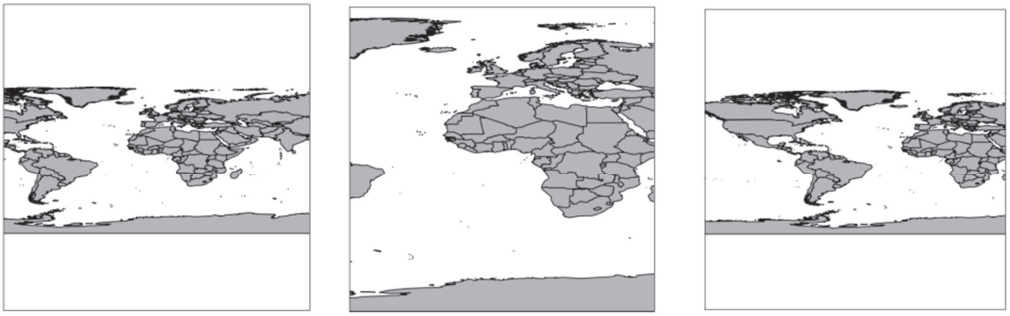
Et webkart-*bibliotek* kan dermed defineres som en samling gjenbrukbar JavaScript-kode med veldefinert oppførsel og dokumentasjon som lar en web-utvikler enkelt sette opp et webkart som fungerer uten ekstra innsats fra brukerens side i alle moderne nettlesere. Vi har behov for et slikt bibliotek med ferdigdefinerte rutiner for å håndtere visning av kart fordi HTML, språket som brukes for å beskrive layout på en web-side, ikke kjenner til konseptet kart. HTML har i utgangspunktet kun fokus på tekst og bilder, selv om HTML5 har fått støtte for ting som lyd og video.

For å komme rundt denne begrensningen bruker man JavaScript. JavaScript er som nevnt et programmeringsspråk som kjører innebygget i nettleseren, og det har tilgang til å redigere DOM (Document Object Model); nettleserens interne representasjon av en webside. Dette gjør at man kan endre på en nettside uten å laste den på nytt fra serveren. En annen fordel med JavaScript er at man kan bruke det for å hente mer innhold fra server ved hjelp av asynkrone HTTP-forespørsler og oppdatere DOM på med disse dataene. Dette konseptet kalles AJAX. DOM-manipulasjon og asynkrone HTTP-forespørsler er de to hovedbyggeklossene i en webkart-løsning. Når man kombiner disse med muligheten til å lytte på brukerens musebevegelser gjenstår det bare et sett med veldefinerte matematiske operasjoner for å vise et «slippy map» i nettleseren, basert på «kartfliser» hentet fra en server.

Dette kan illustreres med et forenklet eksempel, som finnes i sin helhet på nettadressen https://gist.github.com/atlefren/5407085. Vi har et rimelig enkelt JavaScript program som «overvåker» et element på en nettside. Dette elementet har en størrelse på 500x500 piksler, og kartet som initialt vises her har begrensningene [-90, -180, 90, 180] (se figur 1a), det vil si at det viser hele verden. For enkelhets skyld antar vi at vi jobber mot en WMS-server [12], som vil returnere et kartutsnitt uansett hvilket område og oppløsning vi spør om. Vanligvis ville vi brukt en tileserver, basert på WMTS [13], TMS [14] eller lignende standarder, som returnerer ferdiglagede «kartfliser», men prinsippet blir det samme.

For å vise et kartbilde i den enkle kartklienten vår ville vi initielt bedt JavaScript-koden vår sende en WMS-forespørsel til den oppgitte serveren med en forespørsel om et kartbilde på 500x500 piksler som dekker området (bbox) [-90, -180, 90, 180]. WMS-serveren ville da returnert et bilde som koden plasserer i elementet det overvåker. Så langt har vi ikke gjort noe annet enn å vise et statiske bilde av et kart på en nettside. Hvis vi nå overvåker om brukeren beveger scrollhjulet på musen kan vi beregne et nytt område

(bbox). Hvis vi antar at brukeren beveger scrollhjulet for å zoome inn, og at vi reduserer målestokken med en faktor på 2, samtidig som vi beholder kartets senter i midten av kartet (0, 0) vil vi få en ny bbox på [-45, -90, 45, 90] (se figur 1b). Vi sender en ny forespørsel til WMS-serveren, får et nytt bilde tilbake og vi har gitt brukeren en illusjon av zooming i kartet. Hvis så brukeren trykker ned museknappen og beveger musepekeren en fjerdedel over elementet mot høyre vil neste område vi spør om være [-45, -180, 135, 180] (se figur 1c), det vil si at senterpunktet er flyttet til (0, 45).



Figur 1: En enkel webkart-klient i JavaScript som sender WMS-forespørsler basert på brukerinput.

Denne tilnærmingen er veldig forenklet, siden vi ikke tar hensyn til ting som hvor musepekeren er når brukeren zoomer, hvor mye musepekeren beveges, forskjellige kartlag over hverandre, høyde-bredde-forhold samt at vi bruker uprojiserte, geografiske koordinater. Alt dette kan løses ved å bruke riktige formler og transformasjoner, og det er nettopp dette et webkart-bibliotek gjør. Hovedprinsippet er uansett det samme som beskrevet i eksempelet. Problemet med forskjellige projeksjoner og verdensdekkende kart løses ofte ved å bruke en kartserver som støtter den såkalte «Web Mercator»-projeksjonen (eller Pseudo Mercator, se [15]), høyde-bredde-forhold har også en relativt enkel matematisk løsning. Det å beregne nye områder og punkter kan gjøres enklere ved at man har kontroll på «bildekoordinatene» internt i HTML-elementet kartet opererer i. Transformasjon av bildekoordinater til koordinater i ønsket koordinatsystem er også en matematisk enkel oppgave, som igjen kan løses med JavaScript.

Som sagt benyttet vårt eksempel en WMS-server som returnerer et kartbilde uten å sette krav til hvilke oppløsninger og områder man spør om. Slike løsninger fungerer greit for enkle datasett uten stor trafikk, men det krever at kartserveren tegner opp et nytt kartbilde for hver forespørsel. For å unngå dette bruker man ofte såkalte «Tile-servere» for webkartløsninger med høy trafikk og krav til responstid. Disse serverne fungerer på mange måter likt som WMS-servere, men istedenfor å spørre serveren om et kartbilde spør man om en rekke mindre bilder («kartfliser»), som settes sammen i nettleseren slik at brukeren opplever å se et bilde. Disse flisene ligger ferdige som bildefiler på serveren, og har forhåndsbestemt oppløsning og utstrekning. Dermed blir oppgaven med å returnere dem mye mindre. Web-kart-klienten analyserer hvilket område brukeren ser på, finner ut hvilke kartfliser som trengs for å vise dette området og henter disse fra serveren basert på en kjent standard (for eksempel den nevnte WMTS-standarden). Når bildene kommer tilbake fra serveren er det bare snakk om litt enkel koordinasjon for å plassere dem riktig og vise brukeren et sømløst kart.

Denne kombinasjonen av matematiske beregninger, HTTP-forespørsler og DOM-manipulasjon er kjernen i ethvert webkart-bibliotek. I tillegg til dette tilbyr de fleste bibliotekene funksjonalitet for å håndtere flere samtidige kartlag, plassering av markører i

kartet og metoder for å håndtere input fra brukere og annen kode. Alt dette kunne i prinsippet blitt utviklet fra bunnen av hver gang man skulle vise et interaktivt kart på nett, men dette er lite rasjonelt og ville ført til at de fleste webkart-prosjekter ville kostet mer å utvikle og blitt levert senere. Siden funksjonaliteten beskrevet er så generell som den er, er det mer naturlig å løse dette problemet en gang og senere gjenbruke koden: som et webkart-bibliotek.

Punkter og geometrier i en webkart-klient

Generelt kan man si at webkart-klienter bruker enten raster- eller vektordata. Rasterdata kan, når vi snakker om webkart, være data generert på bakgrunn av vektordata på en server og levert til klienten som rasterdata. Dette er ofte tilfellet med WMS- og tile-tjenester, eller man kan få levert «ekte» raster-data via den beslektede WCS-standarden.

Ved å generere et rasterkart, eller et bilde, på serveren flytter man den tunge prosesseringen vekk fra den enkelte bruker og gjør det på en server som ofte har større kapasitet. I lang tid var denne løsningen den eneste mulige: nettlesere manglet kapasitet til å generere kartbilder basert på vektordata, men med fremskrittene i JavaScript-motorer og nye standarder som HTML5 er dette i rask endring. Det å tegne ut vektordata byr på andre problemer enn å håndtere rasterdata som i eksempelet over, en ting er prosesseringskraft og minne i nettleseren, en annen sak er båndbredde og hastighet for å overføre dataene fra server til klient. Med den nevnte nettleser-utviklingen, mer minne tilgjengelig og smarte triks som «vektor-tiling» (se [16] for en grundig gjennomgang av dette) er det nå mulig å lage et webkart basert på «rene» vektordata. Dette har en rekke fordeler sammenlignet med rasterdataene man får via WMS. Skalerbarhet, mulighet for å spesialtilpasse og justere kartografien, samt det å kunne vise og skjule utvalgte geometrier og reprojisering av data i klienten er noen slike fordeler [17]. Utfordringene med å få et dokumentformat som HTML til å vise vektorgeometrier kan løses ved hjelp av programtillegg som Flash og Silverlight [17], men som nevnt har slike tillegg en rekke ulemper og det finnes ulike innebygde løsninger i nettleserne som kan brukes. Dette er løsningene som brukes av de fleste webkart-klienter. Et godt eksempel på en slik løsning er Canvas-elementet som ble introdusert i HTML5. Dette fungerer godt for å vise vektordata, det er i prinsippet et lerret i en del av HTML-dokumentet, indeksert med lokale bildekoordinater der man programmatisk kan tegne geometriske primitiver [18]. En annen løsning er WebGL [19], en web-versjon av grafikkstandarden OpenGL, som også innfører støtte for 3d-grafikk. Før HTML5 ble introdusert ble teknologier som VML og SVG brukt [20]. Disse løser mange av de samme problemene, om enn ikke på en like effektiv måte. Disse teknologiene er fortsatt i bruk i flere webkart-klienter, blant annet OpenLayers [21].

Uansett hva slags underliggende teknologi som brukes for å tegne opp vektordataene i HTML-dokumentet er hovedoppgaven til JavaScript-koden de samme: Holde rede på målestokk (eller zoom-nivå som det ofte kalles) og å transformere koordinater til bildekoordinater i HTML-elementet (som regel 0-indeksert i øvre venstre hjørne med piksler som enhet). I tillegg følger bibliotekene med på musebevegelser, zoom/pan og lar brukeren skru av og på kartlag og lignende funksjoner man forventer av et moderne webkart. De fleste webkart-klienter håndterer både raster- og vektordata, da mange av utfordringene har like løsninger. Det er også vanlig å operere med en kartklient som kombinerer raster- og vektordata i samme løsning, ofte med et bakgrunnskart som raster og overliggende lag som vektorgeometrier.

Tilgjengelige Webkart-bibliotek og APIer

Som nevnt finnes det en rekke tilgjengelige webkart-bibliotek, både som proprietær programvare og under fri-programvarelisenser. De proprietære løsningene kan deles inn i gratis- og betal-varianter. Disse gratisvariantene kalles ofte for webkart-APIer. Google Maps APIet er et godt eksempel på et proprietært, gratis, webkart-API, som inneholder både et JavaScript-bibliotek og innebygde

referanser til bakgrunnskartet til Google Maps. Flere av disse gratis-APIene har begrensninger på bruk og tilbyr ofte betal-løsninger i tillegg som gir tilgang til flere kartoppslag, garantier på oppetid og i noen tilfeller spesialtilpasset kartografi og andre kvalitetsjusteringer.

Fri-programvare-bibliotekene tilbyr som regel kun JavaScript-koden du trenger for å koble deg til en offentlig eller privat kartserver, via standarder som WMS, WMTS og TMS eller varianter av disse. Grunnet åpenheten fri-programvarebibliotekene gir finnes det flere tilfeller der proprietære webkartløsninger baserer seg på fri programvare. Det er også mulig å kombinere kartfliser fra en kommersiell leverandør med fri-programvare-biblioteker. Tabell 2 gir en oversikt over noen tilgjengelige webkart-bibliotek og API-er.

Tabell 2: Oversikt over noen webkart-APIer og bibliotek.

Navn	Type	Nettside
ArcGIS	Gratisbibliotek	http://help.arcgis.com/en/webapi/javascript/arcgis/
Bing Maps	Gratis-API	http://www.microsoft.com/maps/
deCarta	Betal-API	http://developer.decarta.com/
Google Maps	Gratis-API	https://developers.google.com/maps/
HERE Maps	Gratis-API	http://developer.here.com/javascript-apis
jump	Fri-programvarebibliotek	http://eikes.github.io/jump/
Leaflet	Fri-programvarebibliotek	http://leafletjs.com/
MapQuest	Gratis-API	http://developer.mapquest.com/
ModestMaps	Fri-programvarebibliotek	http://modestmaps.com/
OpenLayers	Fri-programvarebibliotek	http://openlayers.org
OpenLayers 3	Fri-programvarebibliotek	http://ol3js.org/
Polymaps	Fri-programvarebibliotek	http://polymaps.org/

Konklusjon

Det finnes en rekke tilgjengelige webkart-bibliotek og de fleste av dem baserer seg på de samme prinsippene, beskrevet relativt detaljert her. Rasterdata og tile-servere har i mange år vært standarden for kart på web, men utviklingen i nettleserteknologi de siste årene har gjort det mulig og i mange tilfeller ønskelig å basere en webkart-løsning på vektordata. De tidligste forsøkene på å lage kartklienter for nettlesere led under manglende standarder og teknologi. I dag beveger nettleserteknologien seg fort og åpner stadig nye muligheter for å lage gode kartløsninger på nett. Den kommende HTML 5-standarden og nettleserteknologier som Canvas og WebGL vil bli viktige for webkartløsninger i fremtiden. Trenden med «enkeltside-applikasjoner» er også et tegn på at webkart-klienter er i ferd med å bli en naturlig del av web-applikasjoner og at det skjer mye med webkart-APIer for tiden [22].

Selv om de tekniske løsningene, standardene og bibliotekene som presenteres i forskjellige webkart kan virke kompliserte er kjerneprinsippene relativt enkle. Ved å forstå på et overordnet plan hvordan et webkart fungerer, hvilke standarder og prinsipper som brukes, står man bedre rustet til å ta beslutninger om hva som skal brukes, hvordan det skal brukes og når man burde bruke de forskjellige løsningene.

Referanser

[1] Stefanakis, E. (2012). Map Mashups and APIs in Education. In *Online Maps with APIs and WebServices* (pp. 37–58). Springer Berlin Heidelberg

[2] Detwiler, Jim & Dutton, John A. (2009). *Evolution of Web Mapping Technology.* Tilgjengelig på: https://www.e-education.psu.edu/geog863/resources/l3_p3.html (Hentet 19.05.2014)

[3] Wikipedia. *History of web mapping*. Tilgjengelig på: http://en.wikipedia.org/w/index.phptit-

le=Web_mapping&oldid=549697306#History_of_web_mapping (Hentet 19.05.2014)

[4] Miller, C. C. (2006). «A beast in the field: The google maps mashup as gis/2». Cartographica: *The International Journal for Geographic Information and Geovisualization*, 41(3):187–199.

[5] Sveen, A. F. (2009). «*Map 2.0. User Interaction in Web Mapping Sites*». Master's Thesis, Norwegian University of Technology and Science, Trondheim, Norway. Tilgjengelig på: http://code.atlefren.net/download/dl.php?id=11

[6] Peterson, M. P. (2012). «Online Maps with APIs and WebServices». In *Online Maps with APIs and WebServices* (pp. 3–12). Springer Berlin Heidelberg

[7] OpenLayers Wiki «OpenLayers History». Tilgjengelig på: http://trac.osgeo.org/openlayers/wiki/History (Hentet 19.05.2014)

[8] Waxman, A. (2000). *ArcIMS 3.0 – An Application Developer's Perspective.* Tilgjengelig på: http://spatialnews.geocomm.com/newsletter/2000/22/arcims.html (Hentet 19.05.2014)

[9] CloudMade *Blog (2011) Announcing Leaflet: a Modern Open Source JavaScript Library for Interactive Maps.* Tilgjengelig på: http://blog.cloudmade.com/2011/05/13/announcing-leaflet-a-modern-open-source-javascript-library-for-interactive-maps/ (Hentet 19.05.2014)

[10] Willmott, S. (2012). *HTML, Javascript and the appification of the Web*. Tilgjengelig på: http://pandodaily.com/2012/12/06/html-javascript-and-the-app-ification-of-the-web/ (Hentet 19.05.2014)

[11] Resig, J. (2008). *JavaScript Performance Rundown.* Tilgjengelig på: http://ejohn.org/blog/javascript-performance-rundown/ (Hentet 19.05.2014)

[12] Open Geospatial Consortium (2006) *OpenGIS Web Map Service (WMS) Implementation Specification.* Tilgjengelig på: http://www.opengeospatial.org/standards/wms (Hentet 19.05.2014)

[13] Open Geospatial Consortium (2010*). OpenGIS Web Map Tile Service Implementation Standard.* Tilgjengelig på: http://www.opengeospatial.org/standards/wmts (Hentet 19.05.2014)

[14] OSGeo *Tile Map Service Specification.* Tilgjengelig på: http://wiki.osgeo.org/index.php?title=Tile_Map_Service_Specification&oldid=62686 (Hentet 19.05.2014)

[15] FMEPedia (2011). *Spherical or Web Mercator Coordinate System.* Tilgjengelig på http://fmepedia.safe.com/articles/FAQ/Spherical-or-Web-Mercator-Coordinate-System (Hentet: 19.05.2014)

[16] Taraldsvik M.(2012) *The future of web-based maps: can vectortiles and HTML5 solve the need for high-performance delivery of maps on the web?* Master's Thesis, Norwegian University of Technology and Science, Trondheim, Norway. Tilgjengelig på: https://github.com/meastp/efficientvectortiles/raw/master/efficientvectortiles.pdf

[17] Lienert, C., Jenny, B., Schnabel, O. and Hurni, L. (2012) «Current Trends in Vector-Based Internet Mapping: A Technical Review». In *Online Maps with APIs and WebServices* (pp. 23–36). Springer Berlin Heidelberg

[18] Canvas «Mozilla Developer Network». Tilgjengelig på https://developer.mozilla.org/en-US/docs/HTML/Canvas. (Hentet 19.05.2014)

[19] Khronos Group (2013) *WebGL Specification.* Tilgjengelig på http://www.khronos.org/registry/webgl/specs/latest/ (Hentet 19.05.2014)

[20] Schnabel, O. and Hurni, L. (2009). Cartographic web applicationsdevelopments and trends. In *Proceedings of the 24th international cartography conference*, Santiago

[21] Hazzard, E. (2011). *OpenLayers: Overview of Vector Layer.* Tilgjengelig på http://www.packtpub.com/article/openlayers-overview-vector-layer (Hentet 19.05.2014)

[22] Leler, W. (2013). *Guide to Google Maps API – and 6 great alternatives.* Tilgjengelig på http://www.creativebloq.com/web-design/google-maps-api-7122779 (Hentet 19.05.2014)