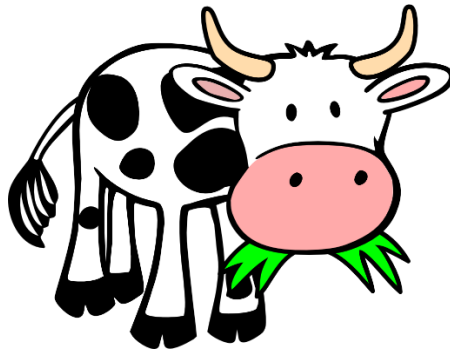

PROSJEKTRAPPORT

i faget TBA4251 Programmering i Geomatikk

Av Torbjørn Auglænd Vilhelmsen



PROSJEKTRAPPORT

INNHold

Introduksjon	3
Oversikt	3
Hvorfor Javascript og HTML	4
Arbeidsprosessen og vurderinger underveis	4
Programmets struktur	5
Biblioteker	6
Leaflet	6
Shapefile-js	6
SOSI.js	6
JavaScript Topology Suite (JSTS)	7
Andre biblioteker	7
Klasser	7
Basemapselector	8
Asynkront arbeid og persistent lagring	8
Web workers	8
Indexed database api	9
Bruk av programmet	9
Installasjon	10
Støtte i nettlesere	10
Behandling av kartlag	10
Legge til kartlag	10
Ordne kartlag	10
Endre farge, gjennomsiktighet og lagnavn	11
Slette kartlag	11
Slå av/på visning av kartlag	11
Velge bakgrunnskart	11
Bruk av verktøyene	11
Buffer	11
Dissolve	12
Simplify	12
Union	12
Intersection	12
Difference	12
Featureextractor	13
Lagring av data	13

PROSJEKTRAPPORT

Problemer	13
Mangler og bugs.....	14
Konklusjon.....	14
Fremtidige forbedringer.....	14
Vedlegg 1 – eksempeløving.....	15
Eksempeløving - introduksjon til et GIS	15
Gjennomføring av øvingen	16
Referanser	17

PROSJEKTRAPPORT

INTRODUKSJON

På NTNU er det lenge blitt brukt en løsning som heter GIST til å introdusere brukerne til GIS gjennom øvinger. Dette er en litt gammel raster basert løsning som dessverre begynner å dra litt på årene i forhold til kompatibilitet med nyere versjoner av Windows. Ved å bruke nettleseren til å gjøre slike analyser kan brukeren uten å installere fort få en innsikt i hvordan et GIS virker og hvordan grunnfunksjonaliteten er bygget opp.

Formålet med oppgaven er å lage et enkelt GIS verktøy med mye av de samme eller tilsvarende funksjonene som finnes i GIST. Dette kunne gjøres enten basert på raster eller vektorer.

I dette prosjektet valgte jeg å basere min implementasjon på vektorer. Programmet er implementert 100% i JavaScript og HTML og resultatet finnes på <http://kugis.projeksjon.no/>.

OVERSIKT

Jeg har valgt å implementere løsningen i ren HTML og JavaScript uten noen form for server funksjonalitet. Dette fordi det er et område i stor utvikling og nye API og løsninger gjør at fullfunksjonalitets programmer stadig blir mer aktuelt å implementere i nettleseren. Særlig Chrome og Firefox har gjort store forbedringer i JavaScript-motoren de siste årene som gjør at bruk av denne er blitt flere ganger mer effektivt.

Jeg har valgt å fokusere på å bruke noen av disse nye API-ene til å gjennomføre denne oppgaven for å få en god opplevelse av et slikt GIS.

Noen av API-ene jeg har brukt er:

- Web Workers
- IndexedDB
- FileReaderAPI

Disse muliggjør at man får litt av funksjonaliteten man forventer av en native applikasjon. Slik som flertrådet-jobbing, persistens mellom sesjoner og lesing av lokale filer.

For å presentere kartdataene er kartbiblioteket Leaflet brukt. Dette er et ypperlig og lett bibliotek med god støtte for å vise komplisert vektorgrafikk.

For bedre formatstøtte er bibliotekene SOSI.js (SOSI-filer) og shapefile.js (ESRI Shape) brukt. I tillegg støttes GeoJSON native i Leaflet.

For de romlige operasjonene er biblioteket JavaScript Topology Suite (JSTS) brukt. Dette er et bibliotek som er basert på funksjonaliteten i Java Topology Suite (JTS). JSTS har funksjonalitet blant annet for buffring og overlay-funksjoner.

Proj4js brukes for å konvertere mellom projeksjoner.

Andre biblioteker som er brukt er jQuery og jQueryUI som hjelpebiblioteker for å forenkle jobbingen med HTML objekter i JavaScript.

Programmet baseres på mye drag and drop funksjonalitet som har vist seg å være intuitivt for nye brukere av systemer.

PROSJEKTRAPPORT

HVORFOR JAVASCRIPT OG HTML

JavaScript og HTML ble brukt for å gi en enkel installasjonsfri opplevelse uten mye vedlikehold. Det er skjedd mye i denne verdenen og muligheten for å lage fullverdige applikasjoner har blitt veldig bra. Særlig gjennom de to foreslåtte standardene IndexedDBⁱ og Web Workersⁱⁱ er det i nyere nettlesere gitt muligheter for flertrådet arbeid og persistent lagring. Dette var to områder jeg ønsket å utforske ved dette prosjektet.

Det har også tidligere vært i dette faget laget webløsninger som benytter en applikasjonsserver til å utføre analysene. Dette er et ekstra vedlikeholds- og kostnadsledd som kan være en hindring i å ta i bruk og drifte et slikt GIS. Jeg ønsket derfor ikke å benytte en slik løsning.

Det er også blitt et veldig aktivt utviklingsmiljø i JavaScript innen GIS og det finnes derfor mange bibliotek som kan benyttes.

Samtidig er det ikke et «ideelt» språk for en slik oppgave og da det er et tolket og ikke et kompilert språk så er det nok ikke like effektivt til en slik oppgave som flere av alternativene. Den ineffektiviteten som oppstår pga. dette mener jeg kan være betydelig og dette merkes spesielt på større datasett. For et slikt demo-GIS som dette mener jeg det allikevel ikke vil være et problem, så lenge man er oppmerksom på de svakheter som finnes.

En webløsning i JavaScript og HTML vil også ha begrenset tilgang til operativ- og filsystemet som gjør at det vil finnes en del begrensinger på hvilke effektiviseringer man kan gjøre.

ARBEIDSPROSESSEN OG VURDERINGER UNDERVEIS

Jeg kom tidlig frem til at en nettløsning var veien og beveget meg litt ut på utforsket område når det kom til GIS-applikasjoner av denne typen. Mye har blitt gjort før men de fleste løsninger jeg fant var enten enkeltstående bibliotek som gjorde en ting eller så baserte de seg på en applikasjonsserver-arkitektur slik som fra for eksempel ESRI eller til og med PostGIS. Dette var et ledd som jeg ikke ønsket å ha i min løsning.

Da det allerede finnes enkelte biblioteker som utfører flere GIS oppgaver i JavaScript ønsket jeg heller å fokusere på å sy dette sammen til en helhetlig løsning med et grensesnitt som er enkelt å bruke.

Jeg bestemte meg så for å benytte kartbiblioteket Leaflet for presentasjon og etter en diskusjon med Alexander Nossum, Keino Valstad og Sverre Wisløff (alle Norkart AS) falt valget på å benytte biblioteket JSTS for å utføre analysene.

OpenLayers var et annet aktuelt kartbibliotek og bruke og da dette har en bedre projeksjonsstøtte kunne nok dette vært et godt egnet verktøy i denne sammenhengen. Jeg valgte å ikke å bruke det grunnet manglende erfaring med hvordan det er i bruk.

Oppstarten av prosjektet var i starten av september. I starten var jeg helt klar på at man i tillegg til å utføre analysene lokalt også skulle kunne bruke egne data. Jeg fant raskt et bibliotek som tok for seg innlesningen av shapefiler over til GeoJSON i JavaScript og integrerte det i min egen kode for bruk med Leaflet. GeoJSON er støttet «out-of-the-box» i Leaflet så alt som trengtes der var å skrive en metode for å lese lokale filer.

I september var jeg så på dagseminaret «FOSS4G i Norge» arrangert av Norkart i Oslo. Her kom vi utfor en «heftig» diskusjon rundt SOSI-formatet. Diskusjonen gikk i ettertid over på Twitter der jeg og Atle Frenvik Sveen (Bouvet) tok initiativet til å lage et JavaScript-bibliotek for å konvertere fra SOSI til GeoJSON og andre formater.

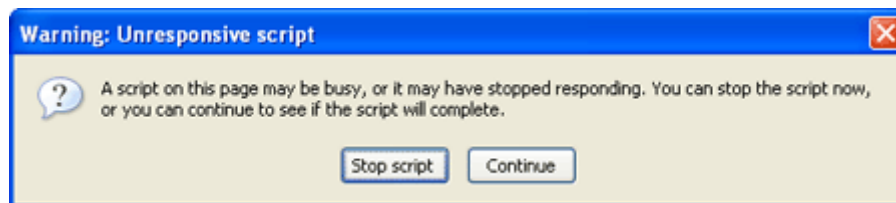
PROSJEKTRAPPORT

Biblioteket kan hentes på <https://github.com/atlefren/sosi.js>. Dette biblioteket er ikke en del av innleveringen da min del av arbeidet ikke lenger er en vesentlig del av det endelige produktet.

Tanker om grensesnittet falt så på plass. Det var naturlig å hente inspirasjon fra programmer som skal utføre noen av de samme oppgavene slik som QGIS og ArcGIS. Begge opererer med et stort panel for kartet og en eller flere side paneler i samme vindu. Samtidig som dette falt på plass så jeg at det kunne være et behov for å kunne reorganisere rekkefølgen kartlagene vises på. Tok igjen inspirasjon fra disse verktøyene og la til en «drag and drop» løsning for dette i tillegg.

Da omfanget på programmet mitt er vesentlig mindre enn de to nevnte alternativene bestemte jeg meg for å kun ha et slikt sidepanel.

I begynnelsen av november/slutten av oktober begynte hoveddelen av programmet å komme på plass men jeg opplevde ikke brukeropplevelsen som så veldig bra da hele nettleservinduet frøys under analysejobbingen og nettleseren stadig ga beskjed om at siden hang.



Dette er ikke en melding som er ønskelig at uerfarne brukere skal få. De vil da gjerne tro at nettleseren har krasjet og gi opp.

Web Workers ble løsningen på dette. Web Workers er et standard API som gjør det mulighet for å gjøre flertrådet arbeid i en nettleser. Dette løste mye men introduserte også et nytt tanke sett som gjorde at mye av koden måtte omstruktureres eller skrives helt om.

Det var også noen av verktøyene som tok veldig lang tid å kjøre, en «Dissolve»-prosess kunne fort ta en halvtime selv på enkle datasett. Så mye av arbeidet gikk heretter på å effektivisere kjøretiden på disse verktøyene og jeg vil si at de fleste verktøyene nå har fått en tilnærmet akseptabel kjøretid på tilpassede datasett.

Etter mye bugfixing og testing var det en ting som irriterte meg veldig: Hver eneste gang jeg måtte laste inn siden på nytt forsvant all dataen jeg hadde jobbet med. Dette måtte fikses. Jeg hadde hørt om et API som heter LocalStorage men aldri brukt det. Da jeg undersøkte nærmere fant jeg fort ut at dette var uegnet siden det var en plass begrensning på 5 MB per domeneⁱⁱⁱ. Da noen av datasettene kunne være vesentlig større enn dette måtte jeg se etter andre løsninger. Løsningen ble Indexed Database API. Her er det ikke spesifisert en slik plass begrensning men mange nettlesere har innført at dersom plassbruken blir for stor vil brukeren få beskjed om dette og valget mellom å avvise eller akseptere den store plassbruken. Denne siste forbedringen ble gjort helt på tampen før innleveringsfristen og kunne nok vært gjort bedre.

PROGRAMMETS STRUKTUR

All koden som ikke ligger i mappen `libs/` er i hovedsak egenprodusert og en del av denne innleveringen.

JavaScript-koden ligger for det meste i mappen `js/` og `workers/` samt at finnes litt kode i `index.html` i rotmappen.

PROSJEKTRAPPORT

BIBLIOTEKER

Programmet er bygget opp av flere biblioteker. Jeg vil i denne seksjonen gå inn på noen av de viktigste og hvilken rolle de har hatt. På slutten av seksjonen vil jeg ramse opp resten med en kortere beskrivelse av bruken.

LEAFLET

Leaflet har vært nevnt flere ganger i rapporten, med god grunn. Det er et lettvekts kartbibliotek med enkle funksjoner for visning av kartdata på vektorformat og tilebaserte bakgrunnskart. Det er en rekke optimaliseringer som gjør at dette biblioteket er særlig egnet for å vise kompliserte vektorbaserte datasett. Særlig er Leaflet effektiv på generaliseringen som gjøres før visning av dataene. Dette gjør at visningen ikke alltid er like presis mens dataen i bakkant er beholdt. En av svakhetene til Leaflet over f.eks. OpenLayers er projeksjonsstøtte mer om dette i Problem-seksjonen.

Hovedoppgaven til Leaflet i dette prosjektet er å vise kartdataene, til dette benyttes map-objektet samt GeoJSON-klassen. Leaflet har også noen funksjoner som er brukt i andre sammenhenger. Blant annet finns det flere hjelpemetoder for å skrive JavaScript objekt orientert innbakt i Leaflet. Det finnes funksjoner for å opprette klasser samt å utvide dem. Noe JavaScript i utgangspunktet ikke har. De klassene jeg har skrevet er avhengige av Leaflet på grunn av dette. Det finnes så klart andre metoder for å gjøre dette, men ved å bruke Leaflets klassefunksjonalitet har dette blitt vesentlig enklere.

Eksempel på hvordan man skriver en klasse ved hjelp av Leaflet:

```
var klasse = L.Class.extend({  
  
  initialize: function (options) { // konstruktør  
  
  },  
  
  var1: «value»,  
  
  metode: function (arguments) {  
  
  }  
  
});
```

SHAPEFILE-JS

Shapefile-js er brukt for å lese inn shapefiler og konvertere de til GeoJSON-objekter. Mye inspirasjon er hentet her fra i hvordan man håndterer å legge til filer lokalt fra maskinen. Mye av drag and drop koden er hentet herfra med noen modifikasjoner. Det er innebygd funksjonalitet her for å lese blant annet zip-filer. Siden en Shape-«fil» i hovedsak består av tre filer.

SOSI.JS

Utviklet i hovedsak av Atle Frenvik Sveen (Bouvet AS) med hjelp og testing av Tomas Hirsch (Kartverket), Sigbjørn Tillerli Herstad (Norkart AS) og meg selv. Dette biblioteket håndterer innlesning av SOSI-filer fra og med SOSI versjon 4.0 og konverterer dem til GeoJSON. Enkelte geometrier som klotoider er ikke støttet og forfatteren av biblioteket vil ikke gå god for robustheten i biblioteket.

PROSJEKTRAPPORT

JAVASCRIPT TOPOLOGY SUITE (JSTS)

JSTS står for mye av analyse arbeidet i prosjektet. Den inneholder flere funksjoner for overlays som Union, Intersection og Difference samt buffering. Det er disse funksjonene jeg har brukt i analyse-verktøyene. Det er mange flere muligheter i dette biblioteket så å bygge videre på dette for å lage en mer komplett GIS-løsning er det gode muligheter for.

ANDRE BIBLIOTEKER

jQuery, jQueryUI forenkler mye av interaksjonen mellom JavaScript og UI-et. «Drag and drop»-funksjonaliteten innad i dokumentet tas hånd om av disse bibliotekene.

FontAwesome brukes for å vise ikoner i HTML-koden.

Proj4JS brukes for å transformere koordinater mellom projeksjoner.

Wicket.js brukes for å konvertere mellom GeoJSON og Well Known Text (WKT).

KLASSER

De klassene jeg har skrevet selv er i hovedsak klassene for sidepanelet kalt «Sidebar» og laglisten, «Sidebar.Layerlist», lagene «Sidebar.Layerlist.Button» og verktøyene «Sidebar.Tool».

Disse klassene er skrevet for å være lett utvidbare. Et verktøy skal lett legges til og nye funksjoner skal enkelt kunne lages i for et lag.

Et verktøy i mitt prosjekt er definert av noen standard funksjoner og variable og en veldig simpel versjon kan se slik ut:

```
sidebar.tool.intersect = sidebar.tool.extend({  
  
  title: "intersection",  
  
  wkt1: null,  
  
  wkt2: null,  
  
  _droppabletext: "drop two layers here in succession to create an  
intersection between them.",  
  
  afterdrop: function (event, context) {  
  
    // hva som skjer etter et lag er blitt droppet på verktøyet.  
  
  },  
  
  createtooloptions: function () {  
  
    element = l.domutil.create("div", "tool-options");
```


PROSJEKTRAPPORT

```
this._droppable = l.domutil.create('div', 'droppable');

var con = this;

$(this._droppable).droppable({

    drop: function (event, ui) {

        con.afterdrop(ui, con);

    }

});

this._droppable.appendChild(document.createTextNode(this._drop
pabletext));

element.appendChild(this._droppable);

return element;

}

});
```

De elementene som kreves er variablene `title` og `_droppabletext` og funksjonene `createtoolOptions`, som definerer hva som vises av opsjoner for verktøyet, og `afterDrop` som definerer hva som skjer etter man har sluppet ett eller flere lag på verktøyet.

BASEMAPSELECTOR

Basemapselector er en klasse som tar inn en liste med `L.TileLayer` og generer en forhåndsvisning av dem. Denne forhåndsvisningen er basert på en et område utenfor Oslo som er veldig representativt for hvordan kartet vil se ut i forskjellig terrengtyper.

ASYNKRONT ARBEID OG PERSISTENT LAGRING

WEB WORKERS

Web Workers er et nytt API som «spawner» en ny tråd for å utføre tyngre arbeid i JavaScript. Dette gjør at man kan offload mye av de tyngre analysene som gjøres til en slik prosess uten at man fryser nettløsningen.

En Web Worker starter som en egen prosess og har ikke tilgang til de variabler og pekere som finnes i det originale dokumentet. Derfor er man nødt til å kopiere disse objektene over til «scopet»-som Web Workeren har. Her er det mange muligheter for å gjøre feil og mangedoble den prosesseringskraften og minnebruken som strengt tatt er nødvendig. Det er også begrensede kommunikasjons muligheter mellom dokumentet og

PROSJEKTRAPPORT

Web Workeren. I prinsippet kan man sende alt man vil frem og tilbake med message-funksjonen men i praksis vil dette være veldig ineffektivt pga. minnebruken som er diskutert før.

Under er et eksempel på et veldig enkel Web Worker og JavaScript som adderer to tall.

Worker.js:

```
onmessage = function(event) {  
  
    var num1 = event.data.num1;  
  
    var num2 = event.data.num2;  
  
    var result = num1+num2;  
  
    postMessage(result);  
  
};
```

Javascript.js

```
var adder = new Worker("worker.js");  
  
adder.onmessage = function (event) {  
  
    // event.data contains the result  
  
}  
  
adder.postMessage({ num1: 2, num2: 7 });
```

Som man kan se over er ikke en JavaScript Web Worker spesielt vanskelig å jobbe med så lenge man er obs på de begrensningene som finnes samt at den kjøres nesten helt på egenhånd.

Jeg har skrevet to Workere som tar for seg tyngre oppgaver i analysen. Disse finnes i workers/buffer.js og workers/dissolve.js. De andre oppgavene som programmet foretar seg kunne nok også ha skjedd i en Worker men på grunn av tidsmessige årsaker er ikke dette implementert.

INDEXED DATABASE API

Indexed Database API (IndexedDB) er en asynkron måte å lagre data lokalt på brukerens maskin. Det er ikke definert noen maksimum lagringskapasitet og siden lagringsfunksjonene er asynkrone vil man merke en minimal innvirkning på kjøretiden på dagens flerkjerne prosessorer.

BRUK AV PROGRAMMET

Programmet baserer seg på bruk av egne kartdata. Lagring til disk utføres automatisk mens man holder på. Dette er en asynkronprosess og vil i de fleste tilfeller ikke merkes. Det er også vedlagt en eksempeløving løst basert på GIST-øvingen i faget Geografisk informasjonsbehandling I.

PROSJEKTRAPPORT

INSTALLASJON

Installasjonen av programmet er ganske enkelt. Eneste som må gjøres er å pakke ut den vedlagte kildekoden til en mappe på en web-tjener. Ingen konfigurasjon skal være nødvendig.

STØTTE I NETTLESERE

Teknologien som er brukt er godt støttet i de nyeste nettleserne til tross for at dette kun er foreslåtte standarder. Begrensningen går i hovedsak på IndexedDB. En tabell over hvilke nettlesere og fra hvilken versjon som støttes (i teorien) er lagt til under.

	IE	Firefox	Chrome	Opera	Opera Mobile	Chrome for Android	Firefox for Android	IE Mobile
From version	10.0	10.0	23.0	15.0	16.0	31.0	25.0	10.0

BEHANDLING AV KARTLAG

LEGGE TIL KARTLAG

Programmet har støtte for 3 formater: GeoJSON, SOSI ver. 4 og opp og ESRI Shape. For shapefiler må .shp, .dbf og .prj filene Zippes i en og samme zip-fil. Det kan være flere Shapefiler i en zip-fil. GeoJSON og SOSI skal ikke være zippet.

For å legge til lagene må man dra de over fra filutforskeren til enten kartvinduet eller til boksen i sidevinduet.

ORDNE KARTLAG

Man kan ordne visnings rekkefølgen på lagene ved å dra dem oppover eller ned over i listen med musepekeren.

PROSJEKTRAPPORT

ENDRE FARGE, GJENNOMSIKTIGHET OG LAGNAVN

Ved å trykke på «tannhjul»-ikonet ved siden av kartlagsnavnet kan man endre enkelte visningsalternativer for kartlagene, man får da opp dette vinduet:



Her kan man endre visningsnavnet på laget, linjefarge og gjennomsiktighet, fyllfarge og gjennomsiktighet.

Opacity går fra 0-1, der 0 er helt gjennomsiktig. Punktum og ikke komma benyttes for å angi desimaler.

Endringene lagres mens man skriver og man lukker menyen ved å trykke på tannhjulet igjen.

SLETTE KARTLAG

Man sletter laget ved å trykke på søppelkassen ved siden av lagnavnene.

SLÅ AV/PÅ VISNING AV KARTLAG

Kartlagene kan slås av/på ved å trykke på huket helt til høyre ved kartnavnene

VELGE BAKGRUNNSKART

Ved å trykke på «Choose basemap» på bunnen av kartlagslisten får man opp en rekke forhåndsdefinerte bakgrunnskart. Disse er hentet fra Kartverkets cache^{iv} eller pluginen «Leaflet.providers»^v.

Man velger ønsket kartlag ved å trykke på forhåndsvisningen.

BRUK AV VERKTØYENE

Løsningen kommer med flere verktøy for å utføre enkle GIS analyser en kort beskrivelse av hvert enkelt verktøy samt hvordan det er i bruk. Verktøyene fungerer slik at man drar et eller flere lag fra laglisten til det aktuelle verktøyet. For noen av verktøyene er rekkefølgen man drar lagene relevant dette vil være indikert i teksten.

BUFFER

Lager en buffer på angitt antall meter for hvert enkelt geografisk objekt for så å slå dem sammen til et eller flere polygoner der det er overlapp.

Skriv først inn antall meter du ønsker bufferen, dra så over kartlaget du vil lage en buffer på.

PROSJEKTRAPPORT

DISSOLVE

Slår sammen polygon/linjer til et polygon/linje. Der det er overlapp/kontakt mellom lagene vil det dannes «singlepart» objekter. Objekter som ikke har overlapp vil gjøres om til «multipart»-objekter. Kompliserte geografiske objekter vil ta lengre tid. Du vil derfor oppleve at prosessen tar lenger tid mot slutten av prosesseringen.

SIMPLIFY

Støtter kun polygon og linjer. Ikke multi-part objekter.

Basert på en modifisert Douglas-Peucker algoritme^{vi} innebygd i Leaflet. Toleransen er antallet grader du tolererer at en linje flytter seg. Endepunktene til en linje vil alltid være bevart.

UNION

Slår sammen to lag til ett.

Det kan i enkelte tilfeller lønne seg og kjøre «Dissolve» på laget først. Spesielt dersom prosessen feiler.

INTERSECTION

Tar inn to lag og lager et nytt lag der kun de områdene som overlapper vil være med. Det kan i enkelte tilfeller lønne seg og kjøre «Dissolve» på laget først. Spesielt dersom prosessen feiler.

DIFFERENCE

Tar inn to lag og lager et nytt lag der kun områdene i lag 1 som ikke er i lag 2 er med. Lagnummeret indikerer rekkefølgen lagene blir sluppet på.

Det kan i enkelte tilfeller lønne seg og kjøre «Dissolve» på laget først. Spesielt dersom prosessen feiler.

PROSJEKTRAPPORT

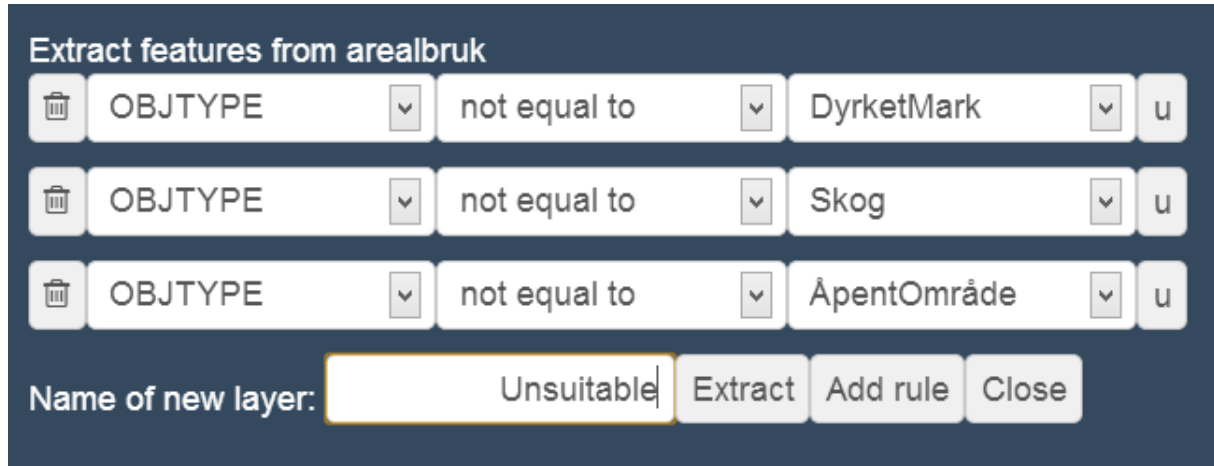
FEATUREEXTRACTOR

En enkel spørremotor som henter ut lag basert på objektenes egenskaper.

Man drar over laget man vil hente ut «Features» fra og trykker så «Add Rule». Her kan man legge til flere regler. Reglene er «AND» og ikke «OR». For å hente ut unike verdier til et felt kan man trykke på «U» til høyre i regelfeltet.

For å utføre «spørringen» trykker man på «Extract».

Merk at de «featurene» man henter ut vil fjernes fra det originale kartlaget.



Extract features from arealbruk

	OBJTYPE	▼	not equal to	▼	DyrketMark	▼	u
	OBJTYPE	▼	not equal to	▼	Skog	▼	u
	OBJTYPE	▼	not equal to	▼	ÅpentOmråde	▼	u

Name of new layer:

LAGRING AV DATA

Av sikkerhetshensyn er det begrensninger i JavaScript og HTML for å lagre data på egendefinerte områder. Lagene lagres likevel lokalt på maskinen av nettleseren og så lenge man fortsetter i samme nettleser og på samme PC vil arbeidet man gjorde tidligere være lagret.

PROBLEMER

Det har vært en del problemer med hastigheten på enkelte av verktøyene, særlig «Dissolve» og «Union» har vært trege når geometriene blir komplekse. I begynnelsen lagde jeg en kø som gikk igjennom alle polygonene én for én og la de til et stort. Dette lagde særlig komplekse og vanskelige polygoner og beregningen for hvor neste polygon skal settes inn i «ringen» ble etter hvert veldig kompleks og tok svært lang tid ofte kunne det ta over 1 time.

Løsningen ble å ta ut to og to polygoner fra køen, sette de sammen med «Union» og så sette de tilbake sist i køen igjen og så iterere gjennom denne køen til kun to polygoner gjenstår. Dette gjorde prosesseringstiden vesentlig kortere til under 1 minutt for de fleste datasett ment for bruk i denne applikasjonen.

Minne bruk er også et stort problem ved bruk av Web Workers, i min applikasjon ble objektene kopiert over og over igjen mellom de to prosessene flere ganger. Dette er særlig ineffektivt på større datasett. Etter å ha satt meg litt mer inn i hvordan en Web Worker fungerer fant jeg fort løsningen og rettet opp denne feilen, problemet var at jeg sendte hele resultatet frem og tilbake for hver oppdatering av analysestatus. Minnebruket er enda relativt stort da det er vanskelig å cache vektorer som ikke vises til disk, men forbedringen var vesentlig.

PROSJEKTRAPPORT

MANGLER OG BUGS

Et av målene med prosjektet var at man kunne bruke egne datasett for å utføre beregninger. Dette er dessverre ikke helt oppnådd. JSTS er litt fintfølende når det kommer til topologi og på grunn av flyttallsfeil som ofte oppstår ved bruken av JavaScript vil JSTS ofte gi beskjed om topologifeil. Det er derfor viktig at man tester de datasettene som er tenkt brukes på forhånd og går igjennom hvordan de er tenkt brukt i en øvingssituasjon.

Det er også mangle på hvordan feil håndteres. Nå spyttes det ut en feilmelding til JavaScript-konsollen, denne kan aksesseres med Ctrl-shift-I i Chrome og Ctrl-Shift-J i Firefox.

«Dissolve»- og «Union» -verktøyene har også en litt skjev fremstilling av fremdriften i prosessen og brukeren kan nok bli lurt og til å tro at prosessen har hengt seg. Dette fordi at de siste stegene tar eksponentielt lenger tid enn de første stegene og statusfeltet tar ikke hensyn til dette.

Det er også noen mangler for støtte av mindre oppløsninger på skjerm dersom kartlagslisten blir lang.

Projeksjonsstøtten i Leaflet er også mangelfull. Leaflet bruker internt Web Mercator (EPSG:38567) men eksponerer geografiske koordinater gjennom. Dette fører til mye reprojisering spesielt når man skal utføre buffringsanalyser i meter og ikke i grader. Hvis man hadde fått gjennomført buffringen i grader ville man i tillegg opplevd en sammen klemming i øst-retning. Siden projeksjonen som brukes internt er Web Mercator og ikke geografisk.

KONKLUSJON

Programmet løser store deler av oppgaven på en god måte. Mye av mest brukte funksjonene i et vektorbasert GIS er implementert. Det er også presentert på en attraktiv måte med enkle metoder for interaksjon med nettsiden. Det er noen feil og mangler i bruk av utestede datasett men for preparerte datasett som tar hensyn til de feil og mangler som finnes i en slik løsning som dette, er dette en ypperlig måte interagere med de. Generalisering og begrensede områder er viktige her.

Løsningen er særlig god på å vise og presentere data. Og har over tid blitt min foretrukne måte å forhåndsvise datasett av typen Shape, GeoJSON og SOSI. Særlig SOSI har det fra før vært flere steg involvert i før man kan vise disse datasettene. Her kan man enkelt dra over en SOSI fil og hente ut enkelte objekttyper for å få en raskt oversikt over hva som befinner seg i filen.

FREMTIDIGE FORBEDRINGER

Dersom topologifeilene JSTS klager over blir fikset er dette en meget robust løsning som kan brukes aktivt for å vise og presentere kartdata samt utføre enkle analyser.

Minnebruket kan også forbedres ved å bruke et lokalt origo. Noe testing viser at minne bruket i enkelte tilfeller kan reduseres med 1/3 avhengig av hvor mye av dataen er koordinater og hvor mye er objektets egenskaper.

Ønsker derfor å jobbe videre med løsningen ut på våren. Samt forsøke å optimere kjøretiden på enkelte av prosessene.

VEDLEGG 1 – EKSEMPELØVING

Denne øvingen er basert på Øving 2 i faget Geografisk informasjonsbehandling 1 og går igjennom mye av de samme prinsippene. Den gir også en grei innføring i bruken av de forskjellige verktøyene.

EKSEMPELØVING - INTRODUKSJON TIL ET GIS

Denne øvinga er meint som introduksjon til hvordan et Geografisk Informasjonssystem kan fungere.

I øvinga vil du prøve noen enkle analyser i øvingssystemet KuGIS.

KuGIS er et vektorbasert system som kan handtere vektorer på SOSI, GeoJSON eller Shape-format.

Du har fått utlevert en zip-fil kalt Demodata.zip. Denne skal ikke pakkes ut. Demodata.zip kan også hentes fra <http://kugis.prosjeksjon.no/Demodata.zip>

Åpne nettleseren til <http://kugis.prosjeksjon.no/>

Etter at siden er lastet inn drar du filen Demodata.zip over til kartvinduet for å laste inn data.

Du har fått i oppdrag av et lokalt Trondheimsfirma og finne et egnet sted for å bygge et nytt lagerlokale. Du har fått følgende krav:

- Skal ligge mellom **30** og **300 meter** fra **bilvei**
- Skal ligge minst **100 meter** fra **vann**
- **Terrenghellingen** må ikke være større enn **8 grader**
- Området som er aktuelt kan klassifiseres etter dagens bruk:
 - 3 Best egnet: Åpent område
 - 2 Godt egnet: Dyrket mark.
 - 1 Middels egnet: Skog
 - 0 Ikke egnet: Tettbebygde strøk, bymessig bebyggelse og andre bebygde områder og andre områder.

PROSJEKTRAPPORT

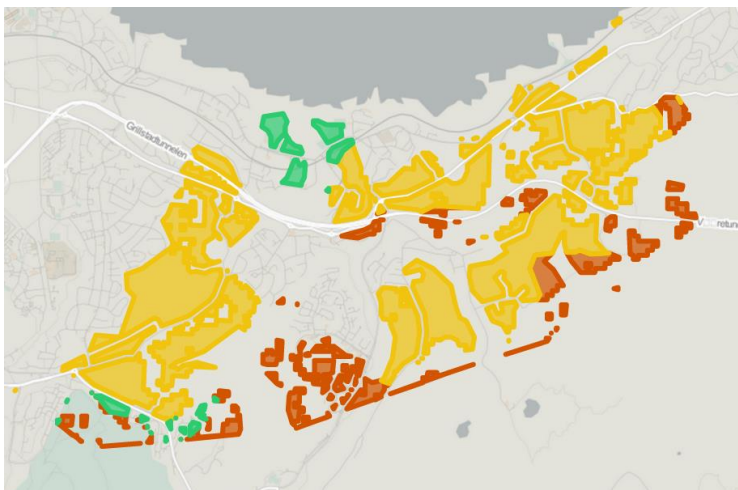
GJENNOMFØRING AV ØVINGEN

1. Last inn lagene ved å dra inn zip-filen demodata.zip over kartvinduet
2. Du vil nå få opp 4 lag. Veg, Arealbruk, Slope og vann
3. Start ved å hente ut det som er bilveg i kartlaget veg.
 - a. Dette gjøres ved å åpne FeatureExtractor og så dra laget veg over til det indikerte valget.
 - b. Trykk så «Add rule». Du vil da få opp tre felter.
 - c. I det første feltet velger du «typeVeg»
 - d. Det andre feltet skal det stå «not equal to»
 - e. Trykk på «U»-en til høyre for det tredje feltet. Velg så Gang- og sykkelvei i den listen som genereres.
 - f. Kall laget f.eks. «BilVeg» og trykk «extract».

Extract features from veg_simp2

	typeVeg	▼	not equal to	▼	Gang- og sykkelvei	▼	u
Name of new layer:			BilVeg		Extract	Add rule	Close

4. Du kan nå fjerne lukke vinduet og fjerne laget «veg» fra visningen.
5. Du skal nå lage to buffere rundt laget du lagde. En på 30 meter og en på 300 meter.
6. Etter at prosessen er ferdig må du trekke fra bufferen på 30 meter fra den på 300 meter. Til dette bruker du verktøyet «Difference». Rekkefølgen man drar lagene inn har noe å si her.
7. Gi laget du lagde med «Difference» et nytt navn ved å trykke på «Tannhjulet» ved siden av lagnavnet «Diference». Kall det for eksempel «Veg30til300m»
8. Lag nå en buffer på 100m rundt laget vann.
9. Bruk så verktøyet «Difference» til å fjerne denne bufferen fra laget «Veg30til300m».
10. Bruk «FeatureExtractor» til å hente ut alle polygoner med en SlopeDeg verdi over 8
11. Bruk så verktøyet «Dissolve» til å gjøre laget fra steg 10 om til et lag.
12. Trekk nå dette laget fra laget du lagde i steg 9.
13. Du kan nå fjerne alle lagene bortsett fra «arealområder» og det siste laget fra steg 12.
14. Bruk «FeatureExtractor» for å lage tre nye lag for områdene med objekttype «ÅpentOmråde», «Skog» og «DyrketMark».
15. Med disse tre lagene bruker du nå verktøyet «Intersection» sammen med laget fra steg 12.
16. Kall disse tre lagene Kategori 1, Kategori 2 og Kategori 3 og gi de nye farger for å skille dem fra hverandre.
17. Resultatet skal se ut ca. som dette:



REFERANSER

- i <http://www.w3.org/TR/2013/CR-IndexedDB-20130704/>
- ii <http://www.w3.org/TR/2012/WD-workers-20120313/>
- iii <http://www.w3.org/TR/webstorage/#disk-space>
- iv <http://www.kartverket.no/Kart/Gratis-kartdata/Visningstjenester/Cache-tjenester/>
- v <https://github.com/leaflet-extras/leaflet-providers>
- vi <http://mourner.github.io/simplify-js/>