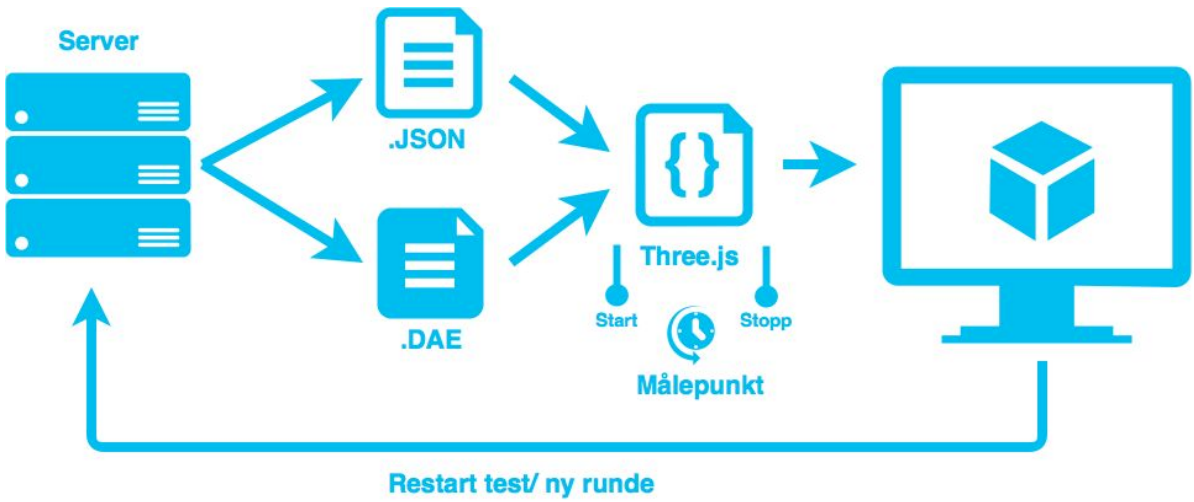
RAPPORT

Bacheloroppgaven 2016

Gruppe 14

Test av Collada- og Jsonloader

av: Eirik Thommessen



Sammendrag

Vi har i denne testrapporten sett på hastigheten Three.js laster inn JSON-objekter og COLLADA-objekter. Three.js konverterer forskjellige formater om til sitt eget THREE.Object3d format før det vises på skjermen. Testen måler hvor raskt rammeverket Three.js konverterer de to forskjellige formatene på klientsiden. Testen måler konverteringen i megabyte per sekund (mb/s).

Bakgrunnen for testen er å se om det er formålstjenlig å gjøre COLLADA-modeller om til JSON når modellene lastet opp til serveren. Det vil si at det generes en ekstra .json fil som ligger på serveren og som blir lastet ned til nettleseren. Vi har derfor i denne testen generert en .json fil basert på den originale COLLADA modellen, og testet dem hver for seg.

Resultatet av testen er meget overbevisende. COLLADA hadde igjennomsnitt over 600 konverteringer en gjennomsnittshastighet på 4,66 mb/s. JSON hadde på samme antall konverteringer en gjennomsnittshastighet på 12,64 mb/s, hele 2.7 ganger raskere enn COLLADA.

Det er en stor gevinst med å gjøre modeller om til JSON. Denne gevinsten kan enten brukes til å benytte seg av mer komplekse modeller og materialer, eller som ren besparelse i lastetiden og responstiden til nettsiden.

Innhold

Bakgrunn

Vi er en gruppe studenter ved Høgskolen i Østfold som gjennomfører bacheloroppgave. Vi skal i denne oppgaven lage et nettsted som skal vise en 3D-modell av en atomreaktor, og hvordan denne fungerer.

Som en del av av oppgaven har vi allerede kommet frem til bruk av rammeverket Three.js. I denne rapporten ønsker vi å se nærmere på filformat ved innlasting av 3D-modeller til nettsiden igjennom rammeverket.

Gjengivelse av 3D-modeller i nettleseren igjennom Three.js

Three.js, som valgt rammeverk, har integrerte loadere for forskjellige filformater. Blant annet for de to formatene COLLADA og JSON som vi skal se på i denne testen.

Når Three.js gjengir 3D-grafikk i nettleseren, gjør de dette igjennom sitt eget format THREE.Object3D. Det vil si at alle formater som kan vises av Three.js blir konverter over til dette formatet, slik at om vi sender inn COLLADA er det ikke COLLADA vi får ut.

COLLADA

Vi bruker i denne oppgaven COLLADA (COLLAborative Design Activity) (.dae) som 3D-format, bakgrunnen for dette er at COLLADA er et ISO-format[1]. Dette fører til at 3D-filene som brukes i prosjektet ikke vil være bundet opp av proprietære formater, som enten er plattformavhengige (mac/windows) eller lisensbelagte. Som ISO-format blir COLLADA støttet av de fleste 3D-modellering programmer. Bruk av COLLADA er også et uttrykt ønske fra oppdragsgiver IFE (institutt for energiteknikk).

COLLADA er et format som er bygget på XML (Extensible Markup Language). Som språk definerer COLLADA elementer, attributter og verdier i form av tagger, som ligner på HTML. Disse starter og slutter med en åpnings- og slutt-tagg.

Dette fører til som vi kan se i *figur 1.* at der <effect> starter er den avhengig av en slutt tagg </effect>. På samme måte vil det være for “diffuse”, “specular”, “color” osv.

[1] International Organization for Standardization: http://www.iso.org/iso/home.html

```
<effect id="ID2">
    <profile_COMMON>
        <technique sid="COMMON">
            <blinn>
                <diffuse>
                    <color>0.8 0.8 0.8 1</color>
                </diffuse>
                <specular>
                    <color>0.2 0.2 0.2 1</color>
                </specular>
                <shininess>
                    <float>0.5</float>
                </shininess>
            </blinn>
        </technique>
    </profile_COMMON>
</effect>
```

**fig 1 . Kodeeksemple på collada*

JSON

Three.js er et javascript rammeverk og har en innebygget converter til JSON[2]. JSON er en forkortelse for Javascript Object Notation. Selv om Three.js er et JavaScript rammeverk, er ikke JSON-versjoner av modeller/objekter det samme som Three.js modeller/objekter. Det vil si at JSON også må konverteres (parses) på lik linje med COLLADA, for så å lastes inn.

JSON er bygget på en annen syntaks enn COLLADA. Der elementer blir annonsert med en nøkkel og en verdi. Og ikke med tagger. Besparelsen ligger i at JSON ikke bruker åpnings tagger og slutt tagger. på den måten sparer JSON i teorien tegn og bytes.

```
"children": [{
    "name": "MilenniumFalcon",
    "uuid": "3FF71E75-3975-3669-9CB8-2A61FB0F9F67",
    "matrix": [1,0,0,0,0,1,0,0,0,0,1,0,0,0,0,1],
    "visible": true,
    "type": "Object",
    "children": [{
        "name": "B_DISK",
        "uuid": "A7084D1D-0EFE-3FFD-ADCF-008CB70FD46C",
        "matrix": [1,0,0,0,0,1,0,0,0,0,1,0,38.7243,-1288.7,0,1],
        "visible": true,
        "type": "Mesh",
        "material": "FBCD6299-1061-3CE5-935C-C114AB343ABF",
        "castShadow": true,
        "receiveShadow": true,
        "geometry": "06CDE1AA-A330-3DFC-A6DC-7350676C11EA"
    },{
```

**fig 2. Kodeeksempel på jsonmodell*

[2] Javascript object notation: http://www.json.org/

Bakgrunnen for at vi i denne testen velger JSON er fordi dette også er et åpent format. Three.js tilbyr en egen converter til JSON, som gjør dette en del av rammeverket og ikke et ekstra tillegg.

Målet for denne testen er derfor å se på hvilken metode som konverteres raskest av Three.js i nettleseren, når brukeren henter de to forskjellige filene fra en server. Og dersom det er en besparelse, se på om denne besparelsen vil ha innvirkning på hvordan vi bruker COLLADA-formatet videre i oppgaven.

Metode

Vi skal i denne testen kjøre to forskjellige modeller igjennom Three.js egen innebygde loader. For å få best mulig resultater har vi valget en COLLADA-modell som i utgangspunktet er på 12.68 megabyte. Med utgangspunkt i denne modellen gjør vi den om til en JSON-modell ved bruk av Three.js egne innebygde .toJSON() metode.

Det er enkle grunner til at vi har valgt denne konverteringsmetoden. For det første finnes det ikke noen ISO-standard for konvertering av COLLADA til JSON. Det finnes flere tilbydere på nett, men forskjellige løsninger og metoder gir en inkonsistent output. Det er derfor ingen garanti for at filen levert av en tredjepart vil fungere optimalt. Når three.js har en egen konverter, bruker vi denne for mest mulig forutsigbarhet.

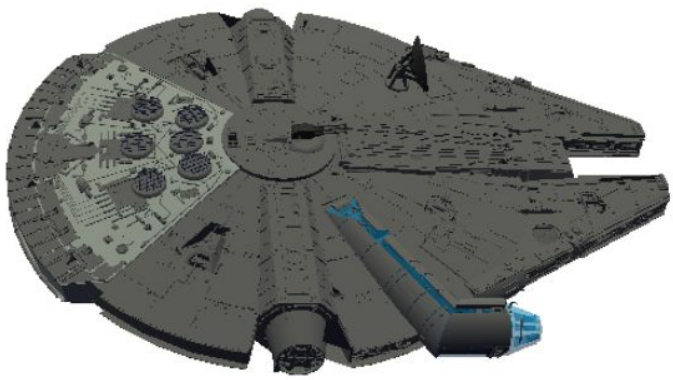


**fig 3. - colladamodel av milleniumfalcon*

Pipeline ved omgjøring til JSON

Modellen blir lastet inn og konverter til Three.js objekt som COLLADA (.dae) fil.

```
loader.load('./modells/mil.dae',
    function(collada){
    model = collada.scene;
    JsonModel = model.toJSON();
    console.log(JSON.stringify(JsonModel));
```

fig 4 . - Omgjøring til Json

Three.js innebygde COLLADA-loader tar tre parameter i sin .load() metode[3]: filen som skal lastes, hva som skal skje når filen er lastet og hva som skal skje når filen lastes inn. Som du kan se av figur 4, har vi valgt å kun benytte oss av de to første parameterne. Her oppgir vi filen som skal lastes inn "mil.dae" som ligger i mappen "modells". Når filen er ferdig lastet inn kjøres en anonym funksjon som tar et parameter. Dette er COLLADA filen som nå er lastet inn i minnet til nettleseren. collada.scene legges så til variabelen "model". Postfikset .scene gjør at alt som tilhører 3D-scenen fra COLLADA-filen, modeller og materialer, blir puttet inn i variabelen "model", som nå er et Three.js objekt. Dette objektet arver fra Object3D og som et Three.js objekt aksepterer den nå metoden .toJSON().

Metoden .toJSON() konverter Three.js-objekter om til JSON-objekter. I figur 4, kan du se at vi nå kan kjøre JavaScript innebygde JSON.stringify() metode på variabelen "JsonModel". I dette eksempelet skrives hele JSON-strengen til konsollen. Vi kan nå bruke resultatet av JSON.stringify() til vår egen JSON-fil.

Oppsett av testen

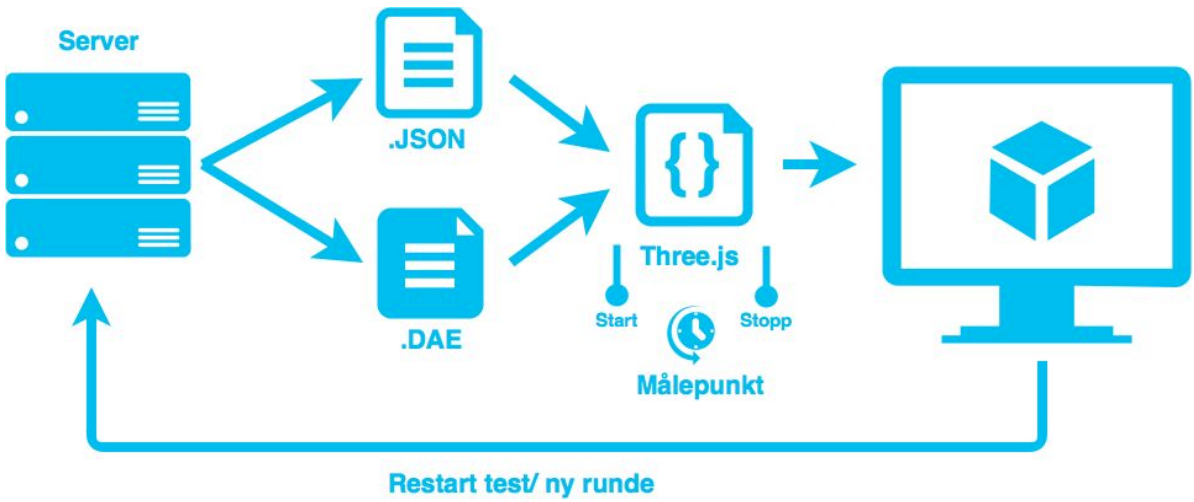


fig 5. flytskjema for testen

[3] Dokumentasjon på colladaLoader : http://threejs.org/docs/index.html#Reference/Loaders/ColladaLoader

Når testen settes i gang henter den enten .json eller .dae fra serveren. Filen leveres til Three.js som starter tidtagningen. Når Three.js har har fullført konverteringen til THREE.Object3D. stopper tidtagningen og runden/testen avsluttes. Testen kan kjøres i flere runder for å få et bedre kvantitativt resultat.

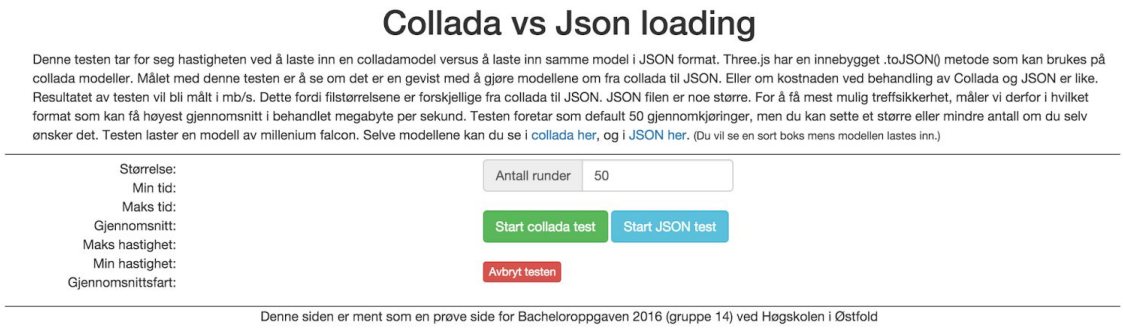


fig 6. Skjerm bilde av testen før st

art

Personen som gjennomfører testen velger mellom å kjøre en COLLADA eller JSON test *(fig 6)*. Testene som korrespondere med startknappene i GUI er som illustrert under i figur 7. men dette er hovedkjernen. i testen.

```
45  function startCollada(){
46          stopWatch.start();
47          var loader = new THREE.ColladaLoader();
48          loader.options.convertUpAxis = true;
49          loader.load(colladaModel,
50              function(collada){
51              nextRound();
52              },
53              function(xhr){
54                  totalBytes = xhr.total/1000000;
55              });
56  }
57
58  function startJson(){
59      stopWatch.start();
60      var loader = new THREE.ObjectLoader();
61      loader.load("./modells/mil2.json", function(object){
62          model = object;
63          nextRound();
64          },
65          function(xhr){
66          totalBytes = xhr.total/1000000;
67          });
68  }
69
```

fig 7. Oppsett og sammenligning av de to testene

Testene er funksjoner som tar i bruk to forskjellige loadere. For COLLADA-filen benyttes THREE.ColladaLoader, og for JSON benyttes, THREE.ObjectLoader. Det finnes også en THREE.JSONLoader, men den vil ikke fungere i vårt eksempel. Forskjellen på THREE.JSONLoader og THREE.ObjectLoader er at Object loaderen tar høyde for at det kan være andre objekter i

JSON-strengen enn kun en modell. Dette skiller object loaderen igjennom ".type" attributtet[4]. Siden vi i denne modellen også har materialer vil vi i JSON loaderen få en feilmelding (fig. 8).

fig 8. feilmelding på JSONLoader

I testen benytter vi oss av tre parametere i loaderen; url, onLoad og onProgress. Til forskjell fra prosessen vi brukte når vi konverterte COLLADA til JSON (fig. 4) benytter vi oss av "onProgress". Dette gir oss tilgang til metoden ".total", .onProgress er en anonym funksjon som tar inn et parameter, objektet som skal lastes inn. Metoden ".total" gir oss størrelsen på filen som lastes inn. sammen med en stoppeklokkefunksjon vil vi kunne måle hastigheten(fart) $f = \frac{størrelse}{tid}$.

```
function nextRound(){
        render();
        stopWatch.stop();
        updateView();
        runNewTest();
        rounds++;
}
```

fig 9. stoppertiden og starter neste rund hvis rounds er mindre enn antall runder

Testen går i loop til antall runder den som tester har satt i input feltet (default 5) er kjørt (fig 10).

```
function runNewTest(){
    if (!startedTest){
        ...
        if(isCollada){startCollada();}else{startJson();}
        startedTest = true;
    }
    else if(startedTest && rounds <antallRunder){
        ...
        if(isCollada){startCollada();}else{startJson();}
    }
    else{
        ...
        startedTest =false;
        showResult();
    }
}
```

fig 10. Stopper når antall runder satt av testeren er nådd

[4] Dokumentasjon på ObjectLoader : http://threejs.org/docs/index.html#Reference/Loaders/ObjectLoader

Premisser for testing

Testingen av de forskjellige loaderne må foregå i samme tidsperiode, på samme maskin og på samme nettverk. Dette for å gi mest mulig like testforhold. Testene i denne rapporten kjøres med 50 runder for å få et kvantitativt resultat.

Måling av resultat

Resultatet av testen vil bli målt i antall bytes som er lastet inn delt på tiden.

$$f = \frac{størrelse}{tid}$$

Det som måles er hvor raskt nettleseren klarer å konvertere formatet som blir sendt inn over til Three,js Object3D format.

Stoppeklokken starter på nytt rett før prosessen settes i gang og stoppes hver gang prosessen er ferdig. Tiden blir lagt til variabelen “totalTime”. Variabelen totalTime vil bli delt på antall runder for å finne gjennomsnitts tiden hver runde tok. Sammen med størrelsen på filen som blir sendt inn vil dette gi oss farten i megabyte per sekund (mb/s). Høyre tall tilsvarer høyere fart på behandlingen av input-filen.

Sluttresultatet er gjennomsnittet av farten det tok å gjennomføre testen.

Testen

Testen ble gjennomført på tre forskjellige på fire forskjellige tidspunkt steder:

Refsnes - Moss - Eirik Thommessen. 2 tester av hver
Remmen - Halden - Joakim Milian Elden. 3 tester av hver
Verket - Moss - Eirik Thommessen. 3 tester av hver
Refsnes - Moss - Eirik Thommessen. 4 tester av hver

Totalt 12 tester av hver loader.

Hver test ble gjennomført med 50 runder, på samme maskin og på samme nettverk. Etter hver gjennomføring av testen ble det tatt skjermbilder av resultatet. Resultatet som kommer opp etter hver gjennomføring er datostemplet. På denne måten er det lett å sammenligne data.

Testen ligger åpent på frigg.hiof.no, og ble kjørt fra serveren under alle testene.

http://frigg.hiof.no/bo16-g14/pages/speedtester/index.html

Resultater

Det ble gjennomført totalt 24 tester av loaderne fordelt 12 på hver. COLLADA hadde en gjennomsnittshastighet på 4,66 mb/s, JSON hadde en gjennomsnittsfart over de tolv testene på 12,64 mb/s. Totalt er det i denne testen gjennomført 600 innlastinger av hver model. JSON er av testresultatet 2,7 ganger raskere enn COLLADA i gjennomsnitt.

Avvik i resultatene

Vi har et avvik i resultatene som følge av en duplikat. Avviket kan ha skjedd vet at den som testet ikke dobbeltsjekket at testen var kjørt på nytt før man tok skjermbildet. Siden resultatene ellers er veldig stabile i forhold til resultat velger vi å legge ved duplikatet, da dette ikke er en topp eller bunn-notering.

Resultatene i tabellen under viser høyeste og laveste hastighet.

Collada

Dato	Tid	Teststed	Tester	Resultat	Merknad
29.01.2016	00:05:40	Refsnes - Moss	Eirik Thommessen	4,49	
29.01.2016	00:10:23	Refsnes - Moss	Eirik Thommessen	4,44	
29.01.2016	00:28:40	Remmen - Halden	Joakim Milian Elden	4,76	
29.01.2016	00:34:25	Remmen - Halden	Joakim Milian Elden	4,91	
29.01.2016	00:40:00	Remmen - Halden	Joakim Milian Elden	4,7	
29.01.2016	12:09:15	Verket - Moss	Eirik Thommessen	5,03	
29.01.2016	12:13:55	Verket - Moss	Eirik Thommessen	4,61	
29.01.2016	12:42:31	Verket - Moss	Eirik Thommessen	4,7	
29.01.2016	22:45:49	Refsnes - Moss	Eirik Thommessen	4,54	
29.01.2016	22:48:29	Refsnes - Moss	Eirik Thommessen	4,8	
29.01.2016	22:56:31	Refsnes - Moss	Eirik Thommessen	4,52	
29.01.2016	22:59:10	Refsnes - Moss	Eirik Thommessen	4,44	

JSON

Dato	Tid	Teststed	Tester	Resulat	Merknad
29.01.2016	00:07:07	Refsnes - Moss	Eirik Thommessen	13,26	
29.01.2016	00:11:34	Refsnes - Moss	Eirik Thommessen	12,48	
29.01.2016	00:31:03	Remmen - Halden	Joakim Milian Elden	12,23	
29.01.2016	00:36:54	Remmen - Halden	Joakim Milian Elden	13,69	
29.01.2016	00:42:00	Remmen - Halden	Joakim Milian Elden	12,13	
29.01.2016	12:06:44	Verket - Moss	Eirik Thommessen	12,72	
29.01.2016	12:20:50	Verket - Moss	Eirik Thommessen	13,09	
29.01.2016	12:20:50	Verket - Moss	Eirik Thommessen	13,09	Duplikat
29.01.2016	22:49:47	Refsnes - Moss	Eirik Thommessen	12,34	
29.01.2016	22:51:05	Refsnes - Moss	Eirik Thommessen	11,93	
29.01.2016	22:52:18	Refsnes - Moss	Eirik Thommessen	12,31	
29.01.2016	22:53:34	Refsnes - Moss	Eirik Thommessen	12,37	

Konklusjon

Igjennom 1.200 innlastninger av modellene i COLLADA- og JSON-format er JSON-formatet betydelig raskere enn COLLADA, når formatet skal konverteres over til Three.js Object3D. I snitt er JSON 2,7 ganger raskere. Det vil si at en side som tar 1 sekund å laste inn med JSON vil ta 2,7 sekunder med COLLADA-formatet .

Dette er ikke en test hvor vi har sett på hvorvidt gruppen skal gå bort fra COLLADA som format. Det vi i denne testen har sett på er hvorvidt det er formålstjenlig å gjøre om COLLADA til JSON-format ved opplasting til nettsiden, og om det er en gevinst i å behandle JSON vs. COLLADA på klientsiden når siden lastes fra serveren.

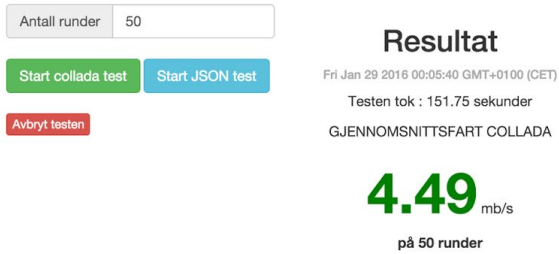
Konklusjonen på dette i denne testen er at det er en stor gevinst for brukeren å bruke JSON kontra COLLADA når siden lastes inn.

Konsekvensene av denne testen er at gruppen kan tåle mer komplekse modeller og materialer dersom formatet konverteres til JSON, på samme lastetid som en mindre kompleks COLLADA-modell. Alternativet er samme kompleksitet, men raskere lastetider.

Vedlegg: Testresultater

collada_refsnes_moss_#1

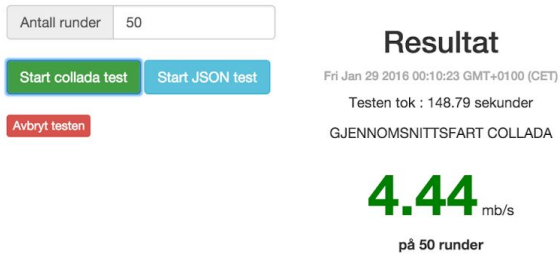
Størrelse:	12.68 mb
Min tid:	0.871 sekunder
Maks tid:	3.938 sekunder
Gjennomsnitt:	3.035 sekunder
Maks hastighet:	3.221 mb/s
Min hastighet:	15.076 mb/s
Gjennomsnittsfart:	4.486 mb/s



---

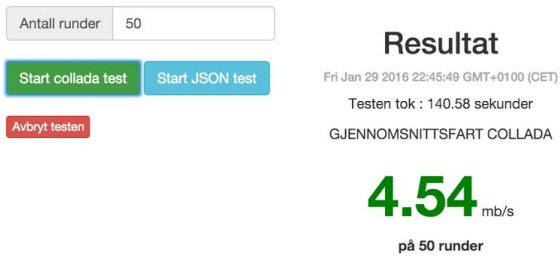
collada_refsnes_moss_#2

Størrelse:	12.68 mb
Min tid:	1.315 sekunder
Maks tid:	3.678 sekunder
Gjennomsnitt:	2.976 sekunder
Maks hastighet:	3.448 mb/s
Min hastighet:	9.986 mb/s
Gjennomsnittsfart:	4.435 mb/s



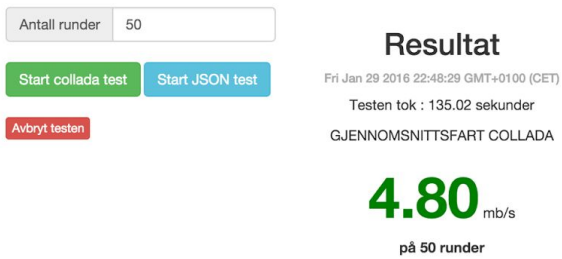
collada_refsnes_moss_#3

Størrelse:	12.68 mb
Min tid:	2.616 sekunder
Maks tid:	4.022 sekunder
Gjennomsnitt:	2.812 sekunder
Maks hastighet:	3.154 mb/s
Min hastighet:	4.848 mb/s
Gjennomsnittsfart:	4.540 mb/s



collada_refsnes_moss_#4

Størrelse:	12.68 mb
Min tid:	1.639 sekunder
Maks tid:	3.631 sekunder
Gjennomsnitt:	2.700 sekunder
Maks hastighet:	3.493 mb/s
Min hastighet:	7.739 mb/s
Gjennomsnittsfart:	4.799 mb/s



collada_refsnes_moss_#5

Størrelse:	12.68 mb
Min tid:	1.349 sekunder
Maks tid:	3.861 sekunder
Gjennomsnitt:	2.933 sekunder
Maks hastighet:	3.285 mb/s
Min hastighet:	9.734 mb/s
Gjennomsnittsfart:	4.523 mb/s

collada_refsnes_moss_#6

Størrelse:	12.68 mb
Min tid:	1.946 sekunder
Maks tid:	5.057 sekunder
Gjennomsnitt:	2.918 sekunder
Maks hastighet:	2.508 mb/s
Min hastighet:	6.518 mb/s
Gjennomsnittsfart:	4.435 mb/s

collada_studentbolig_halden_#1

Størrelse:	12.68 mb
Min tid:	1.783 sekunder
Maks tid:	7.543 sekunder
Gjennomsnitt:	2.777 sekunder
Maks hastighet:	1.682 mb/s
Min hastighet:	7.114 mb/s
Gjennomsnittsfart:	4.761 mb/s

collada_studentbolig_halden_#2

Størrelse:	12.68 mb
Min tid:	1.339 sekunder
Maks tid:	3.396 sekunder
Gjennomsnitt:	2.660 sekunder
Maks hastighet:	3.735 mb/s
Min hastighet:	9.807 mb/s
Gjennomsnittsfart:	4.912 mb/s

collada_studentbolig_halden_#3

Størrelse:	12.68 mb
Min tid:	1.277 sekunder
Maks tid:	4.455 sekunder
Gjennomsnitt:	2.849 sekunder
Maks hastighet:	2.847 mb/s
Min hastighet:	10.283 mb/s
Gjennomsnittsfart:	4.704 mb/s

collada_verket_moss_#1

Størrelse:	12.68 mb
Min tid:	0.981 sekunder
Maks tid:	3.401 sekunder
Gjennomsnitt:	2.635 sekunder
Maks hastighet:	3.729 mb/s
Min hastighet:	13.386 mb/s
Gjennomsnittsfart:	5.032 mb/s

---

collada_verket_moss_#2

Størrelse:	12.68 mb
Min tid:	2.603 sekunder
Maks tid:	3.252 sekunder
Gjennomsnitt:	2.762 sekunder
Maks hastighet:	3.900 mb/s
Min hastighet:	4.873 mb/s
Gjennomsnittsfart:	4.607 mb/s

---

collada_verket_moss_#3

Størrelse:	12.68 mb
Min tid:	1.13 sekunder
Maks tid:	3.817 sekunder
Gjennomsnitt:	2.836 sekunder
Maks hastighet:	3.323 mb/s
Min hastighet:	11.621 mb/s
Gjennomsnittsfart:	4.704 mb/s

json_refsnes_moss_#1

Størrelse:	13.13 mb
Min tid:	0.719 sekunder
Maks tid:	3.006 sekunder
Gjennomsnitt:	1.057 sekunder
Maks hastighet:	4.219 mb/s
Min hastighet:	18.263 mb/s
Gjennomsnittsfart:	13.257 mb/s

json_refsnes_moss_#2

Størrelse:	13.13 mb
Min tid:	0.751 sekunder
Maks tid:	2.865 sekunder
Gjennomsnitt:	1.110 sekunder
Maks hastighet:	4.427 mb/s
Min hastighet:	17.485 mb/s
Gjennomsnittsfart:	12.479 mb/s

---

json_refsnes_moss_#3

Størrelse:	13.13 mb
Min tid:	0.822 sekunder
Maks tid:	1.785 sekunder
Gjennomsnitt:	1.093 sekunder
Maks hastighet:	7.106 mb/s
Min hastighet:	15.975 mb/s
Gjennomsnittsfart:	12.335 mb/s

json_refsnes_moss_#4

Størrelse:	13.13 mb
Min tid:	0.923 sekunder
Maks tid:	1.601 sekunder
Gjennomsnitt:	1.134 sekunder
Maks hastighet:	8.202 mb/s
Min hastighet:	14.227 mb/s
Gjennomsnittsfart:	11.925 mb/s

json_refsnes_moss_#5

Størrelse:	13.13 mb
Min tid:	0.921 sekunder
Maks tid:	1.412 sekunder
Gjennomsnitt:	1.095 sekunder
Maks hastighet:	9.300 mb/s
Min hastighet:	14.258 mb/s
Gjennomsnittsfart:	12.307 mb/s

json_refsnes_moss_#6

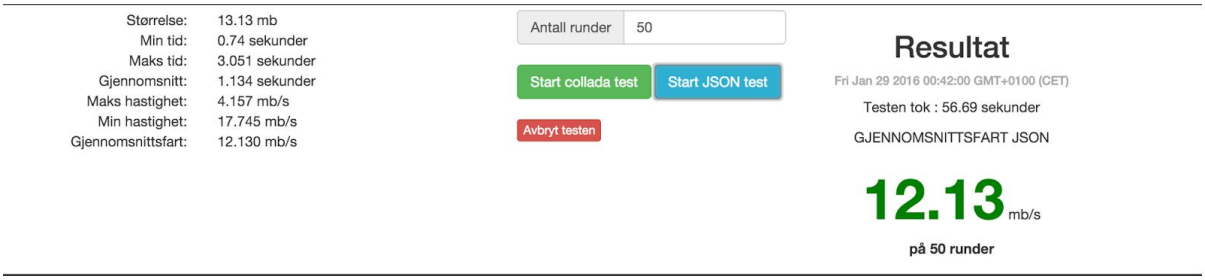
Størrelse:	13.13 mb
Min tid:	0.905 sekunder
Maks tid:	1.371 sekunder
Gjennomsnitt:	1.088 sekunder
Maks hastighet:	9.578 mb/s
Min hastighet:	14.510 mb/s
Gjennomsnittsfart:	12.366 mb/s

json_studentbolig_halden_#1

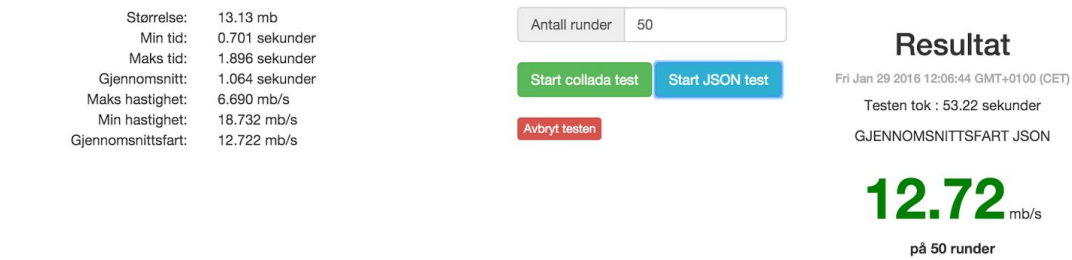
Størrelse:	13.13 mb
Min tid:	0.772 sekunder
Maks tid:	2.347 sekunder
Gjennomsnitt:	1.116 sekunder
Maks hastighet:	5.404 mb/s
Min hastighet:	17.009 mb/s
Gjennomsnittsfart:	12.231 mb/s

json_studentbolig_halden_#2

json_studentbolig_halden_#3



json_verket_moss_#1



json_verket_moss_#2

json_verket_moss_#3