Examensarbete

HÖGSKOLAN I SKÖVDE

Jämförelse av WebGL-teknologier vid rendering av heatmaps utifrån marin miljödata

Jämförelse mellan Three.js och X3DOM

Comparison of WebGL technologies for rendering heatmaps based on marine environmental data

Comparison between Three.js and X3DOM

Examensarbete inom huvudområdet Informationsteknologi
Grundnivå 30 högskolepoäng
Vårtermin 2018

Filip Barkestedt

Handledare: András Márki
Examinator: Gunnar Mathiason

Sammanfattning

En stor kvantitet av miljödata samlas hela tiden in och för att dra nytta av all data behöver den förstås av de användarna som kan applicera kunskapen inom deras område. Visualiseringar tillåter användare att förstå datan och det är därför en viktig del av hur en användare kan ta del av den datan som samlas in. I detta arbete evalueras de WebGL-baserade teknologierna Three.js och X3DOM om vilken som är mest lämplig för att visualisera geospatial data på webben utifrån hur effektiva de är vid rendering av *heatmaps*. Två applikationer utvecklas, en baserat på Three.js och en på X3DOM. Mätningar utförs på varje applikation för att jämföra renderingstiden mellan teknologierna och en kvalitativ studie används för att evaluera deras användbarhet. Resultatet blev att Three.js är mer lämpligt för att visualisera geospatial data på webben.

Nyckelord: visualisation, heatmap, geospatial data, WebGL, Three.js, X3DOM.

Innehållsförteckning

1 Introduktion

Stora mängder av miljödata samlas hela tiden in och det finns ett stort intresse av att ta del av datan i både forskningssyfte samt inom flera olika industrier (James, Moh & Edwards, 2016). Eftersom datan i sin råa form är väldigt svår att förstå behöver den presenteras i ett tydligare format. Genom att visualisera datan går det att göra den betydligt lättare att förstå (Haara, Pykäläinen, Tolvanen & Kurttila, 2018). En lämplig typ av visualisering för just geospatial data såsom miljödata, är att till exempel använda *heatmaps* eftersom den tillåter betraktaren att snabbt känna igen relationer mellan olika områden på kartan (Eichinski & Roe, 2014).

För att rendera *heatmaps* i ett webbaserat format kan en WebGL-baserad teknologi användas. Då det finns ett intresse av att visualiseringen når användaren snabbt för att öka användbarheten, finns det även en vinst med att välja rätt teknologi (James, Moh & Edwards, 2016). Problemet är att det inte finns några speciella riktlinjer av vilket teknologi som kan rendera geospatial data på bästa sätt (Krämer & Gutbell, 2015).

Målet med arbetet är att jämföra olika WebGL-baserade teknologier för att ta reda på vilken teknologi som är mest lämplig för visualisering av geospatial data i form av *heatmaps* i hänsyn till deras syfte att rendera grafik i en webbaserad miljö.

För att ta reda på kapaciteten av hur WebGL-teknologier kan rendera geospatial data genomförs ett experiment där en applikation skapas för respektive teknologi som ska testas och en kvalitativ undersökning görs för att utvärdera deras användbarhet. Prestandamätningar genomförs på varje applikation för att därav ta reda på om det finns en signifikant skillnad av renderingstiden mellan teknologierna.

2 Bakgrund

2.1 Visualisering av data

Data som samlas in från diverse platser är sällan lätt att förstå sig på enbart genom att titta på det i dess råa form. Det är en utmaning att presentera datan på ett sätt som är lätt för en människa att uppfatta och informationen behöver presenteras så den blir enklare att förstå för en person som vill ta del av datan. En visualisering av datan kan göra den mer begriplig och även hjälpa en användare att fatta beslut (Haara, Pykäläinen, Tolvanen & Kurttila, 2018).

Det finns flera olika tillämpningar för visualisering av data och de kan göras på flera olika sätt, bland annat genom *heatmaps*, *contour maps* eller *vector maps*. Två exempel av vad för typer av data som går att gestalta är sjukvårdsdata och marin miljödata (Sopan, Noh, Karol, Rosenfeld, Lee & Shneiderman, 2012; James, Moh & Edwards, 2016). Det är även möjligt att använda visualisering till att förstå flera lösningar genererat av populationsbaserad multiobjektiva algoritmer (Pryke, Mostaghim & Nazemi, 2007). En visualisering kan göra det möjligt att ge en tydlig bild av vad datan betyder, även om den innefattar en stor informationssamling i ett litet format.

För att ge användaren en bra tillgång till informationen som visualiseras samt ett relativt enkelt sätt att använda den, kan visualiseringen ske på webben. Ett webbaserat format av ett visualiseringsverktyg fungerar oftast bättre och är enklare att använda än ett program installerat direkt i datorn då de sistnämnda ofta har en brant inlärningskurva. Om målgruppen som applikationen ska riktas till är bred och inte kan antas ha en god teknisk förmåga är ett webbaserat format bra (Sopan et al., 2012).

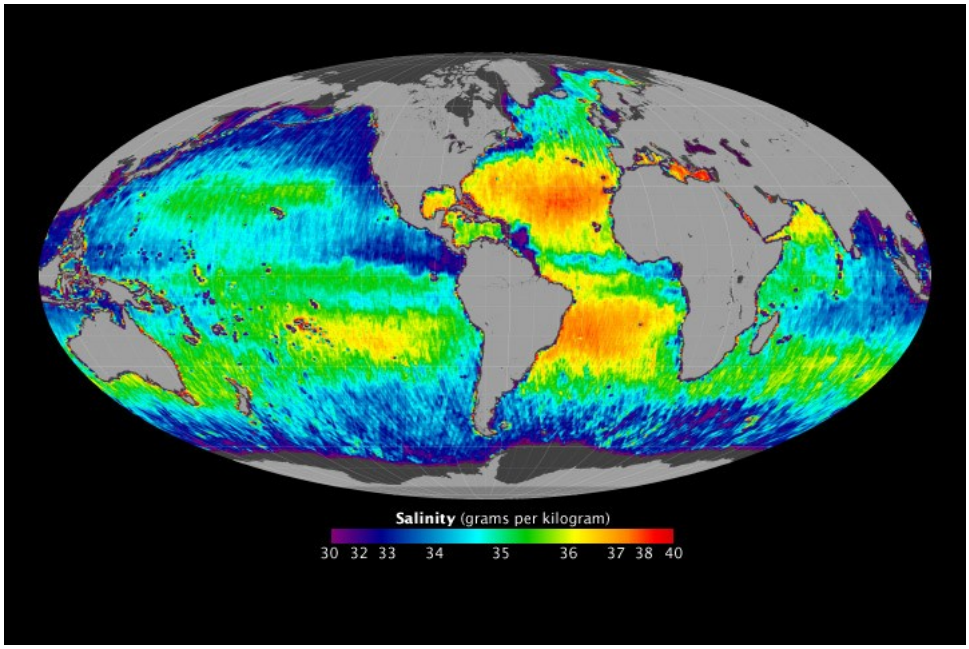
Ett sätt att visualisera data och göra den mer förståelig är med hjälp av *heatmaps* och det finns ett flertal användningsområden för detta visualiseringsalternativ. Ett av områdena är att presentera miljödata genom att generera *heatmaps*. Genom att visualisera miljödatan som samlas in kan användare från flera olika håll få användning av presentationen, både inom forskning samt inom diverse industrier (James, Moh & Edwards, 2016). Ett annat område är att använda *heatmaps* för att förstå och presentera en användares aktivitet på webbplatser. Det kan i sin tur bidra med att kvantifiera den uppmärksamhet en användare ger till olika områden på en webbsida (Lamberti & Paravati, 2015).

2.2 Heatmaps

En *heatmap* är en grafisk representation av data. Den data som ska visas representeras i form av färger och ger en visuell sammanfattning av informationen. Den tillåter betraktaren att snabbt känna igen relationer mellan olika områden på kartan (Eichinski & Roe, 2014). Figur 1 visar hur mängden salt i havsvatten kan visualiseras med hjälp av *heatmaps*.

En *heatmap* kan användas inom flera olika områden men ofta används det för att visualisera platsbaserad data såsom miljödata eller hälsodata (James, Moh & Edwards, 2016; Sopan et al., 2012). De *heatmaps* som nyttjas på detta sättet använder geospatial data vilket innebär att varje värde i datasetet har en geografisk position kopplad till sig. Positionen består av latitud- och longitudkoordinater men kan också innehålla djup vilket gör datan till multidimensionell. Mängden av miljödata växer dock konstant då bland annat bojar, fartyg

och satelliter hela tiden samlar in diverse variabler. Exempel av de variabler som finns i dataseten är salthalt, strömmar och havsnivåer (James, Moh & Edwards, 2016).



Figur 1. Visualiserar salthalten i havet (salinitet). Publicerad av NASA's Earth Observatory under CC BY 2.0 https://www.flickr.com/photos/68824346@N02/7180520093

2.3 WebGL

Web Graphics Library (WebGL) är ett javascript Application Programming Interface (API), är baserat på OpenGL och används för att rendera interaktiv 2D och 3D grafik på webben (Ha, Jin & Lee, 2015). Det fungerar till alla kompatibla webbläsare utan behov av någon form av plugin. WebGL använder HTML5 canvas-elementet för att rendera grafiken. Det finns ett flertal *open source* teknologier som är baserade på WebGL och som erbjuder ett *high level* API. De teknologier som finns skiljer sig dock en del i dess funktionalitet, både i hur de definierar 3D innehåll och deras syfte för olika tillämpningar. (Krämer & Gutbell, 2015; Resch, Wohlfahrt & Wosniok, 2014).

Den största fördelen med att använda WebGL är att det kan köras direkt i webbläsaren utan några plugins och kan dessutom nyttja GPUn istället för CPUn. En tydlig förbättring av prestanda kan därmed erhållas jämfört med andra teknologier som kräver stöd av någon form av plugins. WebGL är också högst relevant då det finns god dokumentation av teknologin och utvecklas för att kunna hantera framtida utmaningar. Det kräver inte heller några licenser för att använda vilket ökar dess tillgänglighet (Resch, Wohlfahrt & Wosniok, 2014).

2.4 Responstider

Responstid syftar på den tid användaren behöver vänta för att interagera med en sida. Responstiden spelar stor roll av hur en användare upplever en webbplats både i hur användbar den är och hur lätt den är att använda (Lin & Lu, 2000). Om det uppstår dröjsmål kan användaren lämna webbplatsen för att söka upp ett annat alternativ. För att användaren

ska bli tillfredsställd och fortsätta använda webbplatsen bör responstiden vara så låg som möjligt (Rajamony & Elmootazbellah, 2001).

3 Problemformulering

Det samlas in väldig stora mängder av miljödata och det är även av intresse att presentera den datan. Datan som samlas in behöver dock visualiseras på olika sätt för att göra den mer förståelig. Genom att visualisera datan som samlas in kan användare från flera olika håll få användning av presentationen såsom inom forskning, fiskeindustrin, olje och gasindustrin, transportindustrin och fritidsbåtägare. Den insamlade datan blir högst relevant då användarna hela tiden har behov för nya uppdaterade visualiseringar eftersom deras förutsättningar är i konstant förändring (James, Moh & Edwards, 2016).

Att visualisera den insamlade datan med hjälp av *heatmaps* gör att informationen aggregeras på ett lämpligt sätt och gör den betydligt mer begriplig för en användare (Eichinski & Roe, 2014). För att användaren ska kunna utnyttja och få god tillgång till den visualiserade datan är webben en lämplig plattform att använda. Att visualiseringen också når användaren snabbt har även det en betydande roll i hur användbar den är (James, Moh & Edwards, 2016).

För att rendera grafik på webben kan en WebGL-baserad teknologi användas. WebGL tillåter en snabb rendering av grafik på klientsidan och lämpar sig för att rendera stora mängder av data (Resch, Wohlfahrt & Wosniok, 2014). Eftersom ett miljödataset ofta innehåller stora datamängder kan just en WebGL-baserad teknologi vara mer lämpligt att använda sig av. Det finns dock många olika teknologier baserade på WebGL och det är inte uppenbart vilken som kan rendera geospatial data på kortast tid (Krämer & Gutbell, 2015). **Problemet** är att det är oklart vilken WebGL-teknologi som är mest lämpligt för rendering av *heatmaps* på webben. *Lämpligt* innebär i det här fallet hur effektivt en teknologi kan rendera en scen och hur användbar teknologin är.

Målet med arbetet är att jämföra olika WebGL-baserade teknologier för att ta reda på vilket teknologi som är mest lämplig för visualisering av geospatial data i form av *heatmaps* i hänsyn till deras syfte att rendera grafik i en webbaserad miljö. Detta sker utifrån en webbutvecklares synpunkt och i kontexten att klientsidan renderar *heatmaps* utifrån geospatial data.

Frågeställningen är vilken WebGL-teknologi som är mest lämpligt för att visualisera geospatial data på webben utifrån hur effektiva de är vid rendering av *heatmaps*.

För att lösa problemet behöver vissa delmål nås.

1. WebGL-baserade teknologier behöver identifieras och väljas ut för att kunna utföra undersökningen.

2. En applikation utvecklas för respektive teknologi.

3. Användbarheten av de valda teknologierna evalueras.

4. Utifrån applikationerna kan mätningar av renderingstid genomföras.

5. Resultatet av mätningarna kan analyseras och utvärderas. Teknologierna kan därmed evalueras utifrån hur de utvecklade applikationerna presterade i mätningarna samt hur användbara de är.

Nollhypotesen är att det inte kommer att finnas någon skillnad mellan de olika WebGL-teknologierna i hur lämpliga de är för att visualisera geospatial data i form av *heatmaps*. Mothypotesen är att det finns en skillnad mellan de olika teknologierna i hur lämpliga de är för att visualisera geospatial data i form av *heatmaps*.

4 Metodbeskrivning och tillvägagångssätt

Ett experiment med teknisk inriktning ska genomföras för att kunna påvisa att det finns en prestandaskillnad mellan teknologierna som undersöks och därmed uppfylla delmål 3 och 4. Anledningen till varför ett experiment ska genomföras är på grund av att situationen hela tiden ska vara under kontroll och det ska vara möjligt att styra beteendet precist, direkt och systematiskt. Det ska alltså gå att kontrollera vad som ska mätas och vilka egenskaper som ska mätas och jämföras (Wohlin, Runeson, Höst, Ohlsson, Regnell & Wesslén, 2012).

En alternativ metod till ett experiment som kan användas är att utföra en fallstudie där en integrerad teknologi i ett befintligt system byts ut mot en annan teknologi . En observation av användarna med en prestandamätning sker på systemet innan och efter förändringen för att samla in data utifrån prestationerna. Datan kan jämföras mot varandra och därmed kan ett resultat fås som också kan evalueras. För att uppnå målet med detta arbetet är dock en fallstudie inte optimalt då inte samma nivå av kontroll kan nås som med ett experiment. För att nå den nivå av kontroll kan inte mätningarna ske i en verklig situation med användare utan måste ske i en sluten miljö (Wohlin et al., 2012).

Trots att ett högt kontrollerat experiment utförs kan det ändå ske fel eller oväntade konsekvenser. Det är viktigt att förstå är att ett experiment med positivt resultat utifrån hypotesen inte nödvändigtvis betyder att hypotesen är korrekt. Det går aldrig att bevisa att hypotesen är sann. Det kan alltid finns en annan förklaring till varför resultatet blev som det blev och det finns en oändlig mängd av orsaker som kan påverka resultatet. Det går endast att säga att resultatet tyder på att hypotesen stämmer eller inte stämmer (Berndtsson, Hansson, Olsson & Lundell, 2002).

Utöver experimentet som ska utföras ska även en kvalitativ undersökning genomföras för att kunna besvara delmål 1 och 3. En kvalitativ undersökning har valts på grund av att det är utanför omfattningen av detta arbete att undersöka alla aspekter för användbarhet på egen hand. Det är därmed möjligt att få en bättre förståelse för användbarhetsaspekten av de teknologier som undersöks. På så vis går det att evaluera om någon av teknologierna som undersöks är mer användbar än de andra.

För att möjliggöra en prestandamätning som faktiskt mäter en teknologis prestanda och inte begränsas av kringliggande variabler så bör realistiska parametrar användas. Realistiska parametrar är parametrar som kan efterliknas i verkligheten och i detta fallet mängden data och positionering (koordinaterna) av datan. Datan bör vara riktig, alltså inte simulerad eller genererad. Olika datastorlekar kan därmed användas för att få en bättre insikt av hur en teknologi presterar vid olika nivåer av belastning. En lämplig storlek på ett dataset som kan användas för att kunna mäta prestandaskillnad är mellan 30 000 till 500 000 mätpunkter (James, Moh & Edwards, 2016). Ytterligare så bör endast tiden av renderingen mätas och därmed undvika variabler som hämtning av data över internet. För att mäta renderingstiden behöver en timer användas som kan mäta tiden det tar för teknologin att rendera en *heatmap*.

4.1 Litteratursökning

För att genomföra den kvalitativa undersökningen användes användbarhetsdefinitionen enligt Abran, Khelifi, Suryn & Seffah (2003) som utgångspunkt. Olika databaser har även

använts för att söka efter vetenskapliga artiklar. Dels för att se den totala mängden av artiklar som nämner de utvalda teknologierna samt för att evaluera hur artiklarna anser att användbarheten uppnås av de utvalda teknologierna. Sökkriterierna var att artiklarna behövde innehålla information om de utvalda teknologierna samt att de skulle diskutera användbarhet i någon form. Sökorden som användes var namnet på den utvalda teknologin i kombination med *efficiency*, *effectivity*, *satisfaction* och *learnability* och är enligt Abran, Khelifi, Suryn & Seffah (2003) användbarhetsegenskaper. Databaserna som användes var främst Google Scholar men även IEEE, ACM och ScienceDirect. Första steget för att avgöra om en artikel var av intresse var att studera dess titel och sammanfattning. Om artikeln fortfarande gav intrycket av att innehålla relevant information gjordes en mer grundlig undersökning av den. Sökningarna gjordes mellan månaderna mars och maj år 2018.

Att mäta rätt parametrar är viktigt. Tiden är den främsta variabeln när det kommer till att presentera data. Beroende på responstiden kan användaren uppleva en bättre eller sämre användbarhet och hur lätt han kan nyttja tjänsten (Lin & Lu, 2000). En teknologi som används för att rendera bör alltså vara så effektivt som möjligt vid renderingen för att användaren ska få goda förutsättningar i sin användning.

För att kunna jämföra prestandan av de olika teknologierna behöver samma data användas vid mätningarna. Datan hämtas från ett allmänt tillgängligt API med fri användning för att brukas. Ett specifikt dataset väljs ut med en lämplig storlek och typ av data. Det utvalda datasetet ska användas i samtliga applikationer som skapats med respektive teknologi. Tillräckligt av mätningar ska också genomföras med samtliga applikationer för att på så vis få en möjlighet att påvisa en signifikant skillnad mellan dem. Datasetet som används till mätningarna i pilotstudien kommer från SMHIs marina miljöövervakningsdatabas och går under CC by 4.0 SE (Havs- och vattenmyndigheten och SMHI, 2018). Datasetet som användes för experimentet kommer däremot från ICES Oceanographic databas (ICES Oceanographic database, 2018). Dataseten består i huvudsak av koordinater som tillsammans med salinitetvärden kan användas till att rendera en *heatmap*.

4.2 Etik

Hot mot validiteten av slutsatsen existerar och måste tas på allvar för att hindra att en felaktigt slutsats fastställs. En stor fördel med att använda statistik är att det går att se korrekta mönster i den data som analyseras. Däremot går det att tolka mönstren fel och därmed få en felaktig slutsats. Vissa tester kan också ha en viss förutfattad mening över sig och genom att bryta mot ett antagande kan konsekvenserna leda till en felaktig slutsats. Vissa statistiska tester kan dock vara mer robusta emot att bryta mot ett antagande. Ett annat hot mot validiteten är att söka efter specifika resultat vilket gör slutsatsen felaktig då den blir vinklad på ett särskilt sätt. Validiteten är även beroende av hur experiment genomförs. Olika mätningstillfällen kan ge olika resultat och utomstående variabler kan också påverka resultatet så som ett överflödigt program som körs i bakgrunden samtidigt som mätningen genomförs. Det är därför viktigt att genomföra flera mätningar där resultatet överensstämmer med varandra (Wohlin et al., 2012).

För att försäkras om att testerna blir så trovärdiga som möjligt så används samma dataset för båda applikationerna. De två applikationerna använder även samma hårdvara och mjukvara, utöver de olika teknologierna, för att undvika att de får någon inverkan på resultatet. Datan som används innehåller inte någon form av personlig data eller känsliga uppgifter. Däremot

används en karta vilken fungerar som en bas för de *heatmaps* som skapas. Om den kartan innefattar olämplig data såsom ett militärt skyddsobjekt eller liknande, kan det innebära problematik ur en etisk synpunkt. Ytterligare så kan det skapa problem om datan som ska användas till genereringen av *heatmaps* har en licens som begränsar fritt användande. Inga försökspersoner ska däremot användas i samband med implementationen eller vid testningsfasen. Därmed uppstår det inga tydliga svårigheter ur den synpunkten.

5 Relaterad forskning

Ett antal andra artiklar har gjort liknande forskning som den som görs i detta arbete. Både James, Moh & Edwards (2016) och Resch, Wohlfahrt & Wosniok (2014) utvecklar ett webbaserat system som kan rendera *heatmaps* utifrån marin miljödata. Fokus för deras arbete var dock att skapa ett system som på ett effektivt sätt kan generera *heatmaps* utifrån miljödata och samtidigt minimera responstiden medans detta arbete riktar in sig på att jämföra olika WebGL-teknologier.

Sopan et al. (2012) presenterar i deras arbete en tjänst för att visualisera hälsodata på webben och för att visualisera datan använder de sig av *heatmaps*. Huvudfokus för deras arbete är att leverera visualisationer som kan underlätta användarna av systemet när de ska fatta beslut, ofta på myndighetsnivå. Det här arbetet hanterar inte användbarheten av den *heatmap* som skapas på samma sätt utan fokuserar mer på att evaluera användbarheten av olika WebGL-teknologier istället. Sopan et al. (2012) tar inte upp hur deras applikation utvecklades eller vad för teknologi som användes utan riktar sig nästan enbart på hur användaren kan få så stor nytta som möjligt av visualisationen. Detta skiljer sig kraftigt mot detta arbete då det mestadels är teknologierna som tas upp.

Krämer & Gutbell (2015) behandlar i sin artikel hur olika WebGL-teknologier kan användas och hur de lämpar sig att rendera geospatial data. Deras fokus ligger i att jämföra olika WebGL-teknologier i hur de kan rendera stora mängder av data, i deras fall består datan av en stadsmodell i 3D-format. Jämförelsen är dock inte i huvudsak prestandamässig utan hur en utvecklare skulle kunna använda teknologin. Skillnad är att detta arbete fokuserar på att visualisera data i en 2D miljö medans Krämer & Gutbell (2015) visualiserar data i en 3D miljö. De olika visualiseringarna har olika syften och därmed kräver något olika egenskaper, främst då datan som används är väldigt olika. Ytterligare så har detta arbete något mer fokus på prestandan, alltså renderingstiden av teknologierna, medans deras artikel riktar in sig nästan enbart på användbarheten i en utvecklares perspektiv.

Boya (2017) undersöker i sitt arbete hur interaktioner från användare kan aggregeras och visualiseras med hjälp av *heatmaps* och *heatelements*. Fokus i Boya (2017) är alltså att interaktionsdata som visualiseras med de olika visualiseringsteknikerna kan ge en noggrann och korrekt representation av datan. Både Boya (2017) och detta arbete visualiserar data på liknande sätt men till skillnad från Boya (2017) så ligger fokus i detta arbete att analysera renderingstekniker och inte hur korrekt visualisationen representerar den ursprungliga datan.

6 Genomförande

Inför utvärderingen av experimentet har en litteraturstudie och implementation genomförts, och beskrivs i detta kapitlet.

6.1 Val av ramverk

För att implementera en realistisk testmiljö så behövs teknologier som är relevanta i dagens webb. Det är även viktigt att använda mjukvara som är *open source* för att säkerställa replikationsbarhet.

För att avgöra vilka WebGL-teknologier som skulle användas i experimentet gjordes en undersökning utifrån följande kriterier. Teknologin ska vara *open source*, ha god dokumentation samt ha ett stort och aktivt *community*. I utvärderingen övervägdes Three.js, X3DOM, SceneJS och BabylonJS.

6.1.1 Three.js

Three.js är en teknologi som kan rendera animerad grafik på webben och använder sig utav WebGL. Det förser ett JavaScript API med funktioner som att manipulera objekt i en 3D scen. Three.js är *open source*, är välkänt och har ett stort *community*. Three.js har inte någon geo-specifik support men har många tillägg som går att använda för att rendera grafik utifrån geospatial data. Eftersom att teknologin däremot använder WebGL så blir den väldigt flexibel och användbar i en mängd olika applikationer (Krämer & Gutbell, 2015).

Three.js gör det lättare för utvecklare att skapa en scen då det finns olika verktyg som kan representera data såsom kameror, shaders och ljus som ritar ut linjer, sfärer, partiklar eller ett plan. Ytterligare finns det olika inställningar som kan underlätta som till exempel utseende, synfält och storlek av objekt (Resch, Wohlfahrt & Wosniok, 2014). Olika sorters kameror kan användas såsom *perspectiveCamera* eller *orthographicCamera* och de kan ändra sättet som det animerade objektet visas på skärmen (Evans, Romeo, Bahrehmand, Agenjo & Blat, 2014).

6.1.2 X3DOM

X3DOM är precis som Three.js en teknologi för att rendera animerad grafik på webben, är *open source* och använder sig av WebGL. Det använder sig av ett deklarativt paradigm vilket betyder att datorn får instruktioner för vad den ska rendera, till skillnad från ett imperativt paradigm som ger instruktioner av hur datorn ska rendera grafiken (Ressler & Leber, 2013). Det är framtaget för att kunna använda det standardiserade filformatet X3D. X3DOM är även designat för att integrera *Document Object Model* (DOM) som tillåter hantering av 3D innehåll genom att lägga till, ta bort eller ändra DOM-element (Evans et al., 2014; Krämer & Gutbell, 2015).

6.1.3 SceneJS

SceneJS är en WebGL-baserad teknologi som används för att presentera 3D animationer på webben på ett mer generellt sätt (Anttonen & Salminen, 2011). Det är *open source* och skapar scener baserad på JSON-data (Książek & Pietruszka, 2012).

6.1.4 BabylonJS

BabylonJS är likt de tidigare nämnda teknologierna baserat på WebGL och kan presentera en 3D scen på webben. BabylonJS är däremot mer inriktad mot spelutveckling än de andra teknologierna vilket gör att det är mindre flexibelt än mot till exempel det mer generiska Three.js. Det har dock en inbyggd kollisionsdetektering och en fysikmotor som underlättar för utveckling av spel (Kaistinen, 2015; Ahire, Evans & Blat, 2015).

6.1.5 Undersökning av ramverken

Både Three.js samt X3DOM uppfyller de utvalda kriterierna då de används flitigt och kan användas för att visualisera geospatial data (Krämer & Gutbell, 2015). För att se antalet sökträffar för respektive teknologier gjordes sökningar på Google Scholar och Stack Overflow med ett tidsintervall mellan åren 2017 till 2018 för vardera av de fyra teknologier som valts att utvärderas. Resultatet kan ses i tabell 1. Stack Overflow kan dock inte visa mer än 500 resultat per sökning. Det är därmed oklart hur mycket mer resultat som finns för "Three.js". Mängden resultat tyder på att både Three.js samt X3DOM aktivt används och därmed är relevanta och välkända. SceneJS och BabylonJS faller på kraven angående god dokumentation samt att ha ett stort och aktivt *community*. SceneJS har ett mindre *community* relativt till både Three.js och X3DOM samt att det inte längre utvecklas vilket gör det mindre lämpligt att använda (Rasys, Hodds, Dawood & Kassem, 2014). BabylonJS har ett större *community* än SceneJS men det är ändå relativt litet i jämförelse med Three.js och X3DOM.

Sökord/Sökmotor	**Google Scholar**	**StackOverflow**
Three.js	1300	500
X3DOM	383	101
SceneJS	9	1
BabylonJS	31	49

Tabell 1. Antal sökresultat för teknologierna på Google Scholar samt StackOverflow.

Att använda SceneJS eller BabylonJS i experimentet är av mindre intresse då användningen av dem och deras syfte inte passar in fullt ut med de fastställda kriterierna. SceneJS används relativt lite och BabylonJS är mer anpassat till spelutvecklare än till visualisation av geospatial data. Då de båda används på en relativt mindre skala jämfört med Three.js och X3DOM är de därmed mindre intressanta att undersöka och nyttan som går att få ut av resultatet från ett experiment relativt liten. Eftersom Three.js och X3DOM används relativt mycket är de däremot mer intressanta att undersöka då de till skillnad från SceneJS används mer kontinuerligt och är mer anpassade för detta arbetets ändamål än BabylonJS. Det går därmed att få större nytta av att undersöka Three.js och X3DOM.

6.2 Progression

Den här sektionen beskriver implementationen av teknologierna som använts i experimentet. De båda applikationerna har utvecklats till att vara så lika som möjligt för att i största möjliga grad undvika att något annat än själva teknologierna ska ge ett varierande

resultat. För att se hur arbetet såg ut i ett tidigare skede se commit 961b540 (https://github.com/a15filba/Exjobb_VT18/commit/961b540e066884acc4725ff33092ffd520196377#diff-51f641e10b8aa2eb498b0108db22c059).

Inspiration för att implementera applikationerna har delvis erhållits från respektive teknologis dokumentation på deras webbplatser. En annan viktig inspirationskälla vid utvecklingen av visualiseringen av *heatmaps* var Sun (2017) som gav idéer om hur koordinater kan omvandlas till värmeelement. Ytterligare gav Boya (2017) också en del idéer för hur visualiseringen av en *heatmap* skulle ske. Eftersom varken Boya (2017) eller Sun (2017) byggt sina lösningar mot något WebGL-baserad teknologi behövde implementationen för applikationerna i detta arbete modifieras gentemot hur deras är uppbyggda.

För att kunna översätta koordinaterna från longitud och latitud till x- och y-koordinater som passar skärmen så användes Proj4. Proj4 är ett bibliotek som översätter koordinater från ett koordinatsystem till ett annat koordinatsystem (Adair, Greenwood, Richard, Irons, Terral & Metcalf, 2014). Det var nödvändigt att ändra positionen för att centrera visualitionen på skärmen.

Under implementationen stöttes en del problem på vilket kunde göra det nödvändigt att mer eller mindre ändra riktning av genomförandet. Val av kameran i Three.js var en sådant problem. En kamera används som utgångspunkt för vyn användaren ser. Till en början användes en *perspectiveCamera*. Dock lämpar sig den bäst för en 3D scen då projektionen liknar det som en människa ser och passar mindre bra i en 2D scen där allt bör projekteras platt på skärmen. För detta ändamålet passade en *orthographicCamera* bättre då det renderade objektet förblir konstant oavsett vilken distans kameran har från objektet (Three.js, 2018).

För att rendera en *heatmap* användes först koordinaternas latitud och longitud i WGS84-format direkt. Det gav en riktig bild av den visualiserade datan men renderades väldigt långt från skärmens mitt. Problemet kunde lösas med hjälp av tjänsten Proj4 som konverterar koordinaternas latitud och longitud till x- och y-koordinater (Adair et al., 2014).

6.2.1 Timer och positionering av koordinater

Timerfunktionen skapas för att kunna mäta renderingstiden och kan ses i figur 2. Variablerna start, stop och resultat skapas för att kunna lagra de nödvändiga tiderna. Beroende på om start- eller sluttiden ska sättas kan samma funktion användas genom att specificeras “start” eller “stop” som en parameter när funktionen kallas. Resultatet lagrar mellanskillnaden av start och stop, och evalueras samtidigt som stop variabeln sätts. Varje tid som räknas ut läggs till i *local storage* och kan på så vis sparas över flera mätningar även när sidan laddas om.

```
function timer(time) {
    if (time == start) {
         start = Date.now();
    } else if (time == stop) {
         stop = Date.now();
         result = stop - start;

         // Save rendering time to localStorage
         var renderedTimes = [];
         var getTimes;
         if(Array.isArray(JSON.parse(localStorage.getItem("time")))) {
              getTimes = JSON.parse(localStorage.getItem("time"));
              renderedTimes = renderedTimes.concat(getTimes);
         }
         renderedTimes.push(result);
         localStorage.setItem("time", JSON.stringify(renderedTimes));

         console.log(JSON.parse(localStorage.getItem("time")));
    } else {
         console.log("Timer was not set correctly");
    }
}
```

Figur 2. Funktionen timer.

Datan som ska visualiseras hämtas från ett json-dokument på servern. Innan en koordinat läggs till i *coordinates* minskas dock x och y från att representeras i meter till 10 km samt att koordinaten flyttas närmare origo, vilket gör det lättare att visualisera *heatmapen* på skärmen. Koordinaterna får ett id, de aktuella x- och y-värdena, samt ett värde som ökar ju fler interaktioner just den koordinaten har vilket visas i figur 3.

```
function addID(point) { // Adds the coordinates and a value to an array
    var id = ((point.x + point.y) % jsonData.length) + "";
    var value = point.value;
    // x and y coordinates in 10km instead of meters
    var x = point.x / 10000;
    var y = point.y / 10000;

    // Move the coordinates closer to origo
    var movedX = x - 30;
    var movedY = y - 780;

    // If coordinates has property of id, average the value
    if(coordinates[id]) {
         coordinates[id].value = (value + coordinates[id].value) / 2;
    } else {
         // Create new coordinate
         coordinates[id] = {value : value, x : movedX, y : movedY};
    }
}
```

Figur 3. Koordinaterna justeras och läggs till i ett objekt i funktionen *addID*.

För att visualisera koordinaterna behöver de kalkyleras utifrån deras mätvärde och position. I funktionen *dotSystem* som kan ses i figur 4, skapas ett rutnät och varje koordinat itereras för att avgöra vart i rutnätet de placeras. Om flera koordinater hamnar i samma ruta tas medelvärdet av deras mätvärde för att senare kunna avgöra deras färg på den *heatmap* som skapas. De beräknade koordinaterna läggs till i objektet *dotCoord* för att senare kunna renderas.

```
for(var id in coordinates) {
    for(var x = -canWidth; x < canWidth; x += gridSize) {
         for(var y = -canHeight; y < canHeight; y += gridSize) {
              if(x <= coordinates[id].x && x + gridSize >= coordinates[id].x && y <=
coordinates[id].y && y + gridSize >= coordinates[id].y) {
                   var dotID = x * y;
                   var value = coordinates[id].value;
                   if(dotCoord[dotID]) { // Make a average of the new and old value for
every new value
                        dotCoord[dotID].value = (value + dotCoord[dotID].value) / 2;
                   } else { // To get a more accurate x and y coordinate, half of the
gridSize is added
                        dotCoord[dotID] = {value : value, x : x + (gridSize/2), y : y +
(gridSize/2)};
                   }
              }
         }
    }
}
```

Figur 4. Ett stycke ur funktionen *dotSystem* där ett rutsystem skapas.

6.2.2 Three.js

En ny scen och en kamera behöver skapas för att kunna visualisera något över huvud taget vilket ses i figur 5. Kameran är ortografisk då enbart en 2D-scen skapas. Även storleken på både kameravyn och renderingsfältet specificeras.

```
var scene = new THREE.Scene();

var width = 600;
var height = 600;
var near = 0;
var far = 5;
var camera = new THREE.OrthographicCamera(width / - 2, width / 2, height / 2, height / -
2, near, far);

var renderer = new THREE.WebGLRenderer({alpha: true});
renderer.setSize(600, 600);
document.body.appendChild(renderer.domElement);
```

Figur 5. En ny scen och kamera skapas.

De punkter som ska bygga upp *heatmapen* representerar de rutor från rutnätet som skapades tidigare i funktionen *dotSystem*. Varje ruta som fått ett värde i *dotSystem* itereras och läggs till i scenen utifrån rutans position samt får en färg tilldelad beroende på vilket mätvärde rutan har. Processen kan ses i figur 6.

```
for(var id in dotCoord) {
    // Convert the value to rgba colors
    var rgba = value2rgba(dotCoord[id].value / max);

    var dotMaterial = new THREE.PointsMaterial({size: 4, sizeAttenuation: false,
transparent: true});
    var dot = new THREE.Points(dotGeometry, dotMaterial);
    dot.material.color.setRGB(rgba[0], rgba[1], rgba[2]);

    dot.position.x = dotCoord[id].x;
    dot.position.y = dotCoord[id].y;
    scene.add(dot);
}
```

Figur 6. Funktionen *createDot* lägger till rutor i scenen som skapar en heatmap.

För att rendera objekten i Three.js används kodstycket som kan ses i figur 7 som är en inbyggd funktion i Three.js. Både den skapade scenen med alla tillagda punkter och kameran som tidigare definierats skapas med hjälp utav den funktionen.

```
renderer.render(scene, camera);
```

Figur 7. Den inbyggda funktionen som renderar scenen.

6.2.3 X3DOM

För att rendera scenen i X3DOM itereras först varje ruta likt den process som användes för Three.js. Till skillnad från Three.js så behöver däremot nya dom-element läggas till i html-filen för att kunna rendera en scen i X3DOM. Figur 8 visar hur en ny nod läggs till med alla element som krävs för att skapa en ny punkt i visualiseringen.

```
function addNode(x, y, color) {
    var transform = document.createElement('Transform');
    transform.setAttribute("translation", x + ' ' + y + ' ' + '0' );

    var shape = document.createElement('shape');

    var app = document.createElement('Appearance');
    var mat = document.createElement('Material');
    mat.setAttribute('diffuseColor', color[0] + ' ' + color[1] + ' ' + color[2]);
    app.appendChild(mat);
    shape.appendChild(app);

    transform.appendChild(shape);

    var node = document.createElement('indexedFaceSet');
    node.setAttribute('coordIndex', '0 1 2 3 -1');

    var coord = document.createElement('coordinate');
    coord.setAttribute('point', '0 0 0, 4 0 0, 4 4 0, 0 4 0');

    node.appendChild(coord);
    shape.appendChild(node);

    var scene = document.getElementById('scene');
    scene.append(transform);
}
```

Figur 8. I funktionen *addNode* läggs dom-element till i html-filen.

Hela skriptet av den beskrivna Three.js-koden kan hittas i *appendix A* och html-koden som används till den beskrivna koden återfinns i *appendix B*. I *appendix C* finns skriptet för X3DOM och motsvarande html-kod kan hittas i *appendix D*.

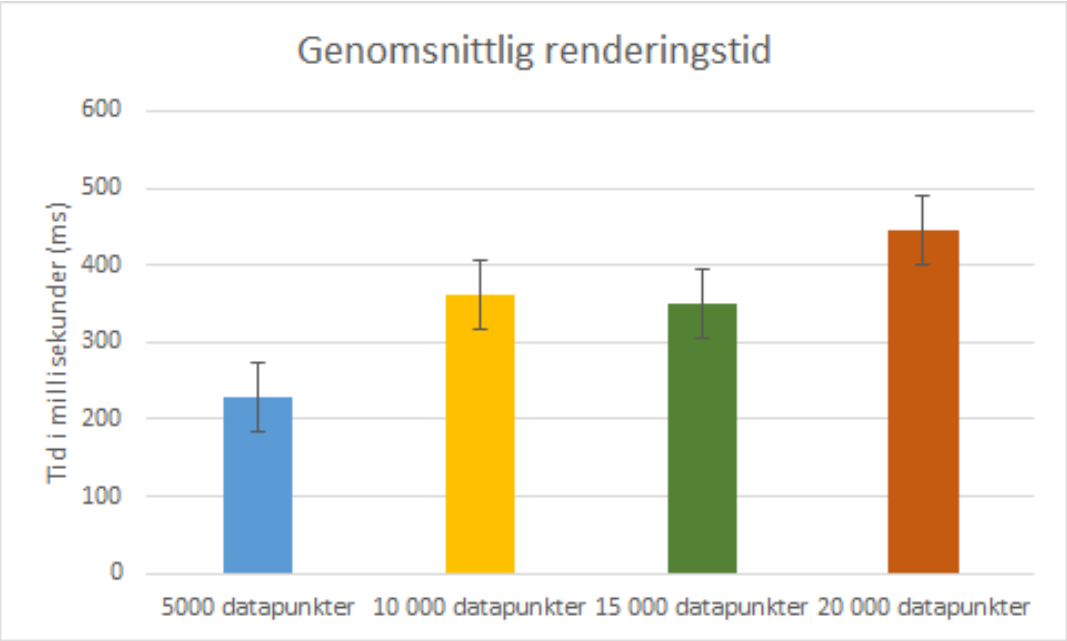
6.3 Pilotstudie

En pilotstudie utfördes för att bedöma om arbetet är utvärderingsbart, om det går att genomföra mätningar samt att identifiera eventuella brister med experimentet. Pilotstudien bestod av 4 olika mättillfällen och samtliga gjordes med Three.js applikationen. En mätning genomfördes med 5000, en med 10000, en med 15000 och en med 20000 datapunkter. 100 mätningar genomfördes per mättillfälle och det tog ca 2 minuter för mätningen med 5000 mätpunkter, ca 3 minuter med 10000, ca 3 minuter och 30 sekunder med 15000 och ca 5 minuter för mätningen med 20000 mätpunkter. Mätningarna utfördes på en Dell XPS 13

9343 med specifikationer enligt tabell 2. Figur 10 visar ett diagram av den insamlade datan från mätningarna.

Processor	Intel(R) Core(TM) i5-5200U CPU (2.20GHz)
RAM	8 GB
Operativsystem	Windows 10, version 1803
Server	Node.js (lokal), version 8.9.4
Webbläsare	Chrome, version 66.0.3359.170

Tabell 2. Lista över hårdvara och mjukvara.



Figur 9. Genomsnittliga renderingstiden med *error bars* av 5000, 10000, 15000 respektive 20000 datapunkter.

Resultaten från figur 9 visar att det finns en skillnad i renderingstiden beroende på antalet datapunkter som ska visualiseras. *Error bars* visar däremot att det finns en variation av de enskilda mätningarna. Det innebär att om olika mängder av datapunkter används i mätningarna så kommer antagligen även renderingstiden att variera. Det kan därmed bli intressant att undersöka hur olika WebGL-teknologier presterar vid mätningar av flera dataset i olika storlekar.

En brist som uppdagat sig efter att mätningarna gjorts var att endast mäta renderingstiden betyder inte nödvändigtvis att det blir en rättvis jämförelse. Ett mer noggrant tillvägagångssätt är att mäta tiden det tar för både processen att räkna ut positionerna av alla koordinaterna samt att rendera scenen. På så vis kan en större del av processen jämföras mellan teknologierna och därmed ta hänsyn till hur effektivt de fungerar på ett bredare spektrum vid visualisation av en *heatmap*. Inte bara hur de renderar en *heatmap* utan även hur de lägger till data till en scen. Ökningen i figur 9 visar inte en linjär ökning när fler

datapunkter används och det är oklart varför det är så. Det kan bero på att inte hela processen för skapandet av en *heatmap* mäts. Om hela processen mäts kan det däremot vara möjligt att få ett tydligare resultat.

Pilotstudien visar att det är möjligt att mäta renderingstiden vid rendering av heatmaps med en WebGL-teknologi. Det är därmed rimligt att kunna jämföra olika WebGL-teknologier och evaluera dem utifrån hypotesen.

6.4 Evaluering av användbarhet

Med utgångspunkt från Abran, Khelifi, Suryn & Seffah (2003) studie användes följande användbarhetsegenskaper *effectiveness*, *efficiency*, *satisfaction* och *learnability* för att avgöra lämpligheten av de valda teknologierna vid rendering av *heatmaps*. För att ta reda på till vilken nivå teknologierna uppfyller användbarhetsmodellen som Abran, Khelifi, Suryn & Seffah (2003) presenterar, undersöks hur användbara de utvalda teknologierna är främst utifrån en utvecklares perspektiv.

6.4.1 Three.js

Effectiveness: Three.js har många funktioner och kan därmed presentera en scen på ett effektivt sätt. Det är därför möjligt att anpassa visualiseringen för att uppnå ett önskat resultat (Ha, Jin & Lee, 2015; Resch, Wohlfahrt & Wosniok, 2014). Three.js har även direkt tillgång till WebGL vilket gör att det är väldigt flexibelt och därmed användbart i flera olika applikationer. Det är därför möjligt att skapa komplexa funktioner vilka kan vara svåra att skapa med andra teknologier såsom minneshantering, vilket ger utvecklare en god kontroll av hur renderingen ska visas och fungera (Krämer & Gutbell, 2015). Three.js kan dock vara svårt att skala upp då det kräver relativt mycket kod för att skapa en scen i jämförelse med X3DOM (Butcher & Ritsos, 2017).

Efficiency: Three.js kan hålla en hög *framerate* även när visualiseringen är av hög detaljrikedom vilket gör att den tillfredsställer användaren (Butcher & Ritsos, 2017).

Satisfaction: Eftersom en scen som skapas med Three.js är lätt att modifiera och ger god kontroll till utvecklaren om hur scenen ska se ut, kan den lätt anpassas till hur den ska uppfattas av användaren (Krämer & Gutbell 2015). En renderad bild kan däremot ge upphov till brister i visualiseringen, såsom överhörning samt brister utav linsförvrängningskorrigering vilket kan ge en obehaglig upplevelse för användaren (Butcher, & Ritsos 2017). En interaktion med ett datorprogram eller liknande kan användas för att höja en användares nöjdhet och prestanda. Three.js tillåter användaren att få möjlighet att interagera med den renderade scenen (Unar, Unar & Patoli, 2016; Resch, Wohlfahrt & Wosniok, 2014)

Learnability: Three.js har tillgång till ett stort *community* och därmed finns det alltid tillgång till hjälp vid behov (Krämer & Gutbell, 2015). Dock kan Three.js ge upphov till komplexa scener vilket kan leda till att det är svårt att skala upp (Butcher & Ritsos, 2017).

Three.js uppnår Abran, Khelifi, Suryn & Seffah (2003) användbarhetsegenskaper mestadels. *Effectiveness* uppnås mestadels genom att ett flertal funktioner ger utvecklaren god kontroll av hur renderingen ska visas. Det begränsas dock något av att det kan vara svårt att skala upp komplexa projekt. *Efficiency* uppnås då Three.js presterar relativt bra och därmed förbättrar användarupplevelsen. *Satisfaction* uppnås mestadels då en scen är lätt att

modifiera och anpassa och kan därmed visa bra visualiseringar för användaren. Dessvärre kan vissa brister i renderingen förekomma och därför uppnås inte *satisfaction* helt. Däremot kan en användare bli mer tillfredsställd då det går att modifiera scenen när den ritats ut. *Learnability* uppnås också delvis då det finns god tillgång till material och andra utvecklare där stöd om Three.js kan finnas men kan däremot bli komplext i större projekt.

6.4.2 X3DOM

Effectiveness: Då X3DOM är baserat på standardfilformatet X3D har teknologin flera funktionaliteter vilket ger fler möjligheter att presentera en scen. X3DOM är dock relativt begränsad när det kommer till funktionalitet i jämförelse med Three.js (Krämer & Gutbell, 2015). Då fler element kan läggas till för att utöka 3D-objektet gör det teknologin väldigt utökbar och lätt att skräddarsy (Di Cerbo, Dodero & Papaleo, 2010).

Efficiency: Tillsammans med filformatet X3D kan teknologin X3DOM rendera geometriska primitiver snabbare än andra standardiserade data modeller (Prieto, Izkara & del Hoyo, 2012).

Satisfaction: En interaktion med ett datorprogram eller liknande kan användas för att höja en användares nöjdhet och prestanda. Eftersom en användare kan interagera med en renderad scen i X3DOM där till exempel ett renderat objekt kan förändras av användaren med enbart muspekaren, höjer det alltså en användares tillfredsställelse (Unar, Unar & Patoli, 2016).

Learnability: X3DOM kan rendera komplexa geometriska figurer utan att komplicerade kodstycken behöver användas, ofta räcker det endast med en kodrad. Utvecklare undviker alltså att lära sig lågnivå WebGL API:t och kan istället skapa en 3D-scen på en väldigt hög nivå, vilket gör det enklare för dem att skapa en 3D-scen (Krämer & Gutbell, 2015). För att definiera en scen använder teknologin sig av HTML-taggar. Det gör det simpelt för en utvecklare att lära sig systemet (Sharakhov, Polys & Sforza, 2013). Teknologin är även lätt att implementera då inga tredje-parts teknologier krävs och då endast fler DOM-element behöver läggas till för att utöka 3D-objektet gör det teknologin enkel att anpassa (Di Cerbo, Dodero & Papaleo, 2010).

X3DOM uppnår Abran, Khelifi, Suryn & Seffah (2003) användbarhetsegenskaper till stor del. *Effectiveness* uppnås då det har ett flertal möjligheter att rendera en scen samt att det är lätt att skala upp, till skillnad från Three.js. Dock har teknologin inte stort omfång av möjligheter som Three.js. *Efficiency* uppnås genom att de snabbt kan rendera en scen utifrån en X3D-fil. *Satisfaction* uppnås då en användare kan interagera med en renderad scen. *Learnability* uppnås genom att relativt lite kod ofta räcker för att rendera en scen vilket gör det enkelt för utvecklare att lära sig och använda. Ytterligare kan utvecklaren återanvända sina kunskaper om HTML för bygga upp en scen vilket minskar tiden det tar att lära sig teknologin.

7 Utvärdering

Mätningarna utfördes på samma enhet som pilotstudien vilket var en Dell XPS 13 9343 med specifikationer enligt tabell 2 och användes för att genomföra mätningarna.

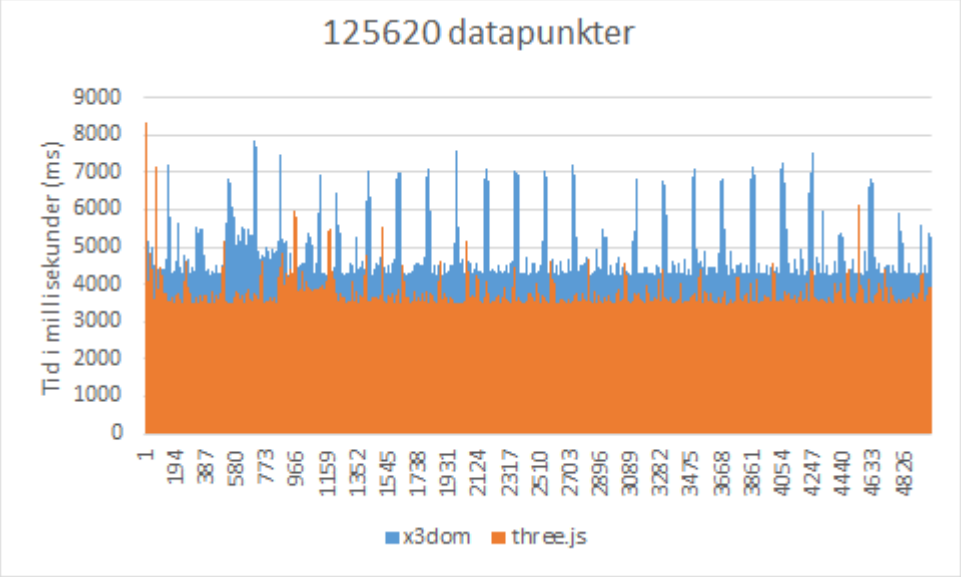
7.1 Presentation av undersökning

Experimentet består av två olika mätgrupper som gjordes med Three.js respektive X3DOM. Varje mätgrupp består av fyra olika mättillfällen som mäter renderingstiden och gjordes med 30000 datapunkter, 60000 datapunkter, 90000 datapunkter respektive 125620 datapunkter. Anledningen till att det sista mättillfället har ett udda antal datapunkter är för att det är storleken av datasetet som användes. För att få ut så mycket som möjligt av datasetet så användes alla tillgängliga datapunkter. Till den första mätgruppen användes Three.js applikationen och till den andra mätgruppen användes X3DOM applikationen. Vid varje mättillfälle utfördes 5000 mätningar.

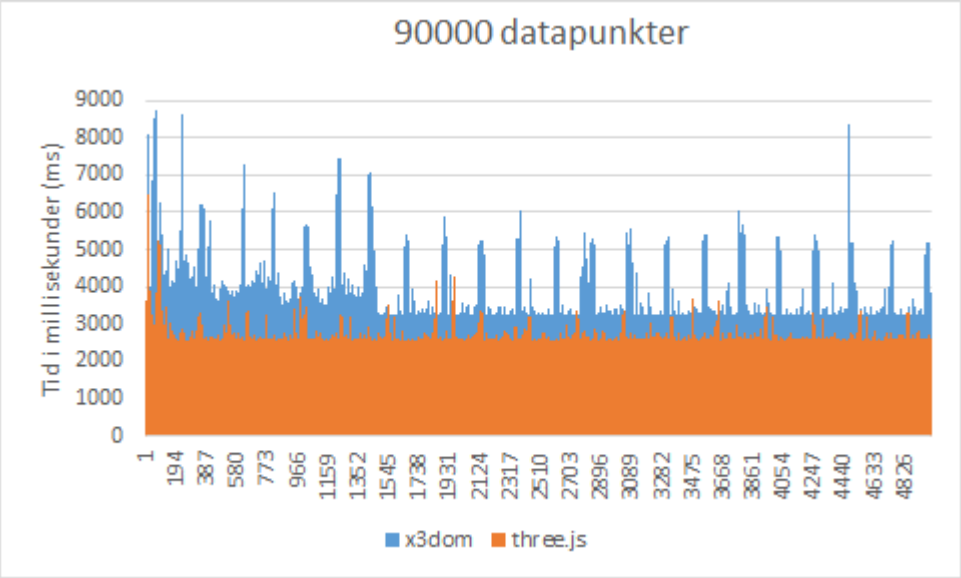
Till skillnad från pilotstudien så mäts inte bara själva renderingstiden i följande mättillfällen utan även processen för att räkna ut hur en *heatmap* ska renderas. Resultatet visar därmed en längre tid men den visar även en större del av förloppet. Då båda applikationerna använder samma sätt att positionera koordinaterna bör skillnaden i renderingstiden mellan applikationerna endast bero på teknologierna, även fast mer än bara själva renderingen mäts. Processen kan därmed jämföras mellan teknologierna och ta hänsyn till hur effektivt de fungerar på ett bredare spektrum vid visualisation av en *heatmap*. Inte bara hur de renderar en *heatmap* utan även hur de lägger till data till en scen.

7.2 Resultat

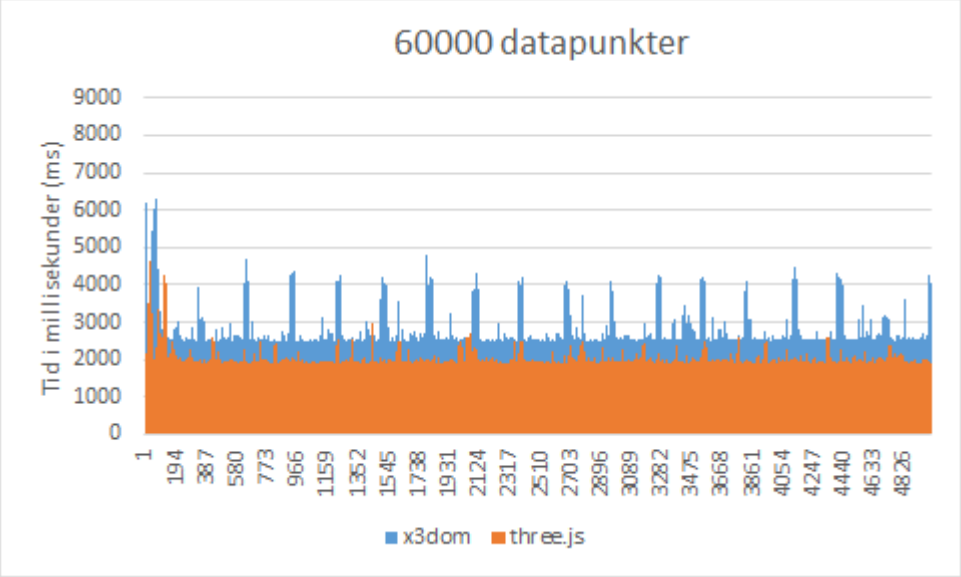
I figur 10, 11, 12 och 13 visas den råa osorterade brusdatan från de olika mättillfällena. Alla fyra figurer visar att Three.js är generellt snabbare än X3DOM. Alla fyra figurer visar även att det det finns flera spikar i båda mätgrupperna men de är dock mest märkbara i figur 10 och 11. Spikarna verkar även drabba X3DOM mer än Three.js. Resultaten verkar dock ligga på en relativt jämn nivå i figur 10, 11, 12 och 13, dock med undantaget att X3DOM i figur 10 och 11 verkar ha perioder tidigt i mätningen som inte helt matchar övriga mättillfället.



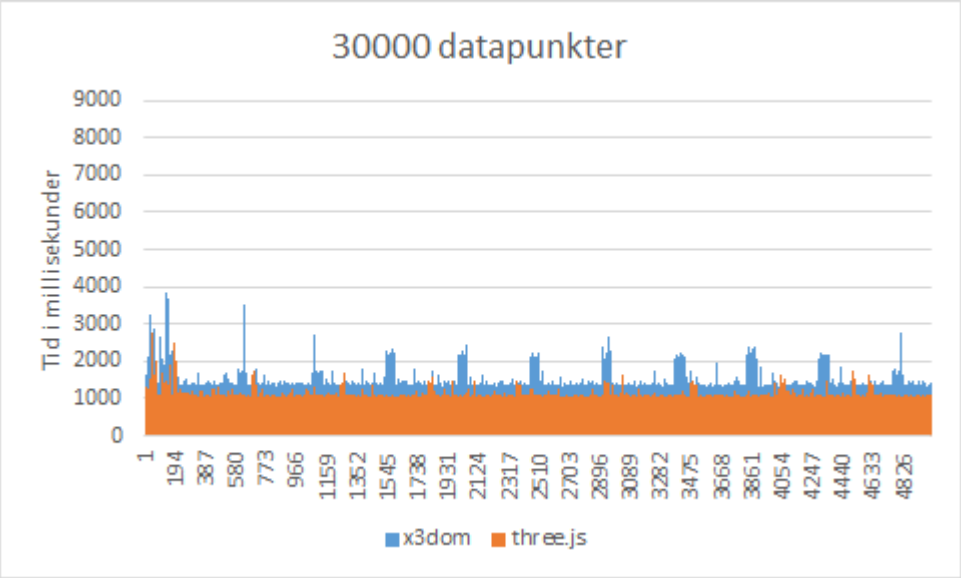
Figur 10. Osorterad rå brusdata av 125620 datapunkter.



Figur 11. Osorterad rå brusdata av 90000 datapunkter.

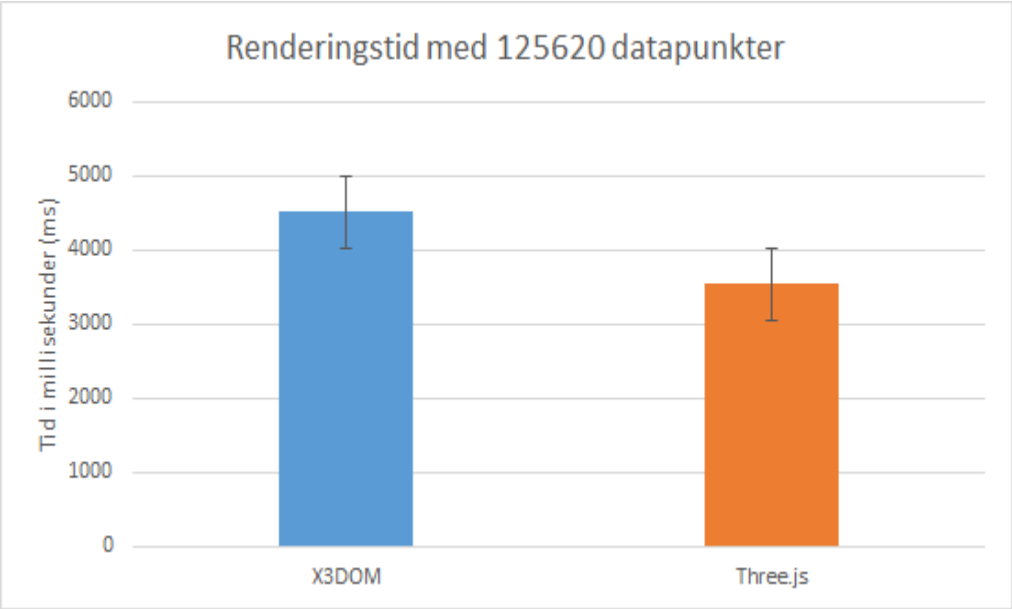


Figur 12. Osorterad rå brusdata av 60000 datapunkter.

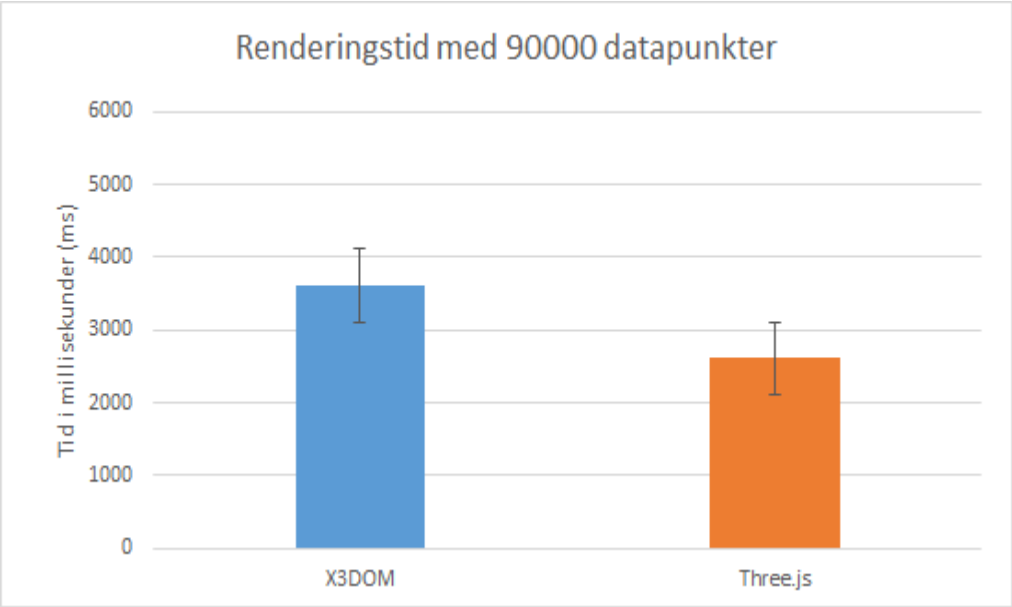


Figur 13. Osorterad rå brusdata av 30000 datapunkter.

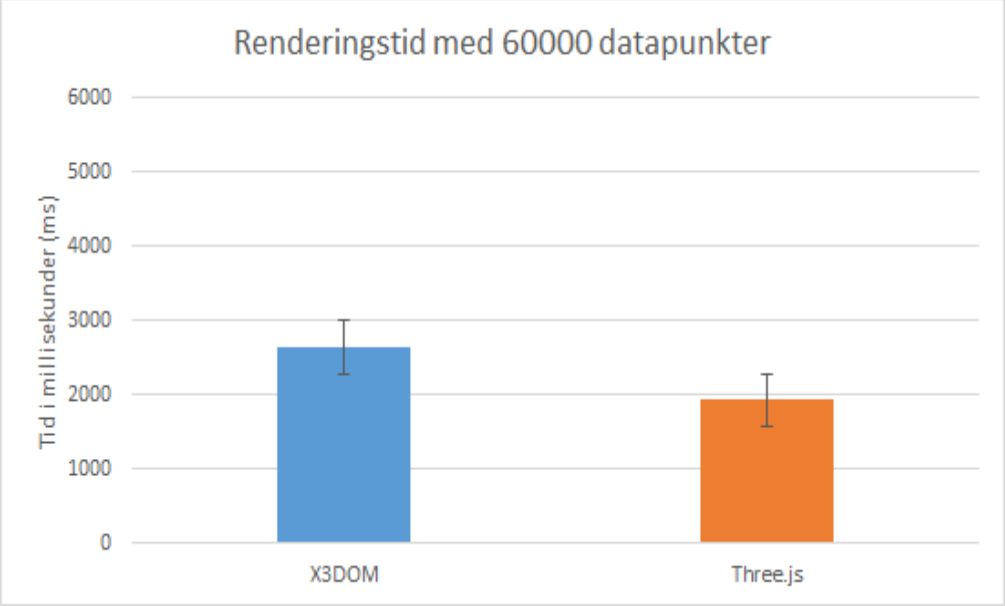
Figur 14, 15, 16 och 17 visar medelvärdet för de olika mättillfällena och en genomgående trend är att X3DOM har en längre renderingstid än Three.js. *Error bars* visar att variationen mellan värdena från de två teknologierna är av liknande storlek.



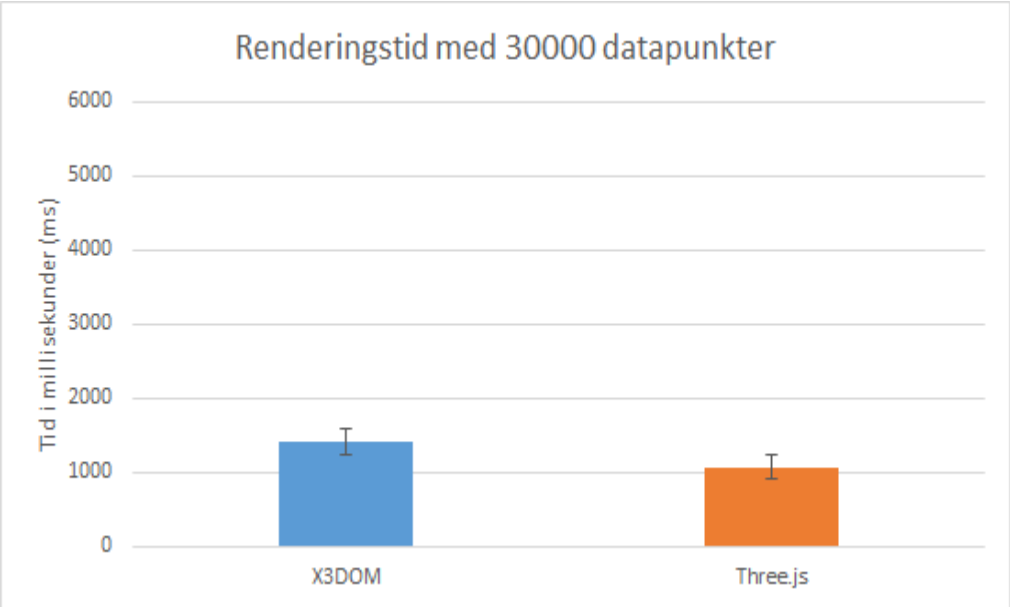
Figur 14. Medelvärde med *error bars* av 125620 datapunkter.



Figur 15. Medelvärde med *error bars* av 90000 datapunkter.

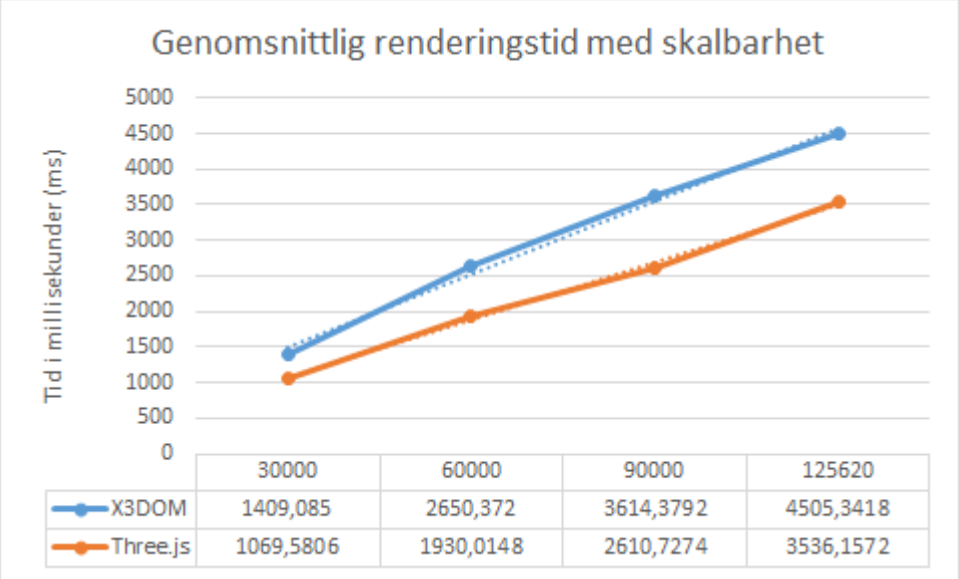


Figur 16. Medelvärde med *error bars* av 60000 datapunkter.



Figur 17. Medelvärde med *error bars* av 30000 datapunkter.

Figur 18 visar den genomsnittliga renderingstiden för både Three.js och X3DOM. Renderingstiden skalas upp stadigt i takt med att antalet datapunkter ökar. Figuren visar även att X3DOM i genomsnitt ca 340 millisekunder (ms) långsammare än Three.js vid rendering av 30000 datapunkter, ca 720 ms långsammare med 60000 datapunkter, ca 1004 ms långsammare med 90000 datapunkter och ca 969 ms långsammare med 125620 datapunkter.



Figur 18. Den genomsnittliga renderingstiden för alla fyra olika mätgrupper. Även hur teknologierna skalas upp representeras av en prickad linje för respektive teknologi.

7.3 Analys

De mätningar som utförts visar att X3DOM har en något högre renderingstid än Three.js. I genomsnitt för de olika mättillfällena var renderingstiden mellan ca 340 till 1004 ms långsammare med X3DOM mot Three.js. Skillnaden ökade däremot inte när antalet datapunkter steg från 90000 till 125620, vilket kan tyda på att vid ett större antal datapunkter jämnar skillnaden ut sig mellan teknologierna. Det skulle även kunna tyda på att skillnaden avstannar så att renderingstiden för X3DOM fortsätter att vara ca 1000 ms högre än den för Three.js.

Spikarna som kan ses i figur 10, 11, 12 och 13 kan bero på dolda bakgrundsprocesser i datorn. De dolda processerna kan ha krävt en betydlig del av beräkningskraften som därmed på något sätt möjligtvis påverkat mätningarna.

Mätningarna i figur 18 visar att om antalet datapunkter ökar, ökar även renderingstiden. Båda teknologierna skalas på samma sätt och skiljer sig inte så mycket mellan varandra. I mättillfällena med mindre antal datapunkter var dock skillnaden mindre än för mättillfällena med fler datapunkter. Three.js skalas inte upp lika mycket som för X3DOM. Med den informationen kan ett antagande göras att vid rendering av större dataset kommer skillnaden mellan teknologierna öka då X3DOM får ännu lägre renderingstid jämfört med Three.js.

Utifrån den kvalitativa sökningen så finns det styrkor och svagheter med både Three.js och X3DOM. Om X3DOM nämns däremot inga direkt negativa egenskaper i de undersökta artiklarna.

Three.js ger en god kontroll av vad som visas men kan vara svårt att skala upp då det har direkt tillgång till WebGL vilket bidrar till mer komplexitet ju större projekt som skapas. X3DOM ger likt Three.js, en god kontroll av vad som ska renderas men till skillnad mot Three.js så kan X3DOM skalas upp relativt enkelt vilket gör det till något mer användbart utifrån *effectiveness*. På grund av X3DOMs simpelhet blir dock möjligheterna mindre i

jämförelse till Three.js då det inte går att styra till exempel minneshantering. Både Three.js och X3DOM presterar dock bra vilket ger dem liknande användbarhet när det kommer till *efficiency*. Three.js kan skapa bra och relevanta visualiationer för användaren men det kan dessvärre uppstå brister i renderingen. Både Three.js och X3DOM tillåter dock användaren att påverka den renderade scenen på något sätt vilket ökar tillfredsställelsen hos användaren. Eftersom båda teknologierna kan tillåta användaren att modifiera scenen som renderas blir båda användbara sett till egenskapen *satisfaction*. Dock på grund av att det kan uppstå brister i renderningen med Three.js hamnar X3DOM något före ur en *satisfaction*-synvinkel. God tillgång till hjälp finns i Three.js *community* vid behov men utvecklingen kan ändå bli komplex i större projekt. X3DOM är däremot relativt lätt att lära sig samt att det kräver mindre kod för att rendera objekt i scenen och på så sätt undviker X3DOM samma komplexitet som Three.js gör. X3DOM är alltså mer användbart än Three.js när det kommer till l*earnability*.

Three.js och X3DOM är båda lämpliga teknologier i de flesta fall. De skiljer sig dock på vissa punkter. Mest synligt är det när det kommer till funktionalitet och komplexitet. Komplexiteten i Three.js kan ses som en styrka då högre komplexitet i vissa fall ger fler möjligheter att skapa en större variation av projekt. X3DOM mister den möjligheten något då den strävar efter att vara simpel att använda.

7.4 Slutsatser

X3DOM verkar enligt den kvalitativa undersökningen vara något mer användbar än vad Three.js är. En kort sammanfattning av användbarheten kan ses i tabell 3. Enligt mätningarna är däremot Three.js mer effektivt vid rendering. En *heatmap* kräver ett mer komplicerat tillvägagångssätt för att kunna produceras. För rendering av *heatmaps,* som var ett delmål för det här arbetet, är mest troligen Three.js bättre anpassat då teknologin är dels något snabbare på att rita ut en *heatmap* samt att den ger fler möjligheter att anpassa hur visualisationen ska renderas. Eftersom Three.js är bättre anpassad för att rendera mer komplexa system än X3DOM så är det mer lämpligt att använda Three.js för rendering av *heatmaps* vilket även besvarar mothypotesen.

Egenskap/Teknologi	**Three.js**	**X3DOM**
Effectiveness	Mycket funktionaliteter men har tendens att bli komplext	Mindre funktionalitet än Three.js
Efficiency	God prestanda	God prestanda
Satisfaction	Ger god kontroll för användaren men kan förekomma defekter i renderingen	Enkel att interagera med en renderad scen
Learnability	Stort community (lätt att få hjälp)	Enkel att använda och lära sig

Tabell 3. Sammanfattning av hur väl teknologierna uppnår användbarhet.

8 Avslutande diskussion

8.1 Sammanfattning

Delmål 1 besvarades genom att göra en utvärdering av olika WebGL-teknologier utifrån specifika kriterier och det framgick i utvärderingen att Three.js och X3DOM var de mest kvalificerade teknologierna att undersöka. Delmål 2 besvarades genom att en applikation utvecklades för respektive utvalda teknologier och resultatet blev två fungerande applikationer där renderingstiden kan mätas. Delmål 3 besvarades genom att en evaluering av användbarheten utfördes utifrån användbarhetsegenskaperna *effectiveness*, *efficiency*, *satisfaction* och *learnability*. Resultatet blev att både Three.js och X3DOM hade liknande egenskaper men X3DOM är simplare att använda medans Three.js är bättre när mer komplexa funktioner ska skapas. Delmål 4 besvarades genom att mätningar utfördes på de två utvecklade applikationerna och resultatet blev att X3DOM var mellan ca 340 till 1004 millisekunder långsammare än Three.js. Delmål 5 besvarades genom att en analys av resultaten genomfördes och resultatet blev att Three.js är mer lämpad att rendera en *heatmap* än X3DOM.

Genom att besvara delmål 1 till 5 besvarades även frågeställningen och resultatet blev att Three.js är mer lämplig att visualisera geospatial data i form av *heatmaps* på webben. Three.js hanterar bättre mer komplexa funktioner samt är snabbare än X3DOM på att visualisera geospatial data. Problemet är därmed löst och resultatet är sammanfattningsvis att Three.js borde användas för att visualisera geospatial data i form av en *heatmap*.

8.2 Diskussion

Resultatet visade att det fanns en skillnad av renderingstiden mellan de två teknologierna men den är antagligen av mindre betydelse. Även om det var upp till en sekund extra väntetid för X3DOM så bör det inte ha så stor påverkan på användaren med tanke på att renderingstiden endast var en liten del av den totala tiden applikationerna tog på sig. För just detta specifika område så borde inte val av teknologi spela någon större roll ur ett tidsperspektiv så länge tidsskillnaden inte ökar. Dock hävdar Rajamony & Elmootazbellah (2001) att en användare föredrar en kortare responstid vilket slutligen ändå gör Three.js mer lämpligt för att rendera geospatial data utifrån ett tidsperspektiv. Krämer & Gutbell (2015) undersöker WebGL-teknologier för deras möjligheter av att användas vid rendering av geospatial data på webben. Deras resultat var likt resultaten från detta arbete. De menar att det finns olika användningsområden för båda teknologierna och ingen av dem är bättre eller sämre i alla olika situationer.

Resultaten av mätningarna behöver dock inte nödvändigtvis reflektera teknologiernas verkliga prestationsförmåga. Det är möjligt att på grund av att maskinen som körde testerna inte var nyinstallerad kan det ha påverka prestandan. Även oförutsägbara bakgrundsprocesser i maskinen kan ha påverkat resultatet genom att ge dem olika förutsättningar än vad de annars skulle ha och därav möjligtvis olika tider. Dock utgick båda teknologierna från samma förutsättningar så skillnaden mellan dem bör inte ha påverkats på en betydande skala.

Artiklarna som undersökts skriver främst positiva saker om både Three.js och X3DOM och de har båda liknande egenskaper. Det fanns dock ett fåtal artiklar där negativa egenskaper

nämns och då endast riktade åt Three.js medan inga tydliga negativa egenskaper fanns skrivna om X3DOM. Om X3DOM är helt fria från negativa egenskaper är något oklart men det finns inget som tyder på det utifrån de undersökta artiklarna. Eftersom inte alla artiklar som finns tillgängliga har undersökts finns däremot en risk att en eller flera artiklar som nämner kritiska delar om någon av de undersökta teknologierna inte hittats och har då inte heller inkluderats i resultatet. Dessa artiklar skulle potentiellt kunna påverka resultatet av denna undersökningen på ett eller annat sätt och därmed bidra med ett helt annat resultat än det nuvarande.

Datan som användes för att generera *heatmaps* i detta arbete bestod av ett flertal datapunkter som alla har tio värden var, varav de mest intressanta är en latitud- och en longitudkoordinat, ett salinitetvärde, temperatur och vattendjup. De värdena som användes i detta arbete var enbart koordinaterna och salinitetvärdet vilket förenklade processen för att skapa en *heatmap*. De övriga värdena användes inte i detta arbete men skulle kunna byta ut salinitetvärdet och därmed skapa en *heatmap* som visar annan potentiellt intressant data över ett visst område.

Resultatet från mätningarna i detta arbete visar att om antalet datapunkter ökar kommer även renderingstiden att öka. Dock är det oklart vad som händer om ett större dataset med fler datapunkter används. Renderingstiden skulle eventuellt kunna öka exponentiellt, plana ut eller fortsätta på samma sätt som den gjort i dessa mätningarna. Detta går att ta reda på om fler mätningar görs med större dataset. Däremot kan olika dataset visa olika resultat. Med just denna data är resultatet på det här sättet men med ett annat dataset kan resultatet bli något annat.

Samhällsnyttan med detta arbete kan erhållas av många olika industrier såsom inom fiskeindustrin, olje och gasindustrin samt transportindustrin. Även forskning och fritidsbåtägare kan dra nytta av detta arbete. Med en bättre förståelse av vilken teknologi som bäst lämpar sig för visualisation av geospatial data kan en användare som tar del av ett system som utvecklas med dessa riktlinjer bättre ta del av datan. Genom att bättre kunna ta del av datan som är av intresse kan användaren också förstå den bättre och därav fatta bättre beslut inom användarens egna område.

Risker som arbetet kan bidra med är beroende på vilket dataset som används till att skapa en *heatmap*. Om miljödata används kommer sannolikt inga risker att uppstå men om någon annan data som till exempel innehåller information om en person eller myndighet kan problem uppstå. Det finns alltid en risk angående integritet med geospatial data om det finns möjlighet att kopplingar till en person existerar.

8.3 Framtida arbete

Möjligheter för kommande arbeten kring samma område skulle med fördel under kort sikt kunna använda sig av ett större dataset. Genom att använda ett större dataset med fler datapunkter kan det undersökas om det blir en ännu större skillnad mellan teknologiernas renderingstid eller om den minskar. Även en ytterligare teknologi skulle kunna undersökas för att få en bredare förståelse av skillnaden mellan olika WebGL-teknologier.

I ett längre perspektiv skulle ett mer ordentligt system kunna skapas som utifrån ett flertal olika dataset från olika källor kan rendera visualiseringar. Det går då att visa med större säkerhet att det finns en skillnad mellan hur de olika teknologierna ur ett tidsperspektiv

visualiserar geospatial data. Flera olika WebGL-teknologier skulle kunna undersökas för att få en ännu bättre och tydligare insikt av vilken teknologi som är mest lämpat för ändamålet.

På lång sikt kan ett väletablerat system skapas med riktiga användare som använder systemet i sina yrken eller på fritiden. Mätningar kan genomföras under lång tid utifrån dataset från en stor databas som hela tiden uppdateras med ny och relevant data. En egen användbarhetsstudie skulle kunna skapas som inte bygger på andras verk utan utgår ifrån en stor mängd av riktiga användare. Andra teknologier än enbart de baserade på WebGL skulle kunna inkluderas i studien för att på så sätt undersöka olika alternativ för visualisering.

Referenser

Abran, A., Khelifi, A., Suryn, W., & Seffah, A. (2003). Usability meanings and interpretations in ISO standards. *Software quality journal*, *11*(4), 325-338.

Adair, M., Greenwood, R., Richard, D., Irons, S., Terral, O., & Metcalf, C. (2014). Proj4js. http://proj4js.org [2018-04-14]

Ahire, A. L., Evans, A., & Blat, J. (2015, June). Animation on the web: a survey. In *Proceedings of the 20th International Conference on 3D Web Technology* (pp. 249-257). ACM.

Anttonen, M., & Salminen, A. (2011). Building 3d webgl applications. Tampereen teknillinen yliopisto. Ohjelmistotekniikan laitos. Raportti-Tampere University of Technology. Department of Software Systems. Report; 16.

Berndtsson, M., Hansson, J., Olsson, B., & Lundell, B. (2007). *Thesis projects: a guide for students in computer science and information systems*. Springer Science & Business Media.

Boya, D. (2017). Interaktionsanalys av responsiva webbplatser med heatmaps och heatelements.

Butcher, P. W., & Ritsos, P. D. (2017, September). Building immersive data visualizations for the web. In *Proceedings of International Conference on Cyberworlds* (CW17).

Di Cerbo, F., Dodero, G., & Papaleo, L. (2010, July). Integrating a Web3D interface into an e-learning platform. In *Proceedings of the 15th International Conference on Web 3D Technology* (pp. 83-92). ACM.

Eichinski, P., & Roe, P. (2014, July). Heat maps for aggregating bioacoustic annotations. In *Information Visualisation (IV), 2014 18th International Conference on* (pp. 88-93). IEEE.

Evans, A., Romeo, M., Bahrehmand, A., Agenjo, J., & Blat, J. (2014). 3D graphics on the web: A survey. Computers & Graphics, 41, 43-61.

Ha, Y. U., Jin, J. H., & Lee, M. J. (2015). Lets3D: A collaborative 3d editing tool based on cloud storage. *International Journal of Multimedia and Ubiquitous Engineering*, 10(9), 189-198.

Haara, A., Pykäläinen, J., Tolvanen, A., & Kurttila, M. (2018). Use of interactive data visualization in multi-objective forest planning. *Journal of environmental management*, 210, 71-86.

James, J. A., Moh, T. S., & Edwards, C. A. (2016, October). Web-Based Visualization of Marine Environmental Data: Performance Analysis of a MatPlotLib Implementation. In *Collaboration Technologies and Systems (CTS), 2016 International Conference on* (pp. 288-293). IEEE.

ICES Oceanographic database (2018). OCEANOGRAPHY. http://ocean.ices.dk/HydChem [Hämtad 2018-04-25].

Kaistinen, M. (2015). Open source solutions for multiplatform graphics programming.

Komninos, A., Besharat, J., Ferreira, D., & Garofalakis, J. (2013, December). HotCity: enhancing ubiquitous maps with social context heatmaps. In *Proceedings of the 12th International Conference on Mobile and Ubiquitous Multimedia* (pp. 52). ACM.

Krämer, M., & Gutbell, R. (2015, June). A case study on 3D geospatial applications in the Web using state-of-the-art WebGL frameworks. In *Proceedings of the 20th International Conference on 3D Web Technology* (pp. 189-197). ACM.

Książek, M., & Pietruszka, M. (2012). Interactive 3D architectural visualization with semantics in web browers.

Lamberti, F., & Paravati, G. (2015, June). VDHM: Viewport-DOM Based Heat Maps as a Tool for Visually Aggregating Web Users' Interaction Data from Mobile and Heterogeneous Devices. In *Mobile Services (MS), 2015 IEEE International Conference on* (pp. 33-40). IEEE.

Lin, J. C. C., & Lu, H. (2000). Towards an understanding of the behavioural intention to use a web site. *International journal of information management*, 20(3), 197-208.

Prieto, I., Izkara, J. L., & del Hoyo, F. J. D. (2012, June). Efficient visualization of the geometric information of CityGML: application for the documentation of built heritage. In *International Conference on Computational Science and Its Applications* (pp. 529-544). Springer, Berlin, Heidelberg.

Rasys, E., Hodds, M., Dawood, N., & Kassem, M. (2014, January). A Web3D enabled information integration framework for facility management. In *Australasian Journal of Construction Economics and Building-Conference Series* (Vol. 2, No. 1, pp. 1-12).

Pryke, A., Mostaghim, S., & Nazemi, A. (2007, March). Heatmap visualization of population based multi objective algorithms. In *International Conference on Evolutionary Multi-Criterion Optimization* (pp. 361-375). Springer, Berlin, Heidelberg.

Rajamony, R., & Elmootazbellah (Mootaz) Elnozahy. (2001, March). Measuring Client-Perceived Response Time on the WWW. In *USITS*.

Resch, B., Wohlfahrt, R., & Wosniok, C. (2014). Web-based 4D visualization of marine geo-data using WebGL. *Cartography and Geographic Information Science*, 41(3), 235-247.

Ressler, S., & Leber, K. (2013, November). Web Based 3D Visualization and Interaction for Whole Body Laser Scans. In *Proc. of 4th Int. Conf. on 3D Body Scanning Technologies, Long Beach CA, USA* (pp. 166-172).

Havs- och vattenmyndigheten och SMHI. (2018). *Marina miljöövervakningsdata*. https://www.smhi.se/klimatdata/oceanografi/havsmiljodata/marina-miljoovervakningsdata [2018-04-16].

Sharakhov, N., Polys, N., & Sforza, P. (2013, November). GeoSpy: a Web3D platform for geospatial visualization. In *Proceedings of the 1st ACM SIGSPATIAL International Workshop on MapInteraction* (pp. 30-35). ACM.

Sopan, A., Noh, A. S. I., Karol, S., Rosenfeld, P., Lee, G., & Shneiderman, B. (2012). Community Health Map: A geospatial and multivariate data visualization tool for public health datasets. *Government Information Quarterly*, 29(2), 223-234.

Sun, N. (2017). *heatcanvas*. GitHub. https://github.com/sunng87/heatcanvas [2018-04-12].

Three.js (2018). http://threejs.org [2018-04-11].

Unar, K. S., Unar, A. M., & Patoli, Z. (2016, August). An interactive system for visualisation of cultural heritage objects of sindh in a web-based environment. In Innovative Computing Technology (INTECH), 2016 Sixth International Conference on(pp. 650-655). IEEE.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. (2012). *Experimentation in software engineering*. Springer Science & Business Media.

Zhang, C., & Mao, B. (2017, August). Distributed Processing Practice of the 3D City Model Based on HBase. In *Advanced Cloud and Big Data (CBD), 2017 Fifth International Conference on* (pp. 159-163). IEEE.

Appendix A – threejs_heatmap.js

```
'use strict';
var scene = new THREE.Scene();

var width = 600;
var height = 600;
var near = 0;
var far = 5;
var camera = new THREE.OrthographicCamera(width / - 2, width / 2, height /
2, height / - 2, near, far);

var renderer = new THREE.WebGLRenderer({alpha: true});
renderer.setSize(600, 600);
document.body.appendChild(renderer.domElement);

// Variables for timing the rendering
var start;
var stop;
var result; // The difference between start and stop

// Timer. Calculate the rendering time
function timer(time) {
  if (time == start) {
      start = Date.now();
  } else if (time == stop) {
      stop = Date.now();
      result = stop - start;

      // Save rendering time to localStorage
      var renderedTimes = [];
      var getTimes;
      if(Array.isArray(JSON.parse(localStorage.getItem("time")))) {
          getTimes = JSON.parse(localStorage.getItem("time"));
          renderedTimes = renderedTimes.concat(getTimes);
      }
      renderedTimes.push(result);
      localStorage.setItem("time", JSON.stringify(renderedTimes));

      console.log(JSON.parse(localStorage.getItem("time")));
  } else {
      console.log("Timer was not set correctly");
  }
}

// Get the data from the json file
var xmlhttp = new XMLHttpRequest();
```

```
xmlhttp.onreadystatechange = function() {
  if (this.readyState == 4 && this.status == 200) {
      var jsonData = JSON.parse(this.responseText);

      // Objects to hold coordinates and relevant values
      var coordinates = {};
      var dotCoord = {};

      var dotQuantity = 0;

      // Source coordinates will be in Longitude/Latitude, WGS84
      var source = new proj4('EPSG:4326');

      // Destination coordinates in meters, global spherical mercators
projection
      var dest = new proj4('EPSG:3785');

      function prepareCoordinates() {
          for(dotQuantity; dotQuantity < 30000; dotQuantity++){
              var coords = newCoordinate();
              var point = convertLatlon(coords.lat, coords.lon);
              point.value = coords.value;
              addID(point);
          }
      }

      function newCoordinate(){
          // Replace any commas with a dot to be able to render the
coordinate
              var lat = jsonData[dotQuantity]["Latitude [degrees_north]"];
              var lon = jsonData[dotQuantity]["Longitude [degrees_east]"];
              var measurementValue = jsonData[dotQuantity]["PSAL [psu]"];
              // Makes value to a number if it is a string
              if(typeof measurementValue == "string"){
                  measurementValue = Number(jsonData[dotQuantity]["PSAL
[psu]"]);
              }
              return {lat: lat, lon: lon, value: measurementValue};
      }

      // Returns a point with x and y coordinates in meters
      function convertLatlon(lat, lon) {
          var point = proj4.toPoint([lon, lat, 0.0]);

          return proj4.transform(source.oProj, dest.oProj, point);
      }

      function addID(point) { // Adds the coordinates and a value to an
```

```
array
          var id = ((point.x + point.y) % jsonData.length) + "";
          var value = point.value;
          // x and y coordinates in 10km instead of meters
          var x = point.x / 10000;
          var y = point.y / 10000;

          // Move the coordinates closer to origo
          var movedX = x - 30;
          var movedY = y - 780;

          // If coordinates has property of id, average the value
          if(coordinates[id]) {
              coordinates[id].value = (value + coordinates[id].value) / 2;
          } else {
              // Create new coordinate
              coordinates[id] = {value : value, x : movedX, y : movedY};
          }
      }

      function dotSystem(){
          var canWidth = 300;
          var canHeight = 300;
          var gridSize = 4;

          for(var id in coordinates) {
              for(var x = -canWidth; x < canWidth; x += gridSize) {
                  for(var y = -canHeight; y < canHeight; y += gridSize) {
                      if(x <= coordinates[id].x && x + gridSize >=
coordinates[id].x && y <= coordinates[id].y && y + gridSize >=
coordinates[id].y) {
                          var dotID = x * y;
                          var value = coordinates[id].value;

                          if(dotCoord[dotID]) { // Make a average of the
new and old value for every new value
                              dotCoord[dotID].value = (value +
dotCoord[dotID].value) / 2;
                          } else { // To get a more accurate x and y
coordinate, half of the gridSize is added
                              dotCoord[dotID] = {value : value, x : x +
(gridSize/2), y : y + (gridSize/2)};
                          }
                      }
                  }
              }
          }
          createDot();
```

```
    }

    var dotGeometry = new THREE.Geometry();
    dotGeometry.vertices.push(new THREE.Vector3( 0, 0, 0));
    function createDot(){
        // Loop through all values and return the number with the
highest value
        var max = 0;
        for(var i in dotCoord) {
            max = Math.max(dotCoord[i].value, max);
        }

        // Loop through all values and apply RGBA colors
        for(var id in dotCoord) {
            // Convert the value to rgba colors
            var rgba = value2rgba(dotCoord[id].value / max);

            var dotMaterial = new THREE.PointsMaterial({size: 4,
sizeAttenuation: false, transparent: true});
            var dot = new THREE.Points(dotGeometry, dotMaterial);
            dot.material.color.setRGB(rgba[0], rgba[1], rgba[2]);

            dot.position.x = dotCoord[id].x;
            dot.position.y = dotCoord[id].y;
            scene.add(dot);
        }
    }

    function value2rgba(value) {
        // define rgb variables
        var r, g, b;
        var h = 1 - value;
        var l = 1 - value * 0.55;
        // convert hue to rgb
        function hue2rgb(p, q, t) {
            if(t < 0) t++;
            if(t > 1) t--;
            if(t < 1 / 6) return p + (q - p) * 6 * t;
            if(t < 1 / 2) return q;
            if(t < 2 / 3) return p + (q - p) * (2 / 3 - t) * 6;
            return p;
            }
        // ternary operator
        var q = l < 0.5 ? l * 2 : l + 1 - l;
        var p = 2 * l - q;
        r = hue2rgb(p, q, h + 1 / 3);
        g = hue2rgb(p, q, h);
        b = hue2rgb(p, q, h - 1 / 3);
```

```
            // return RGBA
            return [r, g, b, value];
        }

        function animate() {
            prepareCoordinates();
            dotSystem();
            console.log('NumberOfCoords: ' +
Object.keys(coordinates).length);
            renderer.render(scene, camera); // Render the scene
        }

        console.log(jsonData.length);
        // Runs the script again to collect a certain amount of data
        if(localStorage.getItem("time") != null){
            if(JSON.parse(localStorage.getItem("time")).length < 1) {
                timer(start); // Start animation render timer
                animate();
                timer(stop); // Stop and calculate the animation render time
                location.reload();
            } else {
                console.log(JSON.parse(localStorage.getItem("time")));
                var getResults = JSON.parse(localStorage.getItem("time"));
                document.getElementById("results").innerHTML = getResults;
            }
        } else {
            timer(start); // Start animation render timer
            animate();
            timer(stop); // Stop and calculate the animation render time
            location.reload();
        }
    }
};
xmlhttp.open("GET", "../Dataset/icesData.json", true);
xmlhttp.send();
```

Appendix B – threejs_heatmap.html

```
<!DOCTYPE html>
<html>
   <head>
       <meta charset=utf-8>
       <title>Heatmap - Three.js</title>
       <style>
           body { margin: 0; }
           canvas { width: 100%; height: 100%; }
           img {width: 600px; height: 427px; position: absolute;  z-index:
-1; margin-left: 0px; margin-top: 60px;}
       </style>
   </head>
   <body>
       <img src="../backgroundMap.png" alt="Map of the north sea”
title="Published by OpenStreetMap under CC BY-SA 2.0
https://www.openstreetmap.org">
       <script src="https://code.jquery.com/jquery-2.2.4.min.js"></script>
       <script src="js/build/three.js"></script>
       <script type='text/javascript'
src='https://cdnjs.cloudflare.com/ajax/libs/proj4js/2.4.3/proj4-
src.js'></script>
       <script src="threejs_heatmap.js"></script>
       <p id='results'></p>
   </body>
</html>
```

Appendix C – x3dom.js

```
'use strict';

// Variables for timing the rendering
var start;
var stop;
var result; // The difference between start and stop

// Timer. Calculate the rendering time
function timer(time) {
  if (time == start) {
      start = Date.now();
  } else if (time == stop) {
      stop = Date.now();
      result = stop - start;

      // Save rendering time
      var renderedTimes = [];
      var getTimes;
      if(Array.isArray(JSON.parse(localStorage.getItem("time")))) {
          getTimes = JSON.parse(localStorage.getItem("time"));
          renderedTimes = renderedTimes.concat(getTimes);
      }
      renderedTimes.push(result);
      localStorage.setItem("time", JSON.stringify(renderedTimes));

      console.log(JSON.parse(localStorage.getItem("time")));
  } else {
      console.log("Timer was not set correctly");
  }
}

// Get the data from the json file
var xmlhttp = new XMLHttpRequest();
xmlhttp.onreadystatechange = function() {
  if (this.readyState == 4 && this.status == 200) {
      var jsonData = JSON.parse(this.responseText);

      // Objects to hold coordinates and relevant values
      var coordinates = {};
      var dotCoord = {};

      var dotQuantity = 0;

      // Source coordinates will be in Longitude/Latitude, WGS84
      var source = new proj4('EPSG:4326');
```

```
      // Destination coordinates in meters, global spherical mercators
projection
      var dest = new proj4('EPSG:3785');

      function prepareCoordinates() {
          for(dotQuantity; dotQuantity < 30000; dotQuantity++){
              var coords = newCoordinate();
              var point = convertLatlon(coords.lat, coords.lon);
              point.value = coords.value;
              addID(point);
          }
      }

      function newCoordinate(){
          // Replace any commas with a dot to be able to render the
coordinate
              var lat = jsonData[dotQuantity]["Latitude [degrees_north]"];
              var lon = jsonData[dotQuantity]["Longitude [degrees_east]"];
              var measurementValue = jsonData[dotQuantity]["PSAL [psu]"];
              // Makes value to a number if it is a string
              if(typeof measurementValue == "string"){
                  measurementValue = Number(jsonData[dotQuantity]["PSAL
[psu]"]);
              }
              return {lat: lat, lon: lon, value: measurementValue};
      }

      // Returns a point with x and y coordinates in meters
      function convertLatlon(lat, lon) {
          var point = proj4.toPoint([lon, lat, 0.0]);
          return proj4.transform(source.oProj, dest.oProj, point);
      }

      function addID(point) { // Adds the coordinates and a value to an
array
          var id = ((point.x + point.y) % jsonData.length) + "";
          var value = point.value;
          // x and y coordinates in 10km instead of meters
          var x = point.x / 10000;
          var y = point.y / 10000;

          // Move the coordinates closer to origo
          var movedX = x - 30;
          var movedY = y - 780;

          // If coordinates has property of id, average the value
          if(coordinates[id]) {
```

```
            coordinates[id].value = (value + coordinates[id].value) / 2;
        } else {
            // Create new coordinate
            coordinates[id] = {value : value, x : movedX, y : movedY};
        }
    }

    function dotSystem(){
        var canWidth = 300;
        var canHeight = 300;
        var gridSize = 4;

        for(var id in coordinates) {
            for(var x = -canWidth; x < canWidth; x += gridSize) {
                for(var y = -canHeight; y < canHeight; y += gridSize) {
                    if(x <= coordinates[id].x && x + gridSize >=
coordinates[id].x && y <= coordinates[id].y && y + gridSize >=
coordinates[id].y) {
                        var dotID = x * y;
                        var value = coordinates[id].value;

                        if(dotCoord[dotID]) { // Make a average of the
new and old value for every new value
                            dotCoord[dotID].value = (value +
dotCoord[dotID].value) / 2;
                        } else { // To get a more accurate x and y
coordinate, half of the gridSize is added
                            dotCoord[dotID] = {value : value, x : x +
(gridSize/2), y : y + (gridSize/2)};
                        }
                    }
                }
            }
        }
        createDot();
    }

    function createDot(){
        // Loop through all values and return the number with the
highest value
        var max = 0;
        for(var i in dotCoord) {
            max = Math.max(dotCoord[i].value, max);
        }

        // Loop through all values and apply RGBA colors
        for(var id in dotCoord) {
            // Convert the value to rgba colors
```

```
            var rgba = value2rgba(dotCoord[id].value / max);
            var x = dotCoord[id].x;
            var y = dotCoord[id].y;
            addNode(x, y, rgba);
        }
    }

    function addNode(x, y, color) {
        var transform = document.createElement('Transform');
        transform.setAttribute("translation", x + ' ' + y + ' ' + '0' );

        var shape = document.createElement('shape');

        var app = document.createElement('Appearance');
        var mat = document.createElement('Material');
        mat.setAttribute('diffuseColor', color[0] + ' ' + color[1] + ' '
+ color[2]);
        app.appendChild(mat);
        shape.appendChild(app);

        transform.appendChild(shape);

        var node = document.createElement('indexedFaceSet');
        node.setAttribute('coordIndex', '0 1 2 3 -1');

        var coord = document.createElement('coordinate');
        coord.setAttribute('point', '0 0 0, 4 0 0, 4 4 0, 0 4 0');

        node.appendChild(coord);
        shape.appendChild(node);

        var scene = document.getElementById('scene');
        scene.append(transform);
    }

    function value2rgba(value) {
        // define rgb variables
        var r, g, b;
        var h = 1 - value;
        var l = 1 - value * 0.55;
        // convert hue to rgb
        function hue2rgb(p, q, t) {
            if(t < 0) t++;
            if(t > 1) t--;
            if(t < 1 / 6) return p + (q - p) * 6 * t;
            if(t < 1 / 2) return q;
            if(t < 2 / 3) return p + (q - p) * (2 / 3 - t) * 6;
            return p;
```

```
            }
        // ternary operator
        var q = l < 0.5 ? l * 2 : l + 1 - l;
        var p = 2 * l - q;
        r = hue2rgb(p, q, h + 1 / 3);
        g = hue2rgb(p, q, h);
        b = hue2rgb(p, q, h - 1 / 3);
        // return RGBA
        return [r, g, b, value];
    }

    function animate() {
        prepareCoordinates();
        dotSystem();
    }

    // Runs the script again to collect a certain amount of data
    if(localStorage.getItem("time") != null){
        if(JSON.parse(localStorage.getItem("time")).length < 5000) {
            timer(start); // Start animation render timer
            animate();
            timer(stop); // Stop and calculate the animation render time
            location.reload();
        } else {
            console.log(JSON.parse(localStorage.getItem("time")));
            var getResults = JSON.parse(localStorage.getItem("time"));
            document.getElementById("results").innerHTML = getResults;
        }
    } else {
        timer(start); // Start animation render timer
        animate();
        timer(stop); // Stop and calculate the animation render time
        location.reload();
    }
  }
};
xmlhttp.open("GET", "../Dataset/icesData.json", true);
xmlhttp.send();
```

Appendix D – x3dom.html

```
<html>
  <head>
      <title>Heatmap - X3DOM</title>

      <script type='text/javascript'
src='http://www.x3dom.org/download/x3dom.js'></script>
       <script type='text/javascript'
src='https://cdnjs.cloudflare.com/ajax/libs/proj4js/2.4.3/proj4-
src.js'></script>
      <script type='text/javascript' src='x3dom.js'></script>
      <link rel='stylesheet' type='text/css'
href='http://www.x3dom.org/download/x3dom.css'></link>
      <style>
          img {width: 600px; height: 427px; position: absolute;  z-index:
-1; margin-left: 0px; margin-top: 60px;}
      </style>
  </head>
  <body>
      <img src="../backgroundMap.png" alt="Map of the north sea"
title="Published by OpenStreetMap under CC BY-SA 2.0
https://www.openstreetmap.org">
      <x3d id='x3d' width='600px' height='600px'>
          <scene id='scene'>
              <OrthoViewpoint DEF='v1' orientation='0 0 0 1' position='0 0
2' fieldOfView='0 0 0 600'></OrthoViewpoint>
          </scene>
      </x3d>
      <p id='results'></p>
  </body>
</html>
```