WebGL:baserad ramverk prestandajämförelse
Mellan Three.Js och Babylon.Js

WebGL: based framework performance comparison
between Three.Js and Babylon.Js

Examensarbete inom huvudområdet Datalogi
Grundnivå 30 högskolepoäng
Vårtermin 2016

David Frisk

Handledare: Mikael Lebram
Examinator: Henrik Gustavsson
Sammanfattning

Webben har utvecklats mycket de senaste åren och blir mer och mer dynamisk. Nya tekniker kommer och går och nya sätt att skapa webbsidor blir allt fler, så länge tekniken förbättras och nya idéer uppstår. Efter 2000-talet så kan det nu skapas bättre 3D interaktiv datorgrafik inom webbläsare, men frågan är dock varför det är så få webbsidor som har 3D-animation. Är det inte 3D-grafik som är framtiden och det frågas ofta varför 2D ofta används vid skapande av webbsidor.

Denna studie går ut på att kartlägga och jämföra olika ramverk för att se vilken kan ge bäst prestanda för att kunna locka utvecklare till att använda 3D-rendering inom webbvärlden. Undersökningen kommer bland annat att redovisa upp bakgrund, metod, problem till varför 3D används så sällan inom webben samt lösning där utvärdering kommer redovisa resultat från mätningarna samt gå på etiska aspekternas nytta av arbetet inför framtida forskning.

**Nyckelord:** WebGL, Three.js, Babylon.js, JavaScript, HTML5, ramverk
Innehållsförteckning

1 Introduktion .............................................................................................................. 1

2 Bakgrund .................................................................................................................. 2
   2.1 JavaScript ........................................................................................................... 2
   2.2 HTML5 .............................................................................................................. 2
       2.2.1 Vad betyder HTML5 för dig ........................................................................ 3
   2.3 WebGL ............................................................................................................... 3
       2.3.1 Goo Engine ................................................................................................. 4
       2.3.2 Blend4web .................................................................................................. 4
       2.3.3 PlayCanvas ................................................................................................ 4
       2.3.4 Copperlicht ............................................................................................... 5
       2.3.5 Three.js .................................................................................................... 5
       2.3.6 Unity ......................................................................................................... 6
       2.3.7 Babylon.js ................................................................................................ 6
       2.3.8 Turbulenz .................................................................................................. 7
   2.4 Valt av ramverk : Three vs Babylon ................................................................... 7

3 Problemformulering ................................................................................................ 9
   3.1 Metodbeskrivning ............................................................................................... 10
   3.2 Metoddiskussion ............................................................................................... 11
   3.3 Hypotes ............................................................................................................. 12
   3.4 Alternativa metoder ......................................................................................... 12
   3.5 Etiska frågeställningar ..................................................................................... 12

4 Genomförande / implementation ............................................................................ 13
   4.1 Förstudie .......................................................................................................... 13
   4.2 Hårdvarukomponenter ...................................................................................... 13
       4.2.1 Stationär datorn .......................................................................................... 13
       4.2.2 Bärbar ....................................................................................................... 14
   4.3 Mjukvarukomponenter ...................................................................................... 14
   4.4 Scen-rendering .................................................................................................. 14
   4.5 Pilotstudie ......................................................................................................... 15
       4.5.1 Three.js objekt-rendering .......................................................................... 16
       4.5.2 Babylon.js objekt-rendering ..................................................................... 18

5 Utvärdering ............................................................................................................... 20
   5.1 Presentation av undersökning .......................................................................... 20
   5.2 Analys ............................................................................................................... 22
   5.3 Slutsatser ......................................................................................................... 29

6 Avslutande diskussion ............................................................................................. 32
   6.1 Sammanfattning ............................................................................................... 32
   6.2 Diskussion ......................................................................................................... 32
       6.2.1 Forskningsetiska aspekter hos arbetet eller undersökningsmetoden .......... 34
       6.2.2 Samhällelig nyttan hos arbetet ................................................................. 34
   6.3 Framtida arbete ............................................................................................... 35

7 Referenser ................................................................................................................. 36

8 Appendix A Första version av mätvärde ................................................................. 39
Appendix B Three.js testapplikation .......................................................... 43
Appendix C Babylon.js testapplikation ....................................................... 47
1 Introduktion

Om en användare surfar ute på nätet så måste det frågas hur ofta den upptäcker eller besöker 3D-baserade webbsidor. Furukawa & Fukumoto (2012) menar att det största skälet till varför det är så få webbsidor som har 3D-innehåll är bland annat svårigheten med att skapa 3D-data och saknaden av en standardiserad 3D-visning, vilket kan förenklas genom den nya teknik som släppts. År efter år har många webbutvecklare försökt leta upp nya sätt att utveckla produkter och det kan vara allt ifrån nya program, små web-plugins, lära sig nya funktioner och kanske till och med programmeringsspråk.


Deras studie var korrekt i sin förutsägelse om de skulle göra en undersökning i dagens IT-användning inom mjukvaror och open source program. För några år sedan så hade många program som kom ifrån exempel Adobe, Microsoft eller något motsvarande, krävt stort utrymme medan andra open source och fria program såsom WebGL inte haft behov av stora mängder utrymme eller uppdateringar.


Frågeställningen som skall besvaras är vilken skillnad det är mellan de olika ramverken som är WebGL-baserade och vilken utav dem som är rekommenderad för webbutvecklare att använda sig av för att kunna få ut hög uppdateringsfrekvens och låg exekveringstid ifrån FPS. Experimentets mätresultat kunde ge svar på att hypotesen höll och denna undersökningen diskuterar också framtida visioner utifrån de resultat som redovisas.
2 Bakgrund


2.1 JavaScript


Mehrara (2011) påpekar att JavaScript har blivit en standard för webben, där hela 95% av alla som surfar har JavaScript påslaget och kommer mer och mer till användning trots dess brister. Mehrara (2011) menar också att JavaScript är långsamt på grund av att det aldrig riktigt har behandlats som ett flerårigt språk, vilket begränsar möjligheten att utnyttja ett flerkännt system.

2.2 HTML5


HTML5 har några fördelar:

- Mindre beroende av plugins.
- Skript ersättas med markup när det är möjligt.
- Offentlig utvecklingsprocess så att folk kan se vad som händer.

Här är några exempel på HTML5 markup taggar som HTML saknades:

- `<header>` och `<footer>` är till för att hjälpa användare att isolera topp och botten i innehållsblock.
- `<nav>` är till för att ange vilka delar som bör övervägas som navigationsblock.
- `<canvas>` är en tagg som låter användare rita grafik med ett separat skriptspråk.
- `<article>` identifierar en specifik del av innehållet, exempelvis ett blogginlägg eller en kommentar.

2.2.1 Vad betyder HTML5 för dig


2.3 WebGL


<table>
<thead>
<tr>
<th>Name</th>
<th>Script</th>
<th>Animation</th>
<th>Audio</th>
<th>Physics</th>
<th>Liensce</th>
<th>Notes and references</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geo Engine</td>
<td>JavaScript</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Freenium or commercial</td>
<td>Game engine with a browser-based editor</td>
</tr>
<tr>
<td>Blend4web</td>
<td>JavaScript</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>GPLv3 or commercial</td>
<td>Authored in the 3D modelling with Blender</td>
</tr>
<tr>
<td>PlayCanvas</td>
<td>JavaScript</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>MIT (engine)</td>
<td>Open-source 3D game engine</td>
</tr>
<tr>
<td>Copperlight</td>
<td>JavaScript</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>based on zlib</td>
<td>Interactive 3D applications using WebGL</td>
</tr>
<tr>
<td>Three</td>
<td>JavaScript</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>MIT</td>
<td>Animated 3D graphics on a Web browser</td>
</tr>
<tr>
<td>Utility</td>
<td>UnityScript / C#</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Proprietary</td>
<td>Offers a WebGL build option since version 5</td>
</tr>
<tr>
<td>Babylon</td>
<td>JavaScript / TypeScript</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Apache License 2.0</td>
<td>JavaScript framework for building 3D games</td>
</tr>
<tr>
<td>Turbulenz</td>
<td>JavaScript</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>MIT</td>
<td>Open source HTML5 game engine</td>
</tr>
</tbody>
</table>

Figur 1   Tabell av ramverk som tas upp under experimentet.
2.3.1 Goo Engine
Goo Engine är en kraftfull WebGL spelmotor som är kopplad till Goo Create, en webbläsarbaserad editor. Det finns verktyg i utvecklingsmiljön som Goo Create erbjuder enkel ”drag and drop” funktion för att kunna lägga till interaktivitet samt importera 3D modeller (Hestman, 2015).

Goo Engine fördelar:
- Fysik och kollisionsmotorer.
- En väl anpassa utvecklingsmiljö.

Goo Engine nackdelar:
- Inte en open source.
- Endast fri att använda under begränsad period.

2.3.2 Blend4web

Blend4web fördelar:
- Open source.
- Har några snygga effektfunktioner.
- Bra dokumentation.

Blend4web nackdelar:
- Att den främst använder Blender.

2.3.3 PlayCanvas

PlayCanvas fördelar:
- Open source under MIT licens.
- Fysik och kollisionsmotorer.
- Stöd för realtidsredigering.

PlayCanvas nackdelar:
- Inga privata projekt i den fria version.
- Inte lika aktiv community som andra ramverk.

2.3.4 Copperlicht

Copperlicht fördelar:
- Open source.
- Har inbyggda fysik och kollisionsmotorer.
- Stöd för animering i ramverket.
- Bra dokumentation.

Copperlicht nackdelar:
- 3D redigeringsverktyget "CopperCube" är inte gratis.
- Har inte den mest aktiva communityn.

2.3.5 Three.js

Three.js fördelar: (Hestman, 2015)
- Three ger användaren en färdig scen, kamera, ljus och andra funktioner, vilket kan minska tid för programmering så projektet bli färdigt snabbare.
- För att skapa ett enkelt objekt så krävs det lite programmering.
- Three kan rendera canvas 2D, WebGL och SVG samt importera modeller från 3D modellerings applikationer.
- Bra dokumentation och stöd av community.
- Fri och open source under MIT licens.

Three.js nackdelar:
- Är ett äldre ramverk.
- Fortfarande i alpha-version och förändras ofta.
- Den har ingen fysik eller kollisionsmotor.

2.3.6 Unity
Unity:s spelmotor och en integrerad utvecklingsmiljö (IDE) är bland av de mest använda ramverken i marknaden idag. Den finns i flertal plattformer, inklusive stationära datorer, spelkonsoler och mobila enheter. Unity är en otroligt kraftfull och interaktivt utvecklingsverktyg som skapades i 2005 (Harshfield, 2015).

Unity fördelar: (Hestman, 2015).
- Lätt att lära sig och använda.
- Har brett community stöd.
- Fysik och kollisionsmotor.

Unity nackdelar:
- Unity Pro är dyrt
- Den fria versionen har inte alla funktioner.
- Är inte open source.
- Dokumentation saknas och är inaktuell för vissa funktioner.

2.3.7 Babylon.js

Babylon.js fördelar: (Hestman, 2015)
- Babylon är modern, ren, fri och open source.
- Den har liknande funktioner som Three.js.
- Den är mycket enklare att använda och har ett bra community stöd.
- Babylon är bra på att göra enkla animerade saker.
- För spel inom design och struktur områden.
- Babylon:s community är aktiv.
- Har ett företag bakom sig.

Babylon.js nackdelar:
- Babylon:s styrka ligger framförallt i att designa spel, vilket minskar stödet för andra områden som webb och film.
- Är ett mer målinrikt tillvägagångssätt för webbaserad spelutveckling.
- Importerade scener måste omvandlas till en fil utav Babylon:s filformat.

2.3.8 Turbulenz
Turbulenz är en 3D-spelmotor i JavaScript som körs direkt i webbläsaren. Motorn har stöd för kollision och begränsningar samt att den har stor samling av inbyggda animations styrenheter (Hestman, 2015).

Turbulenz fördelar:
- Open source under MIT licens.
- Lätt att kombinera Turbulenz med extern teknik.

Turbulenz nackdelar:
- Inte den mest aktiva communityn.

<table>
<thead>
<tr>
<th>Ramverk</th>
<th>Sökträffar i Google</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geo Engine</td>
<td>543 000</td>
</tr>
<tr>
<td>Blend4web</td>
<td>47 400</td>
</tr>
<tr>
<td>playcanvas</td>
<td>216 000</td>
</tr>
<tr>
<td>copperlicht</td>
<td>9 090</td>
</tr>
<tr>
<td>Three</td>
<td>475 000 000</td>
</tr>
<tr>
<td>babylon</td>
<td>724 000</td>
</tr>
<tr>
<td>unity</td>
<td>3 520 000</td>
</tr>
<tr>
<td>turbulenz</td>
<td>36 300</td>
</tr>
</tbody>
</table>

Figur 2   Antal sökträffar i Google för de mest vanligaste ramverk.

2.4 Valt av ramverk: Three vs Babylon
Figur 3 är en scen med flertalet av objekt i rörelse när programmet renderar. Det är vad detta arbete går ut på, nämligen att bevisa vilken av de två tekniker av WebGL som fungerar bäst och kan ge oss utvecklare bäst prestanda hos en websida.

**Figur 3  Kub i rörelse från Three.js**


**Figur 4  Partikel system från Babylon.js**
3 Problemformulering


Utifrån teknikernas skillnader ställs frågan, vilket av ramverken ska webbutvecklare välja för att få ut det bästa resultatet och prestanda för ett projekt? Vad är det som gör Three till ett bättre val än Babylon eller tvärtom? Är det beroende på vad som ska göras i projektet? Som tidigare nämnts visar statistiken att Three kan skapa objekt på kortare tid och kräver mindre programmering medan Babylon är ett nyare och modernare ramverk som erbjuder samma funktionalitet som Three men främst riktar sig mot spel. Finns det en chans att Babylon.js som är ett ramverk för spelmotorer skulle kunna påverka resultatet, och i så fall på vilket sätt?


Det kommer bli mycket fascinerande att se hur många objekt datorn klarar av att objekt-rendera. Målet är att komma upp till minst 5000 objekt och att FPS:en ligger på ungefär 50 FPS. Detta hänger mycket på komponenter i datorn och hur komplex koden är. En annan utmaning är att försöka komma ner till 1 FPS och se hur många objekt som krävs för det. En följdfråga är om resultatet skulle påverkas om det läggs till texturer på objekten, vilket är tanken att försöka belasta arbetskapskapaciteten på både CPU och GPU så mycket som möjligt.

Frame rate är i detta fall ett huvudsål som skall besvaras och många tankar kretsar omkring denna frågeställning. En sak som skall testas är bland annat om storleken på scenen har någon betydelse, som exempelvis 10 x 20 meter scen eller en 5 x 10 meter scen som renderar minst 5000 objekt med texturer kan påverka resultat i rendering eller när kod-filen skall exekveras. Det vill säga om scen-storleken eller storleken på skript-filen kan få CPU och GPU att arbeta mycket hårdare än förväntat.


3.1 Metodbeskrivning


När allt är färdigt, det vill säga att "index.html" filen som innehåller alla skript och sidan kan rendera ut objekt så ska de överföras till experimentens lokala server (Wampserver) där den ska placeras för exekvering. Det är inte klart än om FPS-mätare skall finnas med i själva index.html filen eller vid sidan av. Efter att arbetet genomförts och implementationen är klar så är det dags för mätning. Detta kommer ske genom mätning av svarstider, där svarstider representeras utifrån hur lång tid det tar för filen att exekvera ifrån servern samt att mäta FPS, CPU:n och GPU:n användning under exekvering av filen.


3.2 Metoddiskussion


Vid denna stund så är det svårt att säga vilka typer av renderingar som ska genomföras, eller rent av svårt att säga vilka som överhuvudtaget faktiskt är genomförbara. Första tanken var att göra tre olika renderingar, där den första är att rendera ut bestämt antal av objekt utan några texturer eller mapping på en lagom stor scen. Andra gången blir det samma fast med texturer för att se om CPU, GPU eller RAM:n användning går långsammare, hur FPS:en flyter på samt svarstid. Sista renderingen kommer allt vara samma som i andra renderingen, allt förutom storleken på scenen. Frågan är om det finns möjlighet för till exempel en liten yta med hoptryckta objekt med texture som kan påverka prestandajämförelsen.
3.3 Hypotes

Experiments hypotes är att undersöka huruvida Babylon.js, ett ramverk för spelmotorer, kan ge ut bättre prestanda än Three som är ett brett ramverk inom både spel och webbdesign. Three använder mindre kod i sin programmering, vilket innebär en mindre belastning i koden som kan exekveras snabbare och flyter på bättre i svarstid. Chansen finns att CPU, GPU och RAM-användning inte kommer att behöva arbeta lika hårt om kod mängden är mindre, vilket skapar möjligheter för ett bättre resultat.

3.4 Alternativa metoder

En tänkbar alternativ metod till experimentet skulle kunna vara användning av testpersoner som testar ramverken. Användarstudie som observation och intervjuer skulle i sådana fall kunna göras på testpersoner som är webbutvecklare, då det är den målgruppen som undersökningen är riktad mot. En nackdel är att resultat som kommer ifrån testpersoner, i detta fall webbutvecklare, medför personliga åsikter om ramverken. Detta kan ha en stor påverkan på slutresultatet i hela studien, vilket är något som bör undvikas. Det har funnits tankar om att ramverken inte fungerar som de skall och då kan ett alternativ vara att byta ut ett ramverk mot ett annat, vilket sätter stopp för studien och kan leda till kostnader i form av både tid och resurser. Om en förändring av ramverk skulle ske så måste det alternativa ramverket vara så pass likt de andra två som möjligt för resultatets skull, det vill säga samma programmeringsspråk eller att de erbjuder samma funktionalitet. Det skulle också kunna använda sig av teoretisk studie genom att undersöka skillnaden mellan kodens algoritm och utvärdera det ett värde därifrån.

3.5 Etiska frågeställningar

För att denna undersökning skall kunna påvisa pålitliga mätningar och resultat behöver grundläggande faktorer redovisas om mjukvara och hårdvaruspecifikationer. Samt att redovisa den typ av faktor som ger möjlighet att experiment bli återanvändbar i både kod och resultat inför annan framtida forskning (Cai et.al, 1998). Inga testpersoner kommer att användas för detta experiment på grund av att det inte är lämpligt för att testa hypotesen. När experimentet är färdigt kommer allt som dokumenterats, det vill säga kod, resultat och slutsats att publiceras för allmänheten och information som användes under experimentet. Undersökningen kommer genomföras under en webbläsare som senare väljs ut och för att garantera möjligheten av återupprepbarhet, så kommer valet bli den mest använda webbläsare med den senaste versionen.

4 Genomförande / implementation

Detta kapitel kommer att beskriva ramverkens implementation och vägen till resultat. Det som skall tas upp är bland annat de första idéerna och tankarna om vad för ramverk som ska renderas ut till färdiga applikationer. Kapitlet inkluderar en pilotstudie som visar att mätningar som samlas in kan användas för att analysera resultatet.

4.1 Förstudie

En inspirationkälla var från början en artikel om "Crowd Simulation Rendering for Web" (Savoy & Cabral, 2015) där deras experiment gick ut på att testa animation och rendering av en stor mängd objekt i webbläsare. För att kunna testa hypotesen så utökade de grafikkortets kapacitet för att se om prestandan kunde bli bättre. Detta experiment går ut på att välja WebGL-baserade ramverk för att testa om hypotesen "Can en spelmotor som Babylon.js få ut bättre prestanda än Three som är ett brett ramverk i både spel- och webbdesign?".


4.2 Hårdvarukomponenter

Applikationsrendering testas ifrån stationära och bärbara datorer för att skapa förståelse för varför 3D-renderingen är så lite utbredt i webben. För att vara säker så ska undersökningen inte göras utifrån en enhet utan ifrån flera för att kunna stärka utvärderingen och få ut ett säkrare resultat i experimentet.

4.2.1 Stationär datorn

- Processor : Intel Core i5-3570K 3.40GHz, 4
- Grafikkort : NVIDIA GeForce GTX 960, 4GB
- RAM minne : Kingston DDR3 HyperX 1600MHz 12GB
- Operativsystem : Microsoft Windows 10, 64-bit version 10.0.10240
4.2.2 Bärbar

- Processor: Intel Celeron N2840 2.16 GHz Processor
- Grafikkort: Intel HD Graphics
- RAM minne: 4 GB DDR3L SDRAM
- Operativsystem: Microsoft Windows 8, 64 bit

4.3 Mjukvarukomponenter


4.4 Scen-rendering

Under implementationen så har antal mängder av ändringar skett och det sker främst på idéer om hur rendering skall renderas. Detta var ett stort problem, att hitta en lösning var inte lätt och mycket berodde på att lösningen baseras på vilka funktioner som ramverken använder och det fungerar på båda. Till exempel vilka sorts kameror som Babylon ska använda för att kunna hålla samma mått som Three använder. För att förstå vad detta experiment gick ut på är att först förstå själva idén som det var tänkt, att objektet skulle rotera runt sina koordinater (det vill säga y, x, z leden) i scenen när kamera filmar rörelser ifrån objektet. Det var också tänkt att renderingen skall få arbeta lite hårdare för att kunna se ett utslag i resultatet, så att något verkligen händer. Som andra experiment så finns den första körning, speciellt från Babylon som har några fascinerande situationer i sin tidiga utveckling.

![Babylon första prototyp rendering.](image)

Figur 5

Figur 6  
Jämförelse mellan Three MeshNormalMaterial och Babylon utan ljus.

En annan ändring som nästan infördes i experiment var en gemensam JavaScript-fil som innehåller en for-loop funktion som ska generera fram slumpmässiga positioner för objekt inom båda ramverken. Detta var en abstraktion som kunde fixa problemen med Babylons objekt rendering av den första prototypen, vilket inte hände.

4.5 Pilotstudie


Från början i genomförandet så var tänkt att kameran skulle ha rotationen och inte scenen (stack overflow, 2014), så betyder det att det bara är kameran som behöver rendera frame efter frame, där den går runt scenen och inte antalet av objekt. Detta var en av hypoteser som gällde för båda ramverken.
4.5.1 Three.js objekt-rendering


Figur 7    Fyra rendering med 2000 objekt i olika former.

<table>
<thead>
<tr>
<th>62-63 fps med 0 ms</th>
<th>60-63 fps med 0 ms</th>
</tr>
</thead>
<tbody>
<tr>
<td>62-63 fps med 0 ms</td>
<td>62-63 fps med 0 ms</td>
</tr>
</tbody>
</table>

Figur 8    Fyra rendering med 5000 objekt i olika former.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>17 fps med 0 ms</td>
<td>14-17 fps med 0 ms</td>
</tr>
<tr>
<td>7-10 fps med 0 ms</td>
<td>15-18 fps med 0 ms</td>
</tr>
</tbody>
</table>
4.5.2 Babylon.js objekt-rendering

Babylon.js har varit utmanande under experimentet när det gäller om tid och arbetskraft. Detta beror på att ramverket är nyare än Three.js och att den har funnits ute på marknaden under en kort tid, vilket är en av anledningarna till varför det skall kolla genom tutorial först, och få inspirationer ifrån. Detta gav också en känsla på Babylon.js att de är inte så aktiv som Three.js är idag, som har funnits i mer än sex år och fortfarande skapas det nya applikationer med Three.js. 


Figuur 9 lyckades rendera ut 2000 objekt, men nackdelen med Three och Babylon var att de inte använder samma skala på scen och storleken av objekt, vilket ledde till svårigheter vid matchning av ramverk.

I figur 10 så har antalet objekt ökat till 5000 och den lyckas rendera ut objekt, allt förutom den sista bilden med Torus som gick till 4600 objekt. Den misslyckades ifrån 4600<5000 objekt samtidigt som problemen inte är lösta ännu och som sagt så har Babylon bara exekverats ifrån stationær dator och inför nästa test på andra enheten så kan antalet av objekt ha minskat dramatiskt. Vid denna stund är resultatet inte noggrant framtaget och kan därmed inte säga så mycket, och i sista delen av utvärdering skall det mätas flera gånger för att ersätta de tidigare mätningsresultat.

Figur 10  Rendering på 5000 objekt från Babylon.js.
5 Utvärdering

Följande kapitel kommer att visa upp en utvärdering av flertalet mätningar framtagna från båda ramverken. Det tas även upp flera olika versioner av mätningar, där första versionen är med MSI afterburner och Task manager, den andra versionen med HWinfo med Chrome FPS mätning.

5.1 Presentation av undersökning


Efter första versionens mätningar, som gjordes med MSI och Task manager kändes det inte riktigt som ett trovärdigt resultat på grund utav olika typer av spik i exekvering av mätningen och av den anledningen måste det göras om med ett annat program för ett mer noggrant resultat. Till den andra versionen av mätningen användes programmet Hwinfo (Hwinfo, 2016) som såg mycket lovande ut, det är ett program som kan göra mätningar på allt som sker i datorn under en viss tid.
Figur 12  
HWinfo beskriver allt som ske i datorn.

För att kunna börja med mätningen så måste det finnas ett schema för hur det skall mätas, vad som skall mätas och hur många steg det skall vara. Från de tidigare faserna under pilotstudien var det objekt som Torus, Box, Cylinder och Sphere renderas ut, vilket nu under utvärderings-delan har lett till ändringar och minskning av antal objekt från fyra till tre. De utvalda objekteten blir Torus, Cylinder och Knot som skall mätas i tre områden och tio steg var. Det är främst för att kunna se dess förmåga att kunna sätta press på webbläsaren samt CPU:n och GPU:n beroende hur komplexa objekt det är.
Figur 13  Exempel på schema ifrån stationär dator mätning

Vid denna stund har både ramverken samma funktioner som ljus, färg, objekt, de innehåller bara kod och allt som är kommentarer är borttagna för att se om renderingen kan renders ut snabbare utan dem och med det återstår nu bara mätningen. Det kommer att uppskattas till ungefär 120-240 mätningar med tio sekunder per mätning under experimentet. Under varje sekund som går samlas det in data om vilka värde på mätningsschema var, som sedan multipliceras med antalet sekunder som testet föregås på, vilket är tio för att få ut medelvärdet. Det antalet mätningar som skall köras har gått upp och ner på grund av ändringar av verktys, felmätningar eller extra körning för ett säkrare resultat.

Målet med schemat är att försöka att hitta både tak och golv för FPS eftersom den verkar vara den mest eftertraktade i experimentet och för att kunna använda det resultatet för att fastställa slutsatsen om att det verkligen stämmer överens. Det är därför det kommer att skilja lite mätningsschemat mellan stationär och bärbar, vad gäller antalet objekt. Detta beror helt på känslan av att den bärbara kräver lägre tak för att nå 60 FPS och att golvet kommer vara ganska lågt därmed. Medan den stationära klarar 60 FPS med ett större antal objekt i både tak och golv under rendering.

Figur 14  Exempel på schema ifrån bärbar dator mätning

5.2 Analys

Under analys kommer det bara visas upp version två mätningar för att det är dem mätningar, resultat och diagram som skall bli slutresultat för experimentet medan version ett kommer finnas med under appendix-delen.

Version två innehåller främst mätningar och diagram som beskriver vilka värden som olika steg fick ut under exekveringen. Den har mätts bland annat i CPU användning i totalt och inom GPU D3D-användning under en kortare period. GPU:n D3D som är en grafikutöppning programmering gränssnitt API och den används för att rendera tredimensionell grafik i applikationer där prestanda är viktigt, till exempel spel (Conedweller, 2010).
Första bilderna kommer från renderingar på stationära mätningar och där start positionen är lite större än den bärbara.

Figur 15 Stationär Three rendering på Torus, Cylinder och Knot.

Bilderna säger ganska mycket om hur hårt CPU:n och GPU:n arbetar under mätning, beroende på vilket objekt som renderas ut under tiden. Det ser ut som att Cylinder arbetade minst på GPU:n när de andra två har det lite jobbigt med Torus och Knot. En snabb slutsats
skulle kunna vara att Torus och Knot är lite mer komplexa objekt att renderas ut, jämfört med Cylinder. Kan även bero på antalet polygoner som finns. För FPS:n så är förändringen ganska bra för varje steg som går.

![Diagram](image1.png)

![Diagram](image2.png)
Däremot har Babylon gett ut mycket bättre prestanda på Cylinder, även Torus kunde inte exekvera mer än 3500 < x antal. Samma sak med Knot som har ännu lägre antal på 500 < x. Detta kan bero på att webbläsaren blev överbelastad av antalet polygoner eller för komplexa objekt och det är fortfarande okänt varför det blev så. Om det utgår ifrån Cylinders resultat så är värdet där mycket bättre och i jämförelse med Three:s värde så har den lägre CPU- och GPU-användning. Golvet för Babylon:s Cylinder 50000 objekt och därefter där exekvering och filen går ej köras på webbläsaren. Nackdelen med Babylon är att ju mer objekt som ska renderas, desto längre tid det tar det för sidan att laddas upp än Three.

Figur 16  Stationär Babylon rendering på Torus, Cylinder och Knot.
Som det visas här är att bärbara Three klarar sig inte så pass bra jämfört med den stationäras värde på alla faser i mätningen. Detta kan helt enkelt bero på vilka hårdvaror den bärbara har i jämförelse med vad en stationär har. FPS:n sjunker redan vid antalet 500 och det fortsätter göra så till 3500 < x, vilket det säger vart taket och golvet ligger för den bärbara och hur det skiljer sig ifrån den stationären. Detta skulle kunna besvara hypotesen till varför det är så få som använder sig av 3D-teknik inom webbsidor och att det verkligen spelar roll vilken typ av produkt som ägts.

Figur 18  Bärbar Babylon mätning på Torus, Cylinder och Knot.

Med detta värde, visas det att Babylon tar mer arbetskraft, men kan renderas ut mycket bättre, samt ge en bättre flytande känsla när den renderas och att det även kan ta längre tid. En annan fördel med bärbara Babylon är att den verkar ge ut bättre FPS:n värden jämfört med Three under en längre period.

5.3 Slutsatser

Deras tabeller säger både för- och nackdelar med ramverk, stationär eller bärbar, Torus, Cylinder eller Knot samt taket och golvet. Men det känns fortfarande inte som att det har besvart hypotesen ”är Babylon.js, en spelmotor ramverk som kan ge ut bättre prestanda än Three som är ett brett ramverk både i spel och också webbdesign?”. Vad är det för prestanda som efterfrågas här, är det FPS:n, CPU:n, GPU:n eller allt i ett. Hypotesen har som sagt inte besvarats och kommer kanske inte kunna få ett svar om det inte efterfrågas inom ett mer specifikt område.

Om användaren är ute efter FPS:n så har Babylon gett ut det bästa resultatet i både stationär och bärbar eller om de är ute efter antalet på objekt, så är det Three stationär som gett bäst värden av alla mätningarna. Detta beslut är upp till var och en som får bestämma vad som passar deras ändamål bäst och vad för svar de är ute efter, vilket betyder hypotesen varken kan styrkas eller motbevisas.

För framtida arbete nämndes det också att en hel del arbete finns kvar att göra med förbättring hos renderingsmotorn, volym, bearbetningstiden och sammanfattningsvis, så menas det att både deras och mitt experiment där våra mål och resultat stämmer överens med våra och kom fram till att det finns en möjlighet med att 3D rendering inom webbvärlden finns kvar.

Figur 19 Jämförelse bild mellan ramverken
Figur 20  Standard avvikelse för stationär cylinder
Figur 21  Standard avvikelse för barbär cylinder

Det kunde bara få fram standard avvikelse för cylinder på grund av att andra objektmätningar ej var fullbordade, vilket anses då inte vara med. Figur 20 och 21 visar att CPU:s standard avvikelse var bättre än andra två och även med den så var det inte passande att dra några slutsatser till att svara på hypotesen om den stärker eller motbevisar.
6 Avslutande diskussion

6.1 Sammanfattning

Med alla mätvärden och resultat som har tagits fram med ramverken Three.js och Babylon.js, samt de andra flertals ramverk under fallstudien, hur de förhåller sig till varandra, för och nackdelar som var och en har samt vilken möjlighet de har för att kunna ge användare en fördel inom arbetslivet och forskning. Att 3D-rendering används så allt för sällan inom webb-världen kan bero på många saker, som både berör befintliga och nya användare av tekniken. Till exempel kan svårighetgraden inom programmeringen som många kanske anses vara för komplex för att sitta och försöka att få till, eller så har 3D-tekniken svårare att locka till sig utvecklare för användning än vad 2D har.


Undersökningen kommer bland annat ta upp förstudier, mål med experimentets lösningar, resultat, värden från mätningar samt delsvar från frågeställningen. Ett rakt svar till vilken av de ramverk som ger bäst prestanda för användning kan vara svårt att bevisa i detta experiment men kan fortfarande ge en bra start som kan leda till ett svar på frågan.

6.2 Diskussion

Först och främst har experimentet varit en utmaning ur både fysisk och psykisk aspekt. Detta kan vara allt ifrån att det kommer ett hinder eller när det ska tänkas utanför lådan. Allra första tanken om experimentet var att bara försöka rendera ut något ifrån två olika ramverk, med siktet att fokusera på resultat och ingenting annat. Men allt förändrades efterhand när insikten om att det var mer än att bara rendera ut något och förvänta sig ett resultat. Det var bland annat; hur det skall utföras, vilka metoder skall användas och vilket sätt är det bästa sättet att göra det på?

Experimentets problem var alltid att försöka motbevisa hypotesen och det är att "kan spelmotorers ramverk ge bättre prestanda än vanlig 3D mjukvara som Three". Från en utvecklareperspektiv så har båda ramverken för- och nackdelar, vilket nämns flera gånger. Resultatets trovärdighet är fortfarande oklart och om det är säkert att utgå ifrån det eller inte och detta beror helt på den metod som används och om de verktyg som användes var felfria. Det kanske finns andra sätt som mätningarna kunde har skett på, till exempel så var första mätningen med MSI afterburner mer korrekt än Hwinfo mättvärde.

De har även utfört stickprov i mätningar för att fastställa att resultaten stämmer överens, vilket visat att det var korrekt. Om textur-renderingen hade lyckats, så hade den riktat experimentet fört åt annat håll på grund av nya resultat och värden. Detta hade kunnat bevisat att texturer verkliga spelar roll i 3D-rendering. Figur 22 och 23 på sida 32 är bilder på försöket av att få fram textur i båda ramverken.
Figur 22  Försöket med texture för Three.js

Figur 23  Försöket med Babylon.js texture

6.2.1 Forskningsetiska aspekter hos arbetet eller undersökningsmetoden
Från andra aspekter så kunde undersökningen utföras annorlunda, beroende på vilket ramverk som användes och valdes ut för den specifika fallstudien. Det kunde ha valt att efterlikna ramverk som är specifikt för spelmotorer, det vill säga två spelmotorramverk som Babylon och Playcanvas. Det har hänt under undersökningen att vissa spikar blir borsträckade från beräkningen på grund av att ett värde antingen var noll eller 100, det vill säga onormal.

Från måtvrådena som samlades in så går resultaten inte att lita på tills nya metoder testas eller att en till mätning genomförs för att kunna jämföra mätning A och mätning B. Mätning som utfördes med MSI afterburner och Hwinfo går det inte att dra några slutsatser ifrån, för att mätvråden med MSI ej kunde fullföljas. Om det hade varit fullbordat, hade det varit svårt att jämföra när MSI:s CPU visade resultatet i procent för webbläsaren medan Hwinfo visade den totala CPU användningen under en viss tidsperiod.

I renderings algoritmen hos applikationerna så är de lika varandra och dess syfte är ju att rendera ut föremål, som sagt så går studien ut på att rendera ut föremål på webbapplikationer. Därifrån gjordes det att både applikationernas kod var så lika varandra som möjlig för att kunna dra några slutsatser.

Återupprepning är möjlig, och att återupprepa med uppbyggnad av applikationen från grunden genom att använda koden som har använts går att hitta i (Appendix B) och (Appendix C). Vid återupprepning av studien måste samma version användas på grund av skillnader mellan versionerna. Inom statistik för alla mätvråden som har mätts med mjukvaran "HWinfo" så ansåg jag personligen att det bästa valet var att använda MSI afterburner. Detta stöds genom att använda HWinfo som en mätningstid istället för MSI, som först och främst mäter genom alla hårdvaror som finns i datorn på en gång under en önskad tid än att behöva bläddra mellan olika fönster. Faktum är att mätningarna hade spikar här och där, så är resultatet från HWinfo mycket bättre än resultatet med MSI. Samt att spikar var inräknade i mätresultatet så gjorde det mer att resultatet kan vara helt osäkert att utgå ifrån eller användbart inför jämförelse med andra arbeten.

Samma sak med att använda Chrome:s FPS-mätare istället för att använda olika FPS-mätare som skulle ha varit inbyggda för båda ramverken, vilket skulle ha mätt ifrån renderingsdelen och inte hela webbläsaren som Chrome gör.

6.2.2 Samhällelig nytta hos arbetet
Många år från nu om applikationen har fortsatt att växa så skulle det finnas en chans att detta blir en produkt som riktas till utvecklaren med syftet att först och främst hjälpa utvecklaren genom att erbjuda tjänster som open-source-program inom testning av ramverk. Det vill säga att den kan innehålla flera andra ramverk; Playcanvas, Blend4web, unity m.m. Denna produkt kommer erbjuda utvecklare färdig kod för föremål, tips, hjälp samt innehålla flera "ramverk" med rendering-fas som mål, vilket är för dem att kunna testa rendering på exempel deras arbetsdator. Självklart så kommer utvecklare att kunna ändra om föremål, kod och m.m.

6.3 Framtida arbete

Detta forskningsarbete har många tankar och ideér kvar som inte är infört än i experimentet, teorier som går obesvarade men som fortfarande kan bearbetas i framtiden. Tänkbara tester att jobba vidare med utifrån detta experiment är att också testa med textur-rendering och sedan kunna jämföra det med basic-rendering skulle vara ett intressant scenario, främst att kunna se resultat. Det skulle kunna testas med att ha kameror som roterar runt sig själva eller har ett stort objekt som täcker kameran för att se om det verkligen spelar någon roll, det vill säga om en kamera som filmar flertalet objekt ligger bakom eller bara ett stort objekt som ligger framför allt. Det kan finnas en chans att resultatet bli annorlunda jämfört med nuvarande resultat.

Arbetet skulle kunna tillämpas på större projekt, till exempel till företag och privata utvecklare för att försöka att få renderingen att flyta på bättre genom att öka kapaciteten hos GPU, vilket (Savoy et.al, 2015) gjorde i sitt experiment. Rickardson & George (2012) påpekade att GPU teknik har växt kontinuerlig både när det gäller hårdvara och mjukvara och enhetsarkitekturerna blir allt mer komplexa, vilket ökar kraven på grafikprocessorer mer och mer. För tillämpningar på all hårdvara, inklusive GPU så har många olika faktorer begränsat applikationsprestandan.

För företag så finns det chans att kunna förbättra programmet genom att driva mer på tutorial och best practice, vilket syftar på Babylon som är så pass nyare än Three, samt att försöka locka flera utvecklare att använda dessa open source program för mer idéer och tankar ska kunna komma fram. Finns det till exempel ett bättre sätt att rendera ut objektet?

7 Referenser


Chih, Y. Y., Kuang-Y. C. & Hong, N. G. (2015) Resolution Independent Real-Time Vector Embedded Mesh for Animation, Taiwan University, Taiwan. IEEE. s.1.


8 Appendix A Första version av mätvärde

Stationär Three bacis Torus

Stationär Three bacis Cylinder
9 Appendix B Three.js testapplikation

```html
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  
  <title>Multip objekt - Copy</title>
  <script src="three.min.js"></script>
  <script src="TrackballControls.js"></script>
  <script src="TGALoader.js"></script>
  <script src="stats.min.js"></script>

  <style>
    body {
      margin: 0px;
      overflow: hidden;
    }
  </style>
</head>
<body>

var camera, controls, scene, renderer;
var spotLight1, var spotLight2, var spotLight3;

  var stats = new Stats();
  stats.setMode( 1 );
  stats.domElement.style.position = 'absolute';
  stats.domElement.style.left = '0px';
  stats.domElement.style.top = '0px';
  document.body.appendChild( stats.domElement );

init();
animate();

function init(){
camera = new THREE.PerspectiveCamera(75, window.innerWidth / window.innerHeight,1,25000);
camera.position.z = 4000; // Camera position i z-led
controls = new THREE.TrackballControls(camera);
controls.addEventListener('change', render);
scene = new THREE.Scene();
```
light = new THREE.PointLight( 0xFFFFFF );
light.position.set( 0, 0, 1 );
scene.add( light );

//spotLight1 = new THREE.SpotLight( 0xCCFF00, 100000 );
//spotLight1.position.set( 0.5, 0, 1 );
//spotLight1.position.multiplyScalar( 5000 );
//scene.add( spotLight1 );
//spotLight1.castShadow = true;

//spotLight2 = new THREE.SpotLight( 0x0033CC, 100000 );
//spotLight2.position.set( -0.5, 10, 5 );
//spotLight2.position.multiplyScalar( 5000 );
//scene.add( spotLight2 );
//spotLight2.castShadow = true;

//spotLight3 = new THREE.SpotLight( 0xCC3300, 100000 );
//spotLight3.position.set( -50, 100, -100 );
//spotLight3.position.multiplyScalar( 5000 );
//scene.add( spotLight3 );
//spotLight3.castShadow = true;

// kommentara bort objekt när den inte skall exekvers, bara en kan köras i taget.

var geometry = new THREE.TorusKnotGeometry( 50, 20, 100, 16 );
//var geometry = new THREE.CubeGeometry(100,100,100);
//var geometry = new THREE.TorusGeometry( 100, 10, 16, 100 );
//var geometry = new THREE.SphereGeometry(100,100,100);
//var geometry = new THREE.CylinderGeometry(100,100,100);
//var material = new THREE.MeshNormalMaterial();
var material = new THREE.MeshBasicMaterial({ color: 0xFFFFFF });

setInterval(function () {
    var time = Date.now() * 0.001;
    stats.begin();
    stats.end();
}, 1000 / 60);

for(var i = 0; i < 5000; i++){
    var mesh = new THREE.Mesh(geometry, material);
    mesh.position.x = (Math.random() - 0.5) * 4000;
    mesh.position.y = (Math.random() - 0.5) * 4000;
    mesh.position.z = (Math.random() - 0.5) * 4000;
    mesh.rotation.x = (Math.random() - 0.5) * 4000;
    mesh.rotation.y = (Math.random() - 0.5) * 4000;
    mesh.rotation.z = (Math.random() - 0.5) * 4000;
    scene.add(mesh);
}

renderer = new THREE.WebGLRenderer();
renderer.setSize(window.innerWidth, window.innerHeight);
document.body.appendChild(renderer.domElement);}

function animate(){
    requestAnimationFrame/animate());
    controls.update();
    var time = - performance.now() * 0.0005;
    camera.position.x = 6000 * Math.cos(time);
    camera.position.z = 6000 * Math.sin(time);
    camera.lookAt(scene.position );}
function render(){
    renderer.render(scene, camera); }

function onWindowResize() {
    camera.aspect = window.innerWidth / window.innerHeight;
    camera.updateProjectionMatrix();
    renderer.setSize( window.innerWidth, window.innerHeight ); }
</script>
</body>
</html>
Appendix C Babylon.js testapplikation

<!DOCTYPE html>
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  <title>Multiple object in Babylon</title>
  <script src="http://www.babylonjs.com/hand.minified-1.2.js"></script>
  <script src="http://www.babylonjs.com/cannon.js"></script>
  <script src="http://www.babylonjs.com/oimo.js"></script>
  <script src="http://www.babylonjs.com/babylon.js"></script>
  <script src="rStats.js"></script>

  <style>
    html, body {
      overflow: hidden;
      width: 100%;
      height: 100%;
      margin: 0;
      padding: 0;
    }
    #renderCanvas {
      width: 100%;
      height: 100%;
      touch-action: none;
    }
  </style>
</head>
<body>
<canvas id="renderCanvas"></canvas>
<script>
var canvas = document.getElementById("renderCanvas");
var engine = new BABYLON.Engine(canvas, true);
var createScene = function() {

var Scene = new BABYLON.Scene(engine);
Scene.clearColor = BABYLON.Color3.Black;

var Camera = new BABYLON.ArcRotateCamera("Camera", 0, 0, 0, new BABYLON.Vector3(0, 0, 0), Scene);
Camera.setPosition(new BABYLON.Vector3(0, 0, -470));
Camera.attachControl(canvas, true);

var light0 = new BABYLON.PointLight("light0", new BABYLON.Vector3(0, 0, 10), Scene);
light0.diffuse = new BABYLON.Color3(1, 1, 1);
light0.intensity = 1.0;

var numberOfObject = 2000;
var BoxRange = 200;

var RandomPosition = function(object, i, s) {
    object.position.x = (Math.random() - 0.5) * BoxRange;
    object.position.y = (Math.random() - 0.5) * BoxRange;
    object.position.z = (Math.random() - 0.5) * BoxRange;
    object.rotation.x = Math.random() * 3.15;
    object.rotation.y = Math.random() * 3.15;
    object.rotation.z = Math.random() * 3.15;
    object.color = new BABYLON.Color3(1, 1, 1);

    // object.color = new BABYLON.Color4(object.position.x / BoxRange + 0.5, object.position.y / BoxRange + 0.5, object.position.z / BoxRange + 0.5, 1.0);

    // var mat = new BABYLON.StandardMaterial("mat1", Scene);
    // var texture = new BABYLON.Texture("http://jerome.bousquie.fr/BJS/images/spriteAtlas.png", Scene);

}
// mat.diffuseTexture = texture;

// kommentera bort objekt när den inte skall exekvers, bara en kan köras i taget.
// ändra objekt sker vid själva objekt namn och SPS.addShape "objekt namn"

// var knot = BABYLON.Mesh.CreateTorusKnot("knot0", 8, 2, 100, 64, 2, 3, Scene);
// var box = BABYLON.MeshBuilder.CreateBox("box", {tessellation: 4, sideOrientation: BABYLON.Mesh.DOUBLESIDE}, Scene);
// var box = BABYLON.Mesh.CreateBox("Box", 7, Scene);
var cylinder = BABYLON.Mesh.CreateCylinder("cylinder", 10, 10, 10, 20, Scene);
// var ball = BABYLON.Mesh.CreateSphere("Ball", 10, 10, 10, Scene);
// var torus = BABYLON.Mesh.CreateTorus("torus", 8, 2, 32, Scene, false);
// cylinder.material = mat;

var SPS = new BABYLON.SolidParticleSystem('SPS', Scene, {updatable: false});
SPS.addShape(cylinder, numberOfObject, {positionFunction: RandomPosition});
var mesh = SPS.buildMesh();

Scene.registerBeforeRender(function() {
    light0.position = Camera.position;
    SPS.mesh.rotation.y += 0.008;
});

return Scene;

var Scene = createScene();

engine.runRenderLoop(function () {
    Scene.render();
});

window.addEventListener("resize", function () {
    engine.resize();
});
</script>
</body>
</html>