Jämförelse av 3D-lösningar för HTML5

Rasmus Saxén

Examensarbete

Informations- och medieteknik

2013

EXAMENSARBETE	
Arcada	
Utbildningsprogram:	Informations- och medieteknik
Identifikationsnummer:	4183
Författare:	Rasmus Saxén
Arbetets namn:	Jämförelse av 3D-lösningar för HTML5
Handledare (Arcada):	Johnny Biström
Uppdragsgivare:	

Sammandrag:
Arbetet är en insikt i teknologin bakom nativ 3D-grafik inom HTML5 och en jämförelse mellan tre olika tjänster som används för att visa upp 3D-modeller på nätet. Som forskningsmetoder har både litteraturstudier och praktiska test använts. Arbetet är begränsat till nativ 3D-grafik och tjänster som visar upp 3D-modeller. Plug-ins, interaktiva 3D-webbsidor eller -spel är inte en del av detta arbete. Arbetets syfte är tredelat, få en insikt i tekniken bakom web-baserad 3D-grafik, undersöka några tjänster som använder ifrågavarande teknik och slutligen att testa dem för jämförelse och för att klargöra hur tekniken bakom hårdvaruaccelererad 3D-rendering har kommit i användning. Forskningen går in på HTML5 och canvas-elementet. Det redogörs för hur de utvecklats och vad de för med sig. Arbetet tar upp WebGL, vem som utvecklat det och hur det fungerar. Arbetet förklarar även renderingsmotorer. Deras grundfunktionalitet tas upp och förklaras och renderingsprocessen inom WebGL gås igenom i detalj. Arbetet presenterar 3D-biblioteket Three.js och redogör för dess betydelse. Tre stycken tjänster för 3D-grafik på nätet, p3d.in, Sketchfab och Verold Studio presenteras. Forskningen fokuserar skilt på varje tjänst och går igenom respektives styrkor och speciella egenskaper. Arbetet presenterar ett test som alla tre tjänster utsätts för. Resultaten presenteras separat i detalj för varje tjänst, följt av ett kort sammandrag

Nyckelord:	3D, HTML5, canvas, WebGL, Javascript
Sidantal:	43
Språk:	Svenska
Datum för godkännande:	

DEGREE THESIS	
Arcada	
Degree Programme:	Information and Media Technology
Identification number:	4183
Author:	Rasmus Saxén
Title:	Jämförelse av 3D-lösningar för HTML5
Supervisor (Arcada):	Johnny Biström
Commissioned by:	

Abstract:
The work is an insight into the technology behind native 3D-graphics within HTML5 and a comparison between three different services used to show 3D-models on the Internet. The research is based both on literature studies and practical tests. The work is limited to native 3D-graphics and services that displays 3D-models. Plugins, interactive 3D-webpages or -games will not be covered. The purpose of this work is split into three parts, to get an insight into the technology behind web-based 3D-graphics, to do research on a few services that uses this technology and finally to test them for comparison and to find out how well hardware accelerated 3D-rendering has been put to use. The research brings up HTML5 and the canvas-element. Their development is explained as well as what new possibilities they bring with them. The work brings up WebGL. Its development and functions are explained. The research also brings up render engines. Their basic functionalities are explained and the rendering process of WebGL is examined in detail. The work introduces the graphics library Three.js and its importance is explained. Three services for showing 3D-models on the web, p3d.in, Sketchfab and Verold Studio are introduced. The research focuses on each one of these services and brings up their strong points and special attributes. The work presents a test to be taken by all three services. The results are then presented separately for each service, followed by a short summary.

Keywords:	3D, HTML5, canvas, WebGL, Javascript
Number of pages:	43
Language:	Swedish
Date of acceptance:	

INNEHÅLL

FIGURER

FÖRKORTNINGAR OCH DEFINITIONER

API	Application Programming Interface. Regler för hur olika programvara kommunicerar med varandra.
K3D	Kevs 3D. Renderingsmotor för canvas som inte är beroende av WebGL. Lagad av Kevin Roast.
Khronos Group	Industrikonsortium som fokuserar sig på skapandet av APIn för öppen standard. Ansvariga för bl.a. OpenGL och WebGL.
Plug-in	Datorprogram som inte körs fristående utan innanför ett annat program, som t.ex. Adobe Flash Player i en webbläsare.
RAM	Random Access Memory. En dators huvudminne.
Rendera	Framställa en bild utifrån någon form av data, till exempel en 3D-modell.
SVG	Scalable Vector Graphics. Ett format för skalbar vektorgrafik vars egenskaper definieras i XML-textfiler.
W3C	World Wide Web Consortium. Internationell organisation som utvecklar standarder för webben. Har arbetsgrupp ansvarig för HTML5-specifikation.
WHATWG	Web Hypertext Application Working Group. Utvecklar HTML5 och APIn för webapplikationer i samarbete med webläsartillverkare.

XHTML	Extensible HyperText Markup Language. En striktare vidareutveckling av märkspråket HTML. Baserat på XML.
XML	Extensible Markup Language. Universellt märkspråk.

1 INLEDNING

1.1 Bakgrund

Utvecklingen av HTML5 och specifikt canvas-elementet har öppnat dörrarna för en stor del nya intressanta funktioner inom webutveckling och –design. Ett stort steg framåt är HTML5-standardens försök att få många nya funktioner att klara sig utan externa plug-ins, och bland dessa funktioner finns möjligheten att visa upp 3D-material.

Som 3D-entusiast var jag intresserad av dessa nya funktioner och vad de för med sig både för 3D-industrin och –hobbyister. Jag valde därför att sätta mig in i denna teknologi och några av de möjligheter den har lett till.

1.2 Syfte och mål

Arbetet har tre huvudområden. En insikt i de teknologier som gör det möjligt att visa 3D-modeller direkt i en webbläsare, att göra sig bekant med några av dessa lösningar och slutligen testning av dem.

Den första delen som består av kapitel 2 och 3 beskriver de olika delar som ligger bakom 3D i webbläsare och försöker ge en uppfattning om hur all den nämnda teknologin går ihop. Den andra delen av arbetet presenterar tre olika lösningar för webläsarbaserad 3D-grafik; p3d.in, Sketchfab och Verold Studio. Målet är att jämföra dessa lösningar och ta reda på deras styrkor och svagheter, samt att se vilken potential de kan ha för privatpersoner och/eller firmor.

Den sista delen av arbetet utsätter de tidigare nämnda lösningarna för ett praktiskt test med avsikt att jämföra deras prestanda och funktioner.

1.3 Metoder

Den första delen av arbetet baserar sig på litteraturstudier, både i form av skrivna publikationer och tekniska specifikationer kring den involverade teknologin. Delen av

arbetet som behandlar de tre tjänsterna som undersöks kommer att basera sig på en blandning av deras egen information och instruktioner, andra källors utvärderingar och egna iakttagelser. I testningsskedet kommer en identisk 3D-modell att laddas upp till samtliga tjänster varefter deras prestanda och funktionalitet jämförs.

1.4 Avgränsning

Lösningar baserade på externa plug-ins kommer inte att ingå. Arbetet tar i huvudsak upp lösningar kring visandet av enskilda modeller och prestandan av dessa lösningar, och kommer inte att gå in på till exempel interaktiva 3D-webbsidor eller 3D-spel. Det förväntas att läsaren besitter åtminstone grundkunskaper inom webutveckling och en grund inom någon form av 3D-grafik är till en fördel.

2 GRUNDER

Webläsarbaserad 3D-grafik oberoende av plug-ins bygger på tre grundteknologier. Grafiken programmeras in med Javascript, renderas av WebGL och presenteras i ett canvas-element inom HTML5. Detta kapitel går in på tekniken bakom dessa delar och redogör för deras roll i helheten.

2.1 HTML5

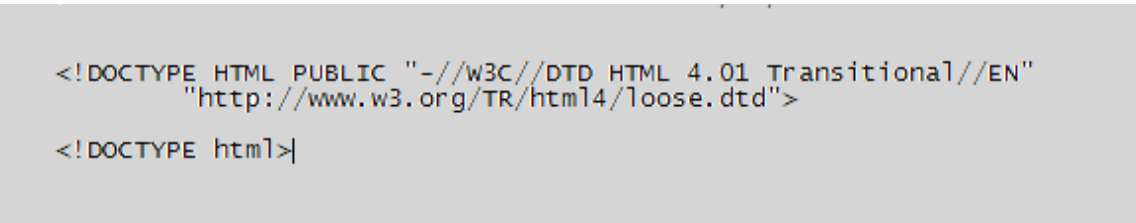
HTML5 är den femte versionen av märkspråket HTML, som ligger bakom största delen av världens webbsidor. Det nya canvas-elementet som har introducerats inom HTML5 utgör planet som 3D-grafik kan visas på.

2.1.1 Utveckling

HTML5 har varit under utveckling sedan 2004 när den första specifikationen skapades av WHATWG (Web Hypertext Application Working Group), en grupp med människor från webläsartillverkarna Mozilla, Google, Apple och Opera. World Wide Web Consortium (W3C) som tidigare kontrollerat utvecklingen av HTML hade lämnat det åt sidan för att fokusera sig på XML och XHTML som i början på 2000-talet ansågs vara

på väg att bli de nya webstandarderna. HTML som vägrade dö ut hade inte uppgraderats sedan version 4.01 år 1999 och målet för WHATWG var en specifikation anpassad för en mer dynamisk web och ett ständigt stigande antal webapplikationer. W3C anslöt sig till utvecklingen av HTML5 år 2006 och lade ner sin utveckling av XHTML 2 några år senare (Lubbers et al. 2011 s.1-3). W3C publicerade en färdig version av HTML5-specifikationen i december 2012 (W3C, 2012a).

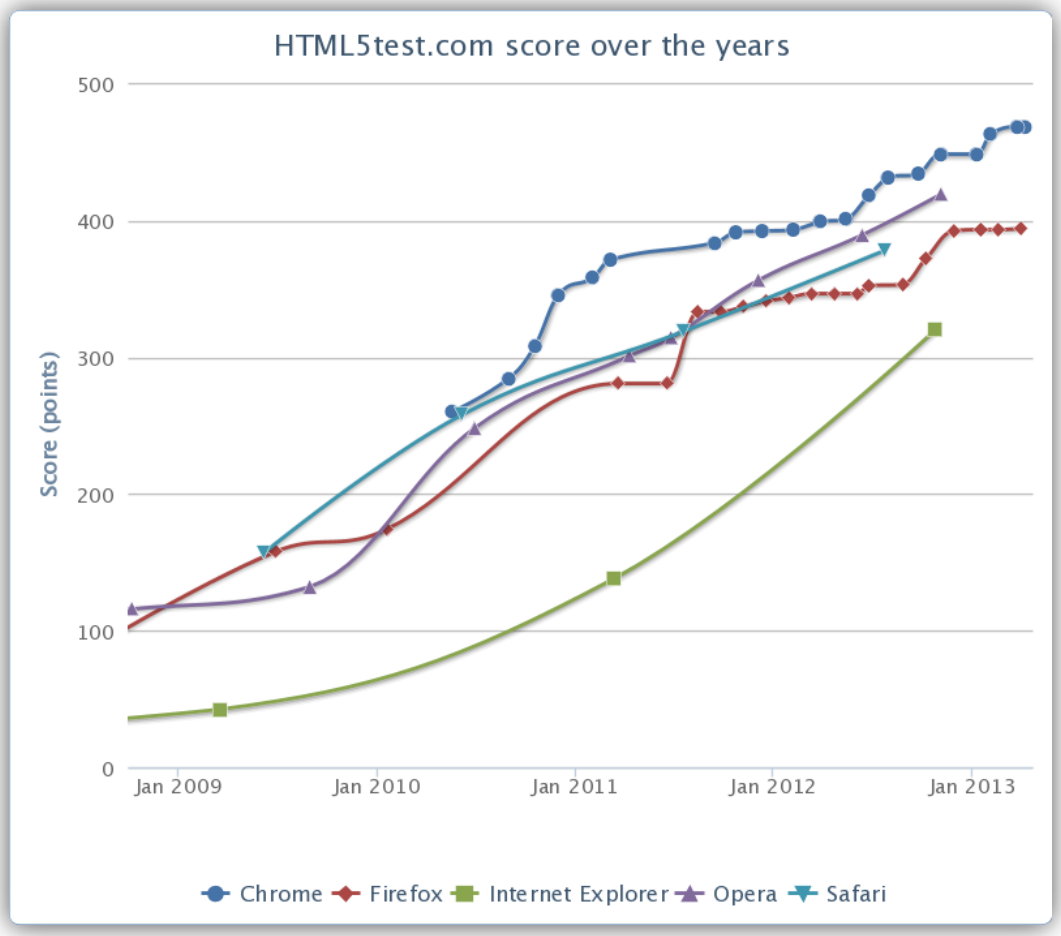
HTML5 introducerar flera nya funktioner för att underlätta och modernisera skapandet av webbsidor. En del av dessa ändringar syns på ett väldigt grundligt plan i form av mera flexibel syntax och en mycket förenklad DOCTYPE-deklaration (se figur 1).



Figur 1. DOCTYPE-deklaration för HTML 4.01 (övre) och HTML5 (nedre) (Lubbers et al. 2011 s.8)

Nya specificerade element såsom `<header>` och `<footer>` skapades som en följd av att analyser av miljontals webbsidor visade att dessa termer ofta användes som namn på mera allmänna `<div>`-element. Trots ett stort antal ändringar strävar HTML5 till att vara kompatibelt med äldre versioners kod. Något som krävs när vi börjar närma oss 20 år av existerande gammal HTML-kod (Lubbers et al. 2011 s.3-10).

HTML5 tar också ett stort steg mot en plug-in-fri web, genom att erbjuda inbyggt stöd för bland annat video och 3D-grafik. Denna inbyggda funktionalitet arbetar tillsammans med CSS och olika Javascript-APIn och skapar en flytande interaktion mellan element som inte fanns förr. Webläsartillverkarna håller aktivt på att implementera stöd för de nya funktionerna inom HTML5 och förbättrar dess specifikation genom deras experimenterande (Lubbers et al. 2011 s.1-6). Websidan www.html5test.com poängsätter webbläsare på en poängskala från 0-500 och innehåller detaljerad information om samtliga webbläsares stöd för de olika egenskaperna inom HTML5 (Sights, 2013).



Figur 2. Fem kända webbläsares olika versioners testresultat från www.html5test.com (Sights, 2013)

2.1.2 Canvas-elementet

Canvas-elementet kan beskrivas som ett bitmap-canvas, där varje pixel har sitt specifika värde efter att grafik ritats in. Grafik som målas på ett canvas kan alltså inte förstoras eller förminskas utan förlust i kvalitet, till skillnad från grafik i SVG-format. Sedan lanseringen av Internet Explorer 9 stöder alla webbläsare canvas-elementet.

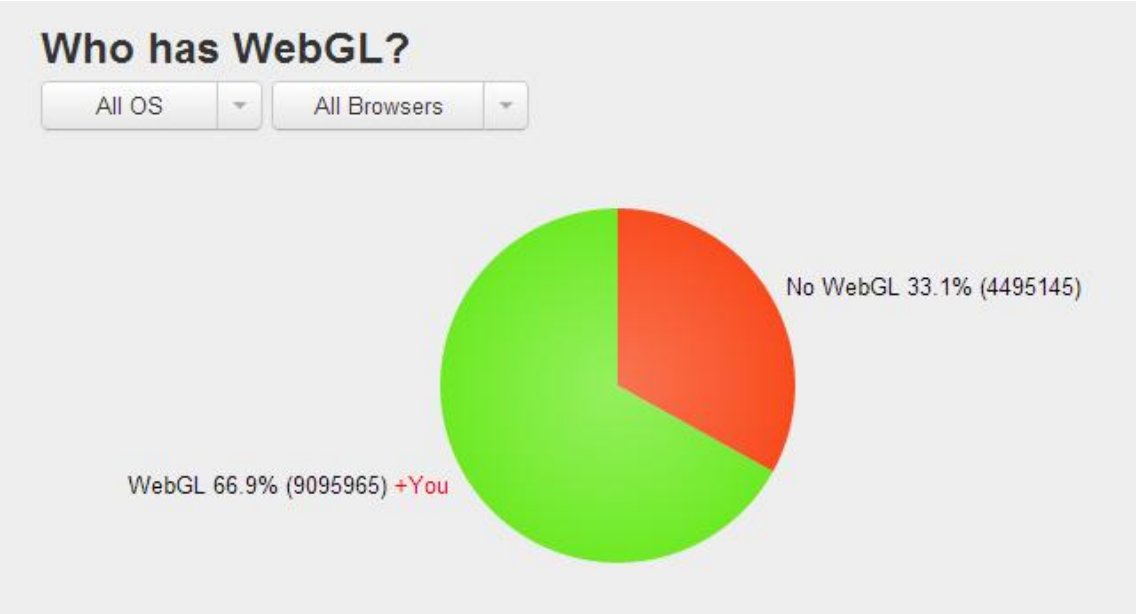
Från början är canvas-elementet endast ett blankt arbetsområde som man sedan lägger innehåll till med hjälp av Javascript (Lubbers et al. 2011, s.23-24). Genom att definiera elementets kontext såsom ”2d” eller ”webgl” bestämmer man vilken sorts API som kan

kontrollera innehållet. Vid en ändring av canvas-elementets dimensioner återställs bitmappen och eventuella kontexter till sina utgångsvärden (W3C, 2012b). CSS kan påverka själva canvas-elementets utseende och kan även beroende på kontext påverka vissa delar av elementets innehåll, t.ex. fonter (Lubbers et al. 2011 s.26). Canvas-elementets ”webgl”-kontext är grunden till avancerad 3D-grafik i webläsarmiljön.

2.2 WebGL

WebGL är ett API som tillåter skapandet av avancerad 3D-grafik på webben. Idén för WebGL skapades ur en prototyp för canvas-baserad 3D-grafik som lagades av Vladimir Vukćević för Mozilla. År 2009 startade Khronos Group en arbetsgrupp för utvecklandet av specifikationen för WebGL, som nådde version 1.0 i mars 2011 (Anyuru, 2012 s.1-2). För tillfället stöder alla större webbläsare förutom Internet Explorer WebGL, men det måste fortfarande aktiveras manuellt i Opera och Safari (Khronos Group, 2012).

WebGL baserar sig på OpenGL ES 2.0, ett API för inbyggda system som i sig är en simplifierad del av OpenGL som är ett av de mest använda APIna för 3D-grafik inom industrin (Khronos Group, 2013). Genom att ge ett 3D-kontext till canvas-elementet ger WebGL webapplikationer möjligheten att använda sig av snabb hårdvaruaccelererad rendering. WebGL är en öppen standard som är gratis att använda och den kör inbyggt i de webbläsare som stöder den (Anyuru, 2012 s.3). Khronos Groups webbsidor för en lista över projekt som involverar WebGL. Listan innehåller linkar både till simpla demonstrationer och fullständiga 3D-grafikmotorer och detaljerade grafikbibliotek (Khronos Group, 2013). Websidan webglstats.com som samlar statistik relaterad till WebGL från ett större antal webbsidor menar att ungefär 67 % av alla undersökta internetanvändare har stöd för WebGL (Boesch, 2012).



Figur 3. Statistik över WebGL-stöd fördelat mellan samtliga operativsystem och webbläsare (Boesch, 2013)

WebGL följer en modell som kallas för immediate-mode API, eller direktläges API. I ett direktläges API måste hela scenen renderas om för varje enskild bildruta. Grafikbiblioteket som tolkar applikationen har inte själv någon modell över hur scenen ser ut, utan får modellen av applikationen varje gång scenen skall renderas. Detta ställer högre krav på applikationen som hela tiden måste ha kontroll över modellen som den håller i minnet, men ger även applikationen en större nivå av kontroll och flexibilitet (Anyuru, 2012 s. 3). Detaljer kring själva renderingsprocessen tas upp i kapitel 3.2.

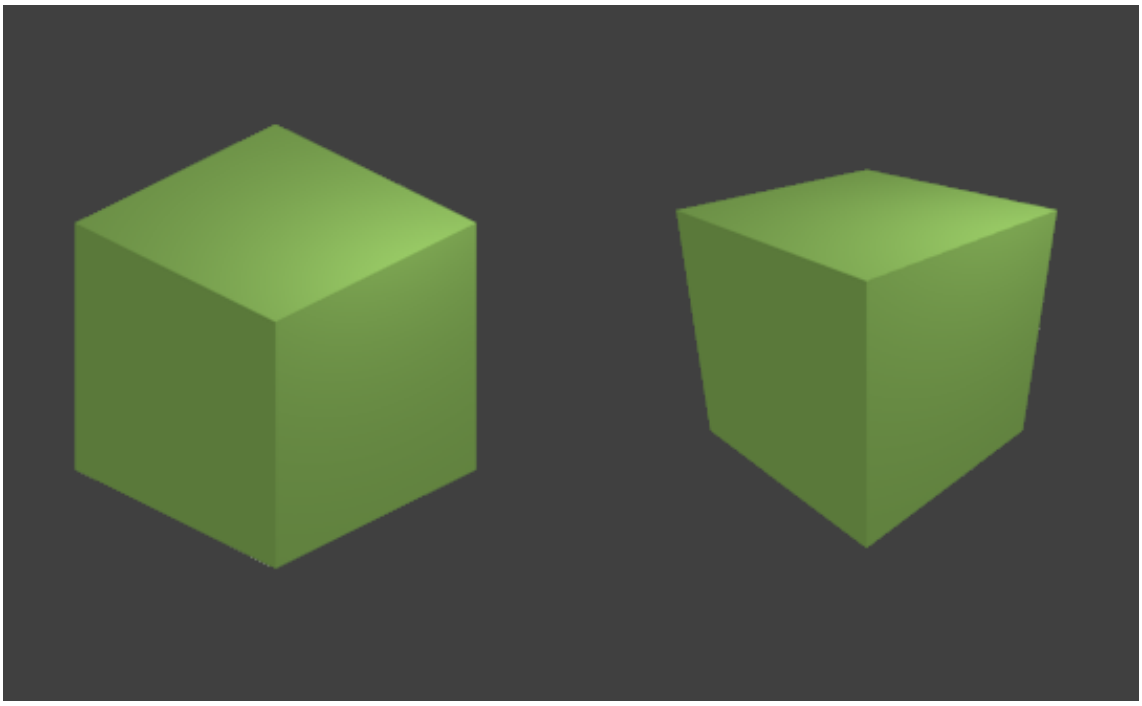
Figur 4. Förenklad representation av ett direktläges APIs funktion. Figuren är en översatt indirekt kopia (Anyuru, 2012 s.3)

3 RENDERINGSMOTORER

Från början är ett 3D-objekt i stort sett endast en serie koordinater i en tredimensionell rymd, inte något som kan visas upp. Detta fixas av renderingsmotorer, som omtolkar dessa koordinater till objekt och ger dem bland annat färg, ljussättning och reflektioner.

3.1 Från 3D till 2D

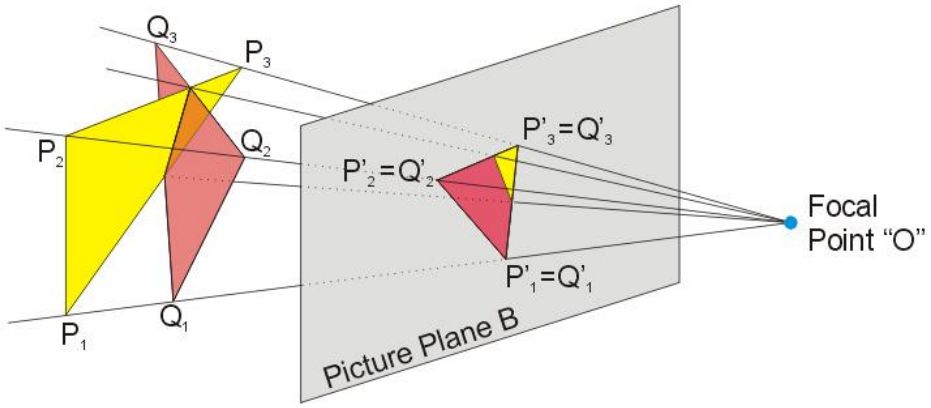
Fastän man talar om 3D-grafik så är det som syns på skärmen alltid tvådimensionellt. Renderingsmotorns jobb är alltså att omvandla 3D till 2D, som görs genom att 3D-filens datapunkter projiceras på ett tvådimensionellt plan (Sloane, 2011). Detta kan göras i form av endera ortografisk eller perspektivprojektion, som båda visas i figur 5 nedan.



Figur 5.Kub renderad med ortografisk projektion (vänster) och perspektivprojektion (höger) (Saxén, 2013)

Ortografisk projektion ger bilder ett onaturligt utseende, speciellt när de placeras vid sidan av en bild med perspektivprojektion. De parallella projektionsstrålarna utan skärningspunkt passar bättre t.ex. till tekniska ritningar (Walker, 2011). Mer utarbetade renderingsmotorer, såsom t.ex. de som används inom Three.js har även stöd för kameraobjekt som skapar ortografiska bilder (Cabello, 2013), men de flesta använder

sig av perspektivprojektion. Kapitlet kommer hädanefter att fokusera på uträkningarna bakom perspektivprojektion.



Figur 6. Exempel på perspektivprojektion (Baecker, 2005)

Grunden till en 3D-renderingsmotor är följande ekvation:

skala=brännvidd/(z+brännvidd)

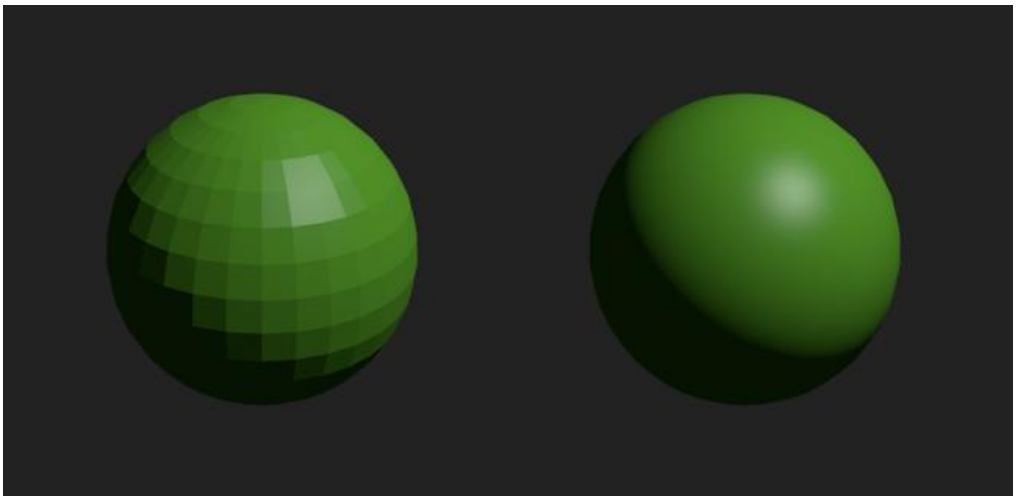
där z representerar z-koordinaten av den punkt man behandlar och brännvidden är avståndet till det tvådimensionella planet från skärningspunkten. Skalan multipliceras med punktens x- respektive y-koordinater och ger som resultat punktens läge på det tvådimensionella planet (Haapoja, 2011). I punktbaserade renderingar kan man för att lätta på mängden uträkningar använda en z-buffert som sparar z-koordinater och ser till att prestanda inte används för att behandla punkter som skulle bli bakom andra (Sloane, 2011). För att sedan forma objekt med ytor krävs ytterligare funktioner, som i många fall inte är komplicerade men väldigt långa att skriva in för hand. Ett exempel på detta är figur 7 nedan, som innehåller kod från renderingsmotorn K3D. Den synliga koden innehåller information som behövs för att skapa en kub (Roast, 2013).

```
init(
   // describe the eight points of a simple unit cube   Specifierar kubens åtta punkter
   [{x:-1,y:1,z:-1}, {x:1,y:1,z:-1}, {x:1,y:-1,z:-1}, {x:-1,y:-1,z:-1},
    {x:-1,y:1,z:1}, {x:1,y:1,z:1}, {x:1,y:-1,z:1}, {x:-1,y:-1,z:1}],
   // describe the twelve edges of the cube   Specifierar kubens 12 kanter
   [{a:0,b:1}, {a:1,b:2}, {a:2,b:3}, {a:3,b:0}, {a:4,b:5}, {a:5,b:6},
    {a:6,b:7}, {a:7,b:4}, {a:0,b:4}, {a:1,b:5}, {a:2,b:6}, {a:3,b:7}],
   // describe the six polygon faces of the cube  Specifierar kubens sex sidor
   [{color:[255,0,0],vertices:[0,1,2,3]}, {color:[0,255,0],vertices:[0,4,5,1]},
    {color:[0,0,255],vertices:[1,5,6,2]}, {color:[255,255,0],vertices:[2,6,7,3]},
    {color:[0,255,255],vertices:[3,7,4,0]}, {color:[255,0,255],vertices:[7,6,5,4]}]
);
```

Figur 7. Kod från K3D för initialiserandet av en kub (Roast, 2013)

3.2 Renderingspipeline och Shaders

När ett objekt ska renderas går det igenom en serie steg, varav vissa är programmerbara. Dessa program, shaders, är del av det som kallas för en renderingspipeline och är alla ansvariga för olika delar av objektet som ska renderas. Varje steg av renderingspipelinen har tillgång till specifika resurser varierande från bilder till buffrar eller geometrisk data (OpenGL, 2012). När man jobbar med 3D inom HTML5 är det WebGL som sköter om att leverera stödet för shaders, och på samma gång möjligheten till hårdvaruacceleration (Khronos Group, 2011).

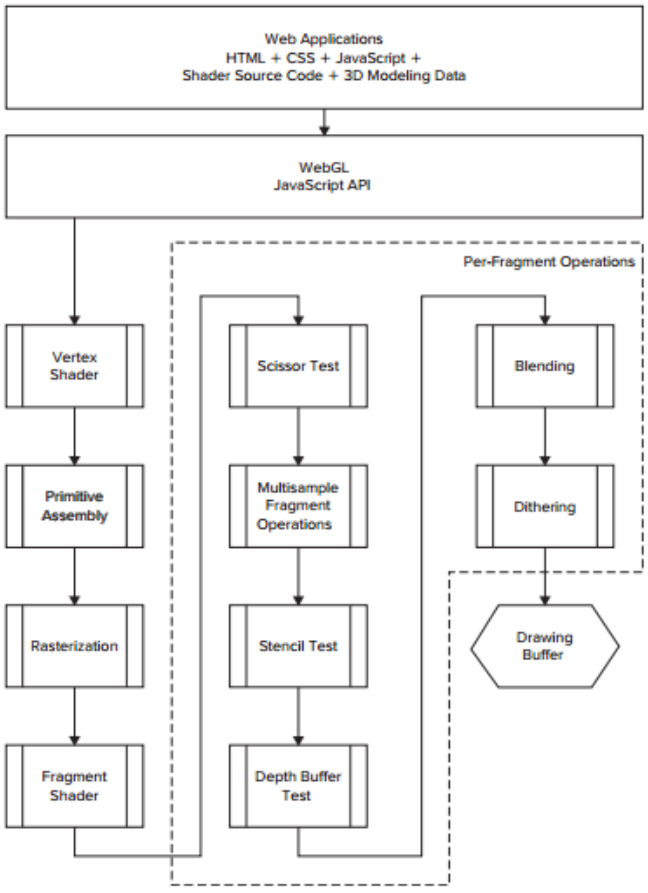


Figur 8. Två objekt med identisk geometri som påverkats av olika shaders (Saxen, 2013)

WebGL ger tillgång till två typer av shaders för användning inom HTML5. Den första typen är en vertex shader. Den har tillgång till att ändra på de enskilda punkterna som objektet består av. Här kan t.ex. distorsion programmeras in om man vill få en ojämn yta på sitt objekt. Den andra typens shader är en fragment shader. Den styr saker som

färg och reflektiva ytor på den färdiga geometriska figuren. I detta skede är det viktigt att påpeka att geometri inom 3D-grafik alltid består av trianglar, även kända som polygoner. Ett fragment betyder i detta sammanhang samma sak som en pixel. Ett fragments innehåll räknas ut genom att kombinera data av de punkter som bildar den triangel fragmentet befinner sig på (Lewis 2011). Objektet till höger i figur 8 är ett exempel på en effekt lagad av en fragment shader. WebGL innehåller inte någon färdigt fungerande renderingspipeline utan ger istället tillgång till en programmerbar pipeline som är kraftigare men som är svårare att lära sig (Lewis, 2011). För att underlätta situationen finns det lösningar som Three.js, som tas upp i följande delkapitel, som kommer med färdiga shader-funktioner som är lätta att modifiera.

Utöver två shaders består renderingspipelinen i WebGL av flera små mellansteg som kan ses i figuren nedan.



Figur 9. Överblick av renderingspipelinen i WebGL (Anyuru, 2012 s. 8)

De två stegen som befinner sig mellan vertex och fragmentshadern är ansvariga för uppbyggandet av objekts geometri. Vertex shadern skickar ut punktdata till steget ”primitive assembly” som skapar synliga punkter, linjer och trianglar av data. Följande steg, ”Rasterization”, skapar fragment av den nya geometriska data och skickar dem vidare till fragment shadern som beskrevs tidigare. Efter fragment shadern passerar ännu varje fragment genom ett antal specialsteg som finslipar färg och genomskinlighet och bestämmer om fragmentet skall skickas vidare till ritbufferten (Anyuru, 2012).

3.3 Three.js

Skaparen av Three.js beskriver sin produkt som ett 3D-bibliotek med låg komplexitet, och har tack vare denna inriktning en stor samling användare, bland annat tjänsterna p3d.in och Verold Studio som presenteras senare i detta arbete. Skaparen av Three.js, Ricardo Cabello, säger att han började arbeta på renderingsmotorn under en tid han kallar ”the demoscene days”, när alla lagade sina egna 3D-motorer bara för att använda dem till en eller två demonstrationer. Han ansåg detta vara slösaktigt och ville skapa något mera varaktigt som flera mänskor skulle kunna använda (Haines, 2013).

Three.js är ett av de mest omfattande 3D-biblioteken inom HTML5. Det stöder canvas- , svg-, CSS3D- och WebGL-rendering, Det kan skapa primitiver (simpla former såsom kuber, klot eller cylindrar), importera färdiga modeller från flera kända 3D-program såsom 3DS-Max och Blender och har stöd för en stor samling med shader-, material- och objektalternativ (Cabello, 2013b). Den totala längden på skriptfilen med alla funktioner är över 35000 rader med kod.

I proportion till sin funktionalitet är API inom Three.js väldigt enkelt att använda. Första sidan på projektets webplats visar ett exempel där en roterande kub kan skapas med endast 25 rader kod (Cabello, 2013a). Detta kan jämföras med koden från K3D i figur 7, där endast skapandet själva kuben tog upp nio rader. Detta gör Three.js till ett intressant alternativ om man vill få sina 3D-modeller ut på webben utan större besvär. Om man har extra tid till att sätta sig in i detaljerna bakom biblioteket så kan man få väldigt stor kontroll över innehållet man producerar.

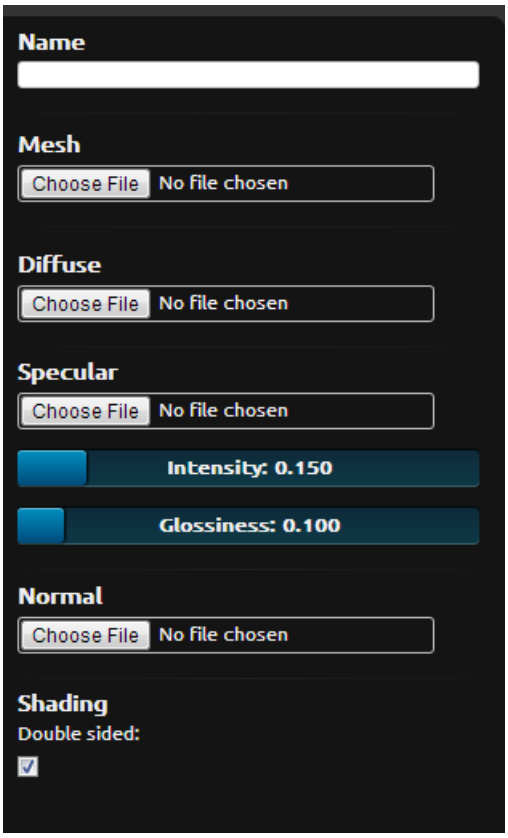
4 TILLÄMPNINGAR

Om man inte vill börja sätta sig in i Three.js eller koda sin egen renderingsmotor från grunden finns det flera enkla sätt att få sina 3D-modeller synliga på nätet. För det här arbetet har jag valt att presentera och sedan testa tre olika lösningar som alla används till att visa 3D-modeller. Alla tre lösningar har en liknande funktionalitet med olika styrkor och svagheter. Man skapar ett gratis användarkonto och sedan är det fritt fram att börja ladda upp sitt 3D-material för att endera kunna se det på tjänsternas sidor eller lägga in visningsfönster på diverse forum eller ens egna webbsidor.

4.1 P3D.in

P3D.in är en enkel men fungerande tjänst som grundades år 2011. Under det senaste året har inga större förbättringar till tjänsten gjorts, förutom ändringar till websidan, men deras nyhetsflöden på Fcebook och Google+ talar om en ny kommande version av projektet som går under kodnamnet ”Jupiter” (p3d.in, 2013). Skaparna av tjänsten säger sig komma från en rad olika specialiseringar inom 3D-industrin och anser att tjänsten kan skapa en omgivning passande en bred samling individer, från studeranden till produktvisualiserare eller firmor som jobbar med 3D-printande (p3d.in, 2012).

Tekniken inom p3d.in är baserad på en modifierad version av Three.js. P3D.in stöder ett allmänt 3D-format känt som Wavefront .obj som finns som exporteringsformat i de flesta programmen för 3D-grafik. Grundversionen av tjänsten stöder texturkartor för färg och reflektion och även ”bump mapping”, en teknik där en höjdkarta läggs ovanpå en modell för att ge illusionen av en mera detaljerad yta. Reflektionskartan kan påverkas av parametrarna glansighet och intensitet (se figur 10). När en modell laddas upp till p3d.in får den en designerad adress, och alla delar av projektet kan sedan ändras på utan att det måste laddas upp från början med ny adress. Detta låter användaren arbeta vidare på sina modeller och sedan enkelt uppgradera den version som skall synas på tjänsten (p3d.in, 2012). Figur 10 visar gränssnittet som används för att bestämma de filer som bildar helhetsmodellen.



Figur 10. Uppladdningsgränssnittet för en användare med grundkonto på p3d.in (p3d.in, 2013)

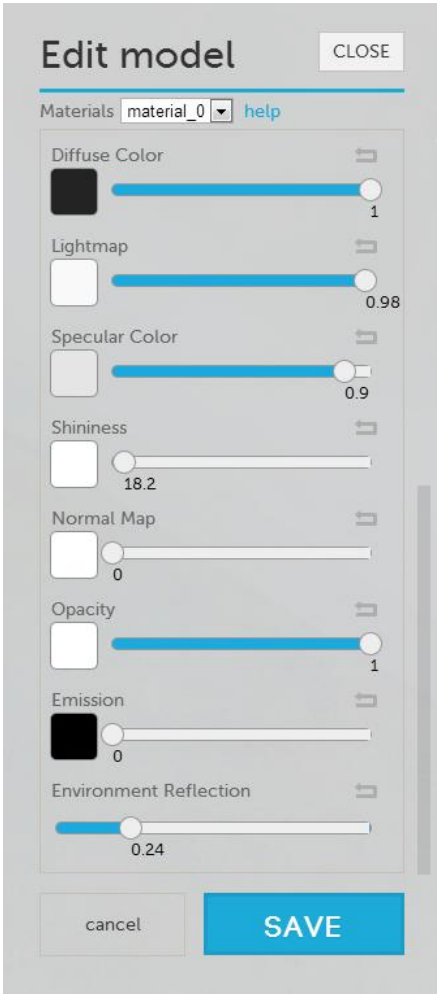
För tillfället finns det två typer av användarkonton för p3d.in. Grundkontot tillåter 100 MB med lagringsutrymme och texturstorlekar på högst 1024x1024 pixlar i jpg-format. Plus-kontot, som för tillfället är i testskedet och därför gratis, ger 2,5 GB totalt lagringsutrymme och texturstorlekar på upp till 4096x4096 pixlar. Png-formatet tillåts även för texturer för användare av plus-konto, och man får möjligheten att ladda upp texturkartor för genomskinlighet och ”ambient occlusion”, en sorts ljuskarta som försöker simulera hur naturligt ljus beter sig. Storleksbegränsningen för en enskild modell är 50 MB, oberoende av användarkonto (p3d.in, 2012).

4.2 Sketchfab

Tjänsten Sketchfab lanserades i Paris i mars 2012. Precis som p3d.in så riktar sig Sketchfab till en stor mängd olika användare, och gör det med hjälp av flera imponerande funktioner. Tjänsten har fem olika kontoplaner som börjar med en

gratisvariant och slutar med Enterprise-planen som riktar sig till större firmainitiativ och innehåller en dedikerad server och speciella arbetsverktyg som ingen annan plan har tillgång till. Tjänsten stöder hela 27 olika 3D-format och ger tillgång till sex olika exporteringsprogram som gör det enkelt att ladda upp modeller om man arbetar med 3ds Max, Maya, Cinema4D, Blender, Solidworks eller Sketchup. De två lägsta kontoplanerna inom Sketchfab har samma storleksbegränsning på modeller som p3d.in, dvs 50 MB, men mängden modeller man får ladda upp är obegränsad. De två följande planerna tillåter modeller på upp till 200 MB att laddas upp och den högsta gradens plan saknar definierad övre gräns (Sketchfab, 2013a).

Till skillnad från p3d.in och Verold Studio, som tas upp i följande delkapitel, så är tekniken bakom Sketchfab inte baserad på Three.js. Istället är Sketchfab baserad på ett WebGL-ramverk som kallas OSG.JS som är skapat av en av Sketchfabs grundare Cédric Pinson. OSG.JS är en Javascriptversion av OpenSceneGraph, ett verktygspaket för 3D-grafik inom OpenGL, som också är skapat av Pinson, som har 12 års erfarenhet av 3D-utveckling. Sketchfab satsar stort på sitt initiativ att hämta 3D till en stor publik, vilket sätter stress på tjänstens system som måste vara kapabelt att behandla en stor mängd modeller så fort som möjligt, och skapa versioner av olika grader av detalj för att kunna tillmötesgå ett stort antal kunder med maskiner av varierande kapacitet (Dorard, 2012). Sketchfab har också en speciell plan riktad åt dem som vill använda tjänsten som grund för sin portfolio. ME-planen som kostar endast fem euro per månad ger tillgång till en egen sida för ens modeller att visas upp på. Sidan har ett smidigt simpelt användargränssnitt som gör det enkelt för folk att kolla på ens material och ett inbyggt kontaktsystem gör att potentiella arbetsgivare lätt får kontakt med en direkt från portfolion (Sketchfab, 2013b)



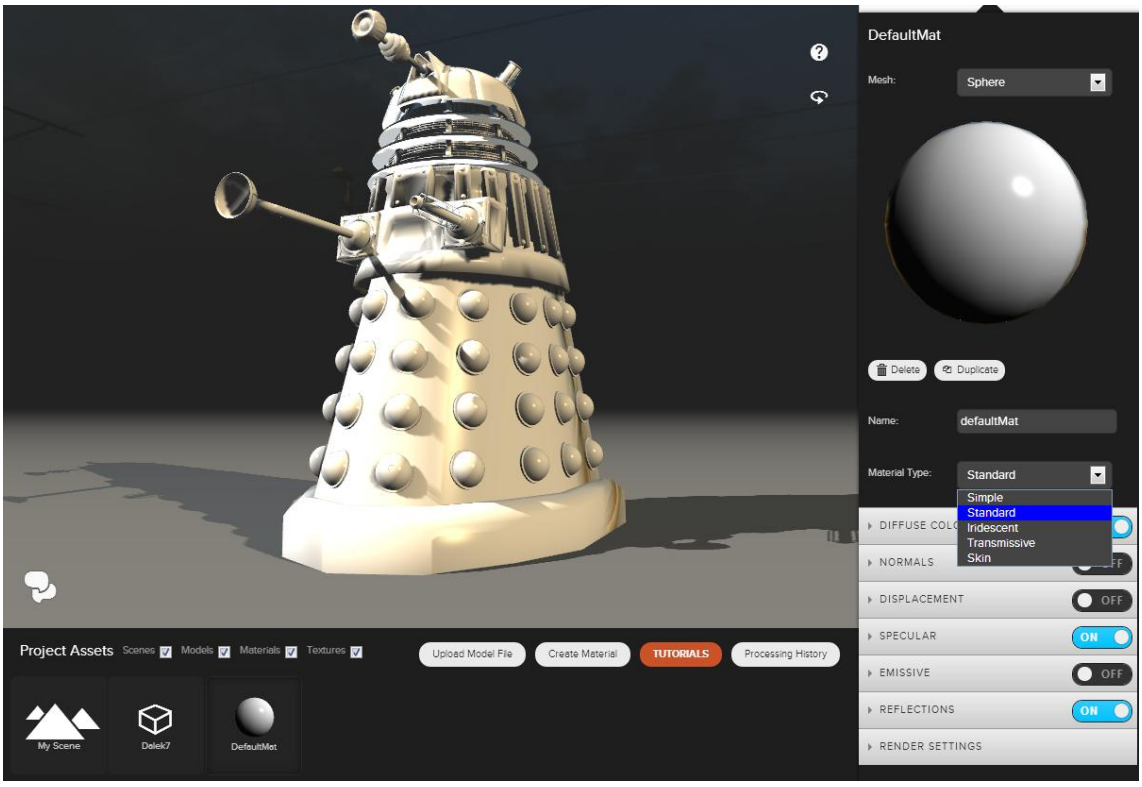
Figur 11. Materialinställningar från Sketchfabs användargränssnitt (Sketchfab, 2013a).

Möjligheterna till texturkartor motsvarar de inom p3d.in, med undantaget att alla användarkonton har möjligheten att ladda upp texturer i png-format. Sketchfab har några ytterligare funktioner såsom möjligheten till emissionstexturer och bilder på omgivningar som ger reflektioner åt renderingarna. I figur 11 syns alla inställningar för ett material inom Sketchfab. Alla materialvärden kan justeras vilket ger tillgång till ett stort antal möjliga konfigurationer (Sketchfab, 2013a).

4.3 Verold Studio

Verold Studio är en 3D-tjänst med en annorlunda affärsplan än p3d.in eller Sketchfab som fokuserar sig på att visa upp modeller. Verold Studio som lanserades under sommaren 2012 satsar på en plattform där flera användare kan se modeller i samma

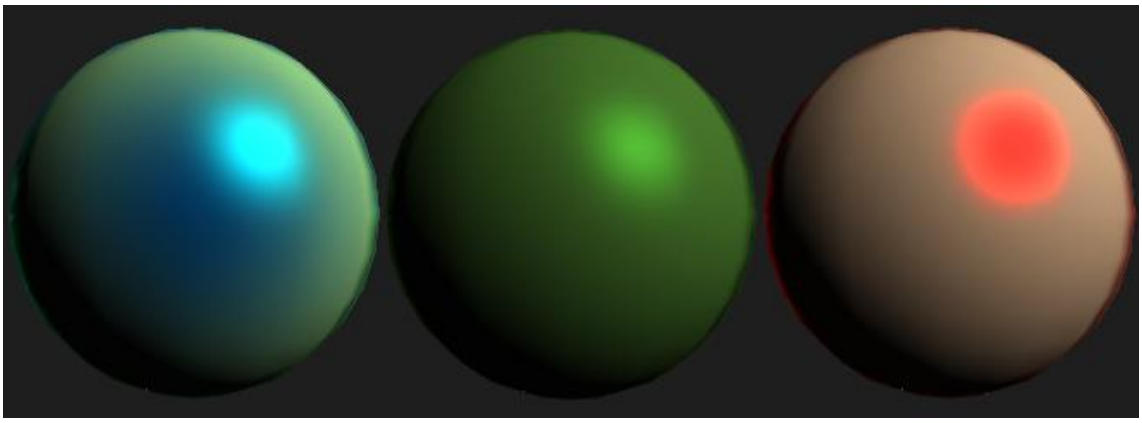
fönster och arbeta på dem i realtid. Meningen är att detta skall förenkla arbetsflödet för grupper som arbetar med 3D-modeller genom att ta bort skedet där alla skickar filer och testrenderingar fram och tillbaka och ersätta det med en gemensam miljö där de olika parterna både kan arbeta med material i realtid inom tjänsten och editera olika delar av modellen och uppdatera den utan att en enda fil måste skickas mellan de enskilda involverade (Verold, 2013). Miljön har även testats inom undervisning och har stor potential bland studeranden som via Verold Studio får ett mycket enkelt sätt att kommunicera och dela med sig av sina projekt till varandra eller till sina lärare (McLean, 2012). Verold Studio håller också på att utvidgas med ett API baserat på Three.js som kommer att låta användarna skapa interaktiva element och kombinera dem med material från Verold Studio (McKegney, 2013).



Figur 12. Bild på Verold Studios användargranssnitt. På den högra kanten synns olika kategorier av materialinställningar medan bottenraden listar alla resurser som hör till projektet (Saxén, 2013).

En annan av Verold Studios styrkor är det väldigt mångsidiga materialsystemet. Även utan texturer kan Verold Studios materialsystem skapa en stor mängd olika effekter genom att kombinera flera färger och olika sorters reflektion (se figur 13), och nya material kan skapas direkt i gränssnittet. Egenskaperna av dessa material kan sedan

kombineras med texturer. Om en textur har ett alpha-värde kan den användas till att skapa ett filter för andra material. Koordinaterna och storleken på texturerna kan även ändras inom gränssnittet för att få dem att passa modellen så bra som möjligt (Bond, 2012).



Figur 13. Exempel på material som lagats med Verold Studios verktyg (Saxén, 2013).

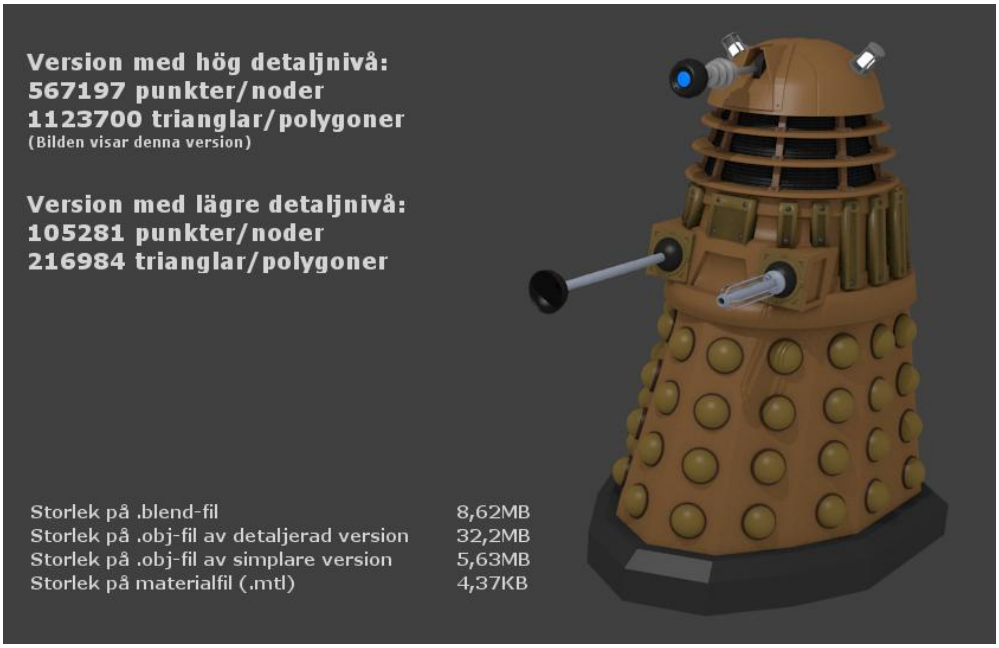
5 TESTNING

5.1 Utgångsläge

Avsikten med testandet var utöver att kontrollera funktionaliteten på de tidigare nämnda tjänsterna att huvudsakligen kontrollera den ansträngning WebGL-baserad grafik sätter på datorns grafikkort. För testandet valde jag en 3D-modell jag hade lagad från tidigare, som med sina 1,12 miljoner polygoner skulle ställa höga krav på rendering i realtid (se figur 14). Tekniska specifikationer på datorn som använts till testningen finns i bilaga 1. Ansträngningen av hårdvaran övervakades med hjälp av programmet Process Explorer. Google Chromes inbyggda webutvecklingsverktyg gav en uppfattning av mängden data som hämtades för varje modell och vilken tid som behövdes för startandet av tjänsterna.

Prestandan är inte mätt från tjänsternas egna sidor, utan alla tre tjänsters visningsfönster har lagts in i mina egna webbsidor för att ge en så simpel testmiljö som möjligt utan extra funktioner eller plug-ins som kan påverka mätningarna. Webbläsaren Google Chrome som användes till testerna startades om före varje individuellt test och hade sin

cache borttagen, för att garantera att laddningstiderna var korrekta och att grafikminnet från tidigare test skulle frigöras.



Figur 14. Bild på och information om modellen som använts för prestandatest (Saxén, 2013)

5.2 p3d.in

Av de tre testade lösningarna är p3d.in klart den enklaste, men den gör sitt arbete utan problem, och lägger minst stress på grafikkortet. Som det syntes på figur 10 i kapitel 4.1 så ger inte p3d.in väldigt många möjligheter att påverka materialfärger eller –egenskaper när man laddar upp en modell, men sju olika shaders finns tillgängliga i visningsfönstret som kan visa modellen bland annat med en standard mjuk shader (som i figur 15) eller genomskinlig eller som wireframe. Detta gör p3d.in till den lösning som ger användarna mest möjligheter att ändra på modellens utseende i realtid. Att navigera sig runt i visningsfönstret går smidigt med de tre möjliga kommandona som finns.

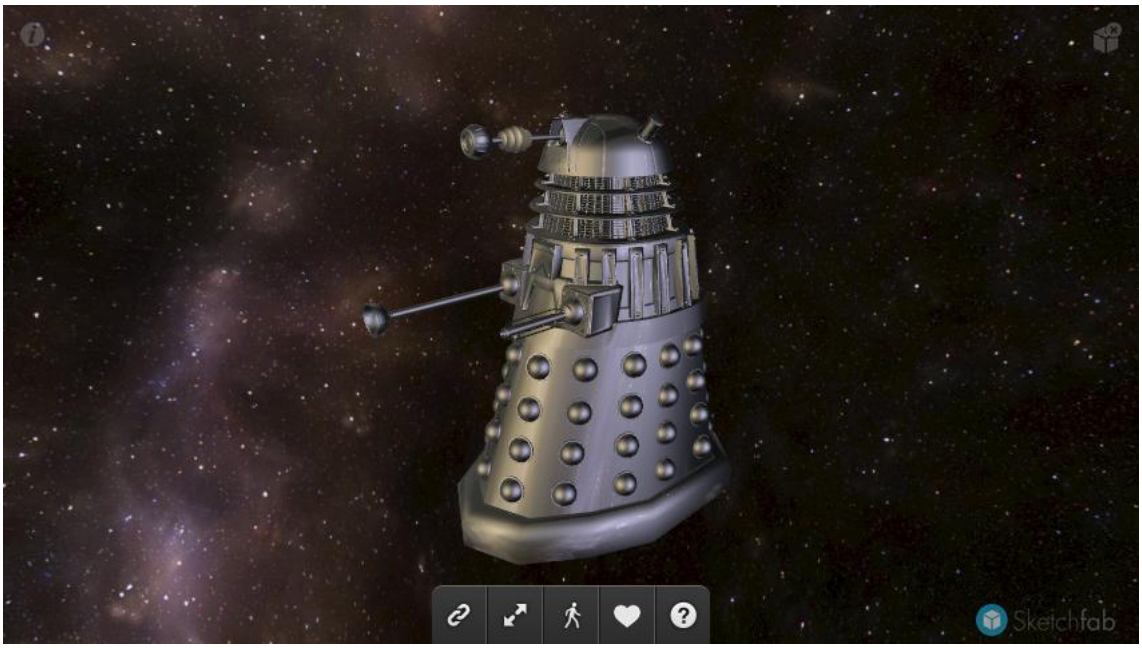
Figur 15. Testmodellen i p3d.in. Till vänster synns shader-alternativ (Saxén, 2013).

När det kommer till prestanda lägger p3d.in minst tryck på grafikkorten bland de tre testade alternativen. Bilaga 2 har skärmbilder och information av testresultaten av p3d.in. P3d.in är också det enda av alternativen som inte lägger konstant tryck på grafikkretsen, utan den används endast då det sker interaktion mellan användaren och fönstret. När ingen interaktion skedde låg användningen av grafikkorten på ungefär 2,5 % medan den steg till 25 % under aktiv interaktion. När det kommer till minnesanvändning var användningen av det vanliga datorminnet relativt högt jämfört med användningen av grafikminnet, med 389,8 MB av datorns RAM i användning mot 111,7 MB dedikerat grafikminne. Fönstret och innehållet laddas snabbt (på endast 8,5 sekunder) och modellfilen som ursprungligen bestod av över 30 MB har krympt till en 7,2 MB stor datafil.

5.3 Sketchfab

Det största som stod ut med den tekniska delen av Sketchfabs test var minnesanvändningen. Alla andra tjänster visade sig ha betydligt större RAM-förbrukning än dedikerat grafikminne. I fallet med Sketchfab var förbrukningen för de två minnestyperna nästan lika stor, med 263,3 MB på RAM och 231,1 MB dedikerat grafikminne. Resultaten för testet med Sketchfab finns dokumenterade i bilaga 3.

Sketchfab höll grafikkortet igång på ungefär 28,5 % vilket är lite mer än för p3d.in, med skillnaden att Sketchfab, till skillnad från p3d.in, krävde konstant användning. Tjänsten laddades också aningen långsammare är p3d.in. Hela scenen med en modellfil på 12,7 MB startade på 15,79 sekunder.



Figur 16. Testmodellen i Sketchfabs visningsfönster (Saxén, 2013)

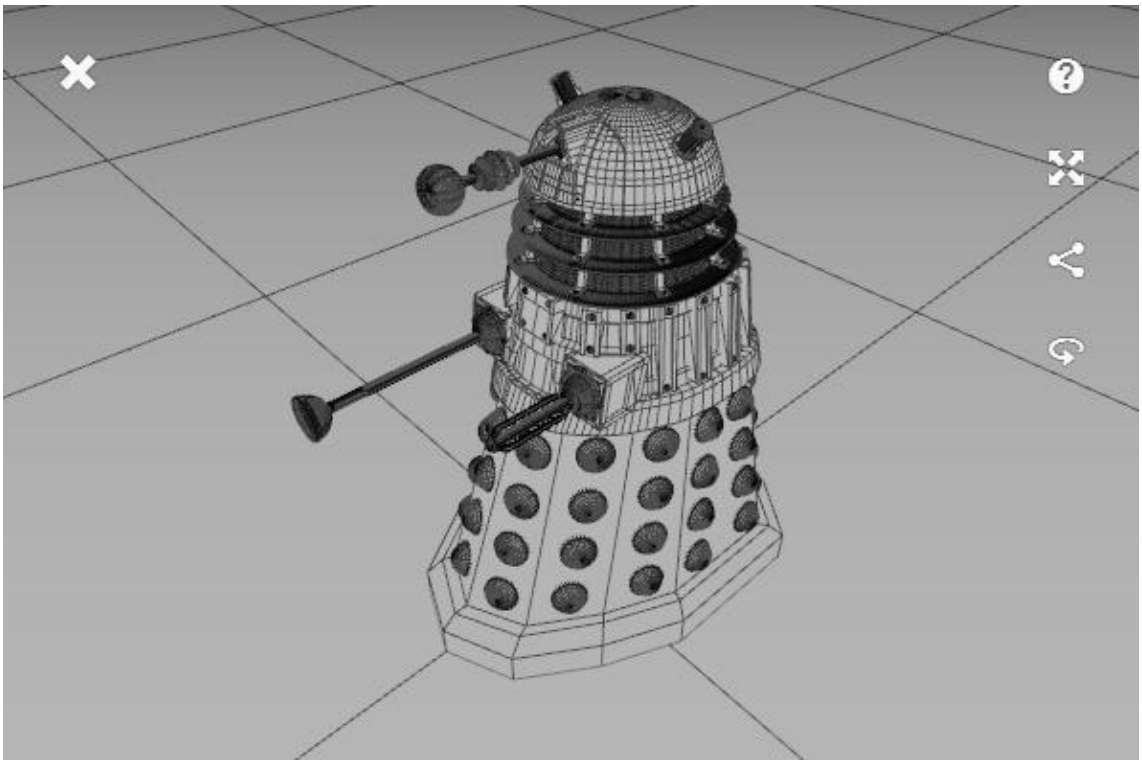
Sketchfabs visare är enkel men väldigt smidig. Kameraobjektet rör sig mjukt omkring och kan även styras i förstapersonsperspektiv. Till skillnad från de andra testade tjänsterna så har Sketchfab inte tillgång till några andra shaders i det externa visningsfönstret. Om modellen visas på tjänstens egen sida finns det ett par alternativ som att visa den utan shader eller lägga på en wireframe, men även dom alternativen är inte så imponerande jämfört med de som erbjuds av p3d.in och Verold Studio. För mera specifika utseenden på modellen är det upp till den som laddar upp modellen att fixa.

5.4 Verold Studio

Det första som bör nämnas angående Verold Studio är att min ursprungliga testmodell överskred Verolds rekommenderade maximum komplexitet med ungefär 450 %. Verold rekommenderar högst 50 enskilda objekt eller 200000 polygoner, medan min modell

stoltserade med lite på 1,12 miljoner polygoner. Jag körde oberoende Verold Studio genom samma prestandatest som de andra tjänsterna men beslöt att även göra en version som föll närmare deras rekommendation på 200000 polygoner, och resultaten av respektive test finns i bilagorna 4 och 5.

De första testerna jag gjorde med den mera detaljerade modellen visade direkt att det inte var något som stöds på ett praktiskt sätt av tjänsten. Med en laddningstid på 7,6 minuter var det det långsammaste test jag utförde, och detta var efter att jag såg till att inget annat använde sig av grafikminne eller -prestanda. Vid tidigare tester utan en optimerad webbläsare kraschade tjänsten upprepade gånger. När modellen laddats upp drev tjänsten grafikkorten konstant på omkring 90 %. Minneskonsumptionen låg på 541,2 MB, som också var det överlägset högsta värdet bland tjänsterna jag testade. Mängden dedikerat grafikminne var dock endast 238,4 MB, och fastän det också var det högsta testvärdet så låg det endast ett fåtal MB ovanför Sketchfabs resultat.



Figur 17. Den mindre detaljerade testmodellen med en wireframe shader inom Verold Studios visningsfönster (Saxén, 2013).

Den för tjänsten mer optimerade modellen gjorde betydligt bättre ifrån sig. Endast 9,73 sekunder behövdes för att ladda upp modellen, men med grafikkorten fast på 41 % konstant förbrukning blev det klart att Verold Studio är en 3D-tjänst som ställer krav. Minnesförbrukningen sjönk till lite under hälften av de tidigare testvärdena för både RAM och grafikminne, som fick värdena 213,9 MB och 119,5 MB. Båda testen med Verold Studio tydde på att den förbrukar ungefär dubbelt så mycket RAM som grafikminne.

Utseendemässigt är Verold Studio imponerande. Med sitt 4-punkts ljussystem erbjuder det den intressantaste ljussättningen av alla de testade lösningarna och med sitt stora utbud av materialinställningar är det inte underligt att grafikkorten måste kämpa extra hårt för att leverera en färdig rendering. Precis som p3d.in så kommer också Verold Studios visningsfönster med ett antal shaders, av vilka kan nämnas wireframe som synns i figur 17, en shader som visar bumpmapping, alltså höjdkartor, och en vy för UV-kartor, som kan förklaras som kartor med texturkoordinater. Trots sitt huvudsyfte att vara samarbetesverktyg fungerar Verold Studio bra som visare för 3D-modeller, om man har lite kraft i sin dator.

5.5 Resultat

Prestandamässigt blev det väldigt olika resultat. P3d.in ställde de minsta kraven på grafikkretsen men var en relativt stor RAM-konsument. Sketchfab var effektivast på att utnyttja grafikminnet, och Verold Studio ställde stora krav både på grafikkorten och komplexiteten på modellen som skulle visas. P3d.in och Verold Studio, som båda är baserade på Three.js, visade liknande minneskarakteristik där mängden använt RAM alltid var åtminstone dubbelt så stor som mängden dedikerat grafikminne. P3d.in var enda tjänsten som inte höll en konstant belastning på grafikkorten. Samtliga tjänster slutade belasta grafikkorten helt när webläsarfönstret minimerades eller en annan flik av webbläsaren sattes i fokus.

Något som i alla fall är entydigt är att hårdvaruaccelererad 3D-grafik med WebGL är en fungerande teknologi. Ingen av tjänsterna satte processorn i arbete utan höll sig till

grafikkorten. Möjligheten till att kunna visa upp krävande 3D-grafik i en webbläsare utan plug-ins har blivit verklighet, och kommer förhoppningsvis att öka i synlighet.

6 SLUTSATSER

3D-grafik inom HTML5 är ett relativt nytt ämne med en stor potential. Tekniken drivs framåt av passionerade grupper och individer och kommer utan tvekan att börja synas allt mer inom de närmaste åren. Jag tror starkt på att tjänster såsom p3d.in och Sketchfab kommer att grunda en ny standard för hur 3D-produkter visas på nätet. Varför nöja sig med några stillbildsrenderingar om man kan få en fullständig modell att vrida på? Jag har själv blivit övertygad att börja jobba på en 3D-portfolio som jag högst antagligen kommer att publicera på Sketchfab.

Jag ser också ett sammanhang mellan denna teknologi och den ökade entusiasmen kring 3D-printrar. Den dag det blir vanligt att köpa 3D-modeller till sin egen 3D-printer kommer man garanterat att villa se vad man köper, så dessa två teknologier kommer garanterat att börja dra mer nytta av varandra. Jag ser även en framtid i Verold Studio med sitt samarbetesgränssnitt. Jag har själv arbetat med 3D-projekt där jag varit beroende av andras kommentarer och möjligheten att visa dem en fullständig modell och få direkt stöd över en webtjänst skulle ha underlättat mitt arbete ordentligt. Industrin kan dra stor nytta av att ha ordentliga samarbetsgränssnitt som följer funktionaliteten inom Verold Studio.

Det enda jag ser som oroande är attityden som Microsoft har visat mot WebGL. De hävdar att det är osäkert, och som följd av det är Internet Explorer nu den enda av de moderna webläsarna som inte har stöd för WebGL. Med tanke på den procent som fortfarande använder Internet Explorer som webbläsare så hoppas jag att Microsoft och WebGL kommer överens någon dag. En öppen standard för web-relaterad 3D-grafik som är accepterad av alla är vad som behövs för att hålla innovasionen igång.

KÄLLOR / REFERENCES

Anyuru, Andreas. 2012, *Professional WebGL Programming: Developing 3D Graphics for the Web*, John Wiley & Sons Ltd, 361 s.

Baecker, Joachim. 2005, [www]. Tillgänglig: http://commons.wikimedia.org/wiki/File:Perspective_Projection_Principle.jpg Hämtad 2.4.2013.

Boesch, Florian. 2013, *WebGL stats*. Tillgänglig: http://webglstats.com/ Hämtad 10.4.2013.

Bond, Michael. 2012, *Making Your Models Look Awesome – Materials Part 2*. Tillgänglig: http://verold.com/blog/2012/10/25/making-your-models-look-awesome-materials-part-2 Hämtad 13.4.2013.

Cabello, Ricardo. 2013a, *mrdoob / three.js*. Tillgänglig: https://github.com/mrdoob/three.js Hämtad 7.3.2013.

Cabello, Ricardo. 2013b, *Features*. Tillgänglig: https://github.com/mrdoob/three.js/wiki/Features Hämtad 7.3.2013.

Dorard, Louis. 2012, *Sketchfab, the "Youtube of 3D content", goes Pro*. Tillgänglig: http://www.rudebaguette.com/2012/08/21/sketchfab/ Hämtad 2.4.2013

Haapoja, Mikko. 2011a, *Build your own HTML5 3D engine*. Tillgänglig: http://www.netmagazine.com/tutorials/build-your-own-html5-3d-engine Hämtad 29.3.2013.

Haines, Eric. 2013, *Interview with three.js Creator*. Tillgänglig: http://www.realtimerendering.com/blog/interview-with-three-js-creator/ Hämtad 1.4.2013.

Khronos Group. 2011, *WebGL Specification*. Tillgänglig: https://www.khronos.org/registry/webgl/specs/1.0/ Hämtad 14.3.2013.

Khronos Group. 2012, *Getting a WebGL Implementation*. Tillgänglig: http://www.khronos.org/webgl/wiki/Getting_a_WebGL_Implementation Hämtad 9.4.2013

Khronos Group. 2013, *User Contributions*. Tillgänglig: http://www.khronos.org/webgl/wiki/User_Contributions Hämtad 2.4.2013.

Lewis, Paul. 2011, *An Introduction to Shaders*. Tillgänglig: http://www.html5rocks.com/en/tutorials/webgl/shaders/ Hämtad 2.4.2013.

Lubbers, Peter; Albers, Brian & Salim, Frank. 2011, *Pro HTML5 Programming, Second Edition*, 2 uppl., Apress Media LLC, 323 s.

McKegney, Ross. 2013, *Announcing The Verold Developer API!* Tillgänglig: http://verold.com/blog/2013/2/8/announcing-the-verold-3d-engine Hämtad 13.4.20163

McLean, Aaron. 2012, *Case Study: Verold Studio in The Classroom At Seneca Colege*. Tillgänglig: http://verold.com/blog/2012/12/17/verold-studio-at-seneca-college Hämtad 13.4.2013.

OpenGL. 2012, *OpenGL Wiki – Rendering Pipeline Overview*. Tillgänglig: http://www.opengl.org/wiki/Rendering_Pipeline_Overview Hämtad 8.4.2013.

OpenGL. 2013, *OpenGL Wiki - Shader*. Tillgänglig: http://www.opengl.org/wiki/Shader Hämtad 2.4.2013.

p3d.in. 2012, *FAQ*. Tillgänglig: http://p3d.in/faq Hämtad 29.3.2013.

p3d.in. 2013, *p3d.in – Google+*. Tillgänglig: https://plus.google.com/118252342153135169696/posts Hämtad 12.4.2013.

Roast, Kevin. 2013, *K3D JavaScript Canvas Library*. Tillgänglig: http://en.wikibooks.org/wiki/K3D_JavaScript_Canvas_Library Hämtad 2.4.2013.

Saxén, Rasmus. 2013, *Material för slutarbete*. Tillgänglig: http://people.arcada.fi/~saxenras/slutarbete Hämtad 2.4.2013.

Sights. 2013, *THE HTML5 TEST*. Tillgänglig http://html5test.com/results/desktop.html Hämtad 9.4.2013

Sketchfab. 2013a, *FAQ*. Tillgänglig: https://sketchfab.com/faq Hämtad 2.4.2013.

Sketchfab. 2013b, *Your professional 3D Portfolio website*. https://sketchfab.com/portfolio?ref=upgrade Hämtad 13.4.2013

Sloane, Andy. 2011, *Donut math: how donut.c works*. Tillgänglig: http://www.a1k0n.net/2011/07/20/donut-math.html Hämtad 2.4.2013.

Verold. 2013, *Verold*. Tillgänglig: verold.com Hämtad 13.4.2013.

W3C, 2012a, *HTML5 Definition Complete, W3C Moves to Interoperability Testing and Performance*. Tillgänglig: http://www.w3.org/2012/12/html5-cr Hämtad 8.4.2013

W3C, 2012b, *HTML5, W3C Candidate Recommendation 17 December 2012*. Tillgänglig: http://www.w3.org/TR/html5/embedded-content-0.html#the-canvas-element Hämtad 9.4.2013

Walker, Cody. 2011, *Working with Orthographic Projections and Basic Isometrics*. Tillgänglig: http://vector.tutsplus.com/tutorials/illustration/working-with-orthographic-projections-and-basic-isometrics/ Hämtad 2.4.2013.

BILAGOR

Bilaga 1 – Tekniska specifikationer för testdatorn

Bilaga 2 – Prestandaresultat från p3d.in

Bilaga 3 – Prestandaresultat från Sketchfab

Bilaga 4 – Prestandaresultat från Verold Studio

Bilaga 5 – Prestandaresultat 2 från Verold Studio

Bilaga 1

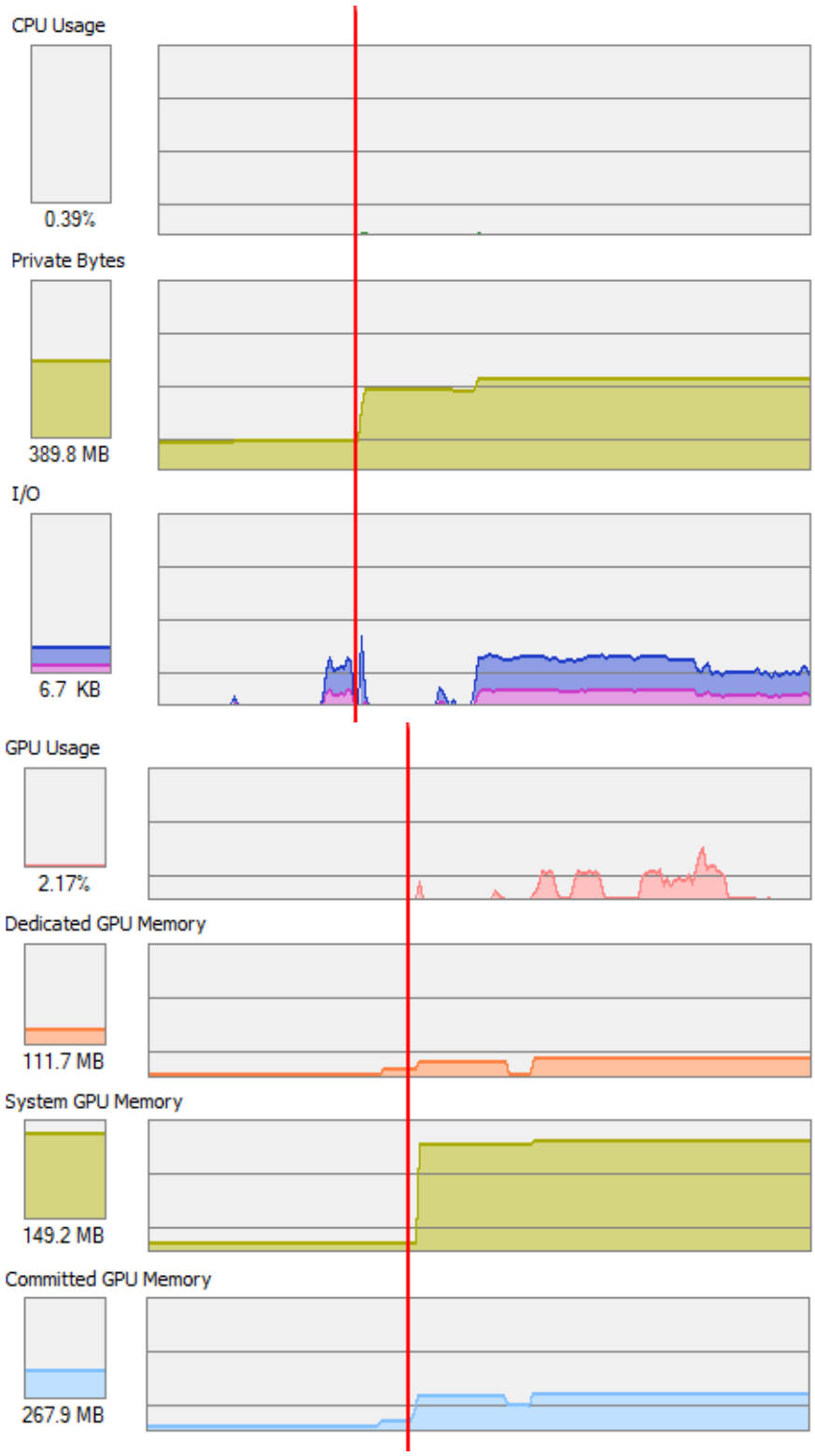
Tekniska specifikationer för testdatorn

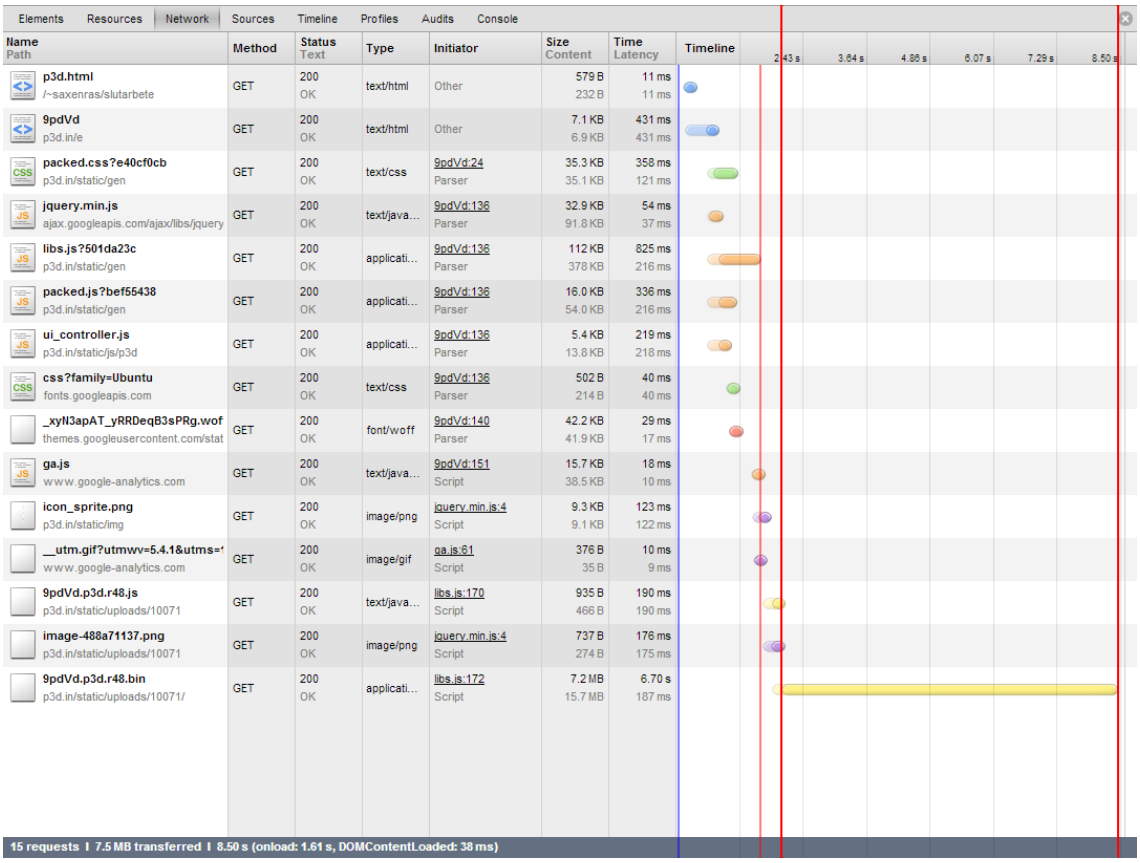
Moderkort	Asus P8P67 Deluxe
CPU	Intel i7 2600K 3,4GHz
Minne	G-Skill 4x4GB Ripjaws DDR3 1600MHz
GPU	2 x EVGA NVIDIA GeForce GTX460 SE i SLI-konfiguration
Webbläsare	Chrome 25
Internethastighet	10 Mb/s upp, 10 Mb/s ner

Bilaga 2

Prestandaresultat från p3d.in.

Taget från http://people.arcada.fi/~saxenras/slutarbete/p3d.html





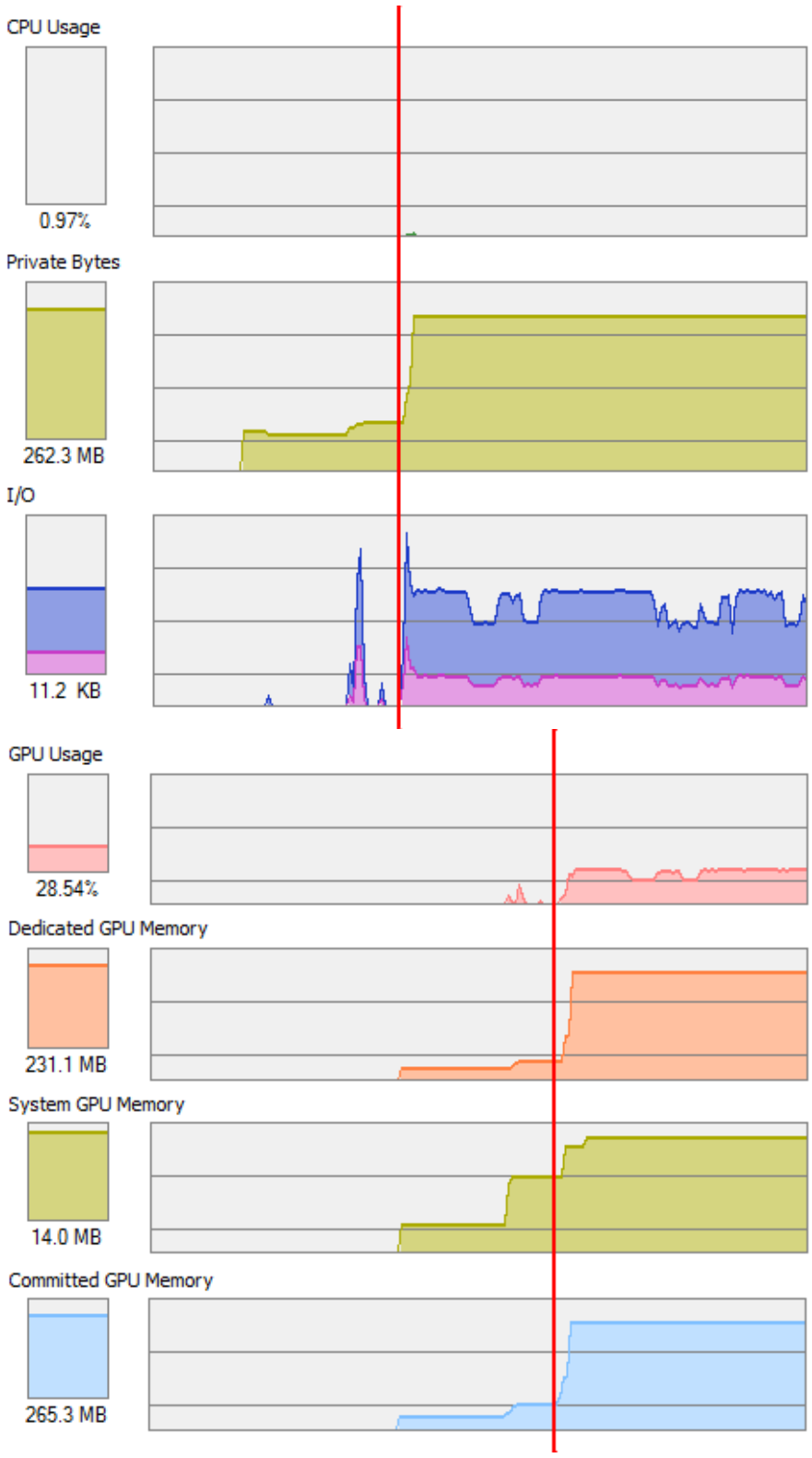
De två första bilderna är skärmbilder från Process Explorer och visar resursanvändningen på processorn, minnet och grafikkorten. Den röda linjen visar var på tidslinjen modellen var nerladdad och började visas. Skärmbilden ovanför är från Google Chromes webutvecklingsverktyg och visar en tidslinje med trafiken till websidan från och med att modellen började laddas ner. Området mellan de två röda linjerna längst till höger visar den del av tiden som behövdes för nerladdning av själva modellen.

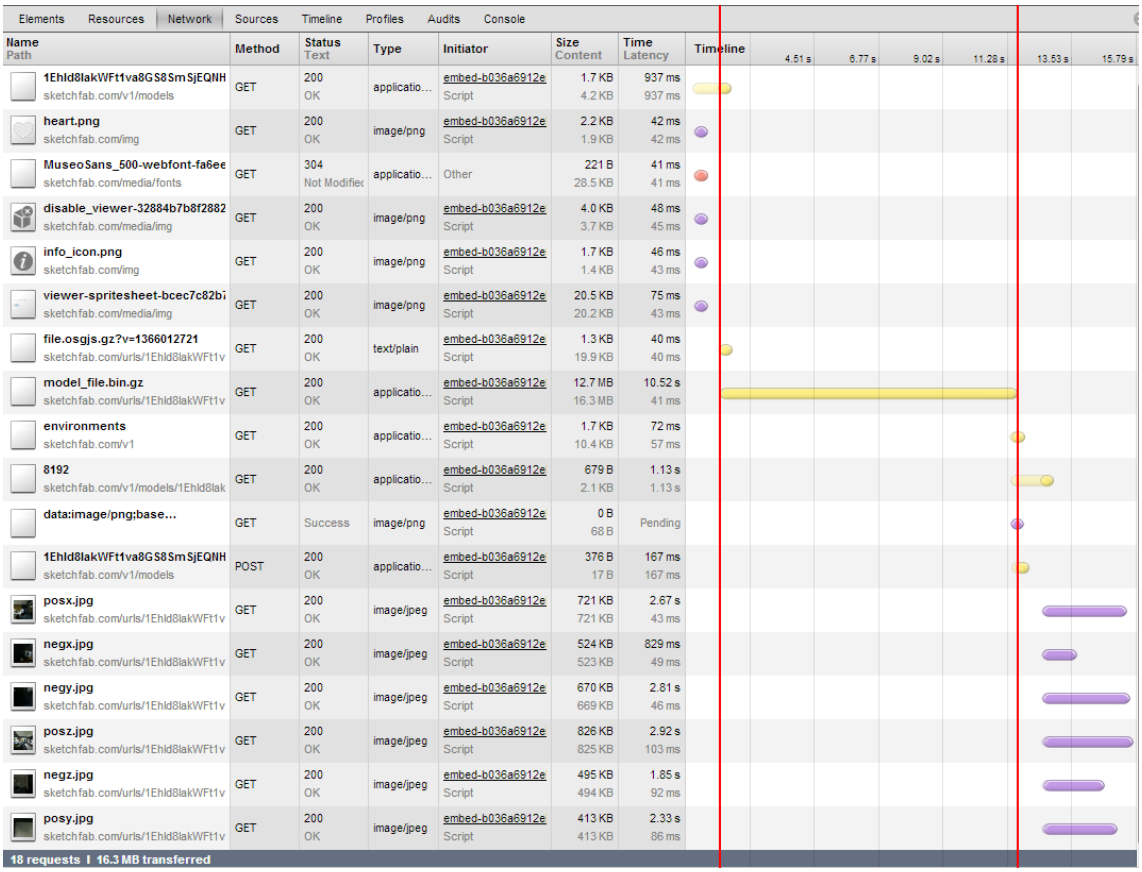
Storlek på modellfilen som laddades ner:	7,2 MB
Laddningstid för sidan:	8,5 sekunder (6.7 för modellen)
Minnesanvändning:	389,8 MB
Grafikkortsbelastning:	~2,5 % i viloläge, ~25 % under interaktion
Grafikminnesanvändning:	111,7 MB

Bilaga 3

Prestandaresultat från Sketchfab.

Taget från http://people.arcada.fi/~saxenras/slutarbete/sketchfab.html





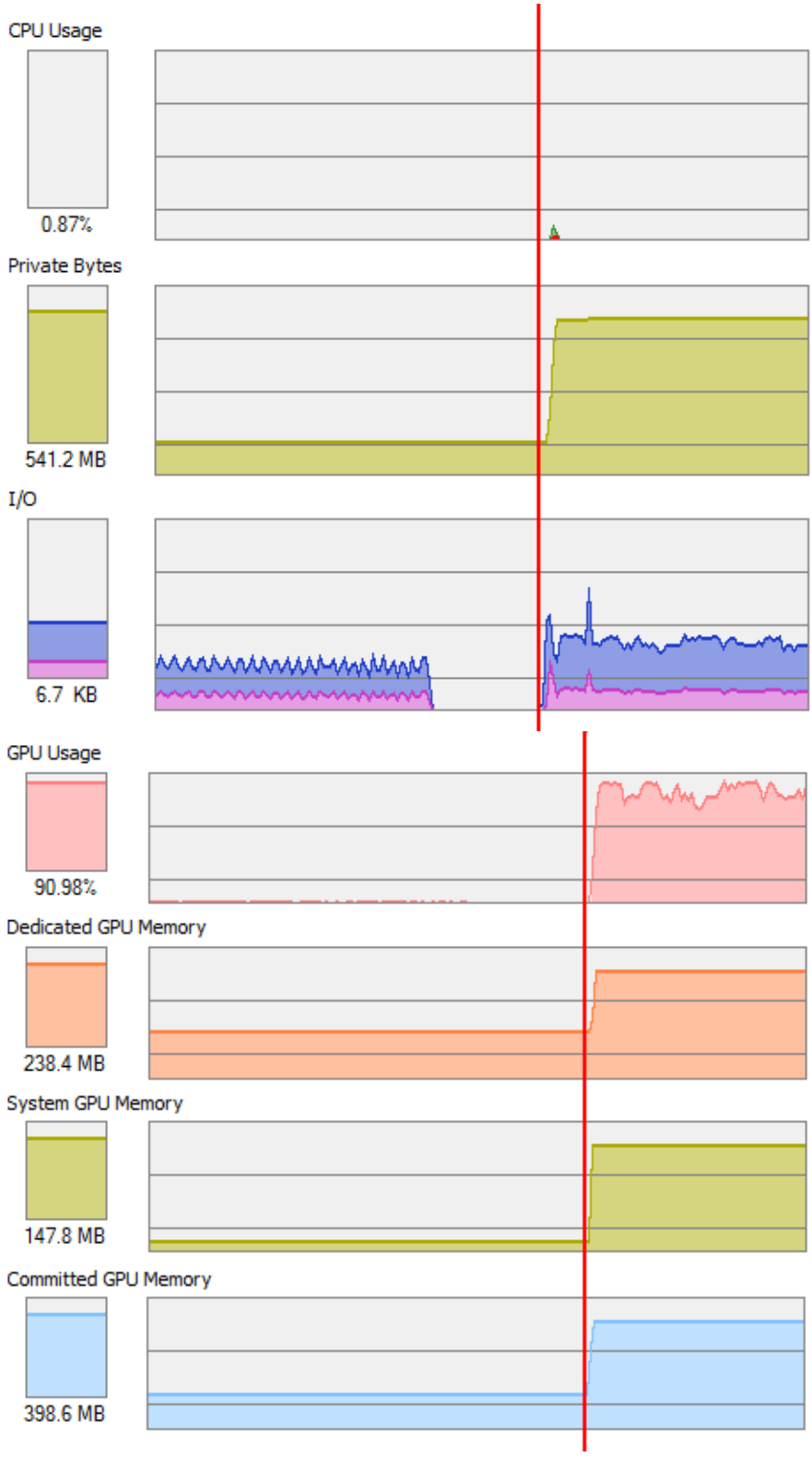
De två första bilderna är skärmbilder från Process Explorer och visar resursanvändningen på processorn, minnet och grafikkorten. Den röda linjen visar var på tidslinjen modellen var nerladdad och började visas. Skärmbilden ovanför är från Google Chromes webutvecklingsverktyg och visar en tidslinje med trafiken till websidan från och med att modellen började laddas ner. Området mellan de två röda linjerna visar den del av tiden som behövdes för nerladdning av själva modellen.

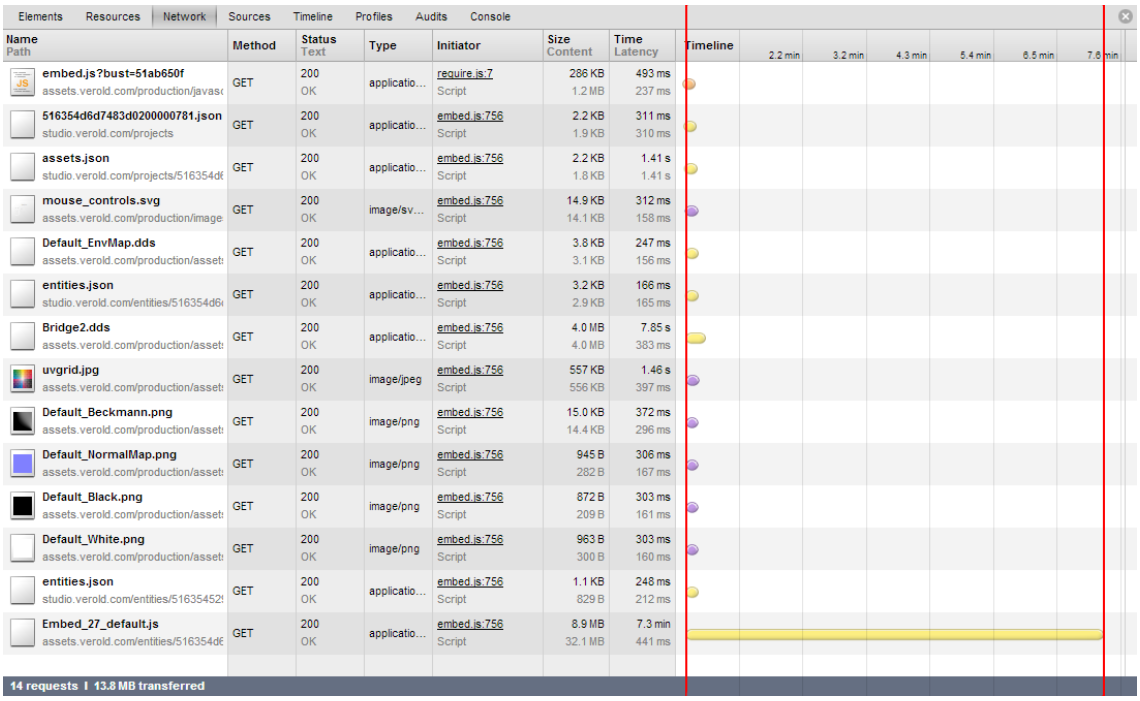
Storlek på modellfilen som laddades ner:	12,7 MB
Laddningstid för sidan:	15,79 sekunder (10,52 för modellen)
Minnesanvändning:	262,3 MB
Grafikkortsbelastning:	~28,5 %
Grafikminnesanvändning:	231,1 MB

Bilaga 4

Prestandaresultat från Verold Studio. Gjort med modell av högre detaljnivå.

Taget från http://people.arcada.fi/~saxenras/slutarbete/verold.html





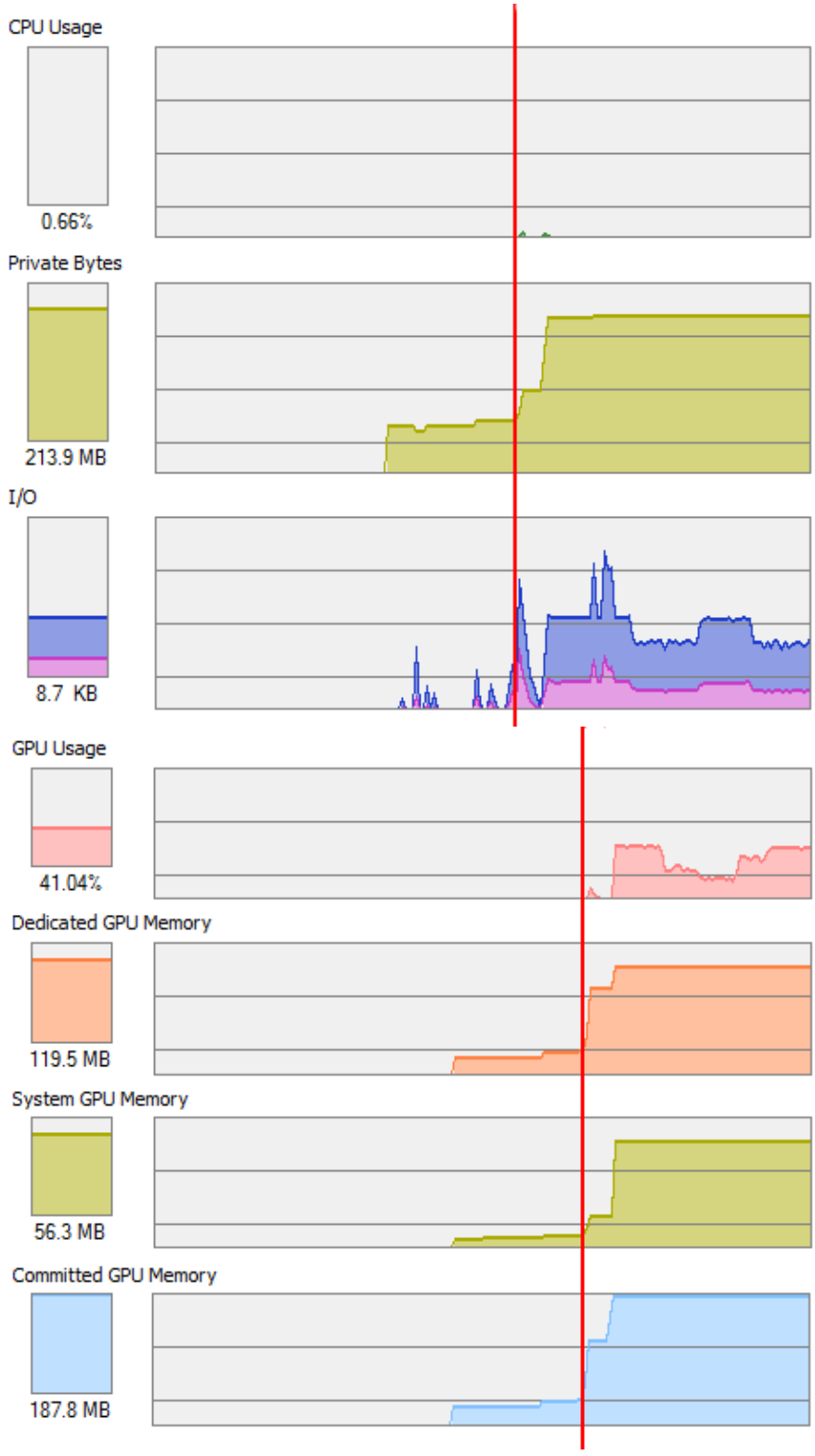
De två första bilderna är skärmbilder från Process Explorer och visar resursanvändningen på processorn, minnet och grafikkorten. Den röda linjen visar var på tidslinjen modellen var nerladdad och började visas. Skärmbilden ovanför är från Google Chromes webutvecklingsverktyg och visar en tidslinje med trafiken till websidan från och med att modellen började laddas ner. Området mellan de två röda linjerna visar den del av tiden som behövdes för nerladdning av själva modellen.

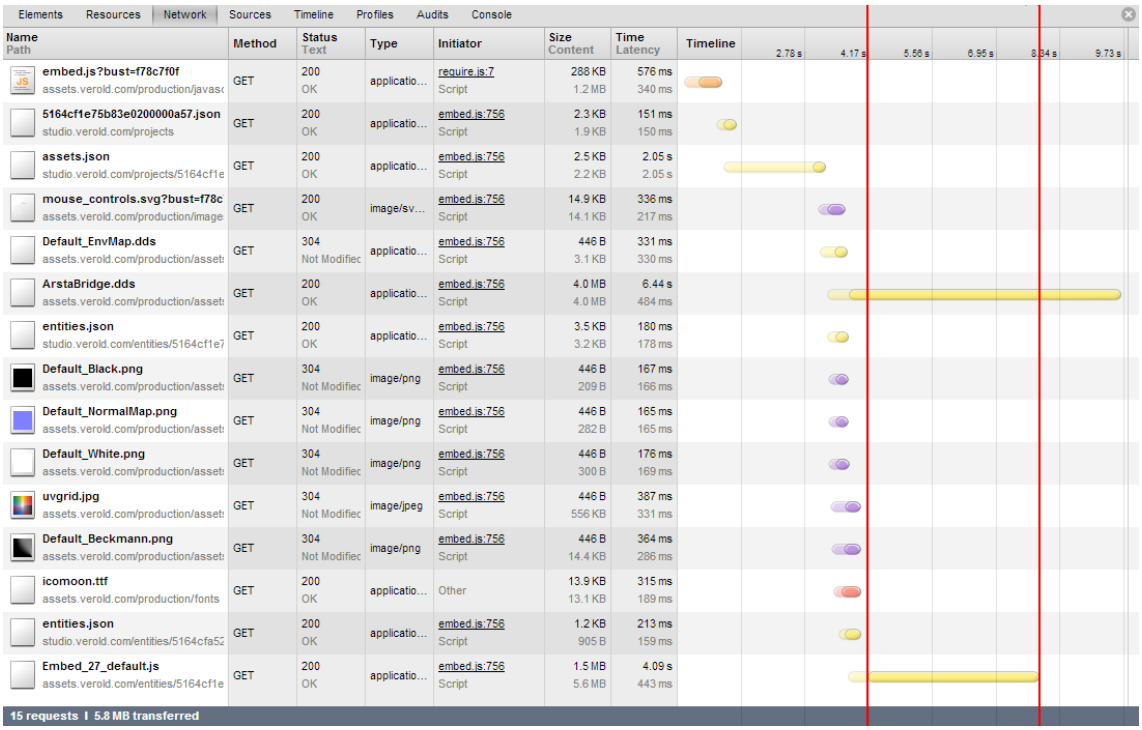
Storlek på modellfilen som laddades ner:	8,9 MB
Laddningstid för sidan:	7,6 minuter (7,3 för modellen)
Minnesanvändning:	541,2 MB
Grafikkortsbelastning:	~92 %
Grafikminnesanvändning:	238,4 MB

Bilaga 5

Prestandaresultat 2 från Verold Studio. Gjort med modell av lägre detaljnivå.

Tagna från http://people.arcada.fi/~saxenras/slutarbete/verold.html





De två första bilderna är skärmbilder från Process Explorer och visar resursanvändningen på processorn, minnet och grafikkorten. Den röda linjen visar var på tidslinjen modellen var nerladdad och började visas. Skärmbilden ovanför är från Google Chromes webutvecklingsverktyg och visar en tidslinje med trafiken till websidan från och med att modellen började laddas ner. Området mellan de två röda linjerna visar den del av tiden som behövdes för nerladdning av själva modellen.

Storlek på modellfilen som laddades ner:	1,5 MB
Laddningstid för sidan:	9,73 sekunder (4,09 för modellen)
Minnesanvändning:	213,9 MB
Grafikkortsbelastning:	~41 %
Grafikminnesanvändning:	119,5 MB