SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 562

Tehnike interakcije 3D objektima u web preglednicima

Marko Pilipović

Zagreb, lipanj 2013.

Zaručnici Kristini za podršku za vrijeme pisanja ovog rada.

SADRŽAJ

1. Uvod

Kad je Tim Berners-Lee napisao prvi web preglednik na svijetu, The WorldWideWeb, nije ni slutio da je gurnuo cijeli svijet u novo informatičko doba [1]. Cilj mu je bio pomoću hiperteksta povezati radove nastale na CERN-u [2]. U međuvremenu Internet je prerastao iz zbirke radova u multimedijalno informatičko središte. Web preglednici mogu prikazivati slike, reproducirati audio i video zapise, izvršavati kodove, itd.

Budući da se promijenio sadržaj web stranica, (prve web stranice nisu mogle niti prikazivati slike), bilo je potrebno mijenjati standard koji se koristi za pisanje web stranica, a tu obvezu je na sebe preuzeo World Wide Web Consortium (W3C) na čelu s Timom Bernes-Leejem. W3C kontinuirano objavljuje nove standarde na svojim stranicama i svi proizvođači poznatijih web preglednika ih se pridržavaju [3].

Najnoviji standard za web stranice, HTML5, omogućava direktno reproduciranje audio i video sadržaja, bez potrebe za programskim dodacima (engl. *plug-ins*), korištenjem oznaka <audio> i <video>. Također omogućuje web pregledniku rad s mikrofonom i web kamerom ako su priključeni na računalo koje pokreće web preglednik. No sa stajališta računalne grafike najvažnija novost u HTML5 standardu je *Canvas* element koji omogućava programsko iscrtavanje 2D i 3D sadržaja koristeći grafičku karticu računala na kojem je pokrenut web preglednik.

Pametni mobiteli i tableti predstavljaju revoluciju u pristupu Internetu, te omogućavaju pregled istog sadržaja kao i na stolnim računalima. Web aplikacije mogu se izvoditi na skoro svim uređajima koji imaju pristup Internetu i, zahvaljujući napretku HTML5 standarda, mogu potpuno uroniti korisnika u interaktivno okruženje.

U ovom radu bit će obrađena upotreba WebGL-a za prikaz i manipulaciju 3D objekata u web preglednicima. Također će biti dan kratki pregled razvoja HTML5 i WebGL tehnologija, usporedba raznih WebGL API-ja te primjer razvoja jedne WebGL aplikacije.

2. Standard HTML5

HTML (Hypertext Markup Language) je standard za prikaz dokumenata na Internetu. Glavna odlika HTML-a jest korištenje oznaka (engl. *tags*) za semantičko označavanje dokumenta, dok se za stilski dizajn koriste druge tehnologije, poput CSS-a (Cascading Style Sheets). World Wide Web Consortium određuje standard od 1993. u obliku preporuka (engl. *recommendations*) koje prihvaćaju proizvođači web preglednika i implementiraju u svojim preglednicima [3].

HTML je standard koji se s vremenom mijenja, mogućnosti se proširuju, dodaju se nove funkcionalnosti dok se neke stare ukidaju, standard se stalno razvija. Slijedi kratak pregled glavnih revizija HTML standarda [4]:

- HTML1.0 (1990) oznake za tekst, hipertext linkovi
- HTML2.0 (1995) oznake za prikaz teksta, dodane slike
- HTML3.2 (1997) dodane tablice i appleti
- HTML4.01 (1999) odvajaju se tekst i prikaz teksta, nastao CSS

Od 2008. u razvoju je HTML5 standard, čija je glavna odlika prikaz multimedije (audio/video) bez upotrebe programskih dodataka, eksternih programa ili java appleta. Također je moguće koristiti web kameru i mikrofon za hvatanje slike i zvuka, te *canvas* element za programsko iscrtavanje 2D i 3D scena.

2.1. Audio i Video elementi

Prije HTML5 standarda audio i video elementi su se ubacivali u web stranice pomoću oznake <object>. Oznaka <object> predstavlja datoteke koje nisu dio HTML standarda te da bi ih web preglednik mogao koristiti potrebno je koristiti programske dodatke. HTML5 je donio oznaku <embed> koja omogućava istu stvar.

HTML5 standard propisuje da web preglednici mogu učitavati audio i video sadržaj preko oznaka <audio> i <video>. Budući da postoji mnogo formata za audio i video datoteke, vlasnici web preglednika se nisu mogli dogovoriti da jedan format

bude službeni format ni za audio niti za video elemente, stoga su u upotrebi tri različita formata za njih.

Tablica 2.1: Audio formati [6]

Web preglednik	MP3	Waw	Ogg
Internet Explorer 9+	Da	Ne	Ne
Chrome 6+	Da	Da	Da
Firefox 3.6+	Ne	Da	Da
Safari 5+	Da	Da	Ne
Opera 10+	Ne	Da	Da

Tablica 2.2: Video formati [6]

Web preglednik	MP4	WebM	Ogg
Internet Explorer 9+	Da	Ne	Ne
Chrome 6+	Da	Da	Da
Firefox 3.6+	Ne	Da	Da
Safari 5+	Da	Ne	Ne
Opera 10+	Ne	Da	Da

Budući da web preglednici ne podržavaju iste formate, i zato što stariji preglednici ne podržavaju HTML5 elemente, potrebno je osigurati izvođenje audio/video elemenata rezervnim pozivima - ako jedan poziv ne uspije, pokušava se pokrenuti sljedeći poziv. Tako je moguće osigurati da će se željeni sadržaj izvoditi na svim preglednicima. Slijedi primjer video poziva [7]:

```
<video width="320" height="240" controls>
 <source src="movie.mp4" type="video/mp4">
 <source src="movie.ogg" type="video/ogg">
 <source src="movie.webm" type="video/webm">
 <object data="movie.mp4" width="320" height="240">
   <embed src="movie.swf" width="320" height="240">
 </object>
</video>
```

2.2. *Canvas* element

Canvas (engl. *canvas* - slikarsko platno) predstavlja dio HTML5 dokumenta po kojem je moguće crtati. Programski se definira scena koja će se iscrtati, rasterizira se, te se na koncu prikaže u dokumentu u obliku bitmap slike.

Canvas se dodaje u web stranicu sljedećim programskim isječkom:

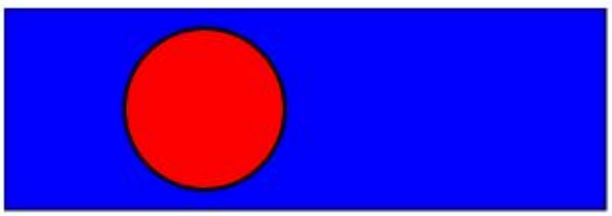
```
<canvas id="myCanvas" width="400" height="300">
 Ovaj tekst je vidljiv ako Vaš internet preglednik ne
    podržava HTML5 Canvas.
</canvas>
```

Ako web preglednik ne podržava HTML5, prikazat će se poruka između oznaka <canvas> i </canvas>. Da bi se nešto iscrtalo na *Canvasu*, potrebno je u skripti koja se vrti na stranici dohvatiti kontekst i zadati redom skup naredbi koje određuju što želimo iscrtati. Primjer jednostavnog 2D crteža:

```
<script>
 var canvas = document.getElementById('myCanvas');
 var context = canvas.getContext('2d');

 // crtanje plavog pravokutnika
 context.fillStyle = "rgb(0,0,255)";
 context.fillRect(0,0,300,100);
 context.strokeStyle = "rgb(0,0,0)";
 context.lineWidth = 1;
 context.strokeRect(0,0,300,100);

 // crtanje crvenog kruga
 context.beginPath();
 context.arc(100, 50, 40, 0, 2 * Math.PI, false);
 context.fillStyle = 'red';
 context.fill();
 context.lineWidth = 2;
 context.strokeStyle = 'black';
 context.stroke();
</script>
```

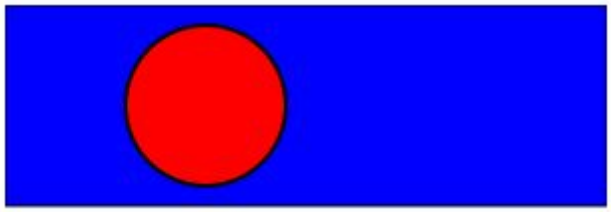


Slika 2.1: Canvas 2D crtež

Jedna primjedba *Canvasu* u 2D kontekstu je to što već postoji standard za iscrtavanje 2D oblika zvan SVG (Scalable Vector Graphics). SVG predstavlja vektorski crtež koji ne gubi na oštrini prilikom uvećavanja slike, može se napraviti izvan stranice, i

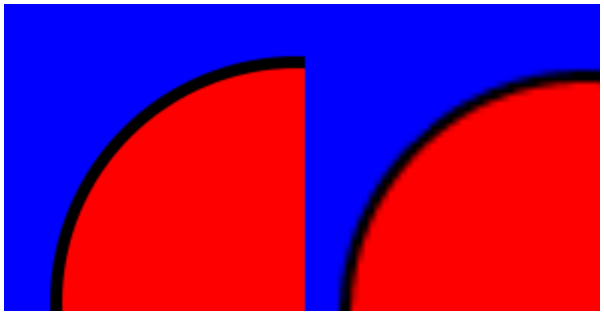
sadržava hijerarhiju elemenata od kojih je sastavljen te je moguće pristupiti pojedinim elementima s događajima (engl. *events*). Primjer jednostavnog SVG crteža:

```
<svg xmlns="http://www.w3.org/2000/svg" version="1.1">
 <rect width="300" height="100"
    style="fill:rgb(0,0,255);stroke-width:1;stroke:rgb(0,0,0)"/>
 <circle cx="100" cy="50" r="40" stroke="black"
    stroke-width="2" fill="red" />
</svg>
```



Slika 2.2: SVG crtež

Razlika se vidi tek kod uvećavanja slika:



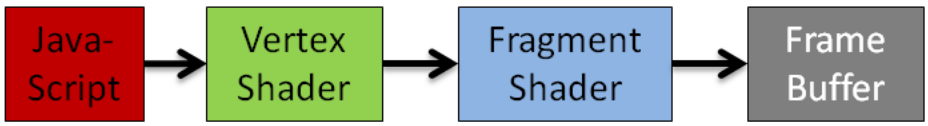
Slika 2.3: Usporedba SVG i Canvas crteža

Canvas ima svoje prednosti - brži je, ponekad želimo rasterski crtež, lako je programski promijeniti sadržaj [5]. Osim za 2D, *Canvas* se može koristiti i za iscrtavanje 3D sadržaja, korištenjem WebGL API-ja.

3. Programsko sučelje WebGL

WebGL je JavaScript API (aplikacijsko programsko sučelje) za grafiku u web preglednicima temeljena na OpenGL ES 2.0 (Open Graphics Library for Embedded Systems). Prva implementacija sklopovski podržane računalne grafike u web preglednicima bila je u *Canvas 3D*, projektu Mozillinog inženjera Vladimira Vukićevića iz 2007. [8]. Na temelju znanja stečenog iz tog projekta, 2009. je nastala Khronos Grupa[1] koja je počela razvoj WebGL-a. U ožujku 2011. je izašla verzija 1.0 WebGL specifikacije.

WebGL programi se sastoje od JavaScript skripti koje se izvode u web pregledniku i pripremaju podatke, te od programa za sjenčanje (engl. *Shaders*) koji se vrte na grafičkom sklopovlju. Programi za sjenčanje su pisani u jeziku GLSL (OpenGL Shading Language).



Slika 3.1: Grafički protočni sustav u WebGL-u

Postoje dvije vrste programa za sjenčanje - programi za sjenčanje vrhova i programi za sjenačanje fragmenata. Program za sjenčanje vrhova je programski kôd koji se izvršava za svaki vrh (engl. *vertex*), a program za sjenčanje fragmenata je kôd koji se izvršava za svaku točku (engl. *pixel* - slikovni element) konačne slike zaslona. Slika 3.1 prikazuje grafički protočni sustav u WebGL-u, odnosno komunikaciju između JavaScripta i konačnog rezultata na zaslonu:

1. JavaScript puni program za sjenčanje vrhova s poljem (engl. *buffer*) vrhova, atributima (engl. *attributes*), uniformnim (nepromjenjivim) varijablama i definira dodatne izlaze iz programa za sjenčanje vrhova u obliku *varying* varijabli.

2. Program za sjenčanje vrhova ima implicitnu *varying* varijablu gl_Position. *Varying*

[1] http://www.khronos.org/webgl/

varijable iz programa za sjenčanje vrhova se linearno interpoliraju i pridružuju rasteriziranim fragmentima.

3. Program za sjenčanje fragmenata ima implicitnu *varying* varijablu gl_FragColor s kojim definira konačnu boju fragmenta nad kojim se izvodi.

4. Spremink okvira (engl. *frame buffer*) sastavlja sliku iz svih dobivenih fragmenata (pixela) i predstavlja rezultat grafičkog protočnog sustava.

Primjer programa za sjenčanje vrhova:

```
<script id="shader-vs" type="x-shader/x-vertex">
 attribute vec3 aVertexPosition;
 attribute vec4 aVertexColor;
 uniform mat4 uMVMatrix;
 uniform mat4 uPMatrix;
 varying vec4 vColor;
 void main(void) {
   gl_Position = uPMatrix * uMVMatrix *
      vec4(aVertexPosition, 1.0);
   vColor = aVertexColor;
 }
</script>
```

Primjer programa za sjenčanje fragmenata:

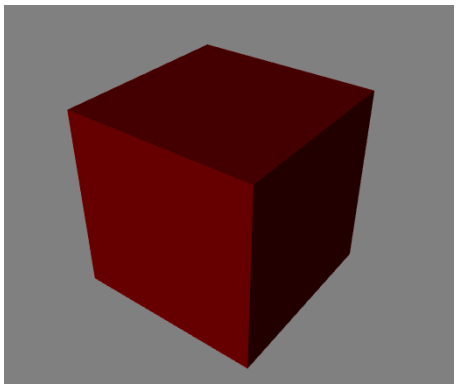
```
<script id="shader-fs" type="x-shader/x-fragment">
 varying vec4 vColor;
 void main(void) {
   gl_FragColor = vColor;
 }
</script>
```

U dodatku A nalazi se kôd za iscrtavanje jednog trokuta u WebGL-u [9]. Sastoji se od preko stotinu linija kôda, te je očito da je teško napisati složenu aplikaciju koristeći samo WebGL. Stoga postoje Javascript paketi koji olakšavaju programiranje u WebGL-u koji s jednostavnim naredbama zamjenjuju desetke linija kôda čistog WebGL-a.

3.1. Grafički JavaScript paketi više razine

Bilo tko može napraviti svoj grafički paket u JavaScriptu, ali to ne znači da su svi paketi jednaki. Kao sa svakom web tehnologijom, programeri mogu pisati svoje pakete koji im pomažu u radu ili koje objavljuju da ih drugi ljudi mogu koristiti. Dobra dokumentacija, širok spektar funkcionalnosti i razumljiv kôd su neki od uvjeta koji određuju hoće li paket opstati ili postati neupotrebljiv.

U sljedećim pododjeljcima bit će dana usporedba nekoliko različitih grafičkih paketa, a u dodatku B se za svaki paket nalaze kodovi potrebni da se iscrta osjenčana crvena kocka, kao na slici 3.2.



Slika 3.2: Rezultat izvođenja kodova iz dodatka B

3.1.1. Paket CopperLicht

CopperLicht[2] je grafički pokretač (engl. *engine*) koji bi se trebao koristiti s komercijalnim alatom za 3D modeliranje CopperCube[3]. CopperCube omogućava kreiranje zatvorenih prostora s dobrom podrškom za teksture. Upravo ta povezanost između CopperLichta i CopperCubea smanjuje korisnost CopperLichta kao samostalnog grafičkog paketa za 3D web aplikacije jer je jedini podržani format za složene scene upravo format nativan CopperCube alatu.

Osim tog velikog nedostatka, CopperLicht ima čudnu logiku grupiranja funkcionalnosti za projekcije i upravljanje mišem u glavnu klasu, i svi objekti moraju imati teksture da bi se mogli iscrtati, nije moguće definirati jednostavan jednobojan materijal. Loša dokumentacija samo pridodaje odluci da se ovaj paket ne koristi za ozbiljnu upotrebu.

3.1.2. Paket PhiloGL

PhiloGL[4] je malo napredniji od čistog WebGL-a, i dalje se trebaju shaderi ručno programirati, i dalje se moraju pozivati WebGL naredbe na dosta mjesta, i dalje se sve definira preko polja vrhova i atributa. Jedina novost koju PhiloGL donosi jest to što enkapsulira naredbe u JSON (JavaScript Object Notation) formatu.

[2] http://www.ambiera.com/copperlicht/
[3] http://www.ambiera.com/coppercube/
[4] http://www.senchalabs.org/philogl/

Budući da je jezik niske razine, u ovom radu se neće navoditi programski isječak za iscrtavanje kocke jer bi isječak sadržavao preko stotinu linija kôda.

3.1.3. Paket X3DOM

X3DOM[5] paket se ističe od ostalih paketa po upotrebi X3D formata. X3D je XML format koji se koristi za zapis 3D modela, nasljednik je popularnog VRML formata. Za korištenje X3DOM paketa potrebno je u zaglavlju pozvati x3dom.css i x3dom.js datoteke, te bilo gdje u web stranici dodati X3D model.

Većina alata za 3D modeliranje može exportati modele u X3D formatu, što je veliki plus za ovaj paket. Interakcija je ograničena na ugrađene kontrole u X3DOM paketu, te nije moguće programirati vlastite kontrole. Velika mana ovog paketa je ta što nije moguće učitavati modele iz datoteka, nego je potrebno ugraditi X3D model u web stranici. Ovaj paket je odličan za jednostavan prikaz 3D modela u web preglednicima, ali nije dobar za izradu interaktivnih 3D web aplikacija.

3.1.4. Paket Three.js

Three.js[6] je najrasprostranjeniji JavaScript API za WebGL. Jednostavan je za korištenje, postoji veliki broj aplikacija izrađenih s njim i učestalo izlaze nove verzije kôda.

Složene modele moguće je izmodelirati u nekom od popularnih alata za 3D modeliranje i exportati u COLLADA ili JSON formatu. Iz tih datoteka moguće je učitati 3D model u Three.js aplikaciju.

Također postoji nekoliko JavaScript paketa koji dodatno proširuju Three.js paket s dodatnim funkcionalnostima, poput Detector.js za detekciju WebGL podrške u web pregledniku i Stats.js za analizu brzine iscrtavanja.

S obzirom na dobru podršku, dokumentiran kôd i širok spektar funkcionalnosti, ovaj paket je najbolji izbor za izradu WebGL aplikacija.

3.2. WebGL podrška u web preglednicima

Sadržaju na Internetu bi se trebalo moći pristupiti s bilo kojeg uređaja. Upravo zato je WebGL temeljen na OpenGL ES 2.0, da se može pokretati na računalima, tabletima i pametnim mobitelima. Nažalost WebGL dijeli jednu manu sa svim ostalim novostima HTML5 standarda - proizvođači web preglednika odlučuju hoće li i kada će ih

[5] http://www.x3dom.org/
[6] https://github.com/mrdoob/three.js/

implementirati u svoje web preglednike. U sljedećim pododjeljcima je dan pregled poznatijih web preglednika, njihova podrška za WebGL i kako ga uključiti ako je podržan. Na kraju poglavlja nalazi se tablica 3.1 s kratkim pregledom WebGL podrške u web preglednicima.

3.2.1. Mozilla Firefox

Mozilla Firefox podržava WebGL od verzije 4.0 (za vrijeme pisanja ovog rada zadnja verzija je Firefox 21.0). Nisu potrebne dodatne postavke, WebGL je aktiviran s instalacijom web preglednika.

Firefox za Android također podržava WebGL od verzije 4.0.

3.2.2. Google Chrome

Chromium i Google Chrome podržavaju WebGL od verzija 9.0 (za vrijeme pisanja ovog rada zadnje verzije su Chromium 29.0 i Chrome 27.0). Nisu potrebne dodatne postavke, WebGL je aktiviran s instalacijom web preglednika.

Chrome za Android podržava WebGL od verzije 25.0. Da bi se aktivirao WebGL potrebno je u chrome:\\flags staviti kvačicu pored opcije "*Enable WebGL*".

Chrome za iOS ne podržava WebGL zbog Appleove zabrane. Čak i da podržava, aplikacije bi bile spore jer na iOS-u nijedan web preglednik osim mobilnog Safarija ne smije koristiti njihov pogon (engl. *engine*) za JavaScript, zvan Nitro JavaScript engine.

3.2.3. Safari

Safari na računalima podržava WebGL od verzije 5.1 (za vrijeme pisanja ovog rada zadnja verzija je Safari 6.0.5). Da bi se aktivirao WebGL potrebno je u glavnom izborniku odabrati opciju *Develop* i staviti kvačicu pored opcije "*Enable WebGL*". Ako se opcija *Develop* ne nalazi na glavnom izborniku, potrebno ju je uključiti u opciji *Preferences*.

Safari za iOS podržava WebGL, ali samo za autore iAds programa. Naime u Safariju za iOS postoji opcija za uključiti WebGL, ali je ta opcija zaključana i skrivena [10]. Postoje dva načina da se ta prepreka zaobiđe:

1. Može se otključati (engl. *jailbreak*) uređaj te instalirati WebGL Enabler koji omogućava WebGL u svim preglednicima na uređaju. Nedostatak ove opcije je potreba za otključavanjem uređaja.

2. Može se izgraditi (engl. *build*) WebGLBrowser[7] na vlastitom uređaju te preko njega pristupati stranicama s WebGL-om. Nedostatak ove opcije je što izgrađeni web preglednik nema Nitro JavaScript engine za ubrzavanje JavaScripti, te je potrebno da korisnik ima Apple iOS Developer Profile da bi uopće mogao izgraditi projekt.

3.2.4. Microsoft Internet Explorer

Microsoft Internet Explorer ne podržava WebGL (za vrijeme pisanja ovog rada zadnja verzija je Internet Explorer 10). Microsoft je odbio implementirati podršku za WebGL u svom web pregledniku zbog sigurnosnih razloga - prvotne verzije WebGL-a su dopuštale čitanje cijele memorije grafičke kartice, rad sa slikama s drugih domena, i općenito mogućnost izvođenja štetnog kôda na grafičkoj kartici [11]. WebGL je u međuvremenu riješio prva dva sigurnosna propusta, dok je treći propust odgovornost proizvođača grafičkih kartica.

Prema neslužbenim vijestima, Internet Explorer 11 će imati podršku za WebGL. U javnost je dospjela nedovršena verzija Internet Explorera 11 i moguće je uključiti WebGL u njoj podešavajući vrijednosti u Registryju. [12].

3.2.5. Opera

Opera podržava WebGL od verzije 12.0 (za vrijeme pisanja ovog rada zadnja verzija je Opera 12.15). Da bi se aktivirao WebGL potrebno je u opera:config podesiti vrijednosti polja "*Enable Hardware Acceleration*" i "*Enable WebGL*" na 1.

Tablica 3.1: Pregled WebGL podrške

Web preglednik	WebGL podrška	Od verzije
Firefox	Da	4.0
Chrome	Da	9.0
Chromium	Da	9.0
Safari	Da	5.1
Internet Explorer	Ne	–
Opera	Da	12.0
Firefox Android	Da	4.0
Chrome Android	Da	25.0
Chrome iOS	Ne	–
Safari iOS	Ne	–

[7] https://github.com/benvanik/WebGLBrowser

4. Edukacijska aplikacija

Programsko sučelje WebGL omogućava izradu interaktivnih 3D web aplikacija, u kojima je moguće manipulirati 3D objektima. Jedan od tipova aplikacija za koje je bitna manipulacija 3D objektima su edukacijske aplikacije. Edukacijske aplikacije omogućavaju korisnicima detaljnije upoznavanje s 3D objektima, lakše pamćenje i svladavanje karakterističnih radnji s objektima.

U suradnji s AVL-AST d.o.o.[1] definiran je zadatak za edukacijsku aplikaciju koja je izrađena u sklopu ovog rada. Sljedeći odjeljci se bave izrađenom aplikacijom. Za izradu aplikacije korišten je JavaScript paket Three.js uz dodatne biblioteke.

4.1. Opis problema

AVL (Anstalt für Verbrennungskraftmaschinen List) je austrijska tvrtka koja razvija tehnološke inovacije u automobilskoj industriji. Svaki automobilski dio koji naprave mora proći detaljno testiranje. Kako postoje mnogobrojni dijelovi za testiranje, tako postoje i mnogobrojni uređaji za testiranje, a jedan od takvih uređaja je AVL Micro Soot Sensor uređaj.

Micro Soot Sensor (engl. *soot* - čađa) je uređaj koji mjeri koncentraciju čađe u automobilskim ispušnim plinovima. Uređaj se može vidjeti na slici 4.1.

Obzirom da je cijena uređaja vrlo visoka, teško je podučavati korisnike kako rukovati uređajem jer ne postoje rezervni uređaji za vježbanje. Upravo zato je osmišljen plan za napraviti edukacijsku aplikaciju s virtualnim modelom na kojem se korisnici mogu naučiti rukovati prije nego krenu koristiti pravi uređaj.

Suradnici iz AVL-a su naglasili da žele pristup s web preglednika, mogućnost pokretanja s bilo koje platforme i posebno su istaknuli da žele moći pristupiti aplikaciji preko iPad uređaja. Također je prednost ako se korisnike ne gnjavi instalacijom dodatne programske potpore. Zbog tih uvjeta WebGL se iskazao kao najbolja tehnologija, te je aplikacija razvijana s Three.js paketom.

[1] https://www.avl.com/avl-ast-d.o.o

Slika 4.1: Micro Soot Sensor uređaj

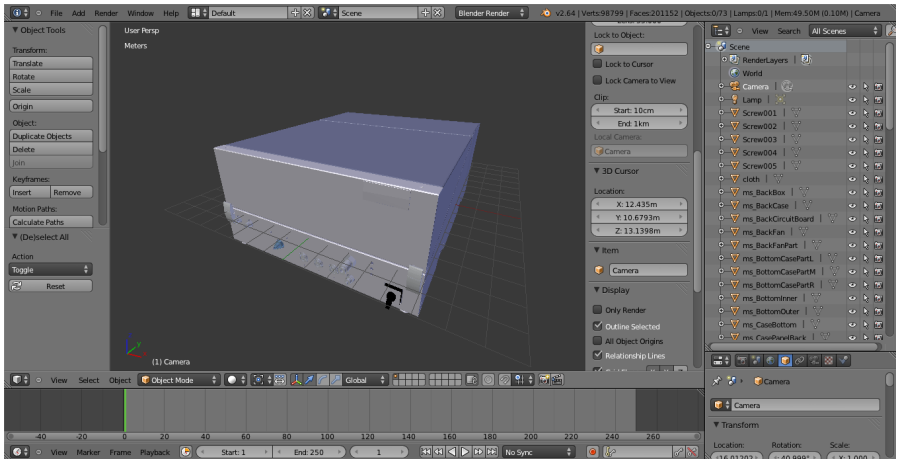
4.2. Programska implementacija

Budući da je u dodatku B već naveden jednostavan primjer korištenja Three.js paketa, u ovom odjeljku se neće posebno objašnjavati struktura Three.js aplikacije, nego će se objašnjavati važniji programski isječci bitni za rad aplikacije.

4.2.1. Programska okolina

Aplikacija je razvijana pomoću tekstualnih editora, a testirana je u web pregledniku Mozilla Firefox s programskim dodatkom Firebug[2], koji omogućava traženje JavaScript grešaka na pokrenutoj stranici.

Dobiveni 3D modeli od suradnika iz AVL-a su dorađeni u alatu za 3D modeliranje Blender[3], te su prebačeni u JSON format s kojim Three.js može raditi. Dorade su uključivale pojednostavljivanje modela (s 300MB na 11MB) i rastavljanje modela na dijelove kao posebne objekte.



Slika 4.2: Blender s 3D modelom Micro Soot Sensor uređaja

[2]https://getfirebug.com/
[3]http://www.blender.org/

4.2.2. Dodatni JavaScript paketi

Kao što je navedeno u pododjeljku 3.1.4, postoji niz dodatnih JavaScript paketa koji olakšavaju izradu Three.js aplikacija.

Paket Detector.js

Paket Detector.js provjerava ima li web preglednik u kojem se pokreće aplikacija aktiviranu podršku za WebGL. Ako nema onda ispisuje poruku u kojoj se navodi poveznica (engl. *link*) na stranicu Khronos grupe, gdje se nalaze upute za aktiviranje WebGL-a u pojedinim web preglednicima. Paket se aktivira s jednom jednostavnom linijom kôda:

```
if (!Detector.webgl) Detector.addGetWebGLMessage();
```

Paket Stats.js

Paket Stats.js služi za provjeru brzine iscrtavanja Three.js scene, a izmjerenu brzinu pokazuje u malom prozoru u obliku iscrtanih okvira po sekundi (engl. *frames per second - fps*), ili u broju milisekunda potrebnih za iscrtavanje jednog okvira. Da bi paket radio potrebno je pozvati metodu *update()* nakon poziva za iscrtavanje scene *renderer.render()*.

Paket Tween.js

Paket Tween.js je zapravo paket za vremensku promjenu varijabli iz jednih vrijednosti u druge, no te se promjene mogu ulančavati pa se korištenjem ovog programskog paketa mogu dobiti jednostavne animacije translatiranja i rotiranja objekata kroz određeni vremenski raspon. Da bi se animacije pokrenule potrebno je pozvati metodu *start()* nad željenom animacijom, te je još potrebno prije poziva za iscrtavanje scene pozvati metodu *TWEEN.update()* koja upravlja svim pokrenutim animacijama. Slijedi programski isječak koji animira pomicanje objekta za dvadeset jedinica u smjeru x-osi za deset sekundi:

```
object.animation = new TWEEN.Tween({x:0})
                           .to({x:20},10000)
                           .onUpdate( function() {
                             object.position.x = this.x;
                           });
object.animation.start();
```

Paket TrackballControls.js

Paket TrackballControls.js omogućava pomicanje kamere. Prilikom inicijalizacije potrebno je predati kao argument kameru kojom će upravljati. Također je potrebno

prije poziva metode za iscrtavanje scene pozvati metodu *update()* da bi se učitale promjene. Kontrole su sljedeće:

- Lijevi klik miša - rotiranje kamere oko gledišta
- Kotačić na mišu - približavanje/udaljavanje kamere gledištu
- Desni klik miša - pomicanje kamere i gledišta u virtualnom prostoru

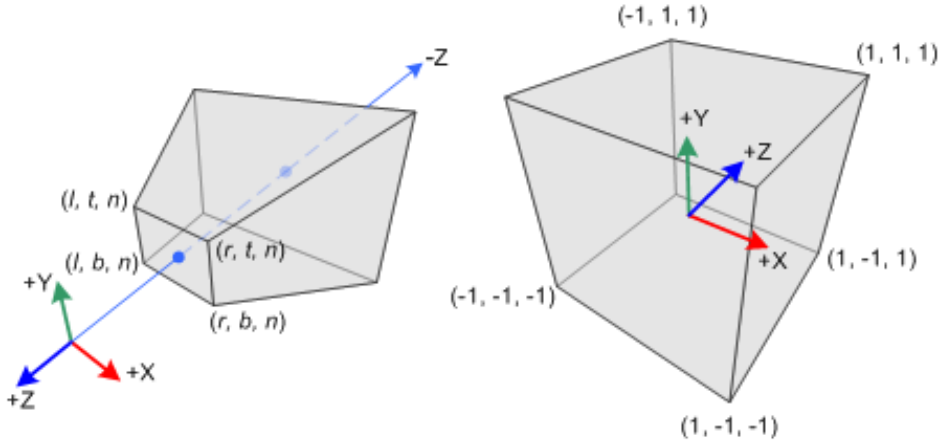
TrackballControls.js također podržava i uređaje na dodir, iste funkcije su pridodane gestama pomicanja jednog prsta, udaljavanju/približavanju dva prsta i pomicanju tri prsta, jednakim redoslijedom kao prije navedene funkcije miša.

4.2.3. Three.js funkcije za rad s projekcijama

Projekcije su vrlo važne pri radu s interaktivnom 3D grafikom - one omogućuju pretvorbu 3D točaka u 2D prostor i obratno. U ovom radu su projekcije bitne i za biranje 3D objekata - kad se klikne na 2D sliku nekog objekta potrebno je 2D točku pretvoriti u 3D točku da se odredi koji objekt je označen. Three.js koristi dva razreda za rad s projekcijama – razredi *Projector* i *Raycaster*.

Projector (hrv. projektor) je razred zadužen za projekcije u Three.js paketu. Sadrži dvije metode – *projectVector(vector, camera)* i *unprojectVector(vector, camera)*. Metoda *projectVector(vector, camera)* projicira 3D točku iz globalnih koordinata u NDC (Normalized Device Coordinates) koordinate kamere. Slika 4.3 prikazuje projekcijski volumen (engl. *view frustum*) prije i poslije prebacivanja u NDC koordinati sustav - za par (x,y), koordinate (-1,-1) odgovaraju donjem lijevom kutu pogleda kamere, a koordinate (1,1) gornjem desnom kutu pogleda kamere, dok z koordinata prima vrijednost 1 za blisku odrezujuću plohu (engl. *near clipping plane*), a -1 za daleku odrezujuću plohu (engl. *far clipping plane*). Metoda *unprojectVector(vector, camera)* radi obratnu stvar, projicira točku iz NDC koordinata kamere u globalni koordinati sustav.

Raycaster (hrv. bacač zrake) je razred zadužen za detekciju presjeka 3D objekata i polupravca u prostoru. Obavezni argumenti konstruktora su početna točka i normalizirani vektor smjera polupravca, a opcionalni argumenti su bliska i daleka granica za detekciju presjeka. Razred sadrži dvije metode – *intersectObject(object)* i *intersectObjects(objects)*. Metoda *intersectObject(object)* provjerava prolazi li polupravac kroz predani objekt, te ako prolazi onda vraća informacije o udaljenosti od početne točke polupravca, 3D točku u kojoj je došlo do presijecanja, poligon kojem pripada točka i referencu na objekt, a ako ne prolazi vraća *null*. Metoda *intersectsObjects(objects)* prima polje objekata, za svaki objekt provjerava presijecanje, te rezultate sprema u polje sortirano po udaljenosti od početne točke polupravca.



Slika 4.3: Projekcijski volumen i NDC koordinate [13]

4.2.4. Interakcija s objektima

Da bi aplikacija bila interaktivna, potrebno je u aplikaciji imati podršku za pojedine događaje (engl. *events*), poput pomicanja miša, početak klika, kraj klika, detekciju dodira, itd. Web preglednici imaju razvijenu podršku za razne događaje, a da bi se programski pristupilo događajima potrebno je:

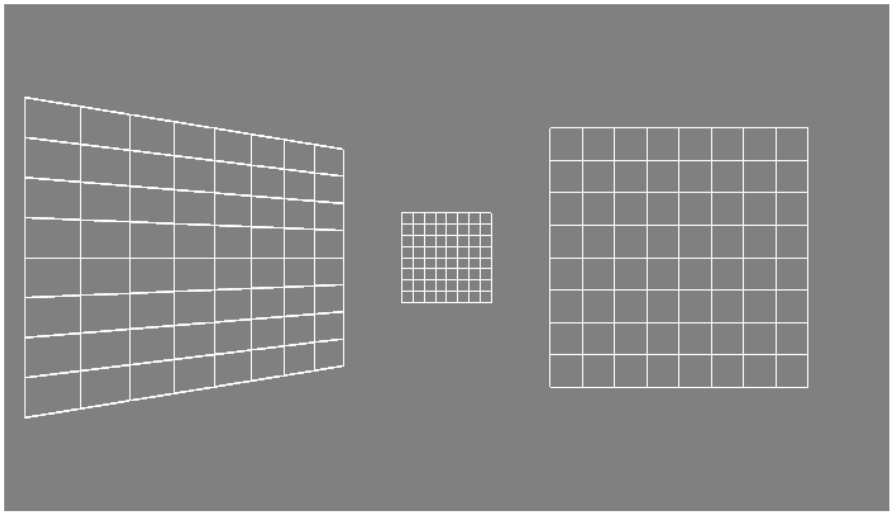
1. Odrediti koji je ciljni događaj
2. Napisati funkciju za obradu događaja
3. Povezati funkciju i događaj s naredbom *addEventListener('eventName',function)*;

Rezultat gornjeg postupka je pokretanje funkcije *function* svaki put kad dođe do detekcije događaja *eventName*. Ova aplikacija koristi događaje: *mousemove*, *mousedown*, *mouseup*, *touchmove*, *touchstart* i *touchend*. Svi navedeni događaji sadrže u varijablama *event.clientX* i *event.clientY* koordinate web stranice na kojem se nalazio miš ili prst u trenutku detektiranja događaja.

Iz apsolutne pozicije događaja potrebno je odrediti na kojem slikovnom elementu (engl. *pixel*) *Canvasa* se događaj dogodio, normirati u NDC koordinate te pomoću projekcija i detekcije presjeka odrediti koji je objekt u sceni selektiran.

Također jedna od bitnih stavki je i pomicanje objekata kroz prostor. Prilikom selekcije objekta, na mjesto detektiranog presjeka stavlja se nevidljiva ravnina i računa se razlika između položaja ravnine i položaja objekta. Zatim se kod detekcije pomicanja miša/prsta gleda samo presjek projicirane zrake i ravnine, te se od točke presjeka oduzima prethodno izračunata razlika i to je nova pozicija objekta. Bitno je za primijetiti da ako se ne napravi dobra orijentacija ravnine, prilikom pomicanja objekta dobit će se

pomicanje i po dubini. Ravnina se orijentira tako da se pozicionira u gledište kamere, pokrene se metoda *lookAt(vector)* koja orijentira ravninu, te se onda pozicionira na željenu poziciju. Slika 4.4 sadrži prikaz loše orijentirane ravnine, ravnine u gledištu i ispravno orijentirane ravnine.



Slika 4.4: Prikaz ravnina različitih orijentacija

4.3. Implementacijski detalji za iPad uređaje

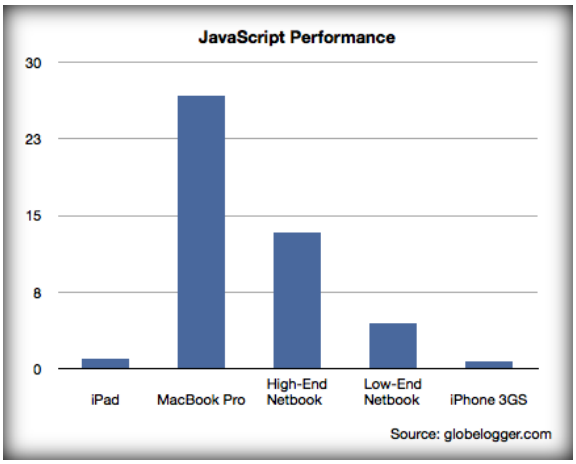
Kao što je navedeno u pododjeljku 3.2.3, web preglednik Safari na iOS uređajima službeno ne podržava WebGL, no postoje načini da se pokrenu WebGL aplikacije. Za ovaj rad korišten je drugi način iz pododjeljka 3.2.3 - WebGL Browser. WebGL Browser se može izgraditi (engl. *build*) na iOS uređaju ako korisnik ima Apple iOS Developer Profile. Aplikacija je testirana na iPad 3 uređaju, i radi pri brzini od 26 fps (za usporedbu, na stolnom računalu je brzina pri normalnom radu 50fps). No ako se pokrene rastavljanje uređaja na dijelove i proba se odabrati neki dio modela, performanse drastično padnu i potrebno je pet do deset sekundi da iPad detektira koji dio je odabran i nastavi s radom.

Najveći razlozi zašto aplikacija zna zastati su manja procesorska moć iPad uređaja i kombinacija sljedećih stavki:

- WebGL Browser nema Nitro JavaScript engine
- Three.js automatski povećava rezoluciju ukoliko se aplikacija pokreće na ekranu s visokim DPI (Dots Per Inch)
- JavaScript je općenito spor na iOS uređajima, vidljivo na slici 4.5

Na slici 4.5 je prikazana relativna brzina JavaScripta Safari web preglednika na različitim platformama u odnosu na brzinu na iPad uređaju. Test je napravljen korištenjem Sun Spider JavaScript Benchmarka[4]. Prema rezultatima testiranja vidljivo je da je JavaScript na iPad uređaju 26 puta sporiji od JavaScripta na MacBook Pro uređaju, te nije čudno što se izrađena aplikacija "smrzava" prilikom računski zahtjevnih operacija.

No unatoč svemu, aplikacija se dobro vrti s obzirom na to da WebGL nije službeno podržan na iPad uređaju. Kada dođe do službene podrške može se očekivati i bolje izvođenje aplikacije na iOS uređajima.



Slika 4.5: Relativna brzina JavaScripta na raznim platformama [14]

4.4. Programski produkt

Aplikacija je zamišljena da zauzima mali označeni prostor u web stranici, tako da se može lagano integrirati u postojeće web stranice koje će obogatiti svojom funkcionalnošću.

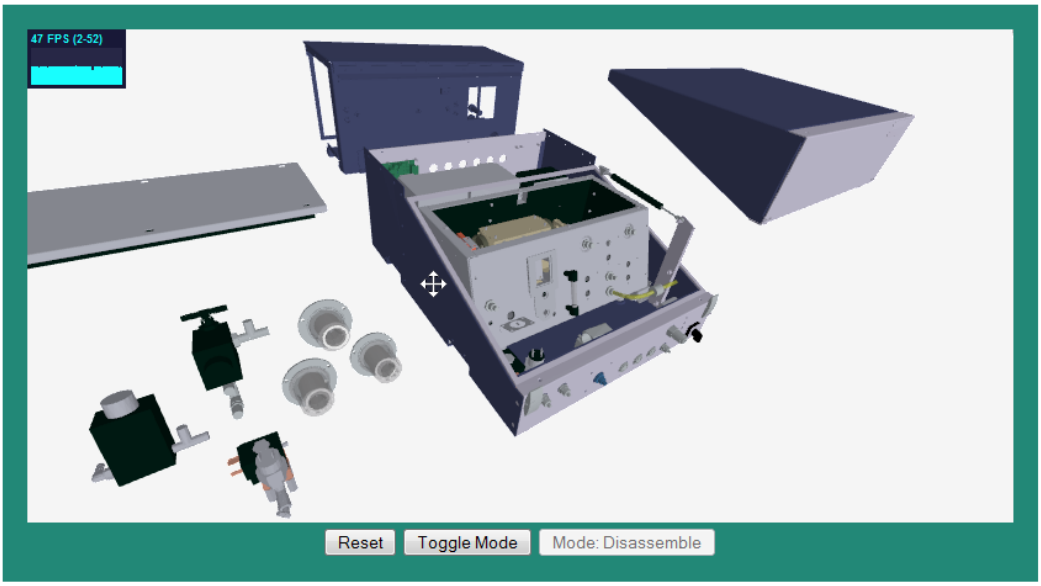
Slika 4.6 prikazuje aplikaciju ubačenu u web stranicu s naslovom "*Microsoot WebGL App Demonstration*" i tekstom "*This is a demo app to show off webGL capabilities.*". U gornjem lijevom kutu može se vidjeti prozor paketa Stats.js koji prikazuje trenutnu brzinu iscrtavanja scene u fps-ovima.

Još jedna odlika aplikacije je da radi na principu promjena stanja - postoji više načina rada koji se izmjenjuju pritiskom tipke "*Toggle Mode*". Izmjena načina rada ostavlja kameru u istoj poziciji i orijentaciji kao u prethodnom načinu rada, dok tipka "*Reset*" vraća kameru na početnu poziciju i orijentaciju.

[4]http://www.webkit.org/perf/sunspider/sunspider.html

Slika 4.6: Aplikacija u "Free Camera" načinu rada

Način rada "Free Camera" predstavlja lagano manipuliranje kamerom da bi se korisnik kretao kroz virtualni prostor i pogledao 3D objekte iz raznih kuteva/gledišta. U ovom načinu rada nije moguće reagirati s 3D modelima, budući da bi interakcija otežala kretanje u sceni ispunjenoj 3D objektima.



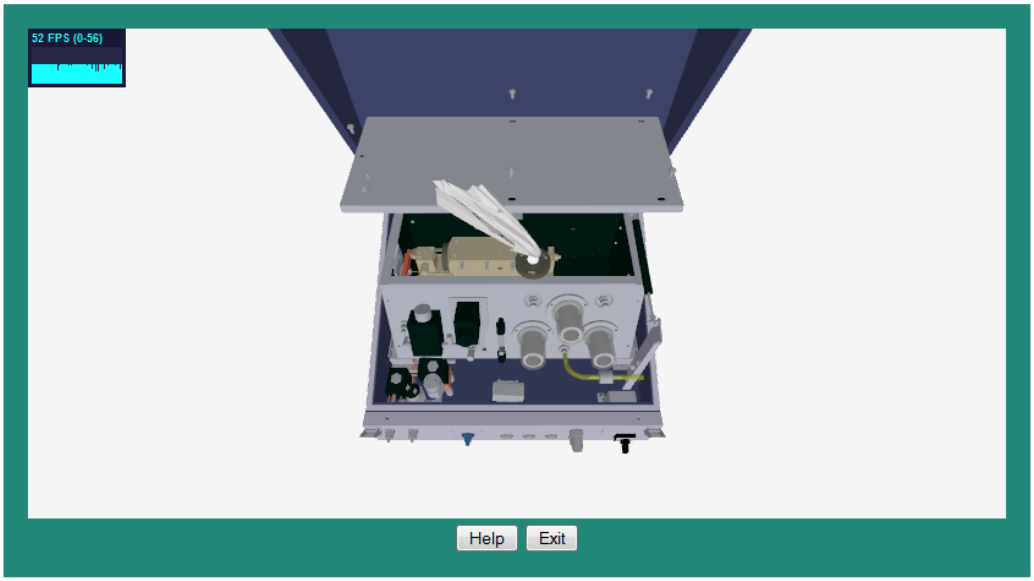
Slika 4.7: Aplikacija u "Disassemble" načinu rada

Način rada "Disassemble" služi za rastavljanje 3D modela. Virtualni 3D model Micro Soot uređaja se sastoji od 70 podobjekata, i svaki od njih je dohvatljiv. Svaki dohvaćeni objekt može se pomicat i prilikom rukovanja s objektom ispod aplikacije se nalazi ispisani tekst "*You have selected imeObjekta.*". Ovaj način rada služi za upoznavanje s unutarnjim dijelovima uređaja, da bi se korisnik upoznao s unutrašnjošću uređaja bez da ga otvara.

U aplikaciji je implementiran obrazac uporabe čišćenja stakalca laserske komore uređaja - postupak se sastoji od četrnaest koraka koji su navedeni u tablici 4.1. Svi koraci su animirani tako da korisnik ima vizualnu predodžbu kako izvršiti čišćenje stakalca. Stoga postoje još dva načina rada koji se bave animacijama - načini rada "Teach Me" i "Test Me".

Slika 4.8: Aplikacija prilikom izvođenja animacije u "Teach Me" načinu rada

Način rada "Teach Me" sadržava niz animacija koje objašnjavaju kako rukovati s uređajem. Svaka animacija ima i svoj popratni tekst koji objašnjava detaljno što se radi u kojem koraku. Prilikom odabira načina rada animacije se neće automatski pokrenuti. Da bi se pokrenule animacije potrebno je kliknuti na tipku "*Run*" koja izmjenjuje vidljive tipke aplikacije i omogućava izvođenje animacija. Izvođenjem animacija upravlja se tipkama: "*Prev*" koja pokreće animaciju prethodnog koraka, "*Play*" koja iznova pokreće animaciju trenutnog koraka, "*Next*" koja pokreće animaciju sljedećeg koraka i "*Exit*" koja prekida izvođenje animacija i vraća program u biranje načina rada. Za vrijeme izvođenja animacija nije moguće pokretati kameru jer je kamera programirana da najbolje prati izvođenje svake pojedine animacije.



Slika 4.9: Aplikacija prilikom izvođenja animacije u "Test Me" načinu rada

Način rada "Test Me" je sličan načinu rada "Teach Me", razlika je u načinu upravljanja animacija. Animacije se pokreću klikanjem po određenim dijelovima modela. Ukoliko korisnik zapne može kliknuti na tipku "*Help*" koja će ispisati tekst sljedećeg koraka. Ovaj način rada je zamišljen za testiranje koliko je dobro korisnik usvojio korake koje je naučio u načinu rada "Teach Me".

S ovim je gotov opis trenutno izrađene aplikacije. Mjesta za daljnji napredak postoji u vidu izbornika koji bi određivao koji obrazac uporabe se podučava/testira, te bi još jedan izbornik služio za odabir uređaja. Također bi trebalo pojednostaviti 3D modele, trenutni modeli zauzimaju oko 11MB prostora, što je ipak dosta na mobilnim uređajima. Pojednostavljivanjem bi se također ubrzao i proces traženja presjeka, te bi se aplikacija brže izvodila na iOS uređajima.

Tablica 4.1: Obrazac uporabe čišćenja stakalca laserske komore uređaja

Korak	Uputa
STEP 1	Switch off the device.
STEP 2	Unlock the locks of the lid.
STEP 3	Open the lid of the measuring unit.
STEP 4	Remove the five screws of the measuring chamber cover.
STEP 5	Lift off the cover of the measuring chamber.
STEP 6	Carefully remove the measuring cell window.
STEP 7	Clean the lens with a cleaning tissue.
STEP 8	Remount the measuring cell window.
STEP 9	Clean the second measuring cell window in the same way.
STEP 10	Close the cover of the measuring chamber.
STEP 11	Tighten the measuring chamber cover with the five screws.
STEP 12	Close the lid of the measuring unit.
STEP 13	Lock the locks of the lid.
STEP 14	Switch on the device.

5. Zaključak

Standard HTML je prošao kroz brojne promjene od svog osnutka 1990. godine, a s trenutnim standardom HTML5 web stranice su postale prava multimedijska središta. Glazba, video i dinamički crteži postali su sastavni dio web stranica te više nije potrebno koristiti programske dodatke da bi se ta funkcionalnost ugradila u web stranice.

Programsko sučelje WebGL dodatno proširuje funkcionalnost web stranica sa sklopovski ubrzanom računalnom grafikom. Nastavkom razvoja ove tehnologije moguće je da će web preglednici postati glavna okruženja za razvoj računalne grafike. U vrlo kratkom roku većina poznatijih proizvođača web preglednika je implementirala WebGL u svojim preglednicima, no pojedini proizvođači se nažalost još uvijek bore protiv uvođenja promjena [10] [11].

Budući da je WebGL programsko sučelje niske razine (zahtjeva pisanje programa za sjenčanje i rad sa spremnicima vrhova/trokuta), za razvoj složenijih aplikacija preporučeno je koristiti naprednije programske pakete za rad s WebGL-om. U trenutku pisanja ovog rada najpristupačniji programski paket (po mišljenju autora) je Three.js.

Za potrebe tvrtke AVL-AST d.o.o. razvijena je web aplikacija koja podučava korisnike u korištenju uređaja AVL Micro Soot Sensor. Izrađena aplikacija pokriva obrazac upotrebe čišćenja stakalca laserske komore.

Primjenom WebGL-a računalna grafika je dospjela na neočekivani medij - Internet. Teško je predvidjeti u kojem smjeru će se nastaviti razvoj ove tehnologije, no definitivno budućnost nosi nove zanimljivosti za sve ljubitelje računalne grafike.

LITERATURA

[1] Tim Berners-Lee. *The WorldWideWeb browser*, 1994. URL http://www.w3.org/People/Berners-Lee/WorldWideWeb.html. Pristupljeno 05.06.2013.

[2] Tim Berners-Lee. *Information Management: A Proposal*. CERN, Svibanj 1990. URL http://www.w3.org/History/1989/proposal.html.

[3] World Wide Web Consortium. *All standards and drafts*, 2013. URL http://www.w3.org/TR/. Pristupljeno 05.06.2013.

[4] Dave Raggett, Jenny Lam, Ian Alexander, i Michael Kmiec. *Raggett on HTML 4*. Addison-Wesley, u drugom izdanju, 1998.

[5] Mihai Sucan. *SVG or Canvas? Choosing between the two*, Veljača 2010. URL http://dev.opera.com/articles/view/svg-or-canvas-choosing-between-the-two/. Pristupljeno 06.06.2013.

[6] W3Schools. *HTML5 Audio/Video Reference*, 2013. URL http://www.w3schools.com/tags/ref_av_dom.asp. Pristupljeno 06.06.2013.

[7] Bruce Lawson i Remy Sharp. *HTML5 Audio and Video: What you must know*, Listopad 2010. URL http://net.tutsplus.com/tutorials/html-css-techniques/html5-audio-and-video-what-you-must-know/. Pristupljeno 06.06.2013.

[8] Vladimir Vukićević. *Canvas 3D: GL power, web-style*, Studeni 2007. URL http://blog.vlad1.com/2007/11/26/canvas-3d-gl-power-web-style/. Pristupljeno 07.06.2013.

[9] Giles Thomas. *WebGL Lesson 2 – Adding colour*, Listopad 2009. URL http://learningwebgl.com/blog/?p=134. Pristupljeno 07.06.2013.

[10] Nathan de Vries. *Enabling & Using WebGL on iOS*, Studeni 2011. URL http://atnan.com/blog/2011/11/03/enabling-and-using-webgl-on-ios/. Pristupljeno 07.06.2013.

[11] MSRC Engineering. *WebGL Considered Harmful*, Lipanj 2011. URL http://blogs.technet.com/b/srd/archive/2011/06/16/webgl-considered-harmful.aspx. Pristupljeno 07.06.2013.

[12] Paul Thurrot. *Blue's Clues: WebGL Support in IE 11*, Ožujak 2013. URL http://winsupersite.com/windows-8/blue-s-clues-webgl-support-ie-11. Pristupljeno 07.06.2013.

[13] Song Ho Ahn. *OpenGL Projection Matrix*, Prosinac 2012. URL http://www.songho.ca/opengl/gl_projectionmatrix.html. Pristupljeno 10.06.2013.

[14] Ben Galbraith. *iPad JavaScript Shockingly Slow?*, Svibanj 2010. URL http://ajaxian.com/archives/ipad-javascript-shockingly-slow. Pristupljeno 11.06.2013.

Dodatak A
WebGL kôd za iscrtavanje trokuta

```
1  <script type="text/javascript"
       src="glMatrix-0.9.5.min.js"></script>
2
3  <script id="shader-fs" type="x-shader/x-fragment">
4   precision mediump float;
5
6   varying vec4 vColor;
7
8   void main(void) {
9     gl_FragColor = vColor;
10  }
11 </script>
12
13 <script id="shader-vs" type="x-shader/x-vertex">
14  attribute vec3 aVertexPosition;
15  attribute vec4 aVertexColor;
16
17  uniform mat4 uMVMatrix;
18  uniform mat4 uPMatrix;
19
20  varying vec4 vColor;
21
22  void main(void) {
23    gl_Position = uPMatrix * uMVMatrix *
         vec4(aVertexPosition, 1.0);
24    vColor = aVertexColor;
25  }
26 </script>
27
28 <script type="text/javascript">
29
30  var gl;
31
```

```
32  function initGL(canvas) {
33    try {
34      gl = canvas.getContext("experimental-webgl");
35      gl.viewportWidth = canvas.width;
36      gl.viewportHeight = canvas.height;
37    } catch (e) {
38    }
39    if (!gl) {
40      alert("Could not initialise WebGL, sorry :-(");
41    }
42  }
43
44  function getShader(gl, id) {
45    var shaderScript = document.getElementById(id);
46    if (!shaderScript) {
47      return null;
48    }
49
50    var str = "";
51    var k = shaderScript.firstChild;
52    while (k) {
53      if (k.nodeType == 3) {
54        str += k.textContent;
55      }
56      k = k.nextSibling;
57    }
58
59    var shader;
60    if (shaderScript.type == "x-shader/x-fragment") {
61      shader = gl.createShader(gl.FRAGMENT_SHADER);
62    } else if (shaderScript.type == "x-shader/x-vertex") {
63      shader = gl.createShader(gl.VERTEX_SHADER);
64    } else {
65      return null;
66    }
67
68    gl.shaderSource(shader, str);
69    gl.compileShader(shader);
70
71    if (!gl.getShaderParameter(shader,
        gl.COMPILE_STATUS)) {
72      alert(gl.getShaderInfoLog(shader));
73      return null;
74    }
75
76    return shader;
```

```
77  }
78
79  var shaderProgram;
80
81  function initShaders() {
82    var fragmentShader = getShader(gl, "shader-fs");
83    var vertexShader = getShader(gl, "shader-vs");
84
85    shaderProgram = gl.createProgram();
86    gl.attachShader(shaderProgram, vertexShader);
87    gl.attachShader(shaderProgram, fragmentShader);
88    gl.linkProgram(shaderProgram);
89
90    if (!gl.getProgramParameter(shaderProgram,
        gl.LINK_STATUS)) {
91      alert("Could not initialise shaders");
92    }
93
94    gl.useProgram(shaderProgram);
95
96    shaderProgram.vertexPositionAttribute =
        gl.getAttribLocation(shaderProgram,
        "aVertexPosition");
97    gl.enableVertexAttribArray(shaderProgram.vertexPositionAttribute);
98
99    shaderProgram.vertexColorAttribute =
        gl.getAttribLocation(shaderProgram,
        "aVertexColor");
100   gl.enableVertexAttribArray(shaderProgram.vertexColorAttribute);
101
102   shaderProgram.pMatrixUniform =
        gl.getUniformLocation(shaderProgram, "uPMatrix");
103   shaderProgram.mvMatrixUniform =
        gl.getUniformLocation(shaderProgram, "uMVMatrix");
104 }
105
106 var mvMatrix = mat4.create();
107 var pMatrix = mat4.create();
108
109 function setMatrixUniforms() {
110   gl.uniformMatrix4fv(shaderProgram.pMatrixUniform,
        false, pMatrix);
111   gl.uniformMatrix4fv(shaderProgram.mvMatrixUniform,
        false, mvMatrix);
112 }
113
```

```
114  var triangleVertexPositionBuffer;
115  var triangleVertexColorBuffer;
116
117  function initBuffers() {
118    triangleVertexPositionBuffer = gl.createBuffer();
119    gl.bindBuffer(gl.ARRAY_BUFFER,
         triangleVertexPositionBuffer);
120    var vertices = [
121         0.0,  1.0,  0.0,
122        -1.0, -1.0,  0.0,
123         1.0, -1.0,  0.0
124    ];
125    gl.bufferData(gl.ARRAY_BUFFER, new
         Float32Array(vertices), gl.STATIC_DRAW);
126    triangleVertexPositionBuffer.itemSize = 3;
127    triangleVertexPositionBuffer.numItems = 3;
128
129    triangleVertexColorBuffer = gl.createBuffer();
130    gl.bindBuffer(gl.ARRAY_BUFFER,
         triangleVertexColorBuffer);
131    var colors = [
132         1.0, 0.0, 0.0, 1.0,
133        0.0, 1.0, 0.0, 1.0,
134        0.0, 0.0, 1.0, 1.0
135    ];
136    gl.bufferData(gl.ARRAY_BUFFER, new
         Float32Array(colors), gl.STATIC_DRAW);
137    triangleVertexColorBuffer.itemSize = 4;
138    triangleVertexColorBuffer.numItems = 3;
139  }
140
141  function drawScene() {
142    gl.viewport(0, 0, gl.viewportWidth,
         gl.viewportHeight);
143    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
144
145    mat4.perspective(45, gl.viewportWidth /
         gl.viewportHeight, 0.1, 100.0, pMatrix);
146
147    mat4.identity(mvMatrix);
148
149    mat4.translate(mvMatrix, [0.0, 0.0, -7.0]);
150    gl.bindBuffer(gl.ARRAY_BUFFER,
         triangleVertexPositionBuffer);
151    gl.vertexAttribPointer(shaderProgram.vertexPositionAttribute,
         triangleVertexPositionBuffer.itemSize, gl.FLOAT,
```

```
            false, 0, 0);
152
153         gl.bindBuffer(gl.ARRAY_BUFFER,
                triangleVertexColorBuffer);
154         gl.vertexAttribPointer(shaderProgram.vertexColorAttribute,
                triangleVertexColorBuffer.itemSize, gl.FLOAT,
                false, 0, 0);
155
156         setMatrixUniforms();
157         gl.drawArrays(gl.TRIANGLES, 0,
                triangleVertexPositionBuffer.numItems);
158       }
159
160       function webGLStart() {
161         var canvas = document.getElementById("myCanvas");
162         initGL(canvas);
163         initShaders();
164         initBuffers();
165
166         gl.clearColor(0.0, 0.0, 0.0, 1.0);
167         gl.enable(gl.DEPTH_TEST);
168
169         drawScene();
170       }
171     </script>
```

Slika A.1: Rezultat izvođenja gornjeg kôda

Dodatak B
Primjeri kodova naprednijih grafičkih paketa

Paket CopperLicht

```
<script type="text/javascript" src="copperlicht.js">
<script type="text/javascript">
 main = function(){
  // create the 3d engine
  var engine = new CL3D.CopperLicht('myCanvas');
  // check for WebGL support
  if (!engine.initRenderer())
    return; // this browser doesn't support WebGL
  // add a new 3d scene
  var scene = new CL3D.Scene();
  engine.addScene(scene);
  // add a cube to the scene
  var cubenode = new CL3D.CubeSceneNode();
  scene.getRootSceneNode().addChild(cubenode);
  // set material texture of the cube
  cubenode.getMaterial(0).Tex1 =
    engine.getTextureManager().getTexture("test.jpg",
      true);
  // add a camera
  var cam = new CL3D.CameraSceneNode();
  cam.Pos = new CL3D.Vect3d(10,10,10);
  cam.setTarget(new CL3D.Vect3d(0,0,0));
  scene.getRootSceneNode().addChild(cam);
  scene.setActiveCamera(cam);
 }
</script>
```

Paket X3DOM

```
<link rel="stylesheet" type="text/css"
      href="http://www.x3dom.org/download/x3dom.css">
</link>
<script type="text/javascript"
      src="http://www.x3dom.org/download/x3dom.js">
</script>

<x3d width="500px" height="400px">
  <scene>
    <shape>
      <appearance>
        <material diffuseColor='red'></material>
      </appearance>
      <box></box>
    </shape>
  </scene>
</x3d>
```

Paket Three.js

```
<script src="three.min.js"></script>
<script>
 var scene = new THREE.Scene();
 var camera = new THREE.PerspectiveCamera(40, 700/500,
    0.1, 1000);
 var renderer = new THREE.WebGLRenderer();
 renderer.setSize(700, 500);
 document.body.appendChild(renderer.domElement);
 var geometry = new THREE.CubeGeometry(1,1,1);
 var material = new
    THREE.MeshLambertMaterial({color:'red'});
 var cube = new THREE.Mesh(geometry, material);
 scene.add(cube);
 var light = new THREE.DirectionalLight( 0xffffff, 0.5 );
 light.position.set( 1, 2, 3 );
 scene.add(light);
 camera.position = new THREE.Vector3(1.5, 1.7, 2)
 camera.lookAt(new THREE.Vector3(0,0,0));
 function render() {
   requestAnimationFrame(render);
   renderer.render(scene, camera);
 }
 render();
</script>
```

Dodatak C
Upute za integraciju aplikacije u vlastitu web stranicu

Pokretanje izrađene aplikacije je jednostavno - potrebno je samo pokrenuti datoteku *DiplomaThesisApp.html* s priloženog CD-a i provjeriti podržava li web preglednik s kojim se otvara aplikacija WebGL po uputama u odjeljku 3.2.

Ako postoji potreba za izmjenom aplikacije potrebno je mijenjati svega tri datoteke: *DiplomaThesisApp.html* za sadržaj web stranice, *myStyle.css* za izgled web stranice i *myScript.js* za izmjenu funkcionalnosti web aplikacije.

Za ubacivanje aplikacije u drugu web stranicu potrebno je kopirati sve skripte s CD-a te napraviti element s atributom *id="Container"*. Taj element određuje gdje će se u web stranici ubaciti *Canvas* element od aplikacije. Također postoje dodatni atributi koji nisu obavezni: *height=""*, *width=""* i *fov=""*. Ti atributi određuju visinu i širinu *Canvasa* aplikacije i kut pogleda (engl. *Field of View*). Ako dodatni atributi nisu navedeni koristit će se standardne vrijednosti koje su: širina 800 slikovnih elemenata, visina 400 slikovnih elemenata i kut pogleda 60 stupnjeva. Također je potrebno kopirati tipke iz *DiplomaThesisApp.html* kako bi aplikacija zadržala svu svoju funkcionalnost.

Tehnike interakcije 3D objektima u web preglednicima

Sažetak

U ovom radu prezentirane su osnovne tehnike interakcije 3D objektima u web preglednicima. Također je ukratko prikazana povijest razvoja web preglednika, navedene posebnosti HTML5 standarda te su još dane upute za izradu interaktivne web aplikacije korištenjem WebGL specifikacije. U sklopu rada napravljena je edukacijska aplikacija za rukovanje AVL Micro Soot Sensor uređajem.

Ključne riječi: web preglednici, HTML5, WebGL, Three.js, interaktivna grafika

Interacting with 3D objects in web browsers

Abstract

In this thesis the basic techniques of interacting with 3D objects in web browsers are presented. The history of web browsers development, highlights of the HTML5 standard and instructions for developing an interactive web application using the WebGL specification are also presented. As part of this thesis an educational applications for handling the AVL Micro Soot Sensor device was developed.

Keywords: web browsers, HTML5, WebGL, Three.js, interactive graphics