

UNIVERZITET SINGIDUNUM
DEPARTMAN ZA POSLEDIPLOMSKE STUDIJE I
MEĐUNARODNU SARADNJU

MASTER RAD

Primer 3D tehnologija u web okruženju

Mentor:

Prof. dr *Aleksandar Jevremović*

Kandidat:

Bojan Madov
2014410145

Beograd, 2015.godina.

Sadržaj

Primer 3D tehnologija u web okruženju	- 1 -
Apstrakt.....	- 3 -
Uvod.....	- 4 -
1. Web 3D tehnologije	- 6 -
1.1 Kriterijumi i opšti principi rada	- 6 -
1.2 Performanse	- 10 -
1.3 Kvalitet prikaza sadržaja	- 12 -
1.4 Interaktivnost i standardizacija	- 15 -
1.5 Vulkan, Mantle, DirectX 12	- 16 -
2. WebGL iz ugla Three.js, prednosti i nedostaci.....	- 18 -
3. Ideja projekta	- 20 -
4. Koraci realizacije.....	- 21 -
5. Kreiranje modela u 3d-u	- 23 -
5.1 Slike modela u 3ds max-u	- 31 -
5.2 Priprema modela za webgl okruženje	- 33 -
6. WebGL aplikacija.....	- 34 -
6.1 Slike.....	- 47 -
7. Zaključak.....	- 48 -
8. Literatura.....	- 49 -

Apstrakt

Ovaj rad bavi se pregledom različitih tehnologija koje su trenutno dostupne i najčešće korišćene u *Web* okruženju. Prikazana je kompleksnost okruženja, istorijski razvoj, tehnološki napredak, ali i ograničenja koje ove tehnologije nameću. Fokus je postavljen na četiri krajnja rešenja koja uključuju *X3D*, *WebGL*, *Unity Engine* i *Unreal Engine*sa posebnim naglaskom na *WebGL* kao glavnom bazom za dalju evaluaciju i standard u odnosu na koji su vršena sva istraživanja. Kroz ovaj rad dati su opis i konkretni koraci u razvoju i implementaciji 3d okruženja. Prikazani su ključni koraci u razvoju 3d modela, scene, pozicioniranje objekata, realističnom postavljanju osvetljenja, kamera i drugih neophodnih elemenata. Takođe, dat je i kompletan kod *webgl* aplikacije koja korisnicima omogućava da se na virtuelan način upoznaju sa okruženjem i da pruži razne funkcionalnosti koje su bile od ključne važnosti za razvoj ovog rada.

Ključne reči: web3d, *WebGL*, javascript, opengl, 3d

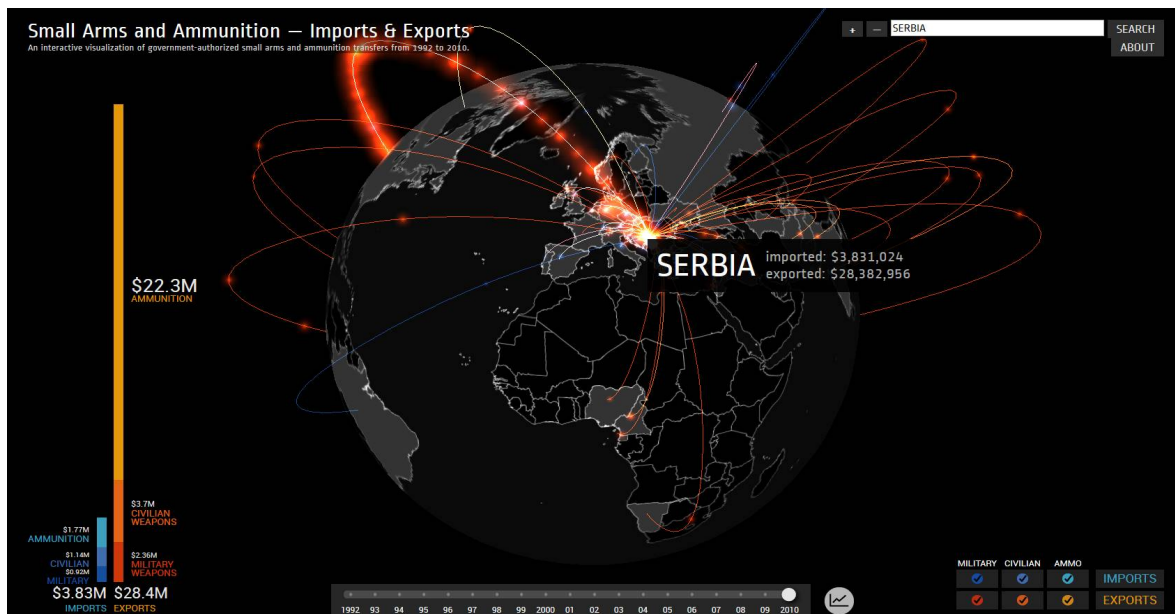
Uvod

Iako se na prvi pogled stiče utisak da je 3D u *Web* okruženju novina koja postaje sve aktuelnija, činjenica je da je još tokom sredine '90.-tih godina nastalo prvo rešenje koje je ponudilo mogućnost da u internet pregledaču demonstrira rad 3D okruženja. Već tada se javila jaka želja za daljim istraživanjem u datoj sferi i sagledavanju mogućnosti koje one pružaju. 3D tehnologije zapravo čini skup različitih tehnologija kako softverskih tako i hardverskih rešenja, njihovu integraciju i standardizaciju koje je u ovoj oblasti itekako bitna za dalji razvoj.

Krajnja funkcionalnost i sama korist svakog do sada prisutnog rešenja u ovoj oblasti na kraju je bilo ništa drugo do sumariizacije mogućnosti i ograničenja koja su se nametala tokom njihovog razvoja. Mogu se videti veliki broj rešenja koja nisu našla put do šireg auditorijuma i pored izvanrednih ideja i novih originalnih mogućnosti samom činjenicom da nisu pružala dovoljnu fleksibilnost kroz razvojno okruženje ili pak su u manjoj ili većoj meri bila nekompatibilna sa već postojećim okruženjem na koje su trebala da se oslone. Kako se broj kompanija koje se aktivno bave razvojem hardvera i softvera u ovoj oblasti višestruko svake godine povećava, od ključne je važnosti standardizacija i mogućnost da se različite tehnologije povežu u jedinstvenu celinu. U ovom radu su prikazane neke od tehnologija koje pružaju izvanredne mogućnosti na ovom polju. Za krajnje korisnike je to jako bitna stavka koja pruža do skoro neviđene mogućnosti po pitanju gde, kada i na kakvim uređajima mogu da pristupaju ovako naprednom sadržaju.

Interfejs čini jedan od najbitnijih elemenata na koje korisnici prvo obrate pažnju prilikom korišćenja neke aplikacije. 3D kao jedno specifično okruženje nudi neke mogućnosti koje nije moguće prikazati koristeći se drugačijim tehnologijama, recimo tekstualnim režimom na ekranu ili klasičnim prikazom 2D elemenata kao što su slike i video. Takođe, razne studije su pokazale da ljudi mnogo intuitivnije i prirodnije doživljavaju 3D okruženje. U radu pod nazivom „Where am i? Redux“ objavljenog od strane Arnold Trehub-a, je dat interesantan uvid u to na koji način ljudi doživljavaju okolinu oko sebe i na koji način poimaju objekte u prostoru. Autor je izneo nekoliko stavova vezanih za činjenicu da ljudi svet oko sebe vide kao aproksimaciju objekata, i činjenicu da sprega bioloških i psiholoških aspekata kod nas stvara sliku okruženja pa i našeg osećaja za nešto što bi moglo da se nazove „svesno iskustvo“. Određeni broj tehnologija koje bi zbirno mogle da se nazovu „virtuelna realnost“ koriste se ovim principima kako bi iskoristile savremene tehnološke inovacije da nadomeste i prošire naše iskustvo viđenja sveta oko sebe. Neke od trenutno prisutnih na tržištu su *Google Glass*, *Oculus Rift* i *Microsoft HoloLens*.

U ovom radu fokus je *Web* okruženje kao primarno za razmatranje jer je ono daleko najprisutnije i najčešće korišćeno od strane korisnika. Mogućnosti da se pristupi *Web* okruženju sa uređaja različitih namena i karakteristika kao i napredak po pitanju hardverskih performansi koje su ostvarene na ovom polju, daju način da se prikaže sadržaj visokog kvaliteta.



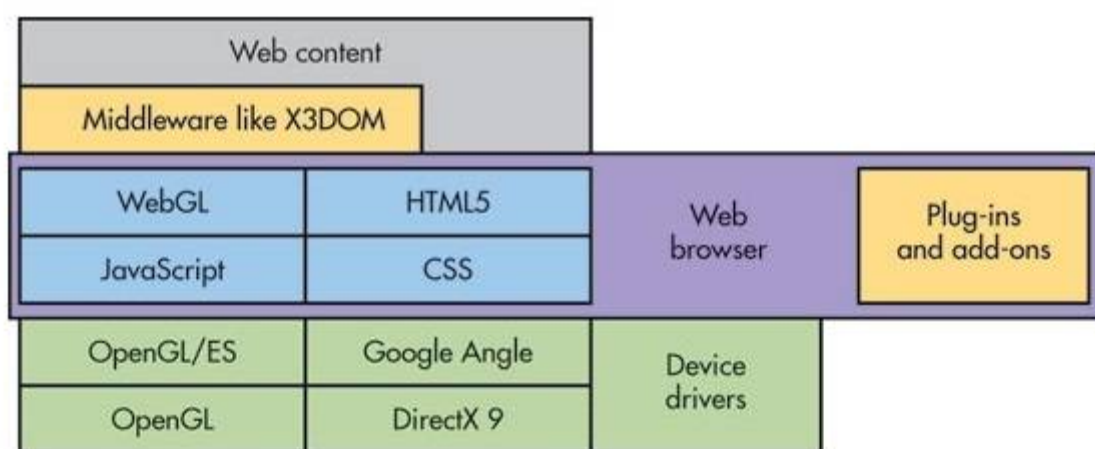
Slika 1- Web strana sa ugrađenim WebGL elementima preuzeto sa: <http://armsglobe.chromeexperiments.com/>

Opisane su sledeće četiri celine: *X3D* kao direktnog naslednika *VRML* i najstarijeg pristupa, *WebGLAPI* okruženje zasnovano na javascript-u, *Unity Engine* jednom od najrasprostranjenije korišćenog *3D engine*-a i svakako najpopularnijeg naročito na tržištu prenosnih uređaja (pametni telefoni i tablet uređaji). *Unreal Engine* rešenje pruža širok spektar mogućnosti i kvaliteta konačnog proizvoda. Biće pomenuta i druga okruženja ukoliko se za tome pokaže potreba u daljem kontekstu ovog rada. Izbor je vođen sa pet kriterijuma: jednostavnost upotrebe, kompatibilnost sa već postojećim platformama, performansama i kvalitetom prikaza, mogućnošću interaktivnosti i na kraju standardizacijom tj. da li je platforma priznata od strane većeg broja relevantnih kompanija i institucija koje se bave ovom oblašću.

WebGl na koji će biti stavljen poseban fokus je svakako pravi izbor po pitanju ovih kriterijuma. Ono što je još posebno bitno naglasiti je i to da ove platforme međusobno omogućuju kompatibilnost pa je tako moguće *x3d* implementirati u *WebGl* pomoću *x3dom*, *Unity* koristi *IL2CPP* komponentu da kod napisanu .Net pretvori u javascript tj. generiše *WebGl* aplikaciju, dok su Mozilla i Epic games koristeći mogućnosti *Firefox*-a i ugrađenog *asm.js* omogućili da se iskoriste prednosti *Unreal engine*-a u *WebGl* okruženju.

1. Web3D tehnologije

Pod pojmom *Web3D* podrazumeva se kolekcija svih tehnologija koje omogućavaju prikaz 3D sadržaja unutar html strane a prikazan pomoću internet pokazivača (*web browser*) uz podršku određenog *plugin* dodatka. Neke od platformi su i direktno podržane od strane proizvođača internet pokazivača pa samim time i ne zahtevaju instaliranje dodatne aplikacije tj. *plugin*-a. Na tržištu je trenutno aktuelno preko dvadeset različitih rešenja, neka od njih su: *vrml*, *x3d*, *java 3d*, *unity* i druge. Aktuelna rešenja kao što je *WebGl* zasnivaju se na podršci kroz *html5* i modernim internet pregledačima gde nema potrebe za dodatnim aplikacijama.



Slika 2-Grafički prikaz implementacije *WebGl*-a u *html* dokument pruzeto sa http://criticaltechnology.blogspot.com/2012_06_01_archive.html

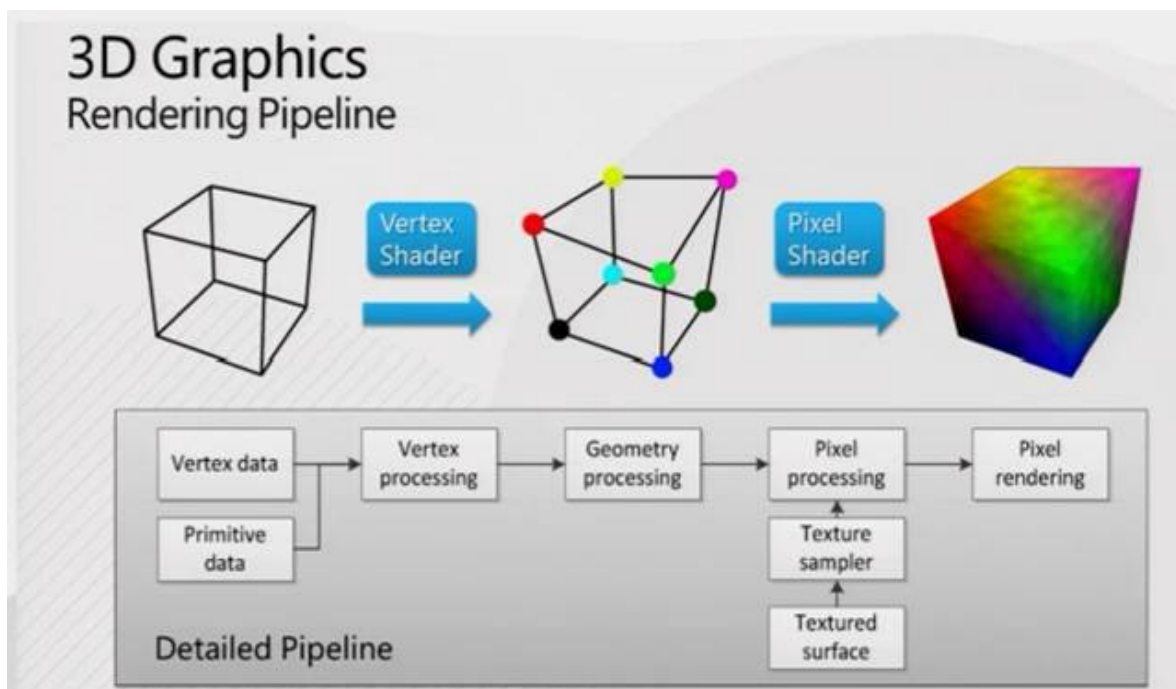
Većina rešenja naročito onih novijeg datuma poseduju podršku tzv. *low-level API*-a kao što su *OpenGL* i *Direct3D* što im omogućava daleko bolje performanse u realnom radu. Rešenja kao što je *WebGl* u mnogome unapređuju svoje performanse kroz direktnu podršku *cpu* i *gpu* jedinica u računaru o čemu će kasnije biti više reči. To omogućava da se vrši evaluacija sistema u različitim uslovima pri korišćenju uređaja različitih performansi i implementiranih softverskih rešenja.

1.1 Kriterijumi i opšti principi rada

Opšti principi rada 3D tehnologija su dati vođeni se sa pet prethodno navedenih kriterijuma. Kao što je već pomenuto jednostavnost upotrebe je jedan od elementarnih kriterijuma za selekciju. Pri tome se rešenja dele u dve grupe, ona koja zahtevaju instalaciju dodatnog softvera tzv. *plugin*-a kao što je to slučaj sa *Unity web player*-om i drugih koja to ne zahtevaju. Svakako da je za korisnike bolje ono rešenje koje omogućava nesmetan rad. Proizvođači internet pregledača takođe kroz različite verzije svojih aplikacija manje ili više podržavaju određena rešenja, tako recimo *Firefox v.35* ima nešto bolju podršku za *WebGl* od *Internet Explorer*-a v.11. Kompatibilnost čini važan deo

razvoja 3D tehnologija. Mogućnost da se ove tehnologije primene na različitim softversko / hardverskim sistemima otvara vrata da se 3D sadržaj pruži korisniku bilo kada ili gde. *WebGl* nudi mogućnost da bude reprodukovana na *Windows*, *Linux* i *OS X* sistemima, na mobilnim platformama koje uključuju *Android*, *iOS*, *Windows Phone*, *Blackberry* i druge...

Opšti princip rada uključuje čuvanje podataka u određenom formatu, njihovo pravilno formatiranje i lakoća prenosa i deljenja. U slučaju *WebGl*-a javascript kod dobija informacije od *canvas* elemenata koji je sastavni deo *Html5Web* strane, sa svim potrebnim podacima kao što su *vertex data*. Ti podaci se dalje prosleđuju u *vertex shader* koji je u stanju da generiše 3D objekat. Odredjuju se koordinate objekata i dodeljuju im se mape ili pridodaju drugi parametri. Takozvani low-level *API* kao što je *OpenGL ES* na koji se *WebGl* direktno i oslanja omogućava da se podaci pošalju na dalju obradu grafičkom podsistemu i time osetno skрати vreme procesiranja podataka i ubrza prikaz na ekranu korisnika.

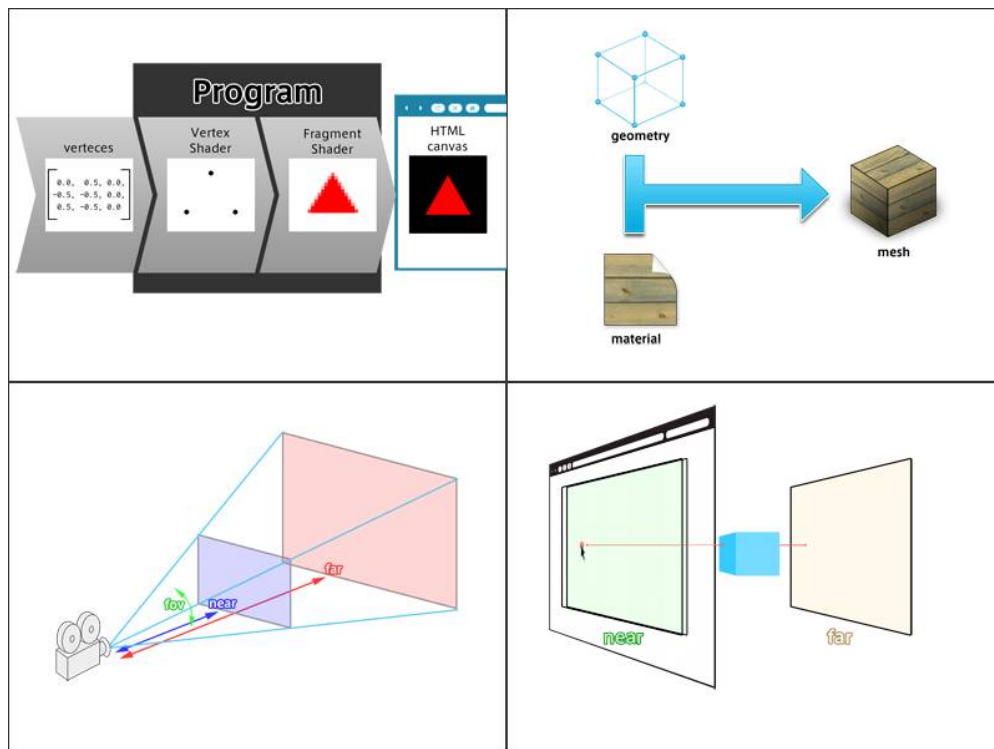


Slika 3- Opšti princip generisanja 3d sadržaja, pruzeto sa <http://www.web-maps.com/gisblog/?paged=2>

Vrlo česta funkcija koju *vertex shader* obavlja je:

$$gl_Position = PROJECTION_MATRIX * VIEW_MATRIX * MODEL_MATRIX * VERTEX_POSITION$$

Ima za cilj da utvrdi koordinate objekta u odnosu na kameru tj. posmatrača pa samim time i da na pravilan način iscrta objekat na ekranu. *View matrix* čini koordinate kamere a u kombinaciji sa *model matrix* dobija se *modelview projection matrix* što omogućava pravilno iscrtavanje na ekranu.



Slika 4- Definisane položaje objekata u world space koordinatama pruzeto sa <https://github.com/yomotsu/figures-to-explain-WebGL-threejs>

Dalji tok zahteva korišćenje *pixel shader*-a koji omogućavaju da se svakoj tački na ekranu dodeli odgovarajuća boja. Ovde postoji veliki broj tehnika koje se koriste, a na kraju se pridodaju i razni efekti kao što su *antialiasing*, *anisotropic filtering* i drugi.

Deojavascript koda koji iscrtava trougao.

```
function setGeometry(gl){
    gl.bufferData(
        gl.ARRAY_BUFFER,
        new Float32Array([
            0,-100,
            150,125,
            -175,100]),
        gl.STATIC_DRAW);
}
```

Sledi dodela boja temenima trougla.

```
function setColors(gl){
    // Pick 2 random colors.
    var r1 =Math.random();
    var b1 =Math.random();
    var g1 =Math.random();
    var r2 =Math.random();
    var b2 =Math.random();
    var g2 =Math.random();

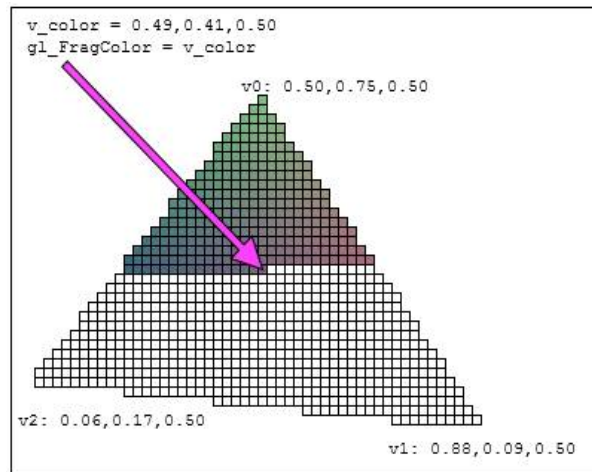
    gl.bufferData(
```

```

gl.ARRAY_BUFFER,
newFloat32Array(
[ r1, b1, g1, 1,
  r1, b1, g1, 1,
  r1, b1, g1, 1,
  r2, b2, g2, 1,
  r2, b2, g2, 1,
  r2, b2, g2, 1]),
gl.STATIC_DRAW);
}

```

Iscrtavanje svakog piksela na ekranu uključuje definisanje koordinata i boje predstavljenu kao *rgbz* zapis. Pri svakom osvežavanju ekrana vrši se novo izračunavanje potrebnih parametara, gde je ovaj proces donekle moguće ubrzati naročito ukoliko ne dolazi do promene položaja određenih elemenata na ekranu.



Slika 5-Dodeljivanje boje svakom zasebnom pikselu na ekranu preuzeto sa <http://WebGlfundamentals.org/WebGl/lessons/WebGl-how-it-works.html>

```

// look up where the vertex data needs to go.
var positionLocation = gl.getAttribLocation(program,"a_position");
var colorLocation = gl.getAttribLocation(program,"a_color");

// Create a buffer for the colors.
var buffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, buffer);
gl.enableVertexAttribArray(colorLocation);
gl.vertexAttribPointer(colorLocation,4, gl.FLOAT,false,0,0);

// Set the colors.
setColors(gl);

```

1.2 Performanse

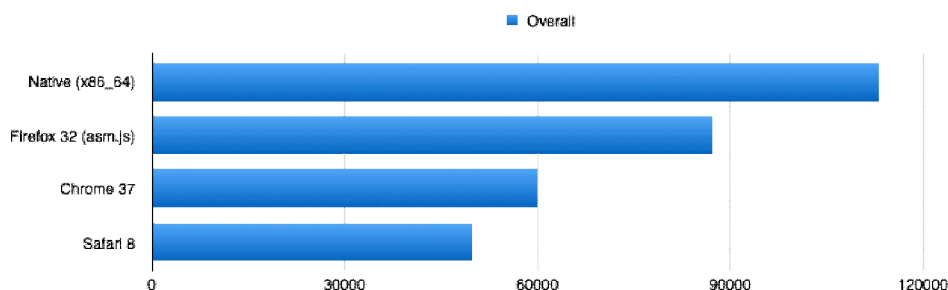
Performanse sistema su uveliko određene izborom softversko / hardverskog rešenja. *WebGl* se kao što je to već navedeno zasniva na javascript-u koji pruža različite performanse zavisno od *cpu* jedinice ali isto tako i od odabira internet pregledača. Na narednom grafiku se jasno uviđaju razlike između internet pregledača i njihovih implementacija javascript rešenja.

Browser Performance - Core i7-860					
Benchmark	IE Old	IE Experimental	Chrome 40	Firefox 35	Percentage Change
Sunspider (lower is better)	149.7ms	144.6ms	260.9ms	220.1ms	3.4%
Octane 2.0 (higher is better)	9861	17928	17474	16508	81.8%
Kraken 1.1 (lower is better)	3781.2ms	2077.5ms	1992.8ms	1760.4ms	45.1%
WebXPRT (higher is better)	913	1083	1251	1345	18.6%
Oort Online (higher is better)	1990	2170	5370	3900	9%
HTML5Test (higher is better)	339	344	511	449	1.5%

Slika 6 – Performanseu raznim javascript testovima

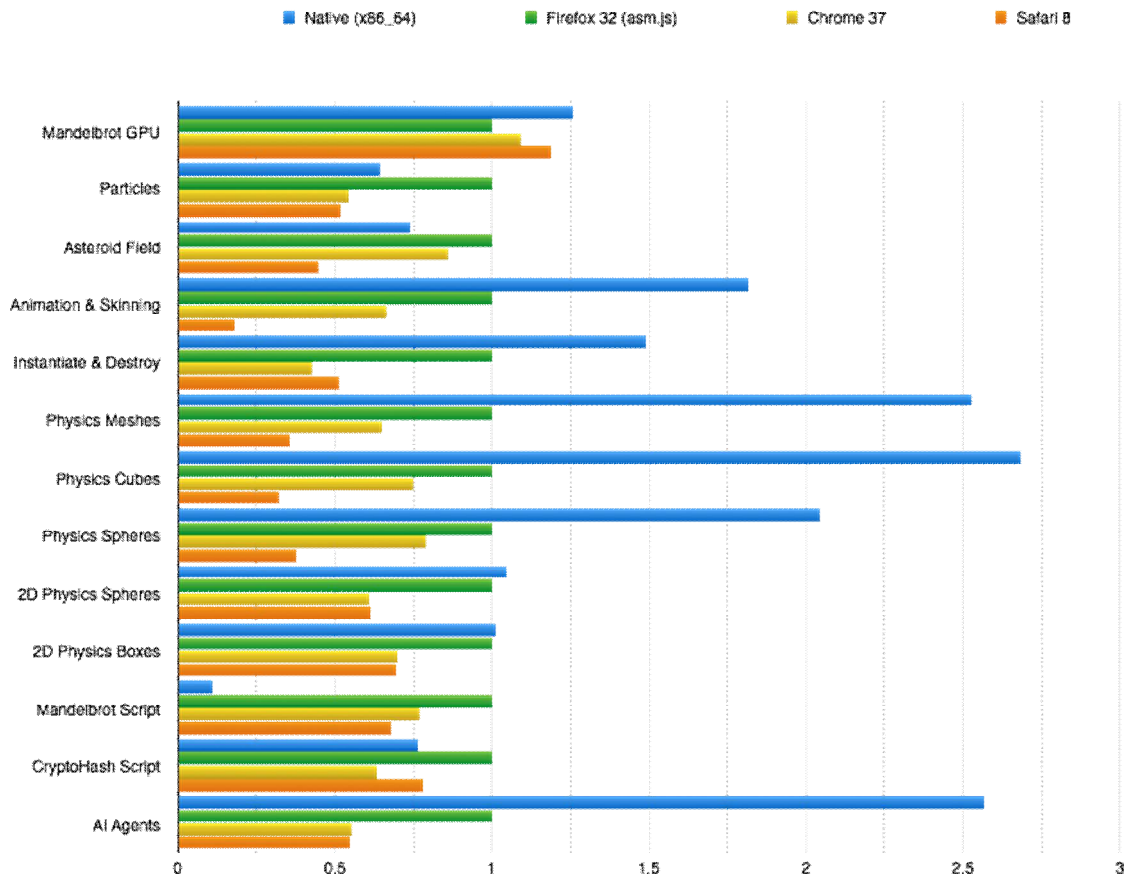
U slučaju nekih drugih rešenja kao što je *Unity*, problem oko performansi se dodatno komplikuje činjenicom da se sav kod napisanu *Unity engine*-u mora pre kompajlirati u *javascript* kako bi mogao da se pokrene u *WebGl* okruženju. Za potrebe toga *Unity technologies* je napisao poseban script eksporter *il2cpp* koji omogućava da se većina funkcionalnosti *Unity engine*-a prenese u *WebGl*. Neke od mogućnosti još uvek nisu prisutne u *WebGl* kao što su *multi-threading* i *simd* pa su samim time i performanse nešto slabije.

Za potrebe testiranja razvijen je *Unity WebGL benchmark* koji meri performanse 2D i 3D fizike, mapiranje objekata, propusnu moć gpu jedinice, animacije, instanciranje objekata, skriptovanje sadržaja i drugo. U narednoj tabeli je moguće videti ostvarene različite performanse pojedinačnih mogućnosti razvijenih u *Unity engine*-u a portovanih u *WebGl*.



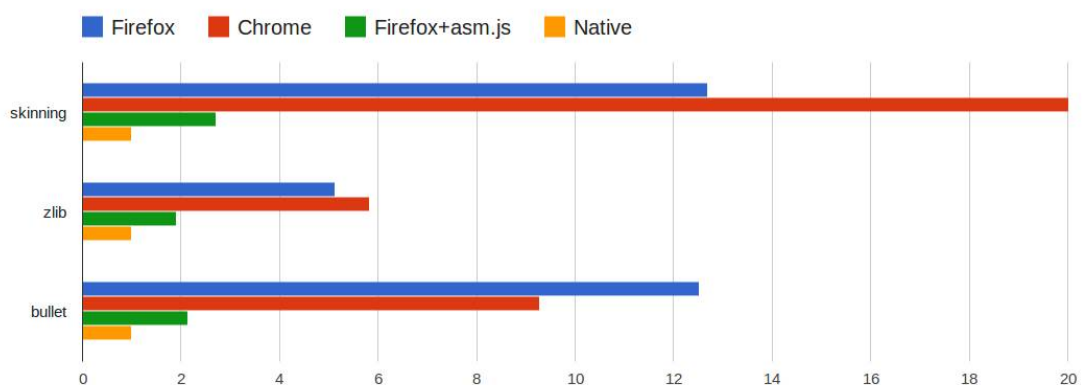
Slika 7 – Zbirne performanse

Na grafiku treba posebno obratiti pažnju na performanse *native* moda kao i očigledno jako dobru implementaciju *Firefox* rešenja koje se ogleda kroz specifično napisan *asm.js*.



Slika 8-Unity WebGL Benchmark testovi preuzeto sa <http://blogs.unity3d.com/2014/10/07/benchmarking-unity-performance-in-WebGL/>

Unreal Engine je kao i *Unity engine* portovan u *WebGl* okruženje zahvaljujući saradnji sa Mozzila fondacijom. Detaljan prikaz je dat kasnije, za sada će biti prikazanesamo performanse istog. I u ovom slučaju je uočeno da najbolje rezultate svakako postižu aplikacije u *native* modu ali se pokazalo da je *Firefox* sa *asm.js* postigao dobre rezultate koji jako malo zaostaju za *native* modom.



Manje vrednosti su bolje

Slika 9 – Performanse Unreal engine-a u WebGL okruženju preuzeto sa <http://html5app.es/juegos-3d-en-firefox/>

Mogao bi da se izvuče zaključak da su performanse *WebGl* okruženja prilično fluktuirajuće. Veliki broj parametara i promenljivih utiče na ukupne performanse pa se

samim time može govoriti i u pravcu daljeg unapređivanja istih. Evidento je da bi korisnici mogli da primete i vidljive razlike u brzini i kvalitetu izvršavanja aplikacija i to je nešto o čemu treba voditi posebnu brigu prilikom razvoja aplikacija ovog tipa. Dosadašnja merenja svakako daju *WebGL*-u prednosti u odnosu na neka starija rešenja tipa *vrml / x3d*, ipak vidi se da je u odnosu na *native* režim rada aplikacija ipak razlika velika, reda veličine i do deset puta su performanse *native* aplikacija bolje u odnosu na *WebGl*.

Što se tiče samih internet pregledača, *Firefox* u kombinaciji sa *asm.js* je dao najbolje rezultate na Windows platformi. Dalje performanse treba istražiti na polju mobilnih platformi gde je prisustvo ovih tehnologija još više prisutno i postaje sve značajnije u odnosu na desktop rešenja.

1.3 Kvalitet prikaza sadržaja

Kvalitet prikazanog sadržaja je često prva stvar koju korisnici ocenjuju u kontaktu sa ovim tehnologijama. Veliki doprinos unapređenju kvaliteta dao je *Html5* standard koji je omogućio da se u okviru *Web* strane prikaže sadržaj koji je do skoro mogao biti prisutan jedino u takozvanim *native* aplikacijama. *WebGLAPI* je zasnovan na *OpenGLAPI*-u koji je dalje prilagođen za potrebe mobilnih platformi pod nazivom *OpenGLES* (konkretno u verziji 2.0). Uz ovo treba naglasiti da *Html5* ima ugrađenu podršku za *Css3d*. Isti poseduje mogućnosti za transformaciju, tranziciju i primenu različitih filtera različitih elemenata koji su implementirani u *Web* stranu. *WebGl* se odlično uklapa u koncept dinamičkih web strana i kao takav je i zamišljen, direktna podrška kroz *Html5* i *Css3d* kao i ugrađeni element *Canvas2d* omogućavaju gotovo neograničene mogućnosti za primenu.



Slika 10 – Demo naprednih rendering tehnika preuzeto sa <https://developer.nvidia.com/content/unreal-engine-4-rivals-demo-running-tegra-k1-google-io>

Važan element u samom prikazu sadržaja je deo *WebGLAPI*-a zvan *shaders*. Neki od njih su već pomenuti kao što je *vertex i pixel shader*, pojednostavljeno rečeno moglo bi da se kaže da oni čine specijalne efekte koji omogućavaju napredne tehnike iscrtavanja

ekrana. Oni su hardverski podržani i zahvaljujući *Direct3D* ili *OpenGLAPI*-u izvršavaju se unutar *gpu* (*graphics-processing unit*) jedinice čime se jako skraćuje vreme obrade podataka. Svi *shader*-i koji se izvršavaju od strane *WebGl* ili *Css* filtera su definisani od strane *OpenGL ES shader language*-a ili skraćeno *GLSL ES*.

Implementacija naprednih efekata i filtera kao što su *raytracing*, *reflections*, *refractions*, *ambient occlusion*, *soft shadows*, *antialiasing*, *caustics* i drugi omogućuju kvalitetno iscrtavanje ekrana (*rendering*) dok se razvojnim timovima ostavlja mogućnost da dodaju druge proceduralne efekte po sopstvenoj potrebi.

Sledeći javascript kod daje uvid u generisanje elemenata na ekranu i dodavanje raznih efekata:

```
varshader=THREE.ShaderLib["cube"];
shader.uniforms["tCube"].value=envMap;

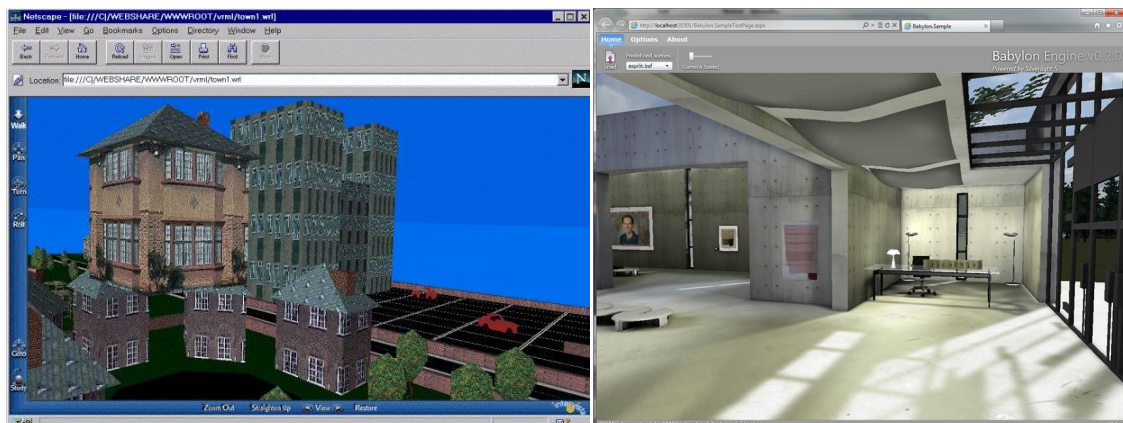
varmaterial=newTHREE.ShaderMaterial({

fragmentShader:shader.fragmentShader,
vertexShader:shader.vertexShader,
uniforms:shader.uniforms,
side:THREE.BackSide

}),

mesh=newTHREE.Mesh(newTHREE.CubeGeometry(500,500,500),material);
scene.add(mesh);
```

Na osnovu priloženog koda vidi se generisanje objekta po nazivu *mesh*, u pitanju je generisana kocka dimenzija 500x500x500 kojoj je pridodat materijal. Za potrebe primera korišćena je biblioteka *Three.js* koja omogućava programerima da na brz način generišu scenu i elemente 3D okruženja.

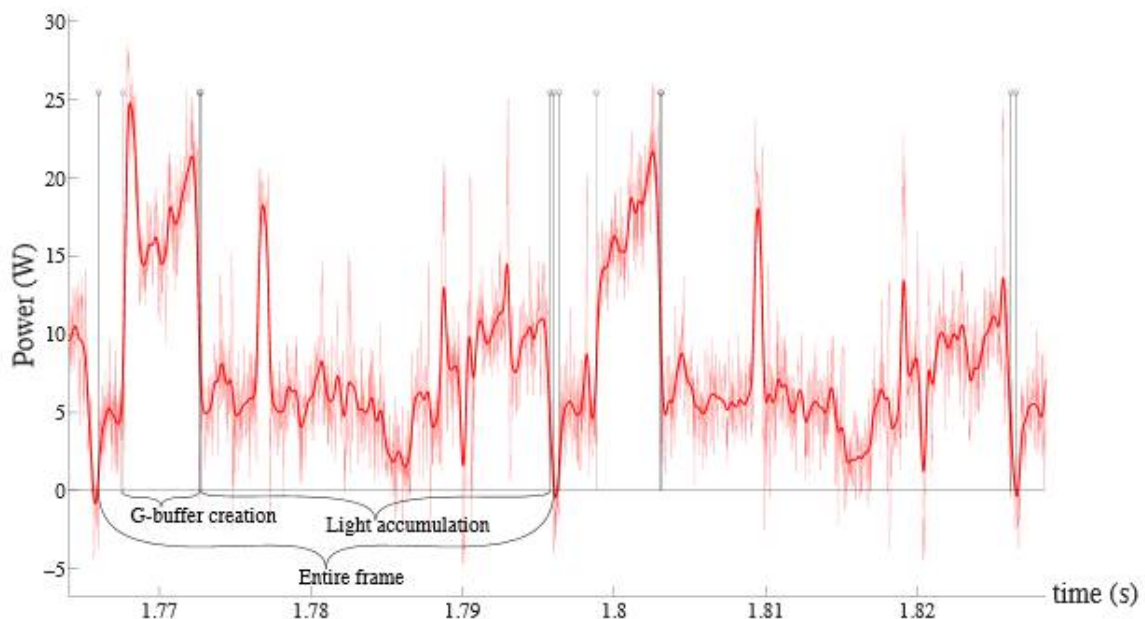


Slika 11 – Iscrtavanje ekranaVRML (levo) i WebGL (desno)

Na slici 11 se vidi evidentan napredak u kvalitetu iscrtavanja ekrana između ranih početaka koje je *vrml* obeležio sredinom '90.-tih godina i *WebGl*-a koji je danas aktuelan.

Zahvaljujući ogromnom napretku grafičkih čipova (*gpu*) kako u klasičnim računarima tako i na mobilnim platformama, omogućeno je da se velika količina procesorske obrade podataka sa *cpu*-a prebaci na specijalizovanegpu jedinice. Sa daljim unapređenjem hardvera naročito na mobilnim platformama omogućiti će se još veća prisutnost ovakvog sadržaja u svakodnevnoj upotrebi.

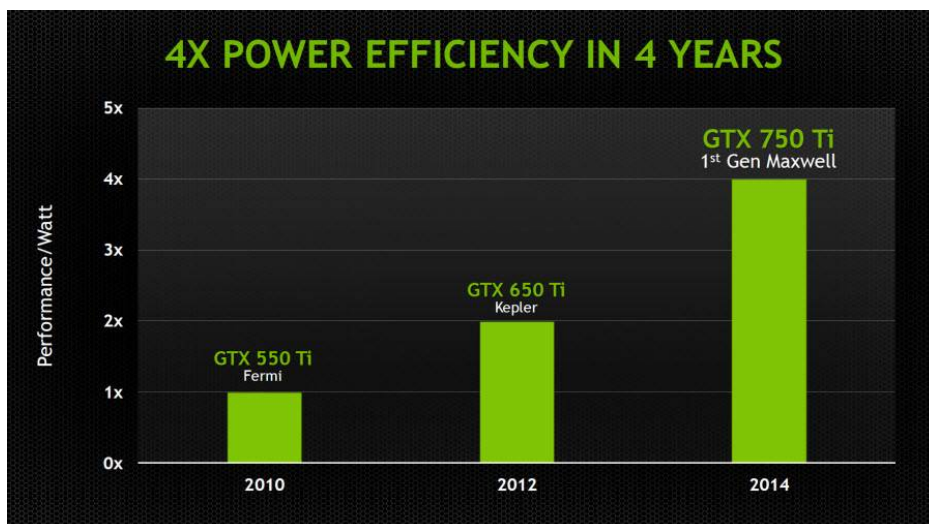
Veliki problem koji se javlja prilikom obrade kompleksnog sadržaja naročito na prenosnim uređajima sagledava se u količini električne energije koja se pri tom procesu potroši. Kako su *gpu* jedinice jedne od najvećih potrošača u računaru logično je da se pretpostavi da bi ovakav sadržaj bio prisutan na ekranu samo određen vremenski period. Na sledećem grafiku mogu da se uoče varijabilne količine električne energije koja se troši za ispunjavanje određenih procesa.



Slika 12 - Potrošnja energija u različitim fazama iscrtavanja ekrana (rendering) preuzeto sa <http://jcgt.org/published/0001/01/03/paper.pdf>

Za potrebe testa autori Tomas Akenine-Moller i Bjorn Johnssonsu koristili *Intel Ivy Bridge cpu* sa integrisanim *HD Graphics 4000 gpu*-om koristeći *OpenGLAPI* za dalja testiranja. Iako autori smatraju da treba izbegavati ovakav vid testiranja potrošnje energije, ipak je moguće izvesti neke zaključke naročito onaj vezan za činjenicu da se najviše energije troši upravo u trenutku kreiranja takozvanog *G-buffer*-a tj. kreiranje i pozicioniranje objekata, zadavanje tekstura i ostalih parametara (u slučaju *WebGL*-a to je posao koji spada pod okrilje *vertex i pixel shader*-a).

Na narednom grafu se može videti kako je odnos performansi i potrošnje energije drastično unapređen tokom zadnjih nekoliko godina razvoja. Zahvaljujući naprednim tehnološkim procesima, proizvođači modernih grafičkih čipova uspeali su ostvare zapaženi uspeh u unapređenju efikasnosti.



Slika 13 – Odnos utrošene energije prema performansama gpu-a preuzeto sa <http://www.ubergizmo.com/2014/02/nvidia-maxwell-gpu-for-geforce-cards/>

1.4 Interaktivnost i standardizacija

Mogućnosti *Web3D* okruženja a posebno *WebGl*-a su sasvim sigurno velike. Činjenica da se *WebGl* prirodno integriše u *Html* okruženje omogućava mu veliki stepen interaktivnosti i komunikacije sa drugim elementima na koje se ovo okruženje oslanja. Javascript u isto vreme pruža velike prednosti u smislu lakoće integracije ali takođe i generiše niz problema zbog kojih je *WebGl* tokom svog razvoja imao problema da bude priznat od određenih subjekata. Veliku podršku ovom standardu dala je kompanija Google Inc. koja je i neke od svojih servisa unapredila korišćenjem *WebGl*-a kao što je to recimo slučaj sa *Google Maps*-om.

Trenutno se sa ovom tehnologijom još uveliko eksperimentiše, najviše je prisutna na polju 3D video igara, naučno istraživačkih radova, generisanju edukativnog softvera, raznim vrstama simulacija i vizualizacijom različitog sadržaja. U okviru vizualizacije sadržaja i interaktivnog manipulisanja istim ostvaren je kvalitetan napredak. Prikaz statističkih podataka, podataka vezanih za prostorno uređenje, vizuelizacija arhitektonskih, građevinskih ili mašinskih elemenata omogućava korisnicima da značajno unaprede svoje poslovanje.

Standardizacija i direktna podrška većeg broja kompanija na tržištu su osnova daljeg razvoja ovih tehnologija. Vreme kada je neka kompanija mogla da samostalno razvije sopstveno rešenje a da pri tome se ne zasniva na kooperaciji sa drugim rešenjima je odavno završeno. Danas je u ovu granu industrije uključen ogroman broj kompanija, razvojnih studija, obrazovnih institucija i pojedinaca koji daju veliki doprinos.

Starije tehnologije kao što su *vrm* i *x3d* su pod okriljem *Web3D Consortium* koji je neprofitna organizacija i okuplja veći broj članova. Kako su ove tehnologije prevaziđene danas je sve manje sadržaja i stiče se utisak da opada interesovanje korisnika za istim. *WebGl* je razvijen od strane Mozilla fondacije, ali je danas u nadležnosti *Khronos* grupe (<https://www.khronos.org/WebGl/>). Ovu grupu čini preko 100 kompanija od kojih su neke: AMD/ATI, Nvidia, Apple, Intel corporation, IBM, Google, Microsoft i druge. U okviru njihove nadležnosti se nalazi još i standardizacija *OpenGL*, *OpenCL*, *OpenGL ES*, *Collada* i

drugih tehnologija. Samim time *WebGl* je našao jaku upor i sigurnu podršku u širem spektru softverskih i hardverskih rešenja na kojima ova platforma može da se nađe.

	<i>WebGl 1.0.2</i> (<i>OpenGL ES 2.0</i>)	<i>WebGl 2</i> (<i>OpenGL ES 3.0.3</i>)
Desktop platforme	Chrome 9 Firefox 4 Safari 6 Opera 11 Internet Explorer 11	n/a
Mobilne platforme	Android 2.2 iOS 8.0 BlackBerry OS 10 Windows Phone 8.1 Raspberry Pi	Android 4.3 iOS Windows Phone BlackBerry OS

WebGl koji je u trenutku pisanja ovog rada još uvek u razvoju zasniva se na novijoj verziji *OpenGL ES 3.0*, tako da podaci na kojim platformama će biti prisutan još uvek nisu poznate. Ono što je sigurno da postoje izvesne razlike pri čemu neće postojati puna reverzna kompatibilnost u odnosu na *WebGl 1.0.2*. Pozitivna strana je to da će u startu postojati podrška za sve aktuelne gpu-e koji se ugrađuju u mobilne platforme kao što su Adreno 3xx i 4xx, Nvidia Tegra, Mali T6xx i druge.

1.5 Vulkan, Mantle, DirectX 12

U ovom odeljku će biti reči o nekim od tehnologija koje će u budućnosti imati jak uticaj i generalno odrediti smer u kome će se buduća rešenja razvijati. Tehnologije koje su do sada spominjane su prisutne na tržištu već duži niz godina pa su samim time uočena i razna ograničenja i nedostaci istih. Jedan od češćih problema i svakako najinteresantnijih za posmatranje je prevelika diverzifikovanost upotrebljenih *API*-ja. Veliki broj platformi koji su danas korisnicima dostupni učinili su da se danas koristi više od 5 low-level *API*-a u paralelnoj upotrebi, kao što su: *Direct3D 9* i *11*, *OpenGL 4.4* i *4.5*, *OpenGL ES 2.0*, *3.0* i *3.1*, *Mantle* i drugi... Hardverski uređaji podržavaju jednu, vrlo često i više rešenja određene verzije što razvojnim timovima zadaje dodatne probleme u razvoju aplikacija.

VulkanAPI koji je predstavila *Khronos grupa* je takozvani *next-gen API* i treba da direktno zameni postojeći *OpenGL standard* i sve *API*-e koje on zastupa. Cilj je da bez obzira na uređaj koji se posmatra bilo to desktop rešenje ili mobilna platforma ovaj *API* bude prisutan kao jedno univerzalno rešenje. *VulkanAPI* će u daljoj budućnosti zameniti *OpenGL* i *OpenGL ES*, a od *Mantle API*-a će preuzeti direktno neka gotova rešenja (sama činjenica da je AMD / ATI direktno podržao dalji razvoj *Vulkan*-a i preusmerio resurse sa *Mantle* na *Vulkan* u mnogome govore o ozbiljnosti ove tehnologije).

Prednosti *VulkanAPI*-a su višestruke, primera kao što su podrška za veliki broj platformi, izuzetna podrška za razne gpu jedinice i smanjenje opterećenja cpu jedinice u daljoj obradi podataka, bolja podrška za *multithreading*, nezavisnost od operativnog sistema, bolja komunikacija sa grafičkim drajverima, veće mogućnosti da programeri po svojoj želji pišu i unapređuju *shader* modele.

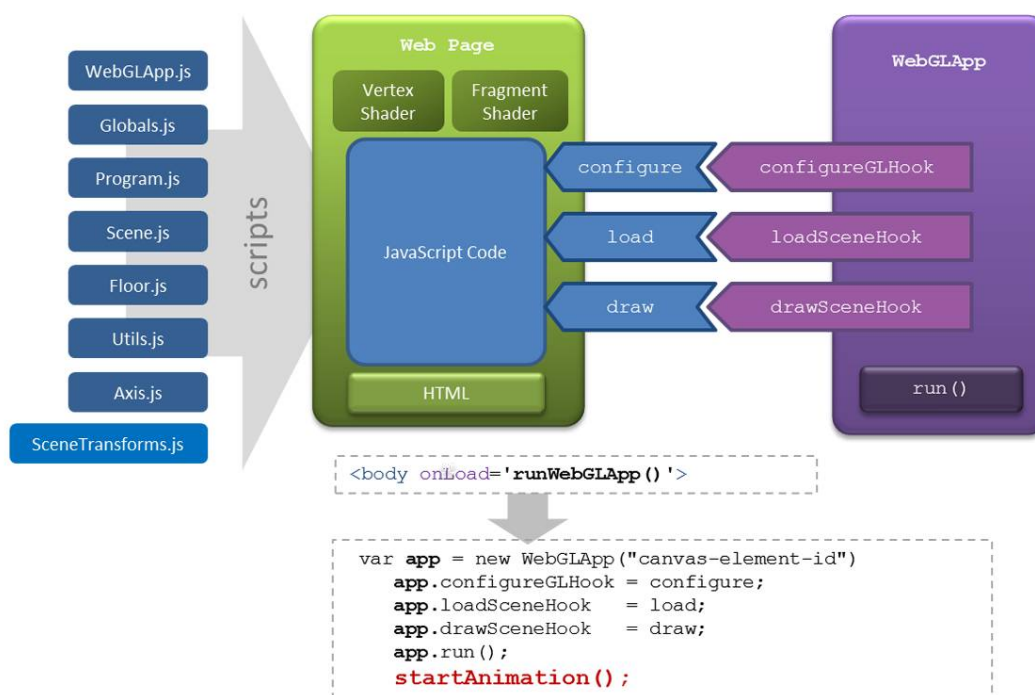
WebGl koji se do sada oslanjao na *OpenGL ES* bi u nekim daljim verzijama mogao da kroz *Vulkan* dalje unapredi performanse koje su trenutno i najveći problem. Testiranja koja su do sada obavljena sa *VulkanAPI* tehnologijama pokazuju da unapređenja performansi idu i preko 50%.

MantleAPI koji je AMD / ATI razvio za potrebe svojih grafičkih čipova pokazuju da na postojećim platforma takođe ima mesta za unapređenje performansi. Konkretno ovo rešenje je zamena za *Direct3D 9* i *11* na *Windows* operativnom sistemu, omogućava da se performanse unaprede i do 45% zavisno od uslova i samih aplikacija. Kako se fokus daljeg razvoja prebacio na *Vulkan*, *Mantle* će ostati vezan za platforme na *Windows 7* i *8.1* operativnom sistemu.

Direct3D 12 koj dolazi u sklopu *DirectX* i *Windows 10* operativnog sistema je jedan od najvažnijih *API*-a u skorijoj budućnosti (*Windows 10* bi trebalo da postane dostupan sredinom 2015.god.). Donosi veliki broj unapređenja naročito po pitanju performansi i efikasnosti. Iz ugla *WebGl* ovo bi moglo da bude jako značajno kako postoji *ANGLE renderer* koji omogućava da se *OpenGL ES* prekompajlira u *Direct3D* (Google Chrome na taj način obrađuje *WebGl* sadržaj već sada). Prema tvrdnjama Microsoft-a *Direct3D 12* donosi unapređenja performansi od 40 do 70 procenata u odnosu na *Direct3D 11*.

2. WebGL iz ugla Three.js, prednosti i nedostaci

Three.js je javascript biblioteka koja se zasniva na *WebGL* standardu a omogućava programerima da na lak i brz način generišu 3D scene. Objavljen tokom 2010. godine do sada je stekao veliku popularnost, dobro je podržan kroz razne primere i dokumentaciju što je od posebnog značaja za programere i individue koje se interesuju za ovu oblast. Neke od prednosti ove biblioteke su: puna podrška za *GLSL*, podrška za veći broj materijala, svetlosnih efekata, animacija i objekata, uvoz podataka raznih tipova i drugo.



Slika 14 – Veza između html, WebGL (three.js) i glsl shader-a, preuzeto sa <https://www.packtpub.com/books/content/WebGL-animating-3d-scene>

Three.js je biblioteka koja već ima ugrađen veliki broj elemenata koji olakšavaju programerima da brzo generišu sadržaj. Neki od najčešće korišćenih elemenata su *camera*, *scene*, *renderer* i drugi. Sledeći kod napisan u *Three.js* pokazuje konkretnu upotrebu istih.

```
var scene = new THREE.Scene();
var camera = new THREE.PerspectiveCamera(75, window.innerWidth /
window.innerHeight, 0.1, 1000);

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

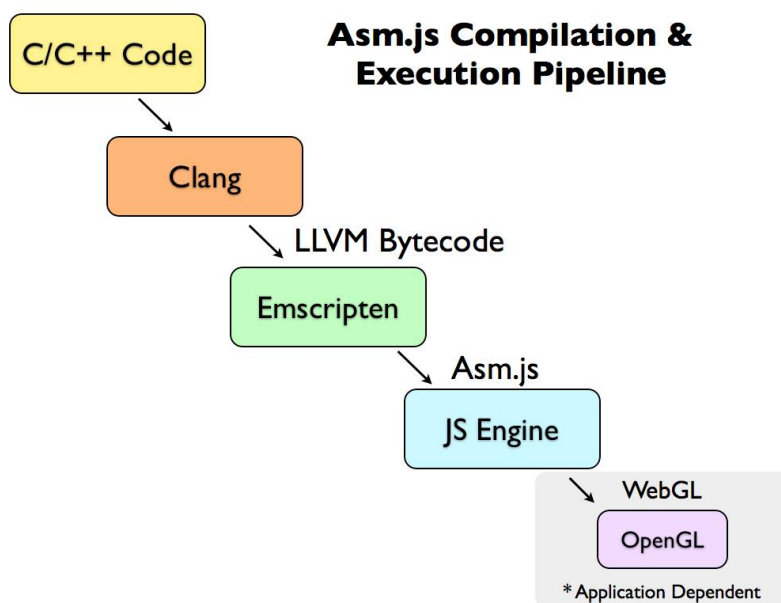
Predefinisani elementi kao što je camera omogućavaju podešavanja o tipu kamere, fokusu, odnosu dijagonale i drugo. Ovakvi ugrađeni mehanizmi omogućavaju da se elementima efikasno rukuje bez mnogo ulaganja truda u razumevanje principa rada samog *WebGL*-a.

U *Three.js* je ugrađen veći broj klasa od važnosti, neke od njih su: *EventDispatcher*, *BufferGeometry*, *JSONLoader*, *Loader*, *WEBGLRender*. Korisnici mogu da dopunjuju i menjaju postojeće klase i na taj način dalje prilagode kod svojim potrebama.

Neki od problema koji prate *Three.js* a i *WebGl* uopšte mogu se generalno nabrojati:

- Slabije performanse u odnosu na aplikacije napisane u *native* kodu
- Izvesni sigurnosni rizici koji uključuju direktan pristup *gpu* i grafičkim drajverima
- Obimniji sadržaji zahtevaju više vremena (*download time*) pri čemu tip i kvalitet konekcije koji korisnici poseduju ima glavnu ulogu
- Varijabilni kvalitet prikaza zavisno od softversko hardverske platforme
- Ograničene sfere upotrebe date tehnologije

Po pitanju prednosti treba pomenuti još samo da *WebGl* omogućava praktično da se kod pisan na bilo kojoj drugoj platformi prekompajlira i koristi direktno iz *Web* okruženja. Na sledećem grafiku dajemo šemu izvršenja aplikacije napisane u *Unreal engine*-u i korišćenjem *asm.js* kompajlirane u *WebGl* javascript kod.



Slika 15 – Prekompajliranje koda tako da bude izvršen kao *WebGl* aplikacija preuzet sa <http://html5app.es/juegos-3d-en-firefox/>

Prednosti koje *Three.js* takođe donosi čine još i mogućnost proširenjem drugim javascript bibliotekama, robustnost okruženja, brzina, objektno orijentisani pristup kreiranju elemenata, podršku za uvoz i izvoz podataka različitih formata i drugo. Ono što se mora naglasiti jeste da *Three.js* nije *3D engine* pa mu samim time neke funkcije nedostaju koje se moraju dodatno programirati.

3. Ideja projekta

Za potrebe projekta kreirana je virtuelna scena u kojoj je prikazan helikopter model Mi-8 transportnog tipa. Ovaj helikopter se uglavnom koristi za transport ljudi i opreme i koriste ga kako civilne tako i vojne službe. Korisnici mogu da se upoznaju sa virtuelnim modelom kako njegovom spoljašnošću, tako i unutrašnjosti samog helikoptera. Posebna pažnja posvećena je dimenzijama, boji i generalnom prikazu kako bi se što vernije preneo utisak korisnicima koji nisu imali prilike da se uživo upoznaju sa ovim modelom helikoptera.

Sam projekat nudi i mogućnost da korisnici identifikuju određene elemente u samom heliopteru kao što su razni uređaji, medicinski, transportni i razni drugi objekti koji bi mogli da se nađu u samoj letelici.



Slika 16 - Helikopter tipa Mi-8 u sastavu vazdušnih snaga vojske Indije, preuzeto sa <http://www.bharat-rakshak.com/media/AeroIndia2007/Static/MiscAircraft/Mil+Mi-8+of+the+Throughbreds.jpg.html>

Kretanje kroz virtuelno okruženje omogućeno je korišćenjem tehnologije poznate kao *free-cam*, što omogućava korisniku da pomoću miša i tastature vrši tranziciju kamere i *point of view* elementa i na taj način ima pogled na scenu iz ugla koji njemu najviše odgovara. Funkcionalnost koja je takođe bitna je i da korisnik može u svakom trenutku da koristi *mouse pointer* kako bi vršio selekciju određenih elemenata koje vidi na ekranu kao i određeni stepen interakcije sa istim.

4. Koraci realizacije

Prilikom izrade projekta korišćene su dve tehnologije, 3d *modeling* kao osnovni element za generisanje potrebnih 3d objekata i *webgl* okruženje koje sa druge strane omogućava razvoj programskog koda potrebnog da se pruže funkcionalnosti kao što su kretanje kroz virtuelnu scenografiju, *free-cam* mod, selektovanje pojedinih elemenata scene, realističan prikaz okruženja i drugo.



Slika 17 - WebGL okruženje

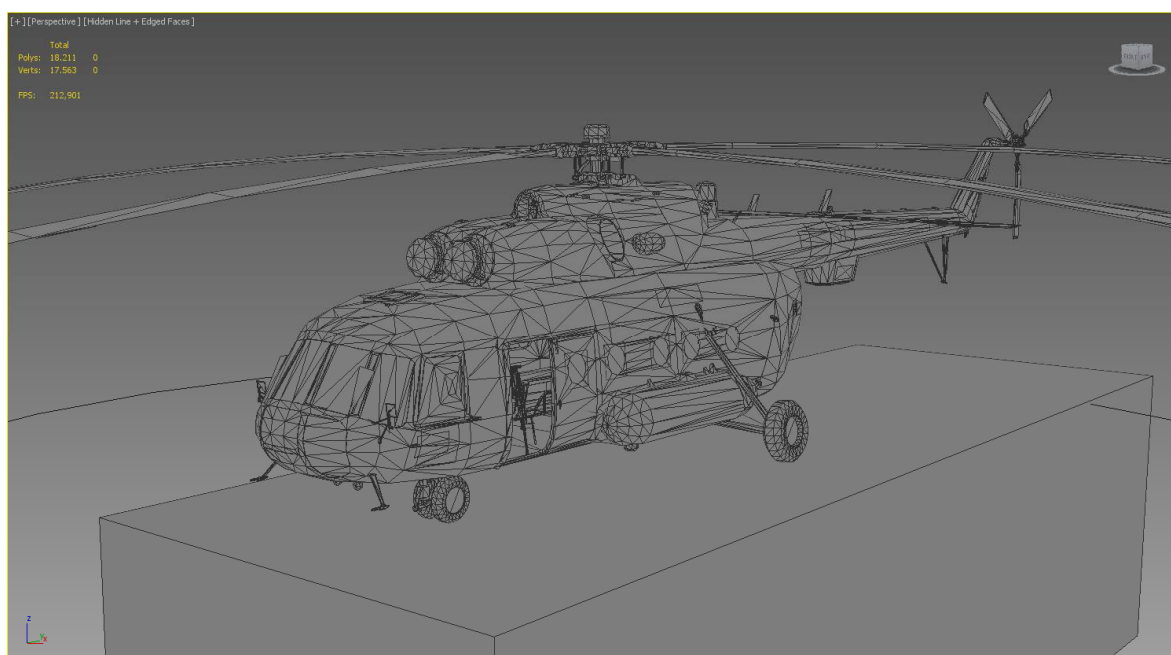
Za potrebe 3d *modeling*-a korišćen je komercijalan softver *3ds Max Design* koji je pružio sve neophodne alate potrebne za razvoj 3d objekata, kreiranje materijala, razna testiranja u *pre-rendering stage* fazi razvoja projekta kao i kreiranje slika visokog kvaliteta. *WebGl* je izabran za razvoj scene, implementaciju elemenata i dodavanje funkcionalnosti što je *javascript* programski kod omogućavao. Konkretno je korišćena *Three.js* biblioteka uz dodatno korišćenje *three.js web editor*-a (ovaj editor je u toku pisanja ovog dokumenta još uvek u razvoju tako da se ne preporučuje njegovo korišćenje u projektima komercijalnog tipa ali je za potrebe ovog projekta omogućio dobre performanse).

Celi razvoj projekta možemo podeliti u sledeće korake:

- Konceptualni model okruženja
- Skaliranje elemenata kako bi pružili tačan odnos veličina i dimenzija
- Kreiranje 3d objekata, materijala, i pomoćne scenografije za testiranje modela
- Kreiranje *webgl* scene kroz programski kod i uvoz 3d objekata i materijala u istu
- Dodavanje raznih funkcionalnosti kroz programski kod



Slika 18 – U razvoju 3d modela korišćene su raspoložive informacije o datom modelu helikoptera

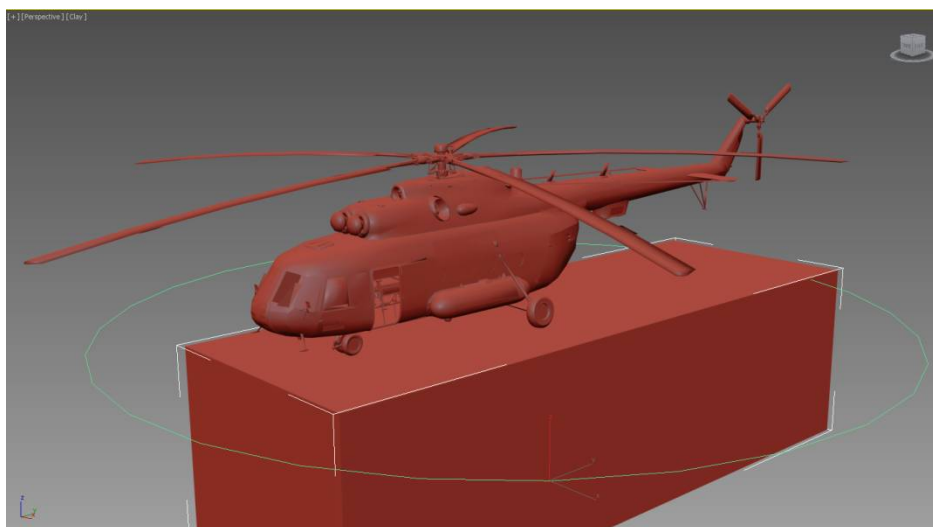


Slika 19 – Wire prikaz modela helikoptera

5. Kreiranje modela u 3d-u

3D objekat datog helikoptera sastoji se od približno 15 000 poligona. Dati model ne spada u visoko realističnu klasu objekata (*hi – poly model*) ali je za potrebe webgl okruženja bio zadovoljavajući. Činjenica da je virtuelno okruženje trebalo da pruži najmanje 30 fps-a (*frame per second*) značilo je da mora da se vodi računa o optimizaciji objekata.

Na narednoj slici može se videti takozvani *clay model* prikaz koji omogućava detaljnu proveru objekta kako se ne bi javili određeni artifakti vezani za *non uniform* normalizaciju objekata („probijanje“ objekta tj. nepravilna vizuelizacija određenih poligona prilikom iscrtavanja objekata).



Slika 20 – Clay model prikaz

Objekat je sačinjen od 6 glavnih grupa i 3 pomoćnih. Svaka grupa je sačinjena od više podobjekata, koji je opet sa druge strane sačinjen od većeg broja elemenata koji omogućavaju zasebnu manipulaciju. Ova složena struktura je uslovljena samim načinom kreiranja objekata u aplikaciji ali i potrebom da se na lakši način vrše razne manipulacije i transformacije na pojedinačnim elementima samog objekta.

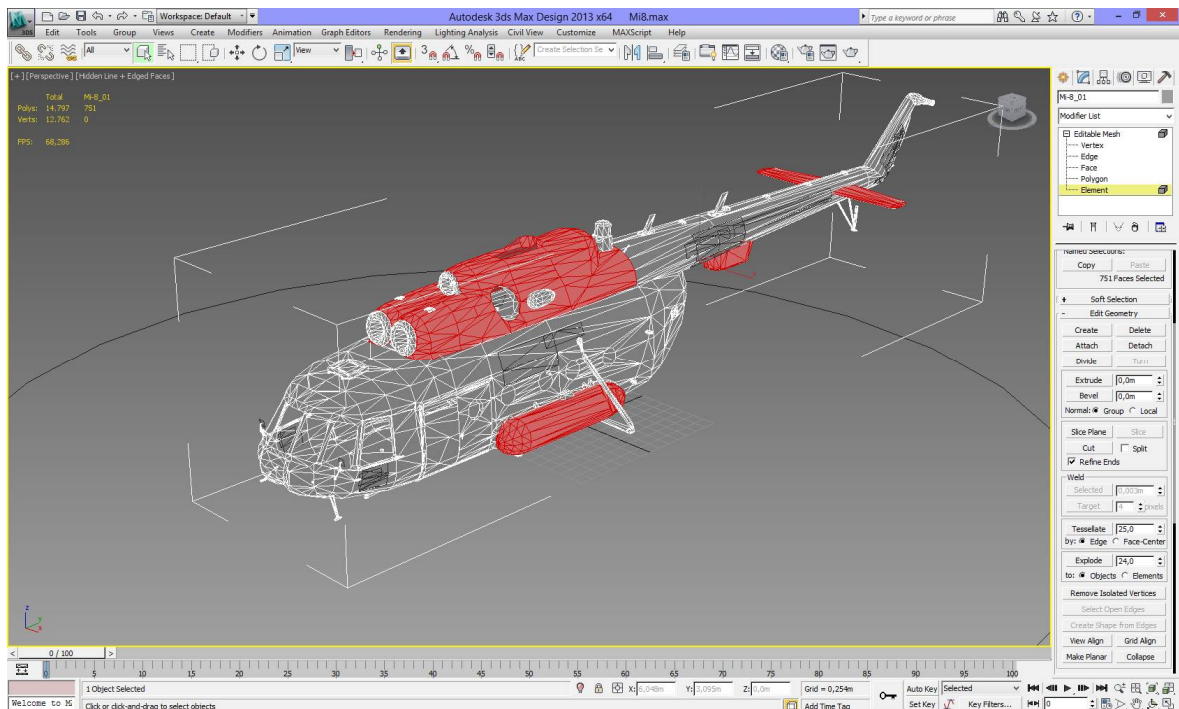


Slika 21 – Jedan od grupisanih objekata

3ds Max aplikacija nudi veliki broj alata za manipulaciju i kreiranje objekata. Svaki objekat se može definisati kao *Mesh* struktura koja se može kasnije prebaciti u *webgl* okruženje i njome dalje manipulirati. Svaki *mesh* objekat poseduje sledeće elemente:

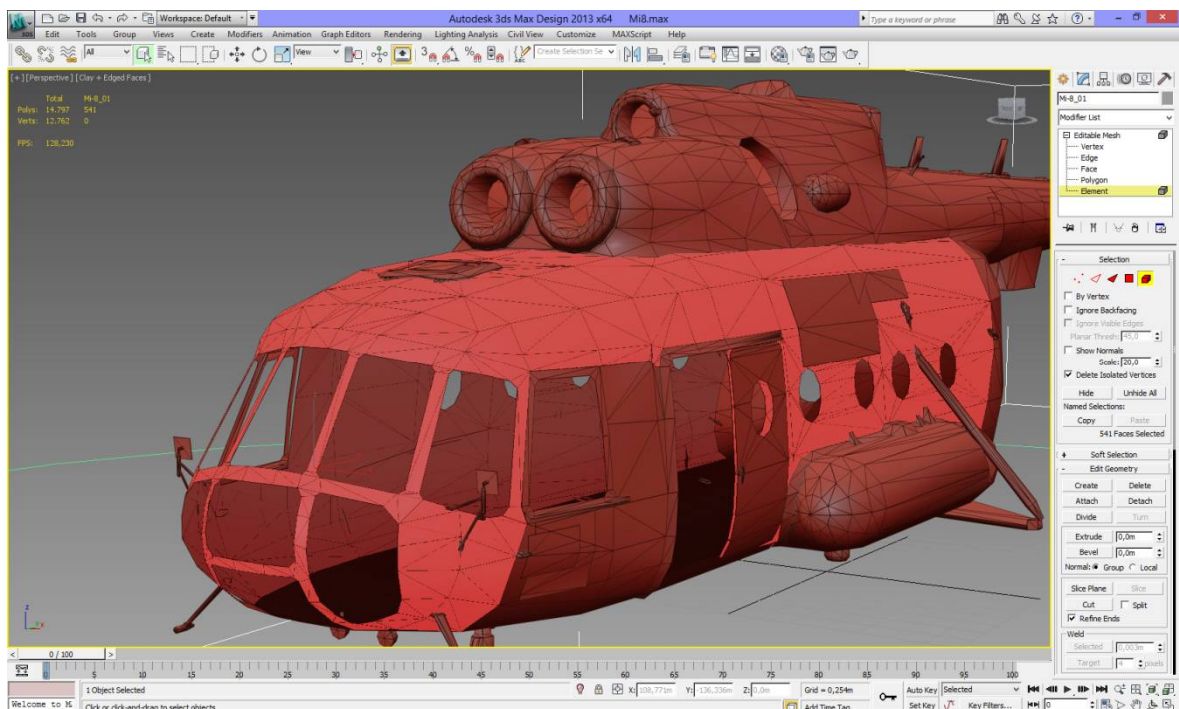
- Vertex
- Edge
- Face
- Polygon
- Element

Najčešće korišćeni elementi su *vertex* koji omogućava rad sa pojedinačnim temenima i *element* koji omogućava selekciju poddelova objekata tj. manipulaciju većeg broja temena odjednom.



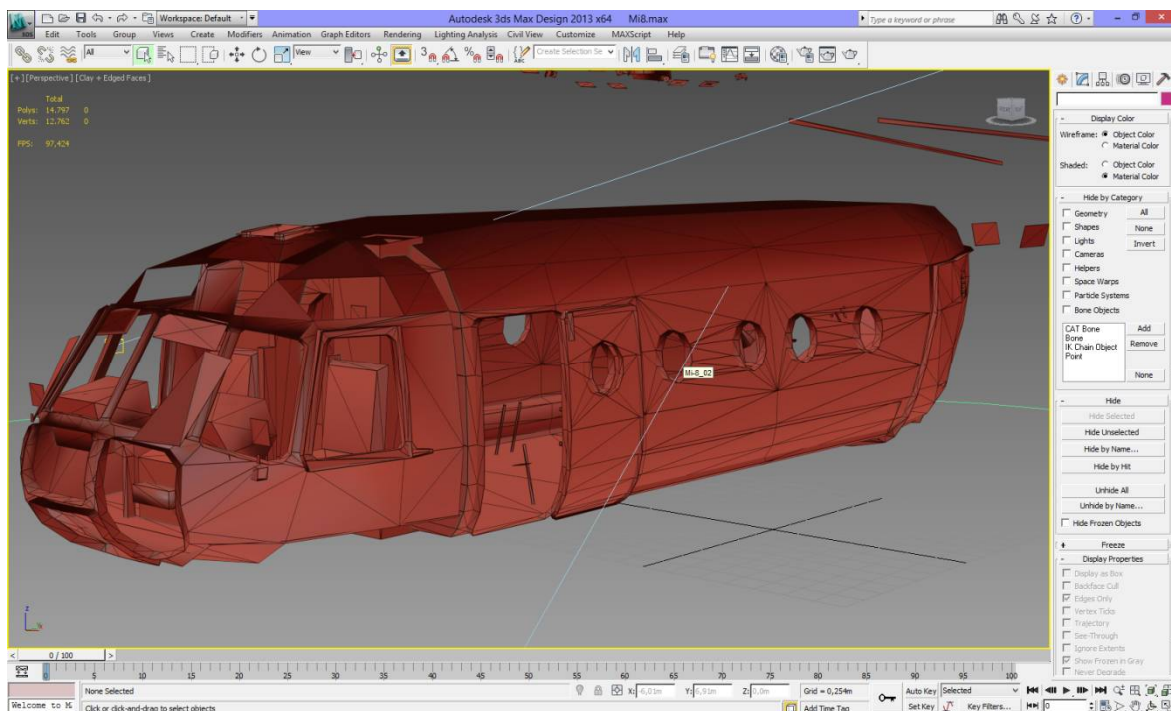
Slika 22 – Rad sa podelementima objekta koji je sveden na editable mesh strukturu

Rad sa podelementima objekta omogućava još jedan bitan aspekt svakog objekta. Kreiranje svakog objekta obuhvata i generisanje pravilnih *normals* definicija svakog poligona kao i generisanje *normals* grupa koje omogućavaju da se texture pravilno iscrtavaju kao i definisanje različitih materijala koji se pridružuju različitim elementima jednog objekta.

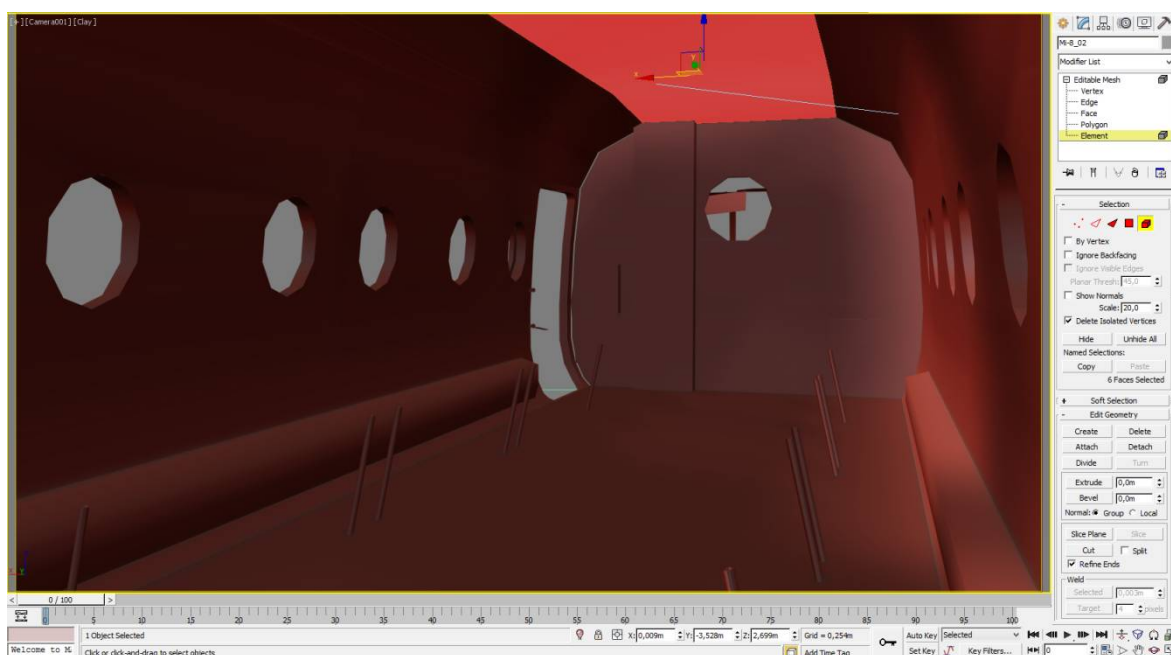


Slika 23 – Dodavanje map id –a odredjenim elementima kako bi se prikazale tačne texture za zadate objekte

Kako je cilj projekta bio da se korisnici podjednako dobro upoznaju i sa unutrašnjošću helikoptera i njegovim delovima, posebno je napravljena unutrašnjost istog. Model omogućava naknadno dodavanje raznih objekata koji čine inventar helikoptera.



Slika 24 – Model unutrašnjosti helikoptera, razmere i dimenzije su verno presene

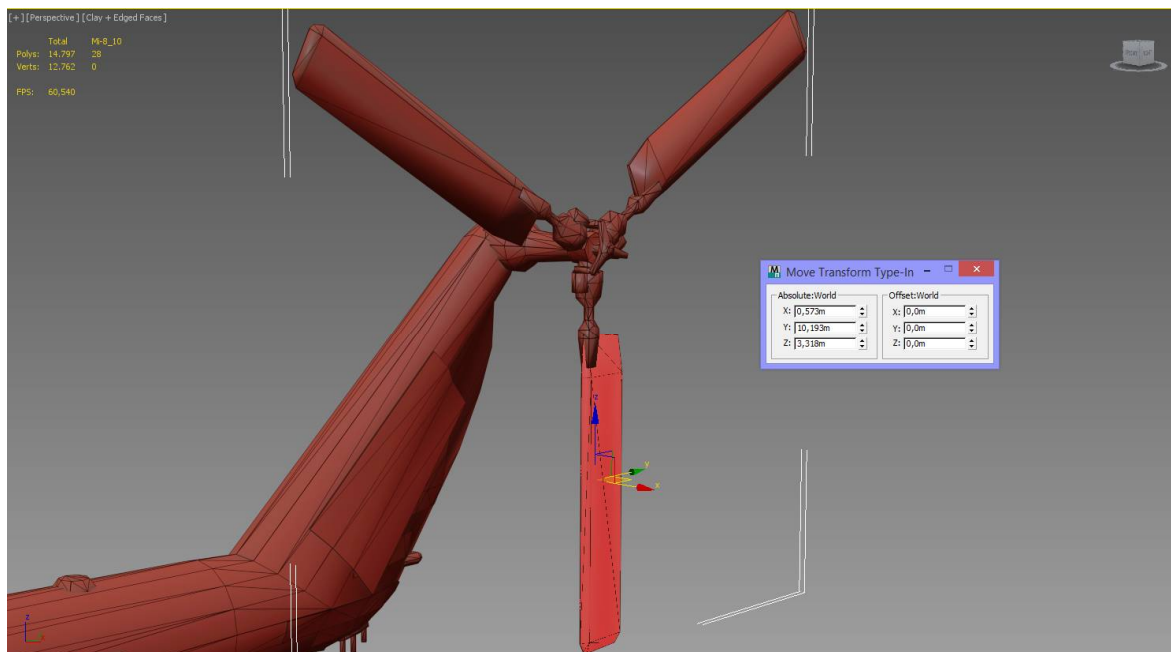


Slika 25 – Prikaz kamere unutar helikoptera

Svaki podelement objekta omogućava tačne procese transformacije tj. pomeranje, rotiranje i skaliranje. Iako konkretan projekat nije zahtevao vrlo visok stepen preciznosti objekti su generisani tako da po dimenzijama verno odlikavaju prave dimenzije helikoptera kako bi korisnici mogli što bolje da se upoznaju sa istim.

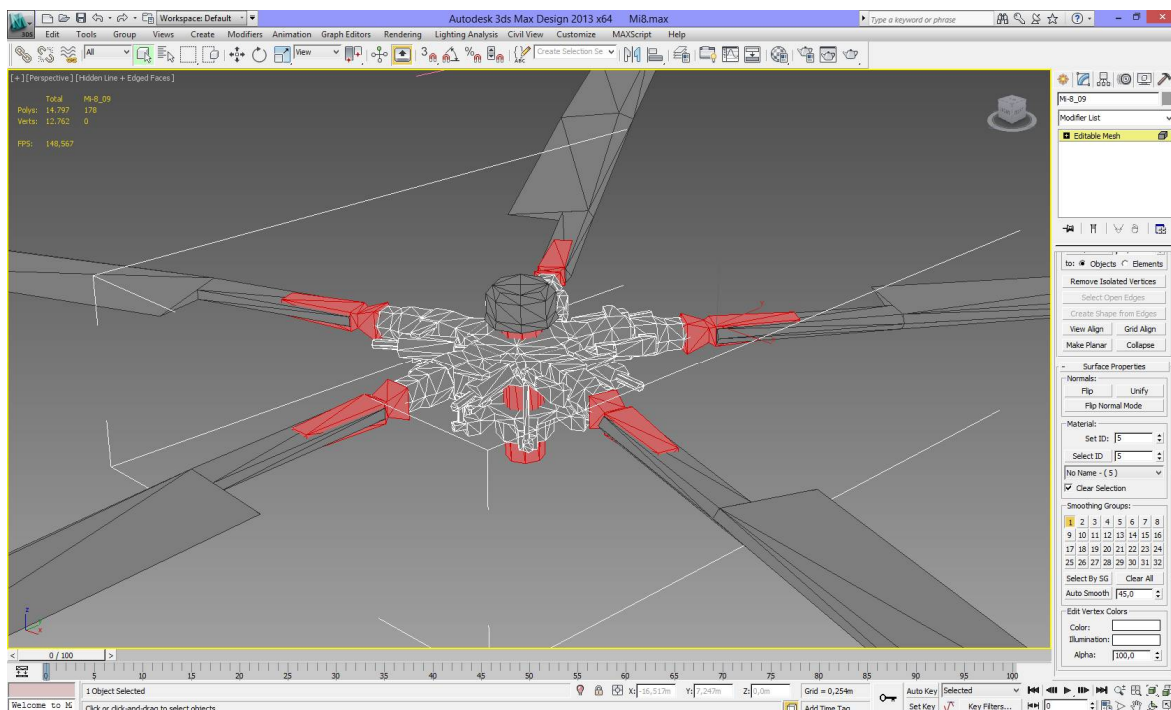
Sam 3ds max nudi mogućnost da se elementi animiraju kao što je recimo kretanje elise, otvaranje i zatvaranje vrata i prozora i slično. Za potrebe projekta razvijen je statični model koji ne poseduje nikakve elemente koji su animirani. Takođe *webgl* kroz programski

kod omogućava određene animacije i transformacije efekte tako da se daljim unapređenjem može povećati realističnost modela i čitave scene.

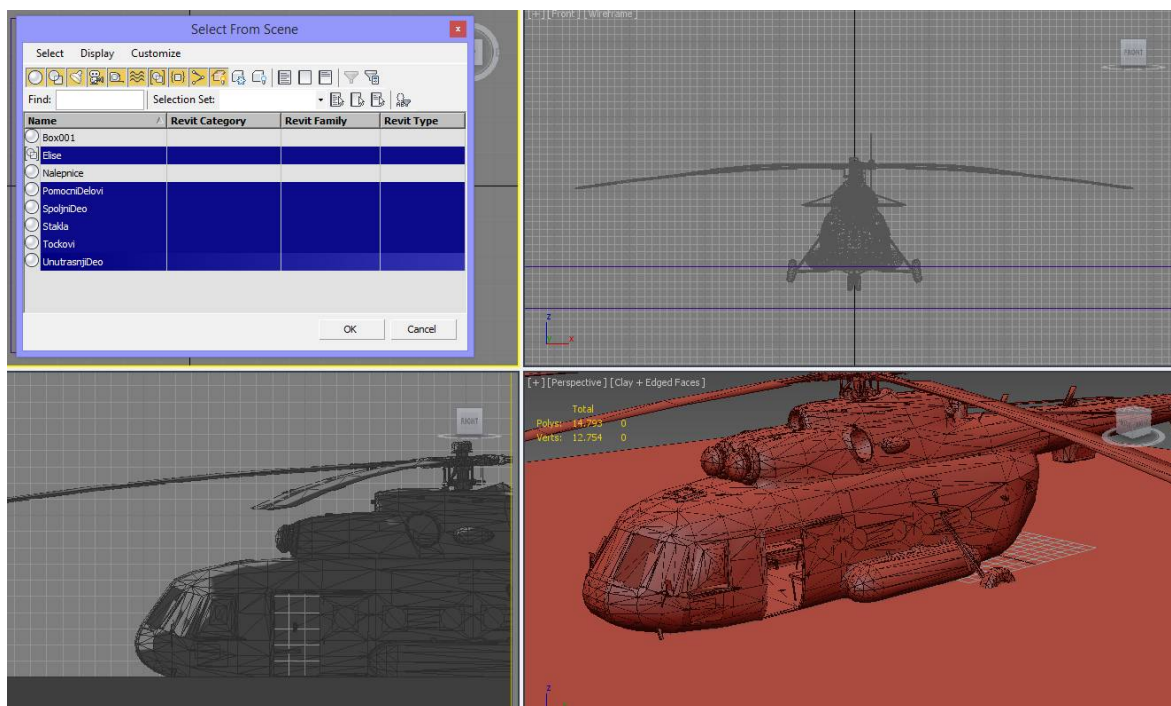


Slika 26 – Primer transformacije određenog podelementa zadnje elise

Grupisanje elementa omogućava daleko lakšu manipulaciju većim brojem podobjekata. Treba naglasiti da webgl okruženje nije u stanju da identifikuje ove podgrupe i da ukoliko postoji potreba za manipulacijom izvesnim elementima u *webgl*-u takvi se elementi prvo moraju u *3ds max*-u izdvojiti kao zasebni objekti i kao takvi posebno uvesti u *webgl*.



Slika 27 – Manipulacija podgrupama elemenata

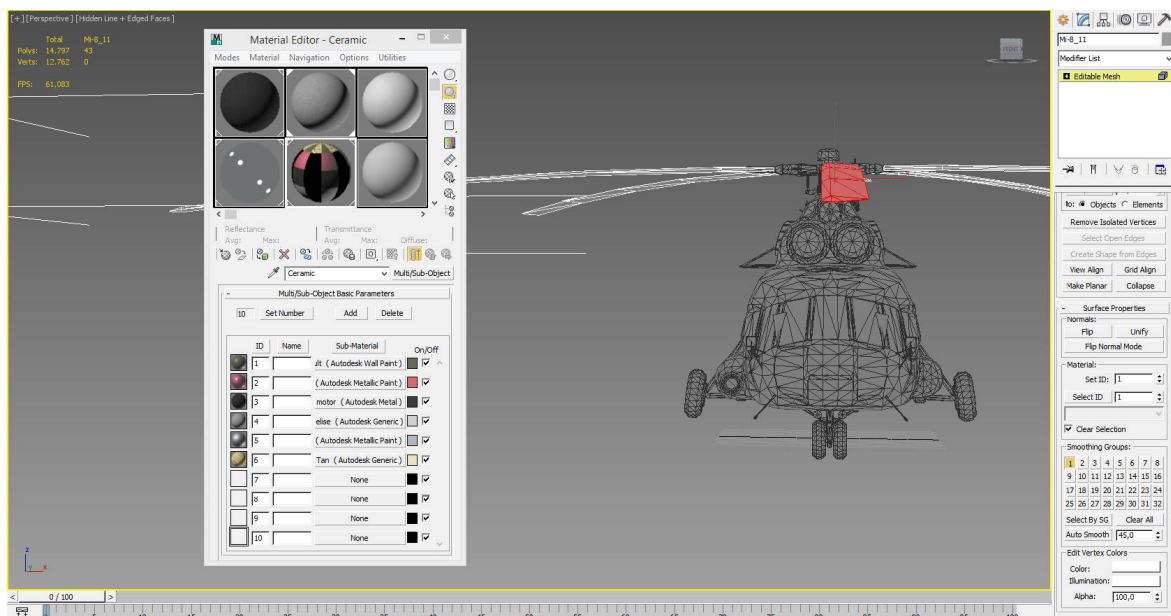


Slika 28 – Grupisanje objekta u 6 glavne grupe. Svaka grupa je ubacena u webgl kao zaseban objekat

Sledeći korak u razvoju modela je dodavanje materijala. Paralelno su generisana dva kompleta materijala, jedan za potrebe vizuelizacije u samom *3ds max*-u i drugi koji je prilagođen eksportovanju u *webgl* okruženje. Materijali namenjeni korišćenju u *3ds max*-u nisu u mogućnosti da se u tom obliku koriste u *webgl*-u što je mana samog *three.js* okruženja. Iz tog razloga napravljeni su nešto jednostavniji materijali koji su lošijeg kvaliteta ali omogućavaju da se pravilno koriste u *webgl*-u.

Sličan pristup u optimizaciji objekta je primenjen i na samom objektu kod koga su izvesni delovi morali da budu odbačeni kao i da se koriste jednostavnije tehnike iscrtavanja ekrana (*rendering*). Razlog ovome su generalno dve stvari, jedna ta da se mora postići dobar odnos kvaliteta i performansi u realnom radu aplikacije koja bi morala da radi i na kompjuterima slabijih performansi. Drugo, samo *webgl* okruženje ne omogućava napredne tehnike koje pružaju moderne aplikacije kao što su *3ds max* pa samim time i nije moguće ostvariti isti kvalitet vizuelnog prikaza kao u takozvanom *native* modu.

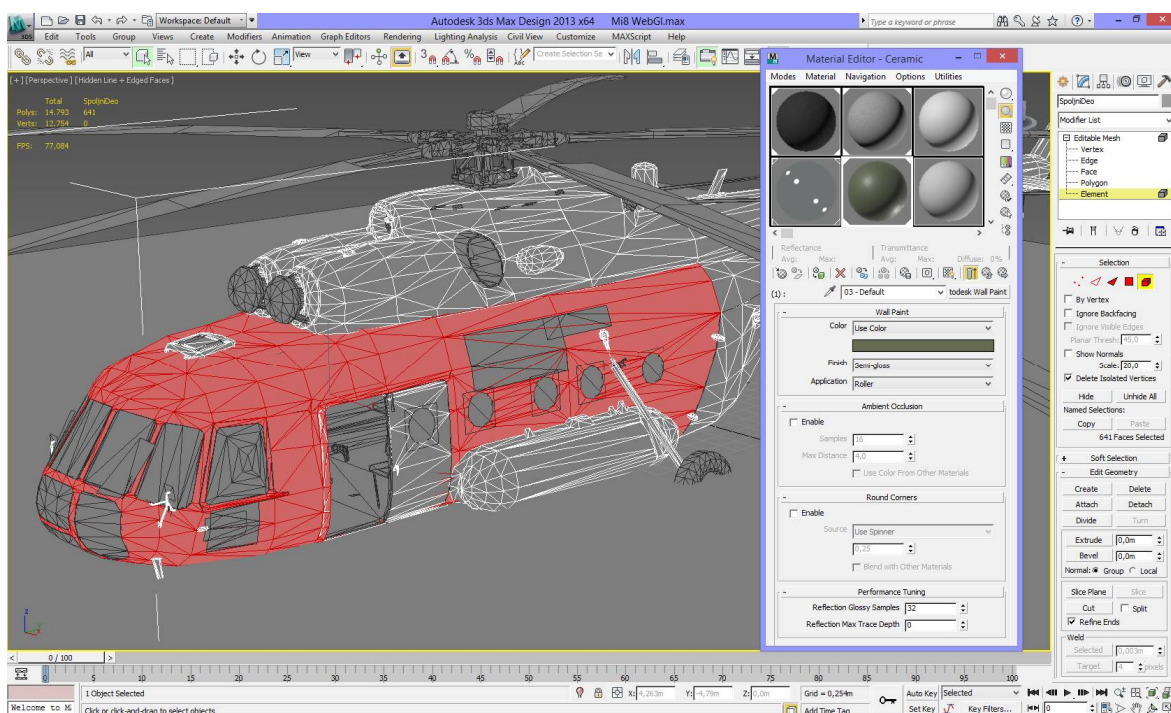
Na sledećim slikama može se videti rad sa *material editorom*, u kome je kreirano 6 primarnih i 5 pomoćnih materijala. Svakom objektu se mora dodeliti *material id*, *smoothing group* i *uvw maping*.



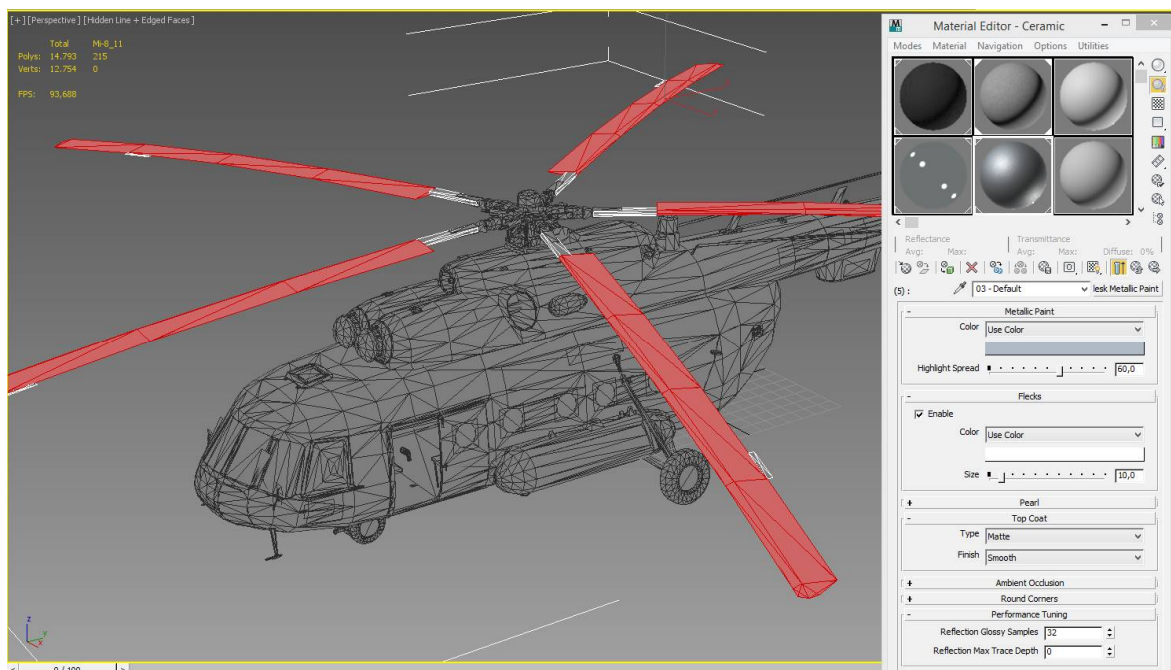
Slika 29 – Dodavanje materijala gotovim objektima

Materijali su generisani na način da verno prikažu izgled helikoptera. Vizuelna šema koja je korišćena podseća na letelice koje se koriste u vazduhoplovstvu Republike Srbije. Određeni elementi su donekle naglašeni kako bi se uneo vizuelno dramatičniji prikaz, kao što je to recimo slučaj sa materijalom prisutnom na staklima.

Dobro grupisani podobjekti i elementi znatno olakšavaju i skraćuju vreme potrebno za realizaciju ovog dela projekta. Iako se praktikuje da se još u toku kreiranja objekata definišu *material id* grupe, to se može u projektima ovog obima uraditi i naknadno ukoliko se vrši dobra nomenklatura podobjekata i elemenata.

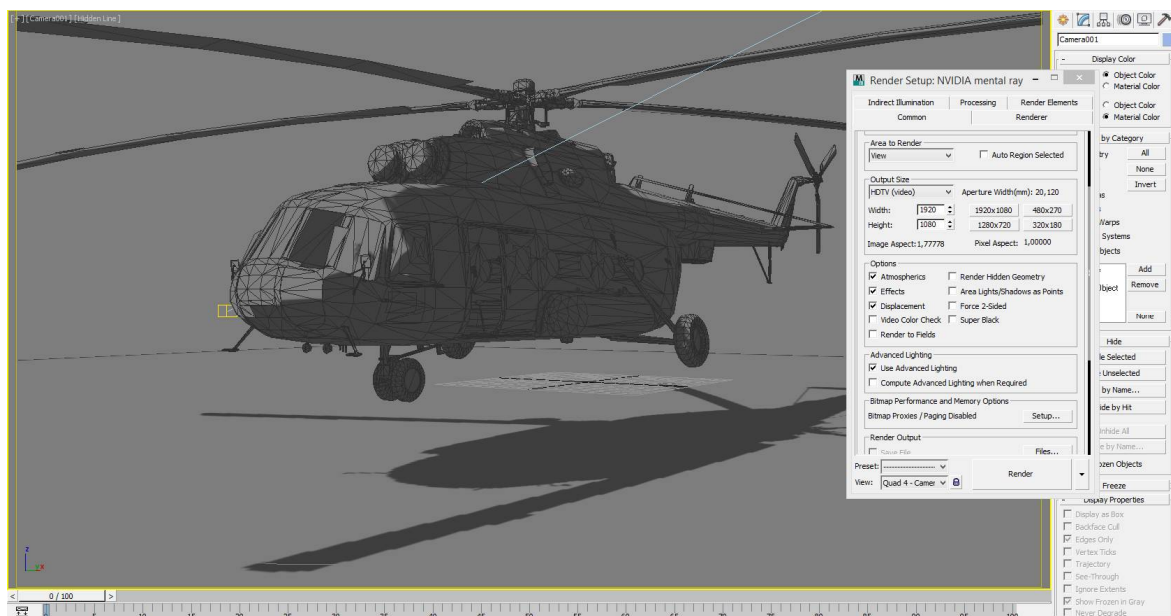


Slika 30 – Dodavanje materijala raznim podgrupama

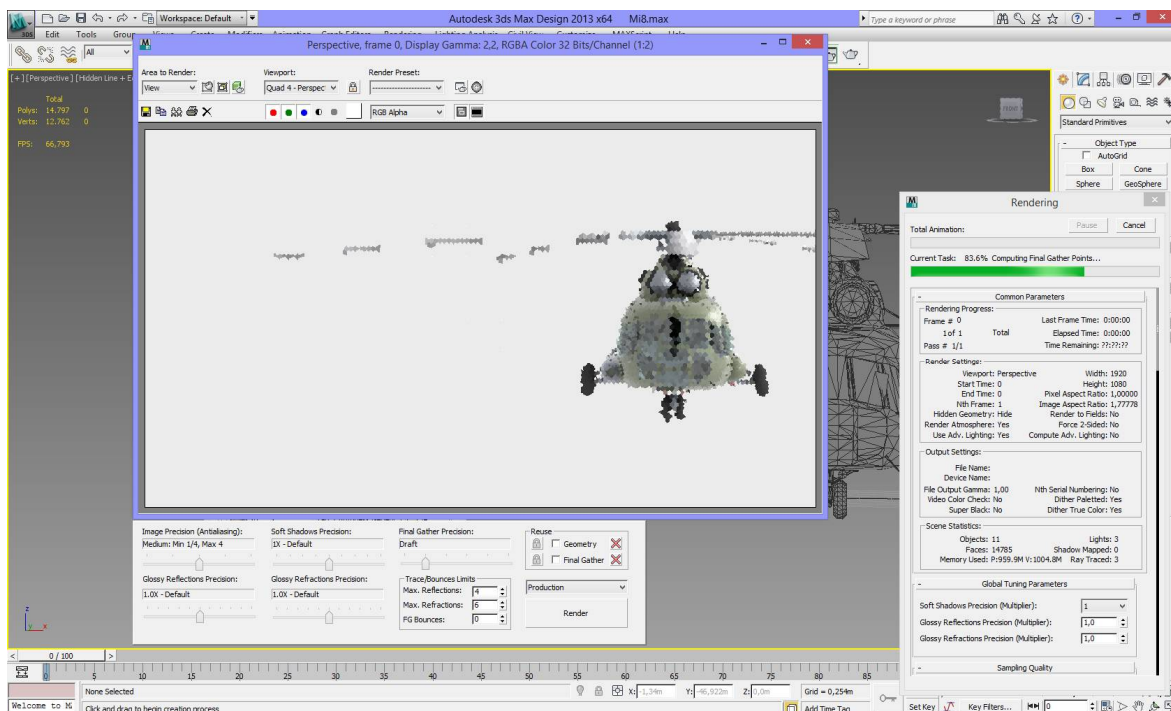


Slika 31 – Primer materijala posebno razvijenog za korišćenje samo u 3ds max-u

Paralelno sa razvojem objekta u 3ds max-u generisana je prosta scena sa nekoliko svetla i kamerom koja je poslužila za *rendering* scene. Na ovaj način mogli smo da pratimo razvoj objekata i imamo uvid u scenu i pre nego da je prebacimo u *webgl* okruženje. To znatno skraćuje vreme razvoja projekta i omogućava da sagledamo maksimum koji se može iz datog modela dobiti.



Slika 32 – Priprema za rendering scene



Slika 33 – Rendering scene može trajati i par minuta u zavisnosti od obima i kompleksnosti iste

Treba naglasiti da rendering scene u 3ds max-u obično traje i po nekoliko minuta dok *webgl* omogućava i do 60 fps-a. To je uslovljeno kvalitetom iscrtavanja koji je u *webgl* okruženju znatno lošiji nego u native modu koji 3ds max pruža.

5.1 Slike modela u 3ds max-u



Slika 34 – Leva strana helikoptera Mi-8



Slika 35 – Desna strana helikoptera Mi-8

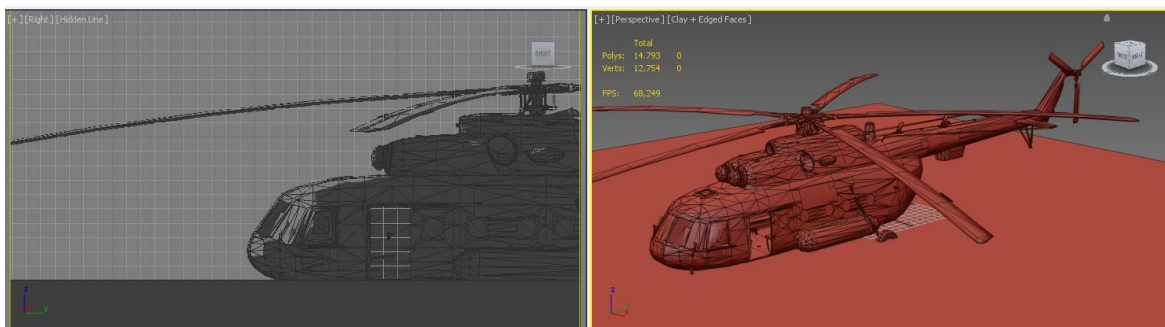


Slika 36 – Mi-8 spolja

5.2 Priprema modela za webgl okruženje

Priprema modela obuhvata sledeće stavke:

- Odstranjivanje određenih elemenata helikoptera
- Kreiranje materijala pogodnih za korišćenje u *webgl*-u
- Pozicioniranje objekata u koordinatnom sistemu kompatibilnom sa *webgl* okruženjem
- Grupisanje i eksportovanje zasebnih podobjekata



Slika 37 – Pozicioniranje objekata

6. WebGL aplikacija

Za potrebe razvoja aplikacije korišćena je three.js biblioteka koja je pružala zadovoljavajuće performanse i mogućnosti razvoja funkcionalnosti koje su nam bile potrebne.

U pitanju je html web strana u kojoj je implementiran *javascript* kod. Okruženje pruža izuzetne mogućnosti za proširenje korišćenjem drugih elemenata i tehnologija koje mogu da se implementiraju u html stranu.



Slika 38 – Iscrtavanje ekrana webgl aplikacije

U head delu html strane podešavamo generalna svojstva web strane što je dato sledećim kodom:

```
<head>
  <meta charset="utf-8">
  <title>three.js - pointerlock controls</title>
  <style>
    html, body {
      width: 100%;
      height: 100%;
    }

    body {
      background-color: 0xffffffff;
      color: 0xffffffff;
      margin: 0px;
      overflow: hidden;
    }
  </style>
</head>
```

```

        font-family: Monospace;
        font-size: 13px;
    }

    #blocker {

        position: absolute;

        width: 100%;
        height: 100%;

        background-color: rgba(0,0,0,0.5);

    }

    #instructions {

        width: 100%;
        height: 100%;

        display: -webkit-box;
        display: -moz-box;
        display: box;

        -webkit-box-orient: horizontal;
        -moz-box-orient: horizontal;
        box-orient: horizontal;

        -webkit-box-pack: center;
        -moz-box-pack: center;
        box-pack: center;

        -webkit-box-align: center;
        -moz-box-align: center;
        box-align: center;

        color: #ffffff;
        text-align: center;

        cursor: pointer;

    }
</style>
</head>

```

Od dodatnih biblioteka koristili smo sledeće:

```

<script src="../../build/three.min.js"></script>
<script src="js/controls/PointerLockControls.js"></script>
<script src="js/loaders/DDSLoader.js"></script>
<script src="js/loaders/MTLLoader.js"></script>
<script src="js/loaders/OBJMTLLoader.js"></script>
<script src="js/loaders/OBJLoader.js"></script>
<script src="js/loaders/ColladaLoader.js"></script>
<script src="js/Detector.js"></script>
<script src="js/libs/stats.min.js"></script>

```

Podešavanja kamere i kontrola iste data je kodom:

```
camera = new THREE.PerspectiveCamera( 45, window.innerWidth / window.innerHeight, 1,
100000 );

    camera.position.y = 30;
    scene = new THREE.Scene();
    scene.fog = new THREE.Fog( 0xE6FAFF , 250, 6000 );

    controls = new THREE.PointerLockControls( camera );
    controls.rotateSpeed = 2.0;
    controls.movementSpeed = 170;
    controls.lookSpeed = 0.15;
    scene.add( controls.getObject() );

    ray = new THREE.Raycaster();
    ray.ray.direction.set( 0, -1, 0 );
```

Posebno je bitno naglasiti postojanje *ray* objekta koji daje mogućnost selekcije objekta na ekranu i tačnog pozicioniranja onoga što korisnik u tom trenutku na ekranu gleda.

U scenu je ubačen veći broj svetla koja imaju za cilj da stvore što realističniji prikaz. Generisanje senki je dodato nekim od njih kako bi se optimizovao proces iscrtavanja ekrana.

```
var ambient = new THREE.AmbientLight( 0x000044 );
    ambient.castShadow = false;
    scene.add( ambient );

    hemiLight = new THREE.HemisphereLight( 0x000044, 0x000044, 0.1 );
    hemiLight.color.setHSL( 0.6, 1, 0.6 );
    hemiLight.groundColor.setHSL( 0.095, 1, 0.75 );
    hemiLight.position.set( 0, -500, 0 );
    scene.add( hemiLight );

    dirLight = new THREE.DirectionalLight( 0xffffffff, 1 );
    dirLight.color.setHSL( 0.1, 1, 0.95 );
    dirLight.position.set( 0, 100, 150 );
    dirLight.position.multiplyScalar( 50 );

    dirLight.shadowCameraNear = 1000;
    dirLight.shadowCameraFar = 1500;
    dirLight.shadowCameraFov = 40;

    dirLight.shadowBias = 0;
    dirLight.shadowDarkness = 0.15;
    dirLight.shadowCameraVisible = true;
    dirLight.castShadow = true;
    dirLight.shadowMapWidth = 2048;
    dirLight.shadowMapHeight = 1024;

    scene.add( dirLight );

    light = new THREE.SpotLight( 0xffffffff, 1, 0, Math.PI / 2, 1 );
    light.position.set( 0, 150, 100 );
```

```

light.target.position.set( 0, 0, -600 );

light.castShadow = true;

light.shadowCameraNear = 1200;
light.shadowCameraFar = 2500;
light.shadowCameraFov = 50;

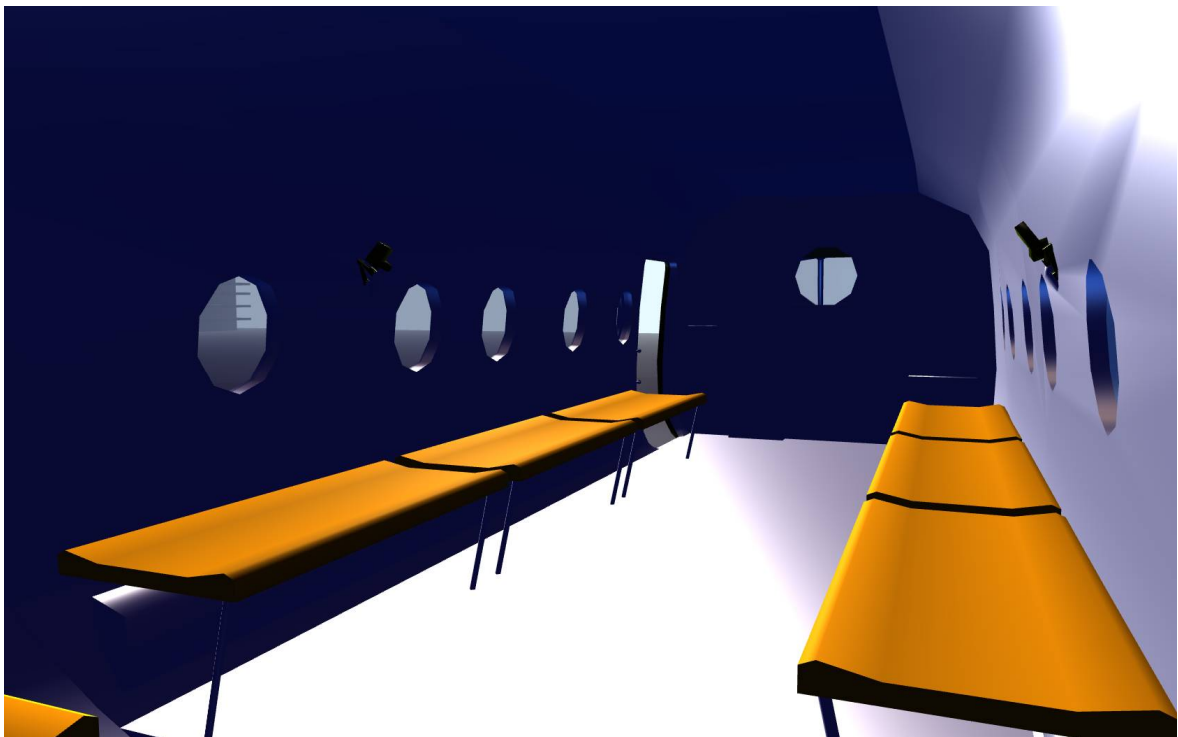
//light.shadowCameraVisible = true;

light.shadowBias = 0.0001;
light.shadowDarkness = 0.15;

light.shadowMapWidth = 2048;
light.shadowMapHeight = 2048;

scene.add( light );

```



Slika 39 – Unutrašnjost helikoptera uz korišćenje pojednostavljenih senki

Najvažniji deo je svakako uvoz gotovih objekata koje smo razvili u 3d-u.

```

var loaderB = new THREE.OBJMTLLoader();
    loaderB.load( 'obj/mi8/mi8spoljasnost.obj',
'obj/mi8/mi8spoljasnost.mtl', function ( objectB1 ) {

        objectB1.position.x = 0;
        objectB1.position.z = -600;
        objectB1.rotation.y = -500;
        objectB1.scale.set(1,1,1);

```

```

        objectB1.castShadow = true;
        objectB1.receiveShadow = true;
        scene.add( objectB1 );

    });

    loaderB.load( 'obj/mi8/unutrasnjost.obj', 'obj/mi8/unutrasnjost.mtl', function (
objectB2 ) {

        objectB2.position.x = 0;
        objectB2.position.z = -600;
        objectB2.rotation.y = -500;
        objectB2.scale.set(1,1,1);

        objectB2.castShadow = true;
        objectB2.receiveShadow = true;
        scene.add( objectB2 );

    });

    loaderB.load( 'obj/mi8/tockovi.obj', 'obj/mi8/tockovi.mtl', function ( objectB3 ) {

        objectB3.position.x = 0;
        objectB3.position.z = -600;
        objectB3.rotation.y = -500;
        objectB3.scale.set(1,1,1);

        objectB3.castShadow = true;
        objectB3.receiveShadow = true;
        scene.add( objectB3 );

    });

    loaderB.load( 'obj/mi8/elise.obj', 'obj/mi8/elise.mtl', function ( objectB4 ) {

        objectB4.position.x = 0;
        objectB4.position.z = -600;
        objectB4.rotation.y = -500;
        objectB4.scale.set(1,1,1);

        objectB4.castShadow = true;
        objectB4.receiveShadow = true;
        scene.add( objectB4 );

    });

    loaderB.load( 'obj/mi8/stakla.obj', 'obj/mi8/stakla.mtl', function ( objectB5 ) {

        objectB5.position.x = 0;

```

```

        objectB5.position.z = -600;
objectB5.rotation.y = -500;
objectB5.scale.set(1,1,1);

objectB5.castShadow = true;
objectB5.receiveShadow = true;
scene.add( objectB5 );

    });

    loaderB.load( 'obj/mi8/nebo.obj', 'obj/mi8/nebo.mtl', function ( sky ) {

        sky.position.x = 0;
        sky.position.z = -600;
sky.rotation.y = -500;
sky.scale.set(1,1,1);

sky.castShadow = false;
sky.receiveShadow = false;
scene.add( sky );

    });

loaderB.load( 'obj/mi8/pod.obj', 'obj/mi8/pod.mtl', function ( pod ) {

        pod.position.x = 0;
        pod.position.z = -600;
pod.rotation.y = -500;
pod.scale.set(1,1,1);

pod.castShadow = true;
pod.receiveShadow = true;
scene.add( pod );

    });

```

Razna podešavanja vezana za iscrtavanje ekrana su data sledećim kodom:

```

renderer = new THREE.WebGLRenderer({ antialias: true });
    renderer.setClearColor( scene.fog.color );
    renderer.setSize( window.innerWidth, window.innerHeight );

    document.body.appendChild( renderer.domElement );
renderer.shadowMapEnabled = true;
//

window.addEventListener( 'resize', onWindowResize, false );

```

Razne funkcije koje se koriste u aplikaciji:

```
function onWindowResize() {  
  
    camera.aspect = window.innerWidth / window.innerHeight;  
    camera.updateProjectionMatrix();  
  
    renderer.setSize( window.innerWidth, window.innerHeight );  
  
}  
  
function animate() {  
  
    requestAnimationFrame( animate );  
  
    controls.isOnObject( false );  
  
    ray.ray.origin.copy( controls.getObject().position );  
    ray.ray.origin.y -= 10;  
  
    var intersections = ray.intersectObjects( objects );  
  
    if ( intersections.length > 0 ) {  
  
        var distance = intersections[ 0 ].distance;  
  
        if ( distance > 0 && distance < 10 ) {  
  
            controls.isOnObject( true );  
  
        }  
  
    }  
  
    controls.update();  
  
    renderer.render( scene, camera );  
  
}  
  
function addObject( geometry, material, x, y, z, ry ) {  
  
    var tmpMesh = new THREE.Mesh( geometry, material );  
  
    tmpMesh.material.color.offsetHSL( 0.1, -0.1, 0 );  
  
    tmpMesh.position.set( x, y, z );  
  
    tmpMesh.rotation.y = ry;  
  
    tmpMesh.castShadow = true;  
    tmpMesh.receiveShadow = true;  
  
    scene.add( tmpMesh );  
  
    return tmpMesh;  
  
}
```



Slika 40 – Mi-8 prikaz u webgl okruženju

Funkcionalnost selekcije izdvojenih objekata na sceni data je sledećim kodom:

```
function onDocumentMouseDown(
event ) {

    event.preventDefault();

    mouse.x = ( event.clientX / renderer.domElement.width ) * 2 - 1;
    mouse.y = - ( event.clientY / renderer.domElement.height ) * 2 + 1;
    raycaster.setFromCamera( mouse, camera );
    var intersects = raycaster.intersectObjects( objects );
    if ( intersects.length > 0 ) {
        intersects[ 0 ].object.material.color.setHex( Math.random() * 0xffffffff
        );
        var particle = new THREE.Sprite( particleMaterial );
        particle.position.copy( intersects[ 0 ].point );
        particle.scale.x = particle.scale.y = 16;
        scene.add( particle );
    }
    /*
    // Parse all the faces
    for ( var i in intersects ) {
        intersects[ i ].face.material[ 0 ].color.setHex( Math.random() *
        0xffffffff | 0x80000000 );
    }
    */
}
```

```
*/
}
```

Funkcionalnost koja onemogućava korisniku da „prođe“ kroz objekat je dala dozu realističnosti koje je bila potrebna ovom tipu projekta. Ovom prilikom do izražaja je došlo i korišćenje *ray* objekta.

```
if ( controlsEnabled )
{
    raycaster.ray.origin.copy( controls.getObject().position );
    raycaster.ray.origin.y -= 10;
    var intersections = raycaster.intersectObjects( objects );
    var isOnObject = intersections.length > 0;
    var time = performance.now();
    var delta = ( time - prevTime ) / 1000;
    velocity.x -= velocity.x * 10.0 * delta;
    velocity.z -= velocity.z * 10.0 * delta;
    velocity.y -= 9.8 * 100.0 * delta; // 100.0 = mass
    if ( moveForward ) velocity.z -= 400.0 * delta;
    if ( moveBackward ) velocity.z += 400.0 * delta;
    if ( moveLeft ) velocity.x -= 400.0 * delta;
    if ( moveRight ) velocity.x += 400.0 * delta;
    if ( isOnObject === true ) {
        velocity.y = Math.max( 0, velocity.y );
        canJump = true;
    }
    controls.getObject().translateX( velocity.x * delta );
    controls.getObject().translateY( velocity.y * delta );
    controls.getObject().translateZ( velocity.z * delta );
    if ( controls.getObject().position.y < 10 ) {
        velocity.y = 0;
        controls.getObject().position.y = 10;
        canJump = true;
    }
    prevTime = time;
}
```

U nastavku je dat alternativni kod aplikacije sa izmenjenim načinom korišćenja kamere kao i promenjenom fizikom kretanja. U ovoj verziji aplikacije ostavljeno je korišćenje miša za potrebe selekcije objekata u prostoru, dok se kretanje vrši korišćenjem samo tastature.

```
var motion = {
  airborne : false,
  position : new THREE.Vector3(), velocity : new THREE.Vector3(),
  rotation : new THREE.Vector2(), spinning : new THREE.Vector2()
};

motion.position.y = -150;

// game systems code

var resetPlayer = function() {
  if( motion.position.y < -123 ) {
    motion.position.set( -2, 7.7, 25 );
    motion.velocity.multiplyScalar( 1000 );
  }
};

var keyboardControls = (function() {

  var keys = { SP : 32, W : 87, A : 65, S : 83, D : 68, UP : 38, LT : 37, DN : 40, RT : 39 };

  var keysPressed = {};

  (function( watchedKeyCodes ) {
    var handler = function( down ) {
      return function( e ) {
        var index = watchedKeyCodes.indexOf( e.keyCode );
        if( index >= 0 ) {
          keysPressed[watchedKeyCodes[index]] = down; e.preventDefault();
        }
      };
    }
    window.addEventListener( "keydown", handler( true ), false );
    window.addEventListener( "keyup", handler( false ), false );
  })(
    keys.SP, keys.W, keys.A, keys.S, keys.D, keys.UP, keys.LT, keys.DN, keys.RT
  );

  var forward = new THREE.Vector3();
  var sideways = new THREE.Vector3();

  return function() {
    if( !motion.airborne ) {

      // look around
      var sx = keysPressed[keys.UP] ? 0.03 : ( keysPressed[keys.DN] ? -0.03 : 0 );
      var sy = keysPressed[keys.LT] ? 0.03 : ( keysPressed[keys.RT] ? -0.03 : 0 );

      if( Math.abs( sx ) >= Math.abs( motion.spinning.x ) ) motion.spinning.x = sx;
      if( Math.abs( sy ) >= Math.abs( motion.spinning.y ) ) motion.spinning.y = sy;
```

```

// move around
forward.set( Math.sin( motion.rotation.y ), 0, Math.cos( motion.rotation.y ) );
sideways.set( forward.z, 0, -forward.x );

forward.multiplyScalar( keysPressed[keys.W] ? -5.1 : (keysPressed[keys.S] ? 5.1 : 0));
sideways.multiplyScalar( keysPressed[keys.A] ? -5.1 : (keysPressed[keys.D] ? 5.1 : 0));

var combined = forward.add( sideways );
if( Math.abs( combined.x ) >= Math.abs( motion.velocity.x ) ) motion.velocity.x = combined.x;
if( Math.abs( combined.y ) >= Math.abs( motion.velocity.y ) ) motion.velocity.y = combined.y;
if( Math.abs( combined.z ) >= Math.abs( motion.velocity.z ) ) motion.velocity.z = combined.z;
}
};
})();

var applyPhysics = (function() {
var timeStep = 5;
var timeLeft = timeStep + 1;

var birdsEye = 100;
var kneeDeep = 0.4;

var raycaster = new THREE.Raycaster();
raycaster.ray.direction.set( 0, -1, 0 );

var angles = new THREE.Vector2();
var displacement = new THREE.Vector3();

return function( dt ) {
var platform = scene.getObjectByName( "platform", true );
if( platform ) {

timeLeft += dt;

// run several fixed-step iterations to approximate varying-step

dt = 5;
while( timeLeft >= dt ) {

var time = 0.3, damping = 0.93, gravity = 0.01, tau = 2 * Math.PI;

raycaster.ray.origin.copy( motion.position );
raycaster.ray.origin.y += birdsEye;

var hits = raycaster.intersectObject( platform );

motion.airborne = false;

// are we above, or at most knee deep in, the platform?

if( ( hits.length > 0 ) && ( hits[0].face.normal.y > 0 ) ) {
var actualHeight = hits[0].distance - birdsEye;

// collision: stick to the surface if landing on it

if( ( motion.velocity.y <= 0 ) && ( Math.abs( actualHeight ) < kneeDeep ) ) {

```

```

motion.position.y -= actualHeight;
motion.velocity.y = 0;
motion.airborne = false;
}
}

if( motion.airborne ) motion.velocity.y -= gravity;

angles.copy( motion.spinning ).multiplyScalar( time );
if( !motion.airborne ) motion.spinning.multiplyScalar( damping );

displacement.copy( motion.velocity ).multiplyScalar( time );
if( !motion.airborne ) motion.velocity.multiplyScalar( damping );

motion.rotation.add( angles );
motion.position.add( displacement );

// limit the tilt at ±0.4 radians

motion.rotation.x = Math.max( -0.4, Math.min ( +0.4, motion.rotation.x ) );

// wrap horizontal rotation to 0...2π

motion.rotation.y += tau; motion.rotation.y %= tau;

timeLeft -= dt;
}
}
};
})();

var updateCamera = (function() {
var euler = new THREE.Euler( 0, 0, 0, 'YXZ' );

return function() {
euler.x = motion.rotation.x;
euler.y = motion.rotation.y;
camera.quaternion.setFromEuler( euler );

camera.position.copy( motion.position );

camera.position.y += 30.0;
};
})();

var start = function( gameLoop, gameViewportSize ) {
var resize = function() {
var viewport = gameViewportSize();
renderer.setSize( viewport.width, viewport.height );
camera.aspect = viewport.width / viewport.height;
camera.updateProjectionMatrix();
};

window.addEventListener( 'resize', resize, false );
resize();

var lastTimeStamp;

```

```

var render = function( timeStamp ) {
var timeElapsed = timeStamp ? timeStamp - lastTimeStamp : 0; lastTimeStamp = timeStamp;

// call our game loop with the time elapsed since last rendering, in ms
gameLoop( timeElapsed );

renderer.render( scene, camera );
requestAnimationFrame( render );
};

requestAnimationFrame( render );
};

var gameLoop = function( dt ) {
resetPlayer();
keyboardControls();
/*jumpPads();*/
applyPhysics( dt );
updateCamera();
};

var gameViewportSize = function() { return {
width: window.innerWidth, height: window.innerHeight
}};

```

6.1 Slike



Slika 26 – Razne perspektive

7. Zaključak

Svrha ovog rada je da pre svega da se da uvid u jednu jako kompleksnu i nadasve dinamičnu oblast. Mogućnosti koje ove tehnologije pružaju još uvek nisu do kraja istražene. Postavlja se pitanje na koji najbolji način izvršiti implementaciju i kako da korisnici osete realnu potrebu za ovakvim okruženjem. Razne platforme će svakako pokazati na koji način korisnici očekuju da ova rešenja budu primenjena pa će dalji napredak ovih tehnologija i ići u tom smeru.

Sfera uticaja koju je nekada činilo dominantno tržište personalnih računara danas polako prelazi ka mobilnim platformama. Veliki broj korisnika je prihvatio tu potrebu za lakoćom i nesmetanim pristupom kao nešto potpuno prirodno. Samim time sadržaj, podaci i informacije postaju fokus onog čemu sve ove tehnologije treba da služe. Bez obzira na krajnju svrhu aplikacija korisnici su ti koji će pokazati na koji način i kroz kakvo okruženje oni žele da taj sadržaj bude impretiran. 3D okruženje nudi neka nova i originalna rešenja. Veliki potencijal leži u činjenici da ove tehnologije postaju sve više dostupne.

Za razvoj aplikacija ovog tipa potreban je svakako dobar set razvojnih alata koji bi omogućili lakši rad. *Webgl* poseduje samo ograničen skup istih, pa samim time nije predviđen za projekte većih razmera. 3ds max je u velikoj meri olakšao razvoj i kreiranje 3d modela.

Performanse aplikacije su u direktnoj vezi sa 3d modelima koji su generisani kao i kompleksnošću same scene koja je definisana u *webgl*-u. Projekti sa većim brojem objekata i njihovom interakcijom ili kvalitetnijim iscrtavanjem ekrana uslovili bi pad performansi koje su direktna mana samog *webgl*-a koje nije lako rešiti. Ograničenja koja *webgl* nameće treba shvatiti ako ne ozbiljno onda bar kao smernicu u kom pravcu vršiti optimizaciju projekta.

Ovaj rad daje uvid u mogućnosti koje *webgl* okruženje pruža. Sa pravilnom implementacijom 3d modela i razumevanjem na koji način *webgl* radi vrlo je lako doći do rezultata koji daju kvalitetan vizuelni kako i funkcionalni element. Proširivost web okruženja na koje se ovaj projekat oslanja omogućava da se naknadno dodaju funkcionalnosti koje bi korisnici mogli da zahtevaju.

Korisnici na probleme sa kojima se ove tehnologije susreću gledaju jako intuitivno. Ima li realne potrebe za 3D okruženjem u radu sa aplikacijama koje nisu tome nikad ni težile? Ono što je svakako sigurno je da će ove tehnologije pratiti veliki broj pitanja na koje malo ko može da da tačan odgovor. Kako su za zadnjih dvadeset godina mnoga rešenja pokazala da je lako napraviti grešku u predviđanjima jedino što se može reći jeste da je budućnost na ovom planu još neizvesnija.

8. Literatura

1. Arnold Trehub, Where Am I? Redux, preuzeto sa <http://people.umass.edu/trehub/where-am-i-redux.pdf>, [Dostupno dana 19.03.2015.]
2. Tomas Akenine-Moller, Bjorn Johnsson, Performance per what?, preuzeto sa <http://jcgt.org/published/0001/01/03/paper.pdf>, [Dostupno dana 19.03.2015.]
3. Tony Parisi, Programming 3D applications with html5 and *WebGL*, preuzeto sa <http://chimera.labs.oreilly.com/books/1234000000802/index.html>, [Dostupno dana 19.03.2015.]
4. J. Behr, Y. Jung, J. Keil, T. Drevensek, M. Zoellner, P. Eschler, D. Fellner A, (2010) Scalable Architecture for the HTML5/ X3D Integration Model X3DOM, Fraunhofer IGD / TU Darmstadt
5. Xiaoyu Zhang, Denis Gračanin, (2013) An Approach to *WebGL* based Distributed Virtual Environments, Virginia Tech
6. Hafez Rouzati, Luis Cruiz, Blair Macintyre, (2013) Unified *WebGL* / CSS Scene-Graph and Application to AR, Georgia Institute of Technology
7. Popović, R., Cvetković, D., Marković, D. (2011) Multimedija, Univerzitet Singidunum, Beograd