SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 3799

Animacija pokretana praćenjem lica kao web aplikacija

Neven Pičuljan

Zagreb, lipanj 2014.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA ZAVRŠNI RAD MODULA

Zagreb, 14. ožujka 2014.

ZAVRŠNI ZADATAK br. 3799

Pristupnik: **Neven Pičuljan (0036464897)**
Studij: Računarstvo
Modul: Računarska znanost

Zadatak: **Animacija pokretana praćenjem lica kao web aplikacija**

Opis zadatka:

Praćenje lica je postupak kojim se tehnikama računalnog vida slijedi ljudsko lice i njegove ključne točke u video slici. Ova tehnologija može se koristiti za pokretanje animacije lica virtualnog lika, što je poznato i pod engleskim nazivom performance animation. Potencijalne primjene su u području igara, zabave i komunikacija a posebno su zanimljive primjene u web aplikacijama.
Na Zavodu za telekomunikacije dostupan je sustav za praćenje lica u stvarnom vremenu. Potrebno je odabrati odgovarajući grafički sustav (preporučen je Three.js) te ga na odgovarajući način povezati sa sustavom praćenja lica.
Vaša je zadaća proučiti postojeće sustave, te predložiti i implementirati izvedbu web aplikacije za animaciju pokretanu praćenjem lica.
Svu potrebnu literaturu i uvjete za rad osigurat će Vam Zavod za telekomunikacije.

Zadatak uručen pristupniku: 14. ožujka 2014.
Rok za predaju rada: 13. lipnja 2014.

Mentor:

Prof.dr.sc. Igor Sunday Pandžić

Djelovođa:

Doc.dr.sc. Tomislav Hrkać

Predsjednik odbora za završni rad modula:

Prof.dr.sc. Siniša Srbljić

SADRŽAJ

1. Uvod

Praćenje lica je postupak kojim se tehnikama računalnog vida slijedi ljudsko lice i njegove ključne točke u video slici. Ta je tehnologija u okviru ovog rada korištena za pokretanje animacije lica virtualnog lika. Animirano je zatvaranje i otvaranje očiju, micanje usana, nosa i slično prema tome kako se miču isti ti dijelovi ljudskog lica koje se snima običnom web kamerom. Osim toga, praćena je i translacija te rotacija ljudske glave pa se ovisno o tome kako se translatira i rotira ljudska glava koja se prati, u realnom vremenu translatira i rotira glava virtualnog lika.

Tehnologija je implementirana tako da se koristi kao web aplikacija kojoj se pristupa pomoću web preglednika, a na način da kao takvoj nisu potrebni nikakvi programski dodaci (engl. *plugin*) što je dodatna prednost jer je danas čest slučaj da se moraju instalirati i koristiti dodaci za preglednik ako se želi pokretati 3D ili 2D animacija na webu.

Iako ideja o 3D sadržajima u web stranicama postoji još od devedesetih godina prošlog stoljeća, do sada niti jedan pokušaj uspostavljanja široko prihvaćene norme za prikaz takvih sadržaja nije uspio, te i dalje ne postoji "standardni" način uključivanja interaktivnih 3D sadržaja u web aplikacije. Tijekom godina pojavile su se međunarodne norme kao VRML, X3D i MPEG-4 koje su, između ostalog, bile namijenjene rješavanju ovog problema; čitav niz pojedinačnih rješenja nuđenih od strane raznih tvrtki, uglavnom zasnovanih na nekoj vrsti plugina za web preglednik; rješenja zasnovana na lažnom 3D prikazu uz korištenje većeg broja slika čijim izmjenjivanjem bi se dobila iluzija 3D grafike; rješenja zasnovana na Java appletima itd. Trenutno možda najizglednija tehnologija za rješavanje problema 3D sadržaja na webu je WebGL, koji je već uključen u sve vodeće web preglednike osim Internet Explorera. Također, vrlo zanimljiva tehnologija je Flash, koji od svoje inačice 10 uključuje mogućnost 3D sadržaja. Microsoftova tehnologija Silverlight također uključuje 3D, no za sada s dosta ograničenim mogućnostima što se tiče interaktivnog upravljanja 3D sadržajem. [1]

Ključni pojmovi, tehnike i alati koji su bili nužni pri realizaciji rada bit će objašnjeni u svakom od idućih poglavlja. Objasnit će se grafički sustav koji je korišten za

izradu web aplikacije te način na koji je on povezan sa sustavom praćenja lica. Bit će riječi o *morphingu*, metodi kojom se vrlo uspješno mogu animirati pokreti ključnih dijelova lica i o *Action Unitima* koji su interni prikaz pokreta lica odabranog sustava koji prati lice. Na samom kraju opisat će se konačno programsko rješenje i način njegova instaliranja i korištenja.

2. Three.js

Kako bi se omogućio prikaz hardverski ubrzane 3D grafike u web pregledniku, a da se ne zahtijevaju programski dodaci, korišten je WebGL (engl. *Web Graphics Library*), točnije Three.js koji je biblioteka temeljena na WebGL-u, a omogućava puno jednostavnije korištenje. WebGL je JavaScript API temeljen na OpenGL ES 2.0. WebGL programi se sastoje od kontrolnog koda napisanog u JavaScriptu i koda za sjenčanje (engl. *shader*) koji se izvršava na grafičkoj jedinici računala (engl. *Graphics Processing Unit*). [2] WebGL je široko rasprostranjen u današnjim web preglednicima kao što su Mozilla Firefox, Google Chrome, Safari, Opera, dok je u Internet Exploreru djelomično podržan.

Three.js je JavaScript 3D biblioteka dostupna na GitHubu koja omogućava lakše korištenje WebGL-a. Biblioteka je otvorenog koda (engl. *open source*). Pomoću Three.js-a mogu se kreirati kamere, objekti, svjetla, materijali i ostalo. Postoji mogućnost odabira načina iscrtavanja (engl. *render*), hoće li se scena iscrtavati koristeći canvas element HTML5 standarda, koristeći WebGL ili SVG (engl. *Scalable Vector Graphics*). WebGL način iscrtavanja iskorištava mogućnosti grafičke kartice, što znači da se skida opterećenje s procesora (engl. *Central Processing Unit*) stoga procesor može procesirati zadatke koji su, na primjer, povezani s korisnikovom interakcijom s web aplikacijom. Izvršavanje web aplikacija koje sadrže 3D grafiku je opterećenje za računalne resurse, stoga je vrlo važno pisati kvalitetan i učinkovit kod.

U nastavku su objašnjeni dijelovi koda ovog rada napisani u Three.js-u, a kojima se omogućava postavljanje scene, kamere, postavljanje modela u scenu i slično.

Dio koda kojim se postavlja scena:

```
// SCENE
scene = new THREE.Scene();
// CAMERA
var SCREEN_WIDTH = 640;
var SCREEN_HEIGHT = 480;
var VIEW_ANGLE = 45, ASPECT = SCREEN_WIDTH / SCREEN_HEIGHT, NEAR =
    0.1, FAR = 20000;
```

```
camera = new THREE.PerspectiveCamera(VIEW_ANGLE, ASPECT, NEAR, FAR);
scene.add(camera);
camera.position.set(0,0,0);
camera.lookAt(scene.position);
// RENDERER
if ( Detector.webgl )
  renderer = new THREE.WebGLRenderer({antialias:true,
     preserveDrawingBuffer:false});
  renderer.setSize(SCREEN_WIDTH, SCREEN_HEIGHT);
```

Nakon što su namješteni scena, kamera i način iscrtavanja (inicijalno je postavljen WebGL način iscrtavanja te je omogućen antialiasing[1]), može se učitati model. Model je zapisan u *Three.js JSON Model formatu*. Nešto više riječi o tom formatu bit će odmah u idućem poglavlju.

Dio koda kojim se učitava model u prethodno postavljenu scenu:

```
var jsonLoader = new THREE.JSONLoader();
jsonLoader.load( "models/robo.json", addModelToScene );
```

U funkciji *addModelToScene*, trebaju se omogućiti *Morph Targeti* koji su detaljno objašnjeni u 4. poglavlju te se trebaju omogućiti boje poligona modela. Model je prekriven *Basic Materialom*, a glavna specifičnost tog materijala je da je vidljiv bez osvjetljenja i podržava boje. Osim *Basic Materiala*, dostupni su *Lambert Material*, *Phong* i neki drugi. Inače, materijali kojima se prekrivaju modeli su jako koristan dio Three.js-a jer se u samom WebGL-u sve što se iscrtava treba sjenčati "ručno", što povlači za sobom da se treba paziti na razne detalje kao što su, na primjer, način refleksije svjetlosti od modela i slično.

```
function addModelToScene( geometry, materials ){
  var material = new THREE.MeshBasicMaterial(
    {
      morphTargets : true,
      vertexColors: THREE.FaceColors
    },
    materials
  );
  robo = new THREE.Mesh( geometry, material );
  robo.scale.set(10,10,10);
    scene.add( robo );
}
```

[1] antialiasing - postupak zaglađivanja nazubljenih rubova oštro iscrtanih krivulja i kosina vidljivih pri velikim povećanjima grafike [3]

Nakon što je i model smješten u scenu, treba se pokrenuti beskonačna petlja koja periodično osvježava (iscrtava) scenu.

```
function animate(){
  requestAnimationFrame( animate );
  render();
}
```

Funkcija za iscrtavanje ovisno o rotaciji i translaciji glave i pomaku dijelova lica rotira i translatira glavu virtualnog lika te pomiče dijelove njegova lica. Informacije o translaciji, rotaciji i pomaku dijelova lica se dobivaju iz objekta faceData razreda FaceData koji je dio Visage|SDK-a, a o njemu će više biti riječi u 6. poglavlju koje će obraditi Visage|SDK. U sljedećem odsječku se vidi, ako je uspješno dodan model u scenu i ako je uspješno instanciran objekt razreda FaceData, da se svaki put kada se osvježi slika, osvježe i koordinate položaja modela. Osim toga, osvježe se i pomaci ključnih dijelova lica, a tome služi lista *morphTargetInfluences* u koje se za svako izobličenje, tj. u ovom slučaju za svaki pomak nekog dijela lica, spremi njegov relativan pomak od neutralnog izraza lica. Ti pomaci su apstrahirani *Action Unitima* u sustavu za praćenje lica, Visage|SDK-u. *Action Unitima* su pridijeljena i imena radi lakšeg snalaženja. U donjem programskom odsječku, u komentarima, mogu se vidjeti imena svih korištenih *Action Unita* (nose wrinkler, lip stretcher i ostali).

```
function render(){
  if (robo && faceData){
    robo.rotation.x = faceData.faceRotation[0];
    robo.rotation.y = faceData.faceRotation[1] * -1;
    robo.rotation.z = faceData.faceRotation[2] * -1;
    robo.position.x = faceData.faceTranslation[0] * 5500;
    robo.position.y = faceData.faceTranslation[1] * 5500;
    robo.position.z = faceData.faceTranslation[2] * -5500;

    robo.morphTargetInfluences[0] = faceData.actionUnits[0];// nose
       wrinkler
    robo.morphTargetInfluences[1] = faceData.actionUnits[6];// lip
       stretcher
    robo.morphTargetInfluences[2] = faceData.actionUnits[11];// brow
        lowerer
    robo.morphTargetInfluences[3] = faceData.actionUnits[10];//
       inner brows raiser
    robo.morphTargetInfluences[4] = faceData.actionUnits[11];// brow
        lowerer
    robo.morphTargetInfluences[5] = faceData.actionUnits[3];//jaw
```

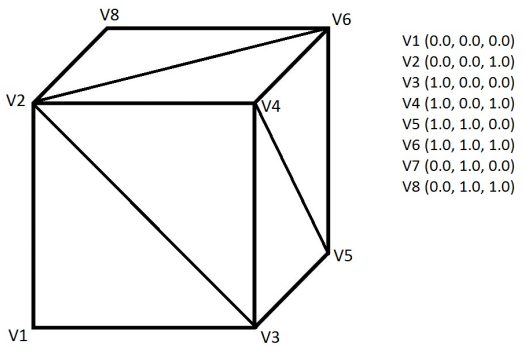
```
        drop
    robo.morphTargetInfluences[6] = faceData.actionUnits[3];//jaw
        drop
    robo.morphTargetInfluences[7] = faceData.actionUnits[9];// outer
         brows raiser
    robo.morphTargetInfluences[8] = faceData.actionUnits[9];// outer
         brows raiser
    robo.morphTargetInfluences[9] = faceData.actionUnits[6];// lip
        stretcher
    robo.morphTargetInfluences[10] = faceData.actionUnits[12];//
        eyes closed
    robo.morphTargetInfluences[11] = faceData.actionUnits[12];//
        eyes closed
    robo.morphTargetInfluences[12] = faceData.actionUnits[10];//
        inner brows raiser
    robo.morphTargetInfluences[13] = faceData.actionUnits[10];//
        inner brows raiser
    renderer.setClearColor(0xe9eaed, 1);
    renderer.render(scene, camera);
  }
}
```

3. Pretvorba OBJ formata u *Three.js JSON Model format*

Najveći problem koji se pojavio pri realizaciji ovog rada pretvorba je OBJ formata zapisa modela, u kojem je bio definiran model koji je bio na raspolaganju (*robo.obj*[1]), u Three.js-ov JSON format *Three.js JSON Model format* koji je trenutno najbolje podržan u toj biblioteci.

OBJ format razvijen je od strane *Wavefront Technologies-a*, otvorenog je tipa i vrlo je često korišten. Sadrži 3D geometriju modela − poziciju svakog vrha u prostoru, popis svih poligona modela i još neke druge informacije koje nisu toliko bitne za ovaj rad. Vrhovi poligona su spremljeni tako da im je redoslijed u smjeru suprotnom kretanju kazaljke na satu gledano izvan tijela, što usmjerava normalu izvan tijela, a ne prema unutrašnjosti što bi bio slučaj kad bi vrhovi bili zadani u smjeru kazaljke na satu. Treba još istaknuti da se vrhovi indeksiraju počevši od jedan.

U nastavku je, za ilustraciju, dan jednostavan primjer tijela i njegovog zapisa u OBJ formatu.[7]



Slika 3.1: Model definiran u datoteci *kocka.obj*

[1]Model je izradio student FER-a, Mihael Grilec, u sklopu svog diplomskog rada pod nazivom *Animacija pokretana praćenjem lica na pokretnim uređajima*, a pod mentorstvom prof.dr.sc. Igora S. Pandžića

kocka.obj

```
v 0.0 0.0 0.0
v 0.0 0.0 1.0
v 1.0 0.0 0.0
v 1.0 0.0 1.0
v 1.0 1.0 0.0
v 1.0 1.0 1.0
v 0.0 1.0 0.0
v 0.0 1.0 1.0
```

```
f 1 3 2
f 3 4 2
f 3 5 4
f 5 6 4
f 5 7 6
f 7 8 6
f 7 1 8
f 1 2 8
f 1 5 3
f 1 7 5
f 2 4 6
f 2 6 8
```

Three.js koristi interni JSON format *Three.js JSON Model format*. Točnije, u ovom radu je korištena njegova verzija 3.1. Na GitHubu u repozitoriju gdje je smješten Three.js dostupna je Python skripta koja pretvara OBJ format u *Three.js JSON Model format* (*convert_obj_three.py*). Pretvorbom, iz datoteke *kocka.obj*, dobiva se datoteka sljedećeg sadržaja:

kocka.json


```
{
  "metadata" : {
  "formatVersion" : 3.1,
  "sourceFile"    : "kocka.obj",
  "generatedBy"   : "OBJConverter",
  "vertices"      : 8,
  "faces"         : 12,
  "normals"       : 0,
  "colors"        : 0,
  "uvs"           : 0,
  "materials"     : 0 },
  "scale" : 1.000000,
  "materials": [{
  "DbgColor" : 15658734,
  "DbgIndex" : 0,
  "DbgName" : "default" }],
  "vertices":   [0.000000,0.000000,0.000000,0.000000,0.000000,1.0000
     00,1.000000,0.000000,0.000000,1.000000,0.000000,1.000000,1.0000
     00,1.000000,0.000000,1.000000,1.000000,1.000000,0.000000,1.0000
     00,0.000000,0.000000,1.000000,1.000000],
  "morphTargets": [],
```

```
  "morphColors": [],
  "normals": [],
  "colors": [],
  "uvs": [[]],
  "faces":  [2,0,2,1,0,2,2,3,1,0,2,2,4,3,0,2,4,5,3,0,2,4,6,5,0,2,6,7
     ,5,0,2,6,0,7,0,2,0,1,7,0,2,0,4,2,0,2,0,6,4,0,2,1,3,5,0,2,1,5,7,
     0]
}
```

Može se primijetiti da su se u listu vertices slijedno redom preslikale sve točke iz *kocka.obj* datoteke. Lista *faces* (lista poligona) je nešto kompleksnija. U njoj su definirani svi poligoni i pojedinačno, za svaki poligon posebno, definirana su neka njegova dodatna svojstva. Prvi član kojim se opisuje poligon predstavlja 8-bitnu masku i on određuje spomenuta dodatna svojstva poligona. Bitovi se u masci promatraju s desna na lijevo i indeks prvog bita je nula. Značenje svih maski je dano tablicom 3.1. [4]

Tablica 3.1: Interpretacije maski u listi faces *Three.js JSON Model formata*

Indeks bita	Vrijednost bita: 0	Vrijednost bita: 1
0	triangle (3 indices)	quad (4 indices)
1	no face material	face material (1 index)
2	no face uvs	face uvs (1 index)
3	no face vertex uvs	face vertex uvs (3 indices or 4 indices)
4	no face normal	face normal (1 index)
5	no face vertex normals	face vertex normals (3 indices or 4 indices)
6	no face color	face color (1 index)
7	no face vertex colors	face vertex colors (3 indices or 4 indices)

Ako se sad konkretno u datoteci *kocka.json* promatra lista *faces*, vidi se da je prvi član liste jednak 2. Binarni zapis broja 2 s 8 bitova je 00000010. Vidi se da je nulti bit postavljen na nulu. Uz pomoć tablice 3.1 otkriva se da se radi o poligonu koji je zadan s tri vrha, tj. kao trokut. Iz tablice se vidi i da odmah nakon maske trebaju slijediti indeksi ta tri vrha i bitno je za primijetiti da se ovdje vrhovi indeksiraju počevši od nule, za razliku od OBJ formata u kojem se vrhovi indeksiraju počevši od jedan. Drugi bit je postavljen na jedan, što znači da se koristi materijal za taj poligon i nakon prethodno zadana tri vrha trokuta treba slijediti indeks tog materijala, a materijali su zadani u listi *materials*. Ostali bitovi, od drugog do sedmog, postavljeni su na nulu, što znači da se ne koriste dodatne mogućnosti koje su navedene u tablici i koje nisu od

posebnog značaja za ovaj rad, osim šestog bita koji definira kojom će se bojom obojati pojedini poligon, a što će biti od koristi kasnije kada bude trebalo obojati poligone modela glave virtualnog lika.

Nakon analize prvog člana koji je 8-bitna maska, vidi se da iza njega slijede, redom kako su postavljani bitovi maske, indeksi vrhova trokuta (poligona) u smjeru suprotnom kretanju kazaljke na satu (0, 2, 1) i indeks materijala (0). Budući da su ostali bitovi nula i da time nisu omogućene nikakve dodatne mogućnosti te stoga više nema potrebe za novim indeksima, prelazi se na novi poligon koji je opet određen svojim prvim članom koji je maska, maska je opet 2 u dekadskom zapisu što znači da ponovno iza maske prvo slijede indeksi vrhova trokuta (2, 3, 1) pa indeks materijala (0) i tako dalje za sve ostale poligone od kojih je sastavljeno tijelo.

Kao što je ranije rečeno, za razliku od ovog jednostavnog modela tijela kojem je šesti bit maske za opis poligona postavljen na nulu, što znači da poligon nije obojan, u modelu glave virtualnog lika šesti će se bit postaviti na jedan. Dodat će se indeks boje, a boje se trebaju nalaziti u listi *colors* u dekadskom zapisu (dekadski zapis se dobije pretvorbom standardnog heksadekadskog RGB načina zapisa boje) i indeksirane su počevši od nule pa će se time dobiti poligon obojan odabranom bojom.

Postoji još jedna znatna razlika modela lica virtualnog lika u odnosu na ovaj jednostavan model kocke na kojem su pojašnjeni OBJ i *Three.js JSON Model format* i razlike među njima, a ta je da je model lica sastavljen od poligona koji su četverokuti, a ne trokuti.

Osim toga za rad će biti posebno važna lista *morphTargets* koja je u ovom slučaju ostavljena praznom, a o tome što su Morph Targeti i na koji način su oni izvučeni iz OBJ formata i prezentirani u *Three.js JSON Model Formatu*, bit će više rečeno u idućim poglavljima.

4. Morphing

Morphing je specijalan efekt u animacijama koji mijenja jedan oblik tijela u drugi. [5] U ovom radu se morphing koristio pomoću Morph Targeta koji su bili definirani u dostupnom modelu. Morph Target je deformirana verzija modela, koji je sada novi model nastao iz izvornog i više nije u neutralnom položaju kao izvorni model, nego ima određeni pomak, npr. u ovom radu izvorni model je model lica virtualnog lika, a Morph Target je model istog tog lica s otvorenim ustima. Između vrhova izvornog modela i vrhova Morph Targeta se interpolira[1] i time dobivamo animaciju, recimo animaciju otvaranja ustiju, animaciju dizanja obrva, zatvaranja očiju i slično.

U OBJ formatu modela koji je korišten u ovom radu, Morph Targeti su bili definirani kao zasebni podmodeli, pomaknuti u prostoru za neki pomak od osnovnog modela koji je sadržavao sve te podmodele u neutralnom položaju. Malo preciznije rečeno, u ishodištu koordinatnog sustava prostora, u kojem promatramo model, bio je smješten čitav neutralni model, sa svim svojim podmodelima (oči, usta, nos i slično), a onda su ti podmodeli bili deformirani tako da predstavljaju maksimum neke kretnje od neutralnog položaja (npr. do kraja podignute obrve ili do kraja otvorena usta) i pomaknuti zasebno, svaki za sebe po x koordinati za neku vrijednost.

Budući da *Three.js JSON Model format* u listu *MorphTargets* kao članove prima Morph Targete koji su određeni svim točkama modela, a ne samo podmodele tj. specifične dijelove modela, trebalo je identificirati gdje se točno nalazi koji dio lica u neutralnom položaju (oči, nos, usta) među svim vrhovima modela i zamijeniti ih s deformiranim vrhovima, s time da se kopiraju i svi ostali vrhovi koji nisu deformirani. Dakle, za svaki od četrnaest Morph Targeta, koliko ih je bilo definirano u OBJ datoteci, bilo je potrebno u listu koja ih prima kopirati sve točke modela u neutralnom položaju, identificirati ključne dijelove lica modela te ih zamijeniti s deformiranim inačicama. O samom tom postupku će se baviti iduće poglavlje.

U nastavku je konkretno definiran jedan Morph Target na kocki iz primjera iz pret-

[1] Interpolacija u prostoru je parametarsko određivanje koordinata na temelju definiranih parametara nad nekim točkama.[8]

hodnog poglavlja. Listi *MorphTargets* je pridijeljen jedan član, definiran imenom i vrhovima te implicitno definiran indeksom nula, budući da je prvi u listi.

```
"morphTargets": [
  {
    "name": "morphTargetExample",
      "vertices":[-2.000000,0.000000,0.000000,0.000000,0.000000,1.00
          0000,1.000000,0.000000,0.000000,1.000000,0.000000,1.000000,
          1.000000,1.000000,0.000000,1.000000,1.000000,1.000000,0.000
          000,1.000000,0.000000,0.000000,1.000000,1.000000]
    }]
```

Pristup Morph Targetu se omogućuje preko podatkovnog člana objekta kojim je definiran model, a koji je lista nazvana *morphTargetInfluences*. U Three.js-u se Morph Targetima u listi *morphTargetInfluences* može pristupati indeksima koji počinju od nule ili im se može pristupati imenom Morph Targeta. U ovom im se radu pristupa indeksima i postavlja im se brojčana vrijednost *X*, koja označava za koliko (*X**100%) se treba linearno interpolirati između neutralnog i deformiranog modela.

Na slici 4.1 a) je prikazan neutralan model (*kocka.json*),

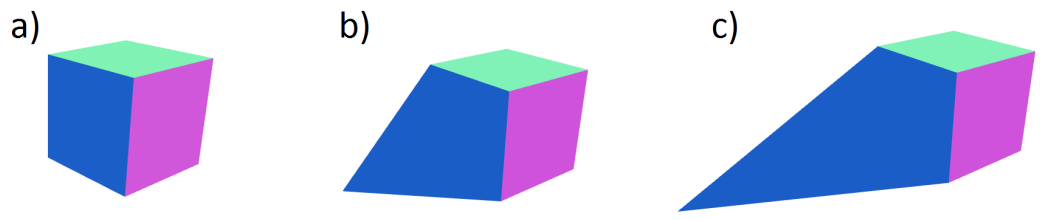
```
kocka.morphTargetInfluences[0] = 0;
```

na slici 4.1 b) je prikazan model interpoliran za 50% između neutralnog modela i Morph Targeta,

```
kocka.morphTargetInfluences[0] = 0.5;
```

a na slici 4.1 c) je prikazan model interpoliran za 100% između neutralnog modela i Morph Targeta.

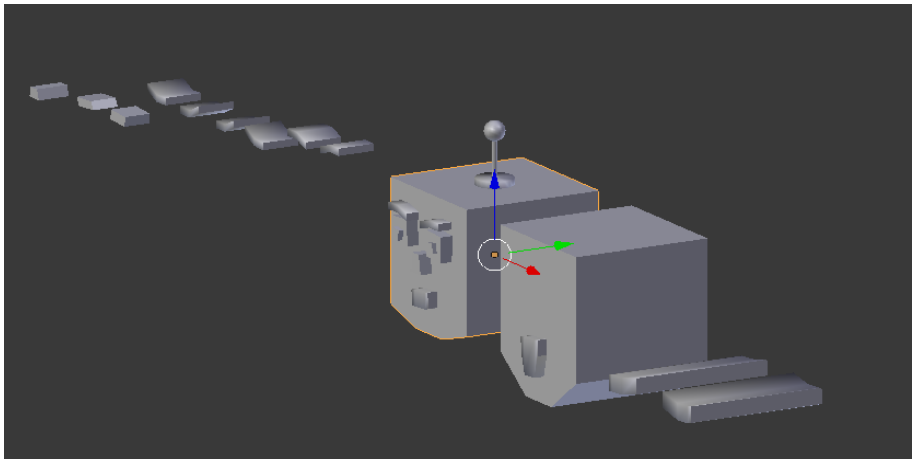
```
kocka.morphTargetInfluences[0] = 1;
```



Slika 4.1: Morphing na primjeru modela definiranog u datoteci *kocka.json*

5. Priprema dostupnog modela (*robo.obj*) za korištenje u Three.js-u

Dalje će se objasniti postupak kojim je iz OBJ formata dostupnog modela (*robo.obj*) dobiven model zapisan u Three.js-ovom JSON formatu sa svim Morph Targetima i bojama. Na početku treba pokazati kako je grafički izgledao dostupan model – slika 5.1. Za to će se koristiti Blender[1]. Ovdje se lijepo vidi ono što je rečeno u uvodu prethodnog poglavlja o morphingu. Neutralni model je u ishodištu koordinatnog sustava scene, dok su Morph Targeti prikazani kao zasebni podmodeli pomaknuti po x koordinati. Može se primijetiti još da su model i podmodeli bezbojni (inicijalno sivi zbog materijala koji se predefinirano koristi) jer OBJ ne podržava boje bez posebnih dodataka. Model će se obojati tek kad se konvertira u Three.js-ov JSON format, jer za sad za time još nema potrebe.

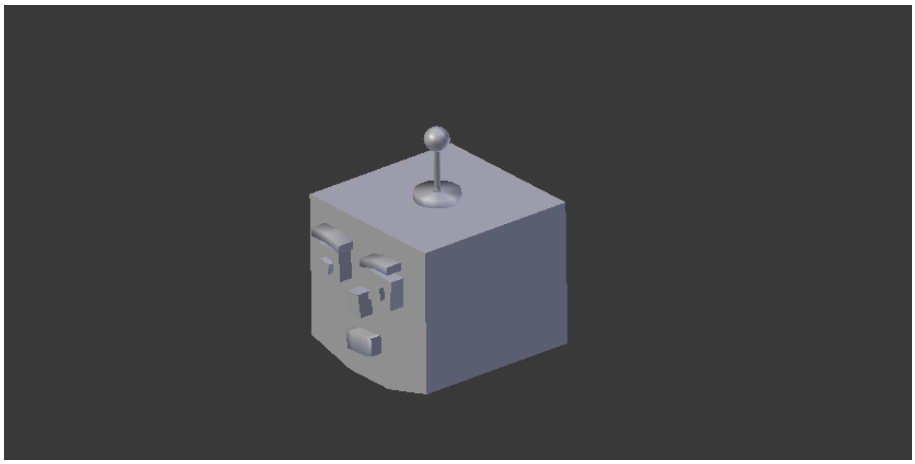


Slika 5.1: Izvorni model u OBJ formatu (*robo.obj*)

Sada se izbrišu svi podmodeli iz scene, tako da ostane samo neutralan model u ishodištu trodimenzionalnog koordinatnog sustava scene, i to se spremi u neku drugu

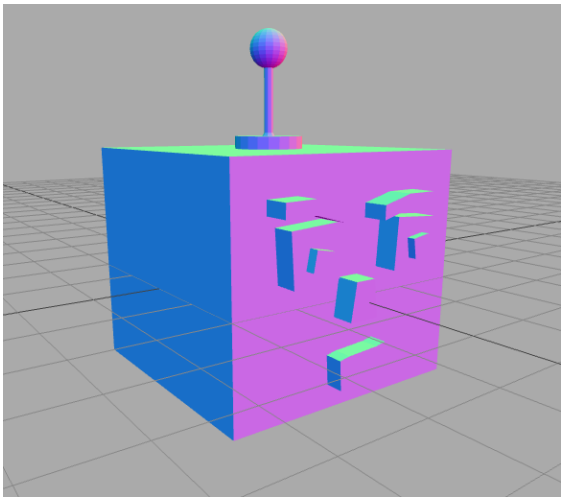
[1] Blender je besplatan skup alata za pregled i izradu interaktivnog 3D sadržaja pod različitim operativnim sustavima

OBJ datoteku, nazvanu, na primjer, *robo_neutral.obj*. Rezultat se može vidjeti na slici 5.2.



Slika 5.2: Neutralan model bez podmodela koji su predstavljali Morph Targete

Nakon ovoga se model definiran u datoteci *robo_neutral.obj* može konvertirati Python skriptom koja pretvara OBJ format u *Three.js JSON Model format* (*convert_obj_three.py*), a koja je dostupna na GitHubu u repozitoriju u kojem je smješten Three.js. Dobivena je datoteka *robo.json*. Na slici 5.3[2] je grafički prikazan model definiran u datoteci *robo.json*, a prekriven je *Normal Materialom*. Lista *morphTargets* je očekivano ostala prazna jer su uklonjeni svi podmodeli koji su predstavljali Morph Targete, a da su bili ostavljeni, ne bi se dobilo ništa korisno jer skripta ne može iz ovakve OBJ datoteke razlučiti Morph Targete i popuniti listu *morphTargets*.



Slika 5.3: Model definiran u *robo.json* datoteci

[2]Slika je dobivena korištenjem Three.js editora dostupnog on-line na stranici http://threejs.org/editor/

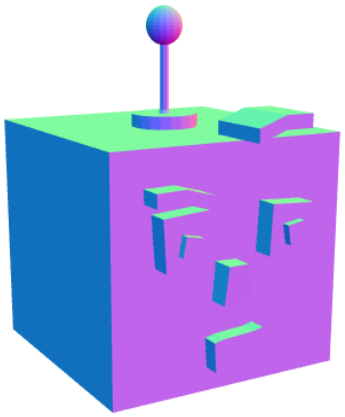
Idući korak je kopirati sve vrhove iz liste *vertices* koja se nalazi u novonastaloj datoteci *robo.json* u drugu listu *vertices* koja je podlista liste *morphTargets*. Potrebno je dati neko ime Morph Targetu u asocijativnom polju *name*. To je prikazano sljedećim isječkom.

```
{
    "metadata" : {
    //...
    },

    "scale" : 1.000000,

    "materials": [
    //...
    ],

    "vertices": [//...], // <- misli se na ovu listu "vertices", a
       ne na donju koja pripada listi "morphTargets"

    "morphTargets": [
    {
      "name": "morphTarget1",
        "vertices":[//ovdje dolaze svi vrhovi iz liste "vertices",
           koja se nalazi iznad, izvan liste "morphTargets"]
      }]

    "morphColors": [],

    "normals": [],

    "colors": [],

    "uvs": [[]],

    "faces": [//...]

}
```

Sada je potrebno u listi *vertices* koja se nalazi unutar liste *morphTargets* pronaći točne vrhove koji predstavljaju pojedini podmodel. Budući da je u datoteci *robo.obj* svaki podmodel (onaj podmodel koji predstavlja Morph Target, ali i onaj koji predstavlja dio neutralnog modela) nazvan prigodnim imenom na što je trebao misliti autor modela, mogu se identificirati njegovi točni vrhovi u listi *vertices* koja je podlista liste

morphTargets. Uzmimo za primjer, za lakše snalaženje u opisu ovog postupka, da je lijeva obrva (*robo_l_eyebrow*) podmodel koji se traži u neutralnom položaju (i koji je nađen!). U listi *vertices*, koja pripada listi *morphTargets*, se izbrišu vrhovi pronađenog podmodela, s tim da je prije brisanja potrebno zapamtiti x koordinate izbrisanih točaka jer su Morph Targeti pomaknuti za neki pomak po x-u, a ovim postupkom se trebaju namjestiti na neutralan model koji je u ishodištu. Sada preostaje u datoteci *robo.obj* pronaći Morph Target (podmodel) za lijevu obrvu koji je isto prigodno nazvan (*robo_blend_shapes_brow_raise_l_eyebrow_lowerer*) i vrhovi tog Morph Targeta se kopiraju na mjesto u podlisti *vertices* liste *morphTargets* na kojem su stajali vrhovi podmodela u neutralnom položaju, s time da se x koordinate točaka Morph Targeta zamijene s prethodno zapamćenim x koordinatama točaka podmodela u neutralnom položaju koji je izbrisan. Ovim postupkom smo dobili jedan Morph Target u *Three.js JSON Model formatu*. Analogno se napravi i za sve ostale.

Na slici 5.3 je, u Google Chrome-u, prikazan model *robo* s interpolacijom između neutralnog modela i Morph Targeta *robo_blend_shapes_brow_raise_l_eyebrow_lowerer* od 100%. Model je, dok još nije obojan, zbog bolje vidljivosti, prekriven *Normal Materialom*.



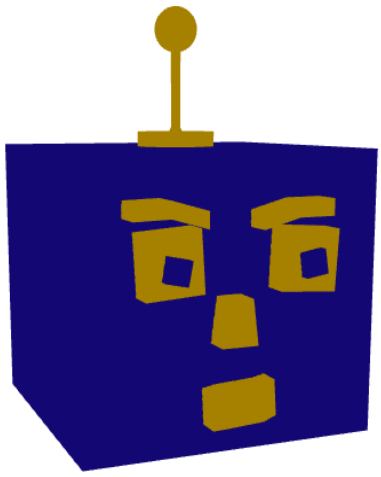
Slika 5.4: Morphing na modelu *robo-a*

Slijedi bojanje modela. Ranije je bilo riječi o dodatnim mogućnostima kod poligona. U listu *colors* će se dodati dvije boje.

```
"colors": [1247347, 10912000]
```

Na odgovarajući način će se uz pomoć tablice 3.1 odrediti maska za svaki poligon tako da se omogući bojanje poligona te će se pridijeliti odgovarajući indeks boje odgovara-

jućem poligonu u listi *faces*.



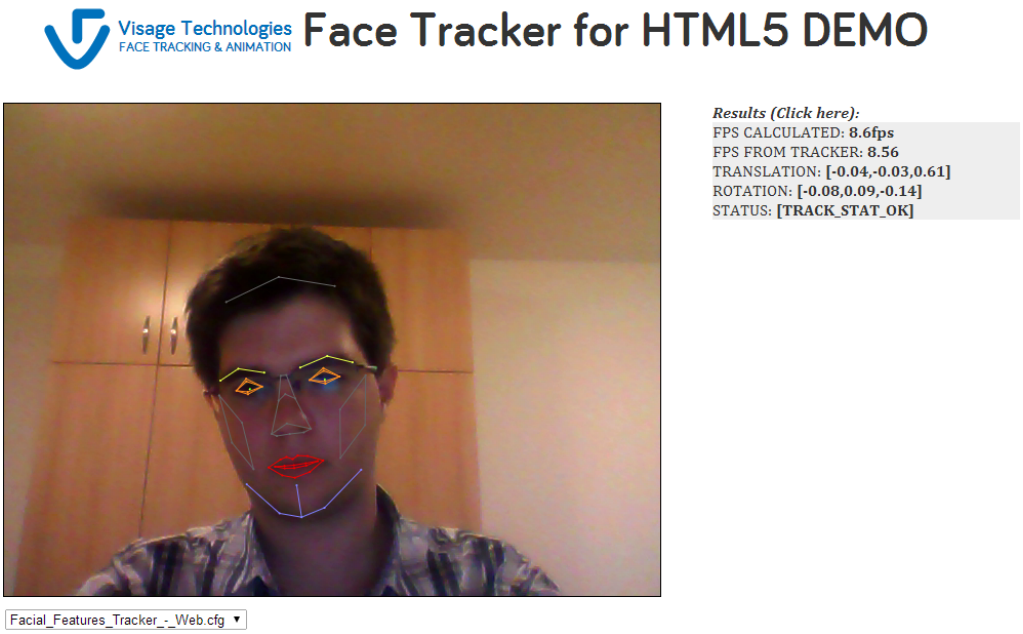
Slika 5.5: Obojani model *robo* u Google Chrome-u

Nakon što je model pripremljen za korištenje u Three.js-u, može se povezati sa sustavom za praćenje lica, Visage|SDK-om.

6. Visage|SDK

Visage|SDK povezuje opsežan broj računalnih tehnologija za animaciju likova i računalni vid u skup razvojnih alata koji se lako koriste i imaju široko područje primjene u različitim aplikacijama, kao što su animacija virtualnih likova, stvaranje virtualne ili proširene stvarnosti, praćenje i analiza pokreta lica. [6] Postoji već gotova verzija Visage|SDK-a za HTML5 koja se vrlo lako može integrirati sa sustavom napravljenim u Three.js-u.

U radu je korišten dio Visage|SDK-a, FaceTracker. FaceTracker prati pokrete glave i lica iz video zapisa koji mu daje kamera ili neki drugi izvor. Tracker daje razne izlazne informacije kao što su 3D položaj glave, izraz lica, smjer gledanja, položaj ključnih točaka lica i potpuni 3D model lica. Izlaznim informacijama se pristupa pomoću objekta faceData, razreda FaceData, koji služi kao spremnik tih svih informacija.



Slika 6.1: FaceTracker za HTML5

Za realizaciju ovog rada je potrebna informacija o translaciji i rotaciji glave. Do

tih podataka se dolazi preko podatkovnih članova razreda FaceData, faceTranslation-a i faceRotation-a.

Osim translacije i rotacije potreban je i izraz lica; treba se detektirati koji se dio promatranog lica miče kako bi se mogao pomaknuti isti taj dio lica virtualnog lika. Izrazi lica su dostupni u obliku *Action Unita* koji su interna reprezentacija kretnji lica koje koristi Tracker. Pristupa im se pomoću podatkovnog člana *actionUnits*, razreda FaceData. Podatkovni član *actionUnits* je lista u kojoj su sadržane trenutne vrijednosti svih definiranih *Action Unita*. *Action Unite* izravno možemo povezati s podatkovnim članom učitanog modela *robo-a*, listom *morphTargetInfluences*. Donjom naredbom je prvom Morph Targetu pridružen *Action Unit* s indeksom nula, a zove se *Nose wrinkler*.

```
robo.morphTargetInfluences[0] = faceData.actionUnits[0];// nose
    wrinkler
```

U tablici 6.1 [9] je popis svih *Action Unita* koje dolaze u standardnoj konfiguraciji VisagelSDK-a.

Tablica 6.1: Action Units

AU1: Nose wrinkler
AU2: Jaw z-push
AU3: Jaw x-push
AU4: Jaw drop
AU5: Lower lip drop
AU6: Upper lip drop (AU10)
AU7: Lip stretcher (AU20)
AU8: Lip corner depressor (AU13/15)
AU9: Lip presser (AU23/24)
AU10: Outer brows rasier
AU11: Inner brows raiser
AU12: Brow lowerer
AU13: Eyes closed (AU42/43/44/45)
AU14: Lid tightener (AU7)
AU15: Upper lid rasier (AU5)
AU16: Rotate eyes left

7. Konačno rješenje

U ovom poglavlju će biti prikazano konkretno ostvarenje aplikacije. Izvorni kod aplikacije nalazi se u datoteci HTML formata (*zavrsni_rad.html*). Ključni dio te HTML datoteke sastoji se od dva dijela JavaScript koda. Prvi dio koristi biblioteku Three.js i tu se postavljaju scena, kamera i ostalo što je objašnjeno u poglavlju 2, a drugi dio JavaScript koda je preuzet iz FaceTracker Sample-a koji dolazi s Visage|SDK-om, a prikazuje žičanu masku lica preko lica koje se prati i taj dio služi za usporedan prikaz lica koje se miče u realnom vremenu i modela lica virtualnog lika.

Model koji se koristi u aplikaciji je smješten u *robo.json* datoteci. Način dobivanja datoteke tog formata iz datoteke OBJ formata prikazan je u poglavlju 5, a njena struktura je objašnjena na jednostavnom primjeru u poglavlju 3.

Preduvjet za pokretanje i korištenje aplikacije je instalirana kamera na računalu. Aplikacija je testirana na operacijskom sustavu Microsoft Windows 8.1. Korišten je virtualni server WampServer 2.5 koji je open source. U mapu www WampServera, u koju virtualni server sprema datoteke koje će prikazivati u Internet pregledniku, stavljene su sve biblioteke i pomoćne datoteke (izvorni kod Three.js-a, datoteke koje su potrebne pri radu Visage|SDK-a i slično), izvorni kod aplikacije i *robo.json* datoteka u kojoj je opisan model. Nakon pokretanja WampServera u odabrani Internet preglednik (aplikacija je testirana u Google Chrome-u) se upiše URL koji vodi do glavne HTML datoteke (*zavrsni_rad.html*), npr., *http://localhost/zavrsni_rad/robo/robo/zavrsni_rad .html*. Web preglednik postavlja upit za dozvolu korištenja kamere i nakon što mu se to omogući, pokreće se aplikacija.

Aplikacija se sastoji od dva canvas elementa koji su definirani novim HTML5 standardom. U lijevom canvas elementu se nalazi scena virtualnog lika, *robo-a*, a u desnom scena koju snima kamera, a u kojoj se treba nalaziti ljudska glava koja se prati. Kada sustav za praćenje lica, Visage|SDK, prepozna lice u sceni koju snima kamera, na istom tom mjestu u sceni virtualnog lika pozicionira se glava virtualnog lika. Dalje se glava virtualnog lika translatira, rotira te se pomiču dijelovi lica ovisno o tome kako iste te kretnje izvodi i ljudska glava koja se snima kamerom, tj. prati sustavom za praćenje

lica, Visage|SDK-om.

Na slici 7.1 je prikazan rad aplikacije.

Slika 7.1: Rad aplikacije

8. Zaključak

U sklopu ovog rada uspješno je implementirana web aplikacija pokretana praćenjem lica pomoću trenutno najizglednije tehnologije za rješavanje problema 3D sadržaja na webu − WebGL-a. Iako je korištena biblioteka Three.js zasnovana na WebGL-u koja je relativno mlada, u razvoju i u eksperimentalnoj fazi, sve iznenađujuće dobro radi u web pregledniku. Može se jedino dati mala zamjerka na brzinu izvođenja. Pažljiviji korisnik će primijetiti malo kašnjenje (engl. *delay*) u canvas elementu u kojem je prikazana scena koja se snima kamerom. To bi se vrlo vjerojatno moglo riješiti daljnom optimizacijom koda jer je poznato da je izvršavanje 3D aplikacija opterećenje za računalne resurse pa je vrlo važno pisati kvalitetan i učinkovit kod, tj. vrlo je važno paziti i na najsitnije detalje. Vrlo vjerojatno će se taj problem ublažiti i daljnjim razvojem WebGL-a i biblioteke Three.js. Jedan od najvećih problema u ovom radu je predstavljala pretvorba OBJ formata u *Three.js JSON Model format* koja je većim dijelom izvedena "ručno", a postupak je opisan u 5. poglavlju. Inače je dosta teško konvertirati bilo koji format zapisa 3D modela u neki drugi, pa je bilo predvidljivo da će to zadavati probleme. Taj bi se postupak mogao u potpunosti automatizirati, pisanjem učinkovite skripte, ali to prelazi okvire ovog rada. Sve u svemu, rad je vrlo uspješno priveden kraju, ostvareni su svi ciljevi zadani na početku, s većom ili manjom učinkovitošću. Three.js se pokazao kao odlična biblioteka za rad s 3D grafikom na web-u, Visage|SDK je kao sustav praćenja lica također dao vrlo dobre rezultate, a sve skupa ukomponirano u jednu web aplikaciju koja prati lice i pokreće virtualni lik izgleda jako dobro, iako ima još dosta prostora za razna poboljšanja.

LITERATURA

[1] HOTLab. Teme za studentske radove: 3D grafika na webu. Datum zadnje promjene: 14.01.2014. URL http://hotlab.fer.hr/HOTlab/students_hr/teme. Datum pristupa: 28.05.2014.

[2] Računalna animacija na Webu. Godina nastanka: 2013. URL http://www.zemris.fer.hr/predmeti/irg/Zavrsni/13_Komar/sadrzaj_webGL.html. Datum pristupa: 28.05.2014.

[3] anti-aliasing. Datum zadnje promjene: 15.05.2010. URL http://hr.wiktionary.org/wiki/anti-aliasing. Datum pristupa: 28.05.2014.

[4] JSON Model format 3. Datum zadnje promjene: 11.05.2014. URL https://github.com/mrdoob/three.js/wiki/JSON-Model-format-3. Datum pristupa: 28.05.2014.

[5] Morphing. Datum zadnje promjene: 19.05.2014. URL http://en.wikipedia.org/wiki/Morphing. Datum pristupa: 28.05.2014.

[6] VisagelSDK. URL http://www.visagetechnologies.com/products/visagesdk/. Datum pristupa: 28.05.2014.

[7] Čupić M., Mihajlović Ž.. Interaktivna računalna grafika. Zadatci za laboratorijske vježbe. radna verzija, 2013.

[8] Mikuš B. Programi za sjenčanje. Završni rad. Fakultet elektrotehnike i računarstva. 2010.

[9] *VisageTracker Configuration Manual*. Visage Technologies AB. inačica 2.0.

Animacija pokretana praćenjem lica kao web aplikacija

Sažetak

Praćenje lica je postupak kojim se tehnikama računalnog vida slijedi ljudsko lice i njegove ključne točke u video slici. Ta je tehnologija u okviru ovog rada korištena za pokretanje animacije lica virtualnog lika. Animacija je ostvarena *morphingom*, metodom kojom se interpolira između neutralnog i deformiranog tijela. Grafički sustav koji je korišten pri izradi aplikacije je biblioteka Three.js temeljena na WebGL-u, a korišten sustav za praćenje lica je Visage|SDK. Grafički sustav je na odgovarajući način povezan sa sustavom praćenja lica. Model dostupnog virtualnog lika je bio izvorno definiran u OBJ formatu, pa je pretvoren u *Three.js JSON format* kako bi se mogao koristiti u Three.js-u.

Ključne riječi: pokretanje animacije lica virtualnog lika, praćenje lica, WebGL, Three.js, HTML5, Visage|SDK, Morphing, Morph Target, Action Unit, Three.js JSON Model format, OBJ format

Performance-driven animation as a web application

Abstract

Face tracking is a process where computer vision is used to locate human face and its feature points in video image. In this paper, face tracking is used to gain motion in the face of animated virtual character. *Morphing* is used in the process of animation. *Morphing* is a method where interpolation between neutral and deformed body is made. Graphic system used to build the application is Three.js library based on WebGL. Face tracking system used to build the application is Visage|SDK. Graphic system is compatible with face tracking system. Model of the virtual character was initially defined in OBJ format, then it was converted to *Three.js JSON format* so it could be used in Three.js.

Keywords: Performance Animation, Face Tracking, WebGL, Three.js, HTML5, Visage|SDK, Morphing, Morph Target, Action Unit, Three.js JSON Model format, OBJ format